集成开发环境(CodeArts IDE)
2.5.0
用户指南

文档版本 01

发布日期 2025-10-30

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 快速入门

1.1 使用 CodeArts IDE for Java 快速创建 Java 工程

CodeArts IDE for Java是一个Java集成开发环境，将文本编辑器和强大的开发工具（如智能代码补全、导航、重构和调试）集成在一起。

本示例为您演示如何使用CodeArts IDE for Java快速创建Java工程。

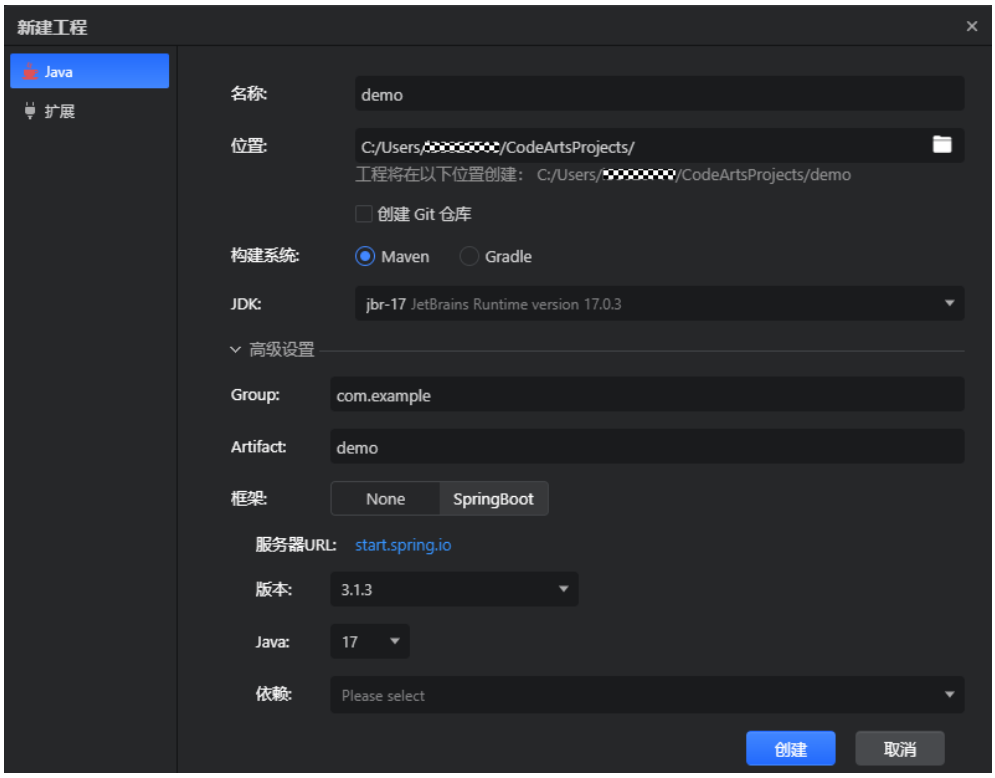
前置条件

- 已下载并安装CodeArts IDE for Java。
- 已安装Java JDK 1.8及其以上版本。
- 引入spring-boot-starter-web、spring-boot-starter-thymeleaf和lombok（可选）相关依赖。

创建 Java 工程

步骤1 在本地CodeArts IDE页面，选择“新建工程”创建Java工程。

其中“构建系统”选择“Maven”，“框架”选择“SpringBoot”。



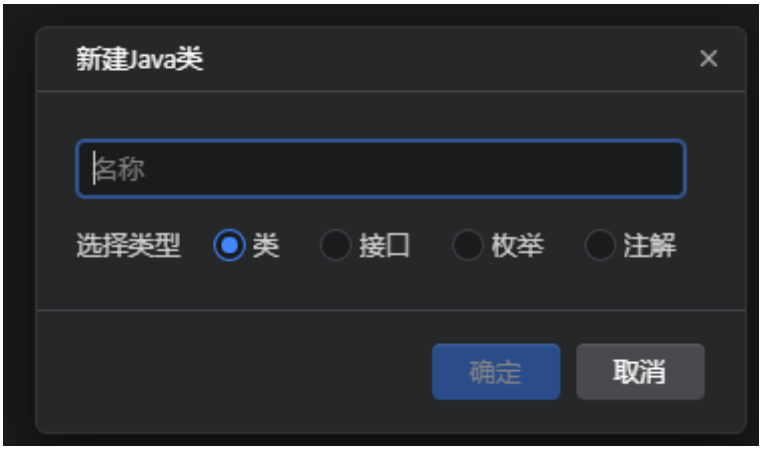
步骤2 单击“创建”，项目开始加载，语言服务初始化过程中会启动相关服务、下载依赖的Jar包及进行Indexing，此过程受计算机性能、网速等因素影响会耗费一定的时间。

----结束

编辑代码

步骤1 创建Java类。

1. 在项目资源管理器中，右键单击src/main/java/com/example目录。
2. 选择“新建>Java类”，在弹出的对话框中输入类名，例如Main。单击“确定”。

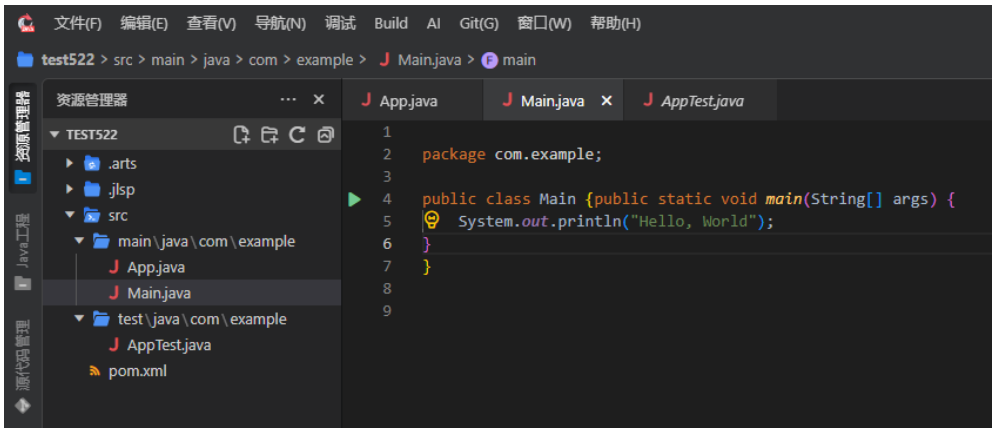


步骤2 编辑Java类。

1. 双击Main.java文件，打开编辑器。

2. 输入以下简单的Java代码：

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

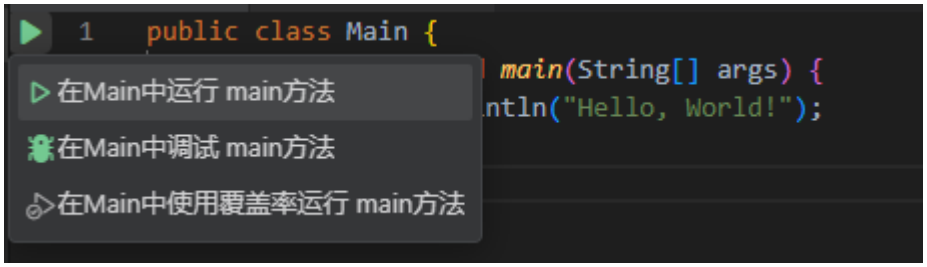


3. 按 **Ctrl + S** 保存文件，或者单击顶部菜单中的“文件 > 保存”。

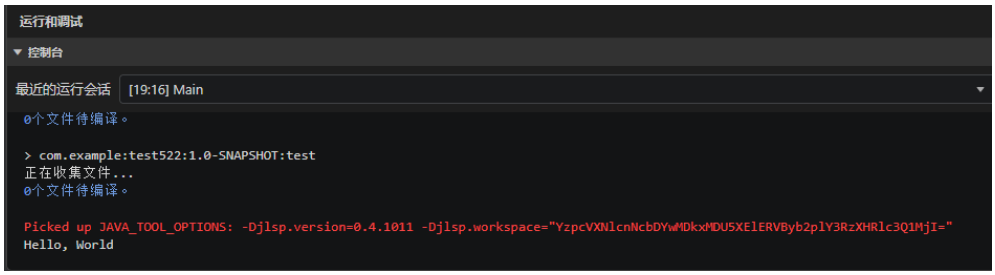
----结束

运行代码

步骤1 在编辑器窗口，单击“public class Main”代码行左侧的运行按钮，选择“在Main中运行 main方法”。



步骤2 运行结果会显示在底部的窗口中，会看到 Hello，World的输出。



----结束

调试代码

步骤1 **设置断点**：在需要调试的代码行左侧单击，设置断点（会出现一个红点）。

步骤2 **启动调试**：单击“public class Main”代码行左侧的运行按钮，选择“在Main中调试 main方法”。

步骤3 使用调试工具栏中的按钮来控制调试过程。

----结束

1.2 入门实践

当您完成了CodeArts IDE客户端的下载、安装和激活，可以根据自身的业务需求使用CodeArts IDE提供的一系列常用实践。

本文汇总了基于CodeArts IDE常见应用场景的操作实践，为每个实践提供详细的方案描述和操作指导。

表 1-1 CodeArts IDE 最佳实践一览表

最佳实践	说明
使用CodeArts IDE for Cpp开发OpenGl示例工程	本实践向您介绍如何基于CodeArts IDE快速创建简单的C++工程，CodeArts IDE面向开发者提供的智能化可扩展桌面集成开发环境，结合行业和产业开发套件，实现一站式开发体验。
使用CodeArts IDE for Java开发简单的Java工程	本实践向您介绍如何基于CodeArts IDE快速创建简单的Java工程，CodeArts IDE for Java是一个JAVA集成开发环境，将文本编辑器和强大的开发工具（如智能代码补全、导航、重构和调试）集成在一起。

2 用户指南

2.1 下载、安装和卸载 CodeArts IDE 客户端

2.1.1 登录 CodeArts IDE 运营面

步骤1 使用浏览器，以VDC管理员或VDC业务员**登录ManageOne运营面**。

步骤2 在页面左上角单击☰，打开服务列表。

步骤3 单击服务列表中的“软件开发生产线 > 集成开发环境 CodeArts IDE”进入CodeArts IDE运营面控制台。

如果登录用户第一次进入CodeArts IDE运营面控制台，页面将弹出提示框，单击“同意并继续”即可继续使用服务。

----结束

2.1.2 下载 CodeArts IDE 客户端

步骤1 在CodeArts IDE运营面首页左侧选择“CodeArts IDE > 下载客户端”。

步骤2 在下图所示的页面中，可以选择“Cpp”或者“Java”这两种开发环境的客户端。

步骤3 单击下图所示页面中间的“下载”按钮，选择Windows操作系统或者Linux操作系统的安装包进行下载。

----结束

2.1.3 安装 CodeArts IDE 客户端

Cpp客户端与JAVA客户端安装过程相同，本文以Cpp客户端为例，在不同操作系统安装。

- **Windows版本安装**。
- **Linux Arm版本安装**。
- **Linux X86版本安装**。

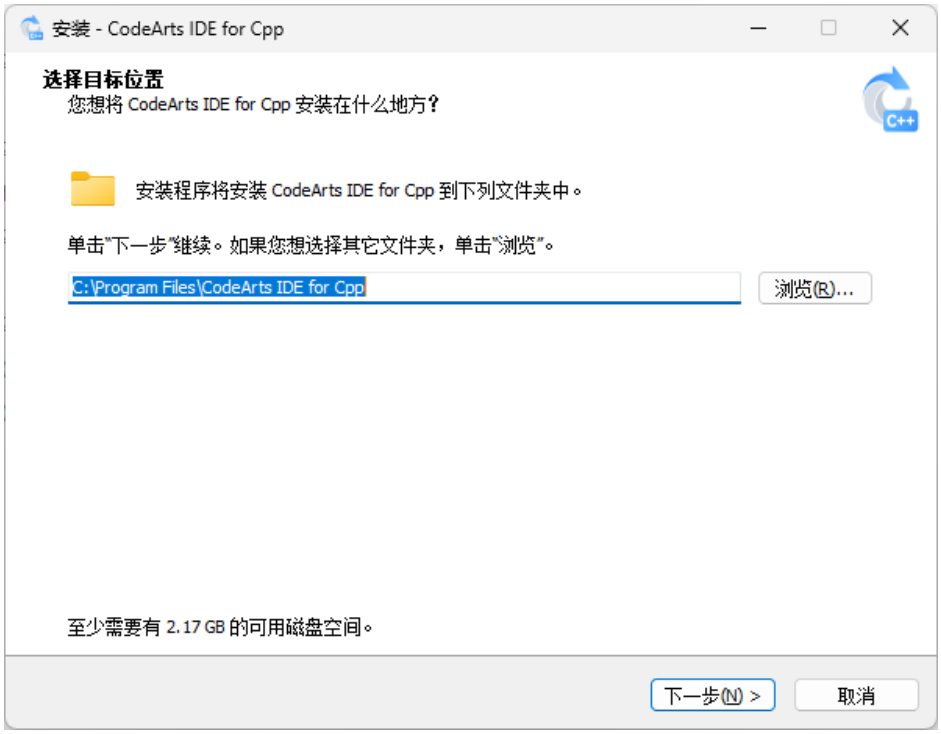
2.1.3.1 Windows 版本安装

约束限制

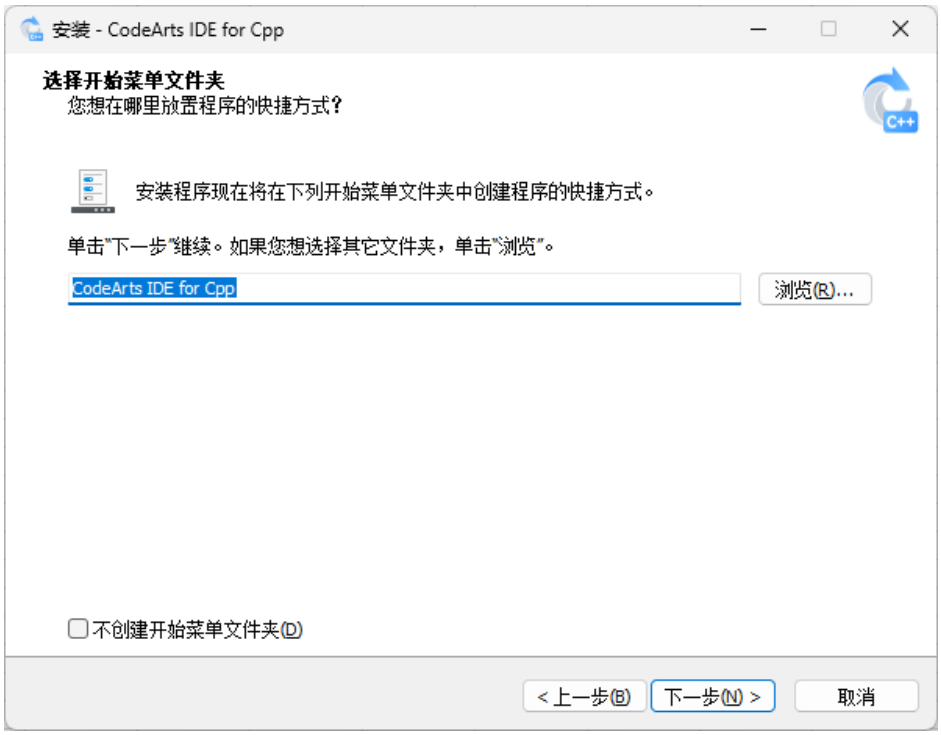
- 参考**下载CodeArts IDE客户端**，下载Windows版本CodeArts IDE安装包。
- 在安装软件包时，建议为解压和安装过程预留至少2.15GB的磁盘空间。

安装步骤

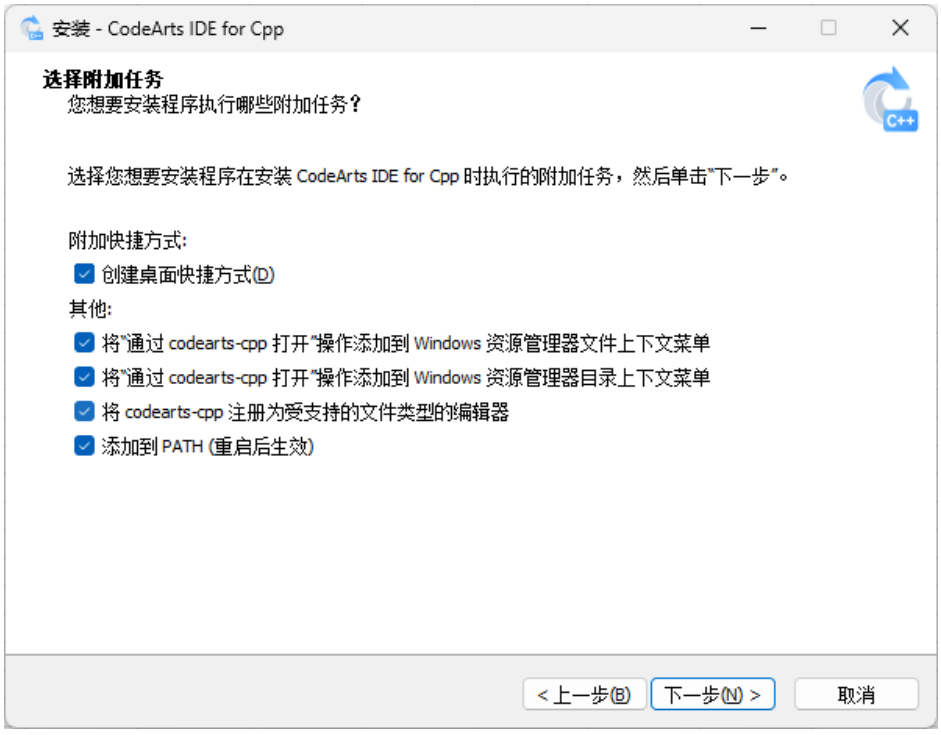
步骤1 选择目标位置，单击“下一步”。



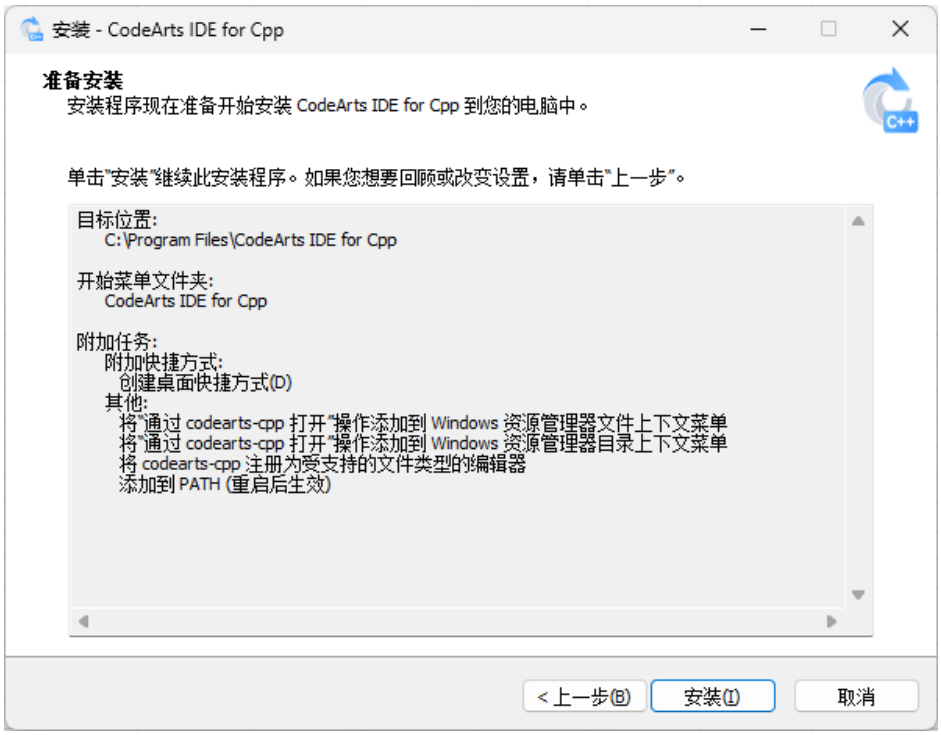
步骤2 选择开始菜单文件夹，单击“下一步”。



步骤3 按需选择附加任务，单击“下一步”。



步骤4 单击“安装”按钮，等待安装完成。



步骤5 安装完成后显示“CodeArts IDE for Cpp安装完成”，单击“完成”按钮，退出安装程序。

----结束

2.1.3.2 Linux Arm 版本安装

约束限制

- Linux机器需要具有图形化界面的操作系统。
- 参考**下载CodeArts IDE客户端**，下载Linux Arm版本CodeArts IDE安装包。
- 在安装软件包时，建议为解压和安装过程预留至少1GB的磁盘空间。

安装和启动步骤

步骤1 以具备sudo权限的用户（非root用户）登录Linux机器。

步骤2 执行**cd ~/Downloads**进入安装包所在目录，此命令表示安装包所在目录为“/home/用户名/Downloads”。

步骤3 执行**sudo dpkg -i codearts-cpp-arm64-linux-xxx.deb**安装客户端，此命令中的安装包名需替换为实际安装包名。

步骤4 安装完成后，在终端窗口根据安装包的版本（C/C++或JAVA）执行如下对应命令，启动CodeArts IDE客户端。其中，C/C++客户端执行**codearts-cpp**，JAVA客户端执行**codearts-java**。

----结束

2.1.3.3 Linux X86 版本安装

约束限制

- Linux机器需要具有图形化界面的操作系统。
- 参考**下载CodeArts IDE客户端**，下载Linux X86版本CodeArts IDE安装包。
- 在安装软件包时，建议为解压和安装过程预留至少1GB的磁盘空间。

安装步骤

步骤1 以具备sudo权限的用户（非root用户）登录Linux机器。

步骤2 执行**cd ~/Downloads**进入安装包所在目录，此命令表示安装包所在目录为“/home/用户名/Downloads”。

步骤3 执行**sudo dpkg -i codearts-cpp-x64-linux-xxx.deb**安装客户端，此命令中的安装包名需替换为实际安装包名。

步骤4 安装完成后，在终端窗口根据安装包的版本（C/C++或JAVA）执行如下对应命令，启动CodeArts IDE客户端。其中，C/C++客户端执行**codearts-cpp**，JAVA客户端执行**codearts-java**。

----结束

2.1.4 卸载 CodeArts IDE 客户端

Windows系统可参考以下方式卸载客户端并删除数据，以CodeArts IDE for Java为例。

步骤1 打开“控制面板”，单击“卸载程序”，找到已安装的CodeArts IDE for Java程序，右键单击“卸载”，在弹窗中单击“是”即可完成卸载。

步骤2 如果想要进一步删除用户数据和自定义安装的插件数据，删除文件夹“**%APPDATA%/codearts-java**”和“C:\Users\用户名\.codearts-java”即可。

----结束

Linux系统可参考以下方式卸载客户端并删除数据，以CodeArts IDE for Java为例。

步骤1 输入**sudo apt-get purge codearts-java**命令卸载CodeArts IDE客户端。

步骤2 如果想要进一步删除用户数据和自定义安装的插件数据，输入如下命令删除对应文件夹：

```
rm -rf ~/.config/codearts-java
rm -rf ~/.codearts-java
```

----结束

2.2 激活管理

2.2.1 申请激活码

步骤1 参考**登录CodeArts IDE运营面**，在CodeArts IDE运营面首页左侧选择“CodeArts IDE > 激活管理”进入“激活管理”页面，再单击右上角的“申请激活码”。

步骤2 下图所示，输入需要申请的激活码数量，再单击当前页面右下角的“立即申请”即可申请成功，并自动跳转到“激活管理”页面查看已申请的激活码。

----结束

2.2.2 绑定激活码并激活

步骤1 以激活CodeArts IDE for Cpp为例，打开已下载的CodeArts IDE客户端，会弹出提示并提供“机器码”，单击下图中的复制按钮，复制机器码，如**图2-1**。

图 2-1 复制机器码

步骤2 根据**申请激活码**，打开CodeArts IDE运营面首页的“激活管理”页面，选择上述**申请激活码**申请的激活码，如下图所示，单击右侧**“绑定”**，粘贴上一步获取的机器码，完成绑定。

图 2-2 绑定机器码

步骤3 如**图2-3**，从运营面的下载客户端页面复制“激活服务器地址”，在**步骤1**的CodeArts IDE客户端弹出的提示界面的“服务器地址”中粘贴，展开高级设置，取消勾选“代理Strict SSL”。单击**“激活”**按钮，如**图2-4**，即可完成激活。

图 2-3 复制激活服务器地址

图 2-4 输入服务器地址并激活

----结束

须知

CodeArts IDE客户端激活成功后，会自动和许可激活服务器通信，客户端和许可服务器通信中断超过7天后，激活客户端将不可用。

2.3 基本操作

须知

CodeArts IDE的不同子产品默认会使用不同的快捷键方案。CodeArts IDE for Java默认使用的是“IntelliJ IDEA快捷键方案”，CodeArts IDE for Cpp默认使用的是“CodeArts快捷键方案”且仅有此快捷键方案。基本操作章节涉及到快捷键默认是“CodeArts快捷键方案”。如果用户使用的是CodeArts IDE for Java，涉及到快捷键的部分，若快捷键失效，请先参考**默认键绑定**切换到“CodeArts快捷键方案”再做尝试。

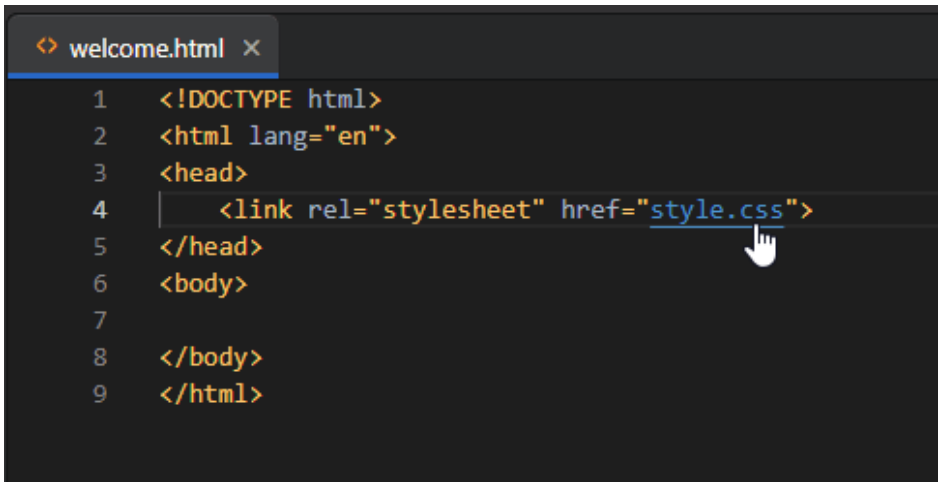
2.3.1 代码编辑

2.3.1.1 基本信息

CodeArts IDE作为编辑器，它包含了用户所需要的高生产力的源代码编辑功能。本节将带用户了解编辑器的基本知识，并帮助用户编写代码。

快速打开文件

步骤1 如下图所示，光标移动至文件链接处style.css。



步骤2 按下“**Ctrl**”，光标单击“style.css”快速打开文件或图像。

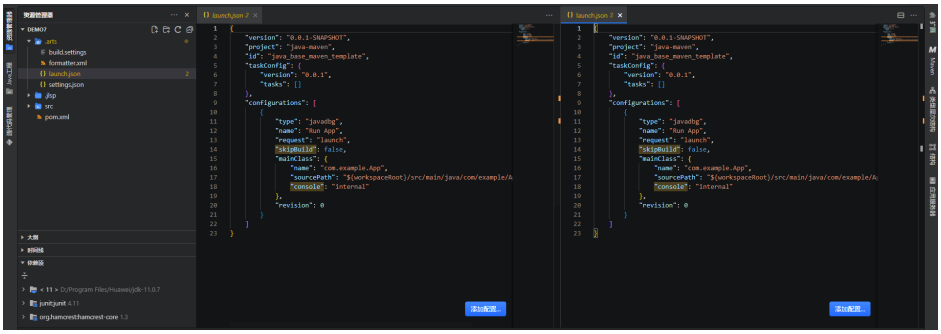
----结束

并排编辑

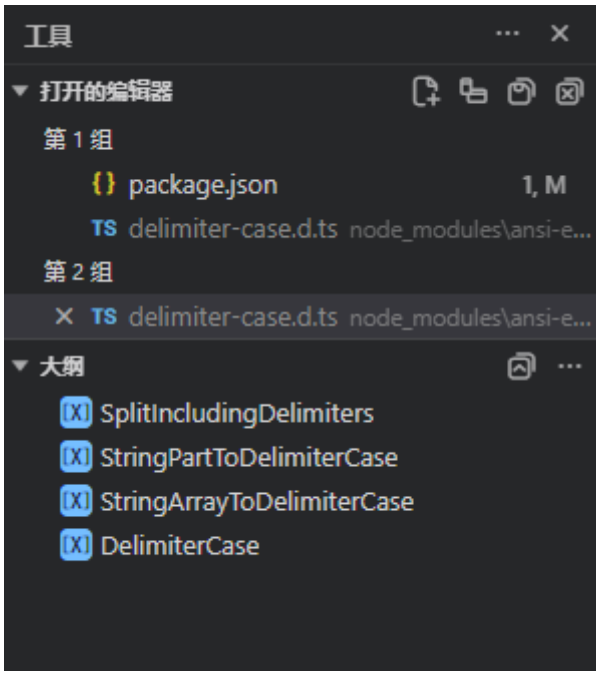
用户可以通过编辑器布局功能在垂直和水平方向打开编辑器。如果用户已经打开了一个编辑器，可以通过以下方式在现有编辑器的一侧打开另一个编辑器：

- 按住“**Alt+单击**”或“**Ctrl+双击**”**资源管理器**中的一个文件。
- 按“**Ctrl+**”将现有的编辑器一分为二。
- 在资源管理器选中文件右键后，上下文菜单中选择“在侧面打开（Ctrl +Enter）”。
- 单击编辑器右上角的“向右拆分编辑器（Ctrl+\）[Alt]向下拆分编辑器”按钮（）。
- 可以将文件拖动到编辑器区域的任意一侧。

拆分编辑器时，将创建编辑器组，这些编辑器组可以容纳多个选项卡。



可以通过打开编辑器视图查看所有创建的组，可以通过单击“资源管理器 > 更多 > 勾选打开的编辑器”打开该视图。

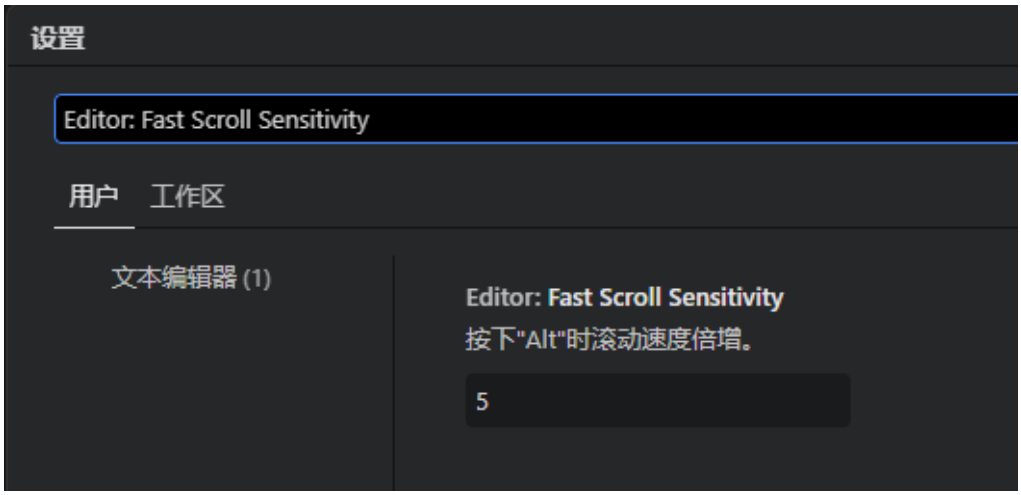


要快速聚焦编辑器组，请使用键盘快捷键**Ctrl+1**（聚焦第一组）、**Ctrl+2**（聚焦第二组）、**Ctrl+3**（聚焦第三组），以此类推。

新打开的文件将作为当前重点组内的选项卡打开。如有需要，用户可以锁定组，以防止在其中打开新选项卡。要执行此操作，请单击组标题中的“更多操作”按钮（···），然后从弹出菜单中选择“锁定组”。

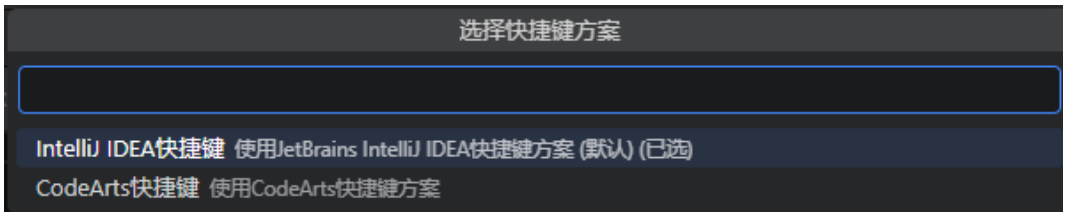
快速滚动

按“**Alt**”键的同时滚动鼠标滚轮可在“编辑器”和“**资源管理器**”中快速滚动。默认情况下，快速滚动速度倍增系数为5，但用户可以使用“**Editor: Fast Scroll Sensitivity**”设置项来调整它。



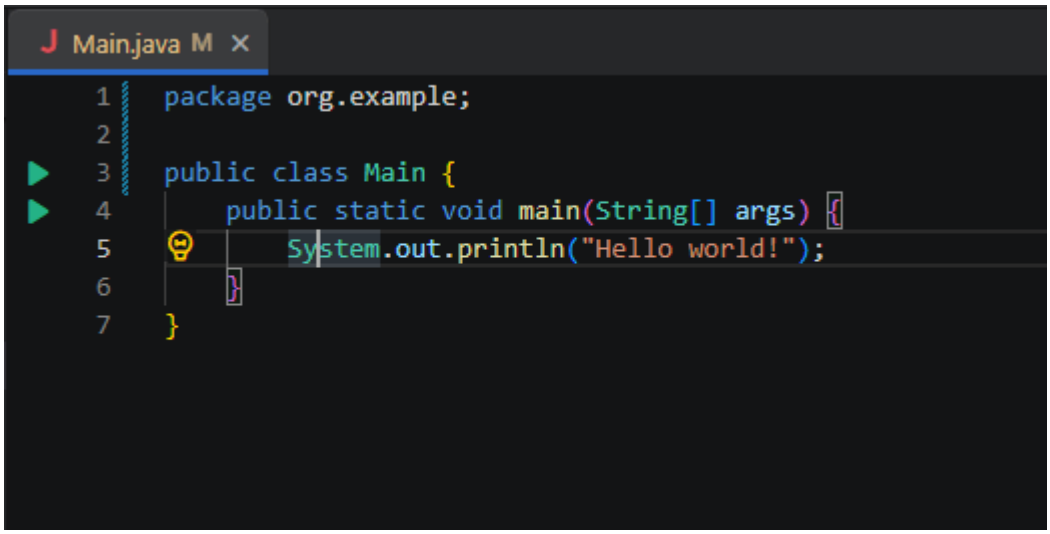
切换快捷键方案

CodeArts IDE for Java提供了两套不同的快捷键方案，分别是：IntelliJ IDEA快捷键方案和CodeArts快捷键方案。Java版本默认使用IDEA快捷键方案，也可以通过左下角“管理 > 切换快捷键方案”来切换不同的快捷键方案。

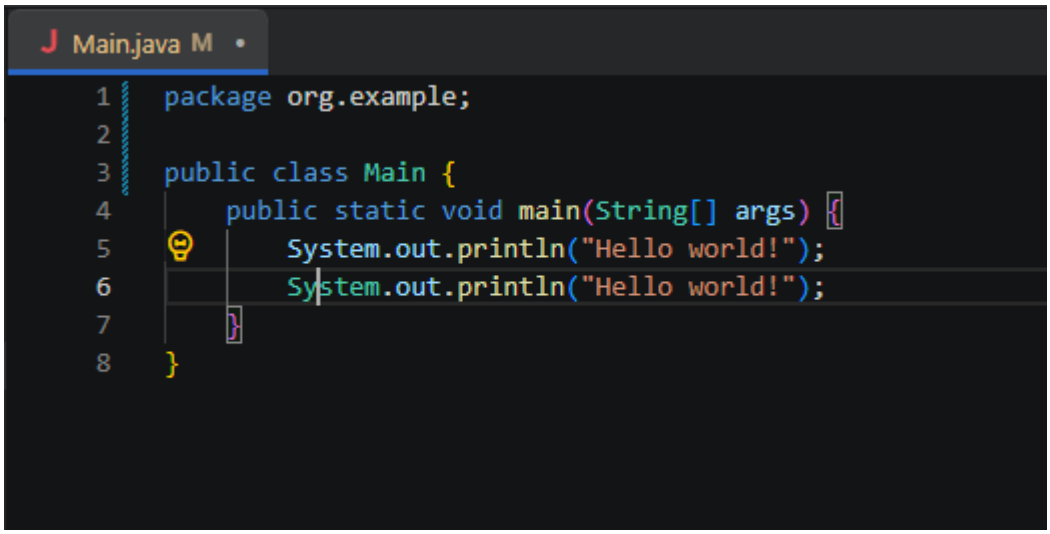


向上/向下复制行

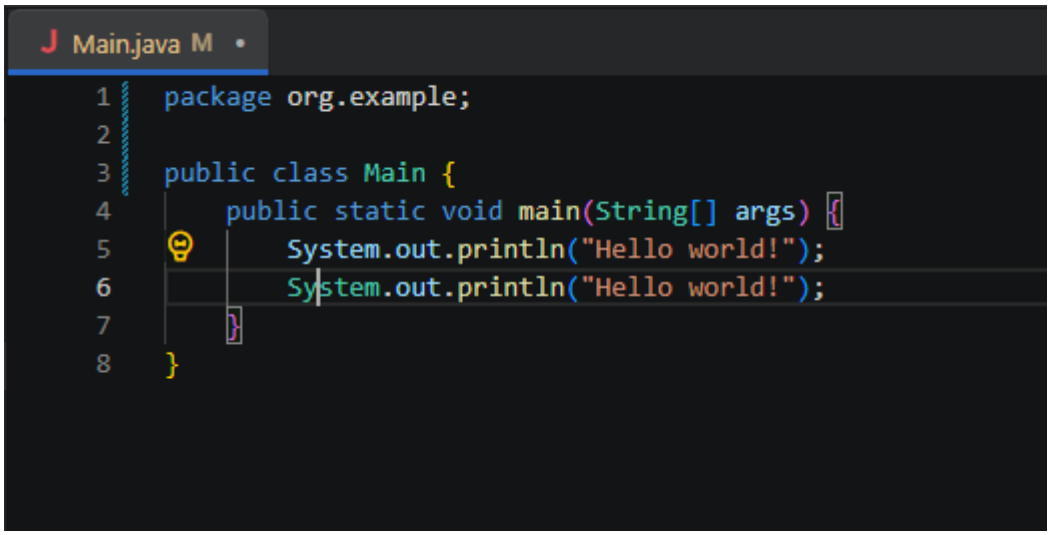
步骤1 光标定位到想要复制的行上面。



步骤2 使用快捷键“**Shift+Alt+Up**”或“**Ctrl+D**”（IDEA快捷键方案）向上复制一行。



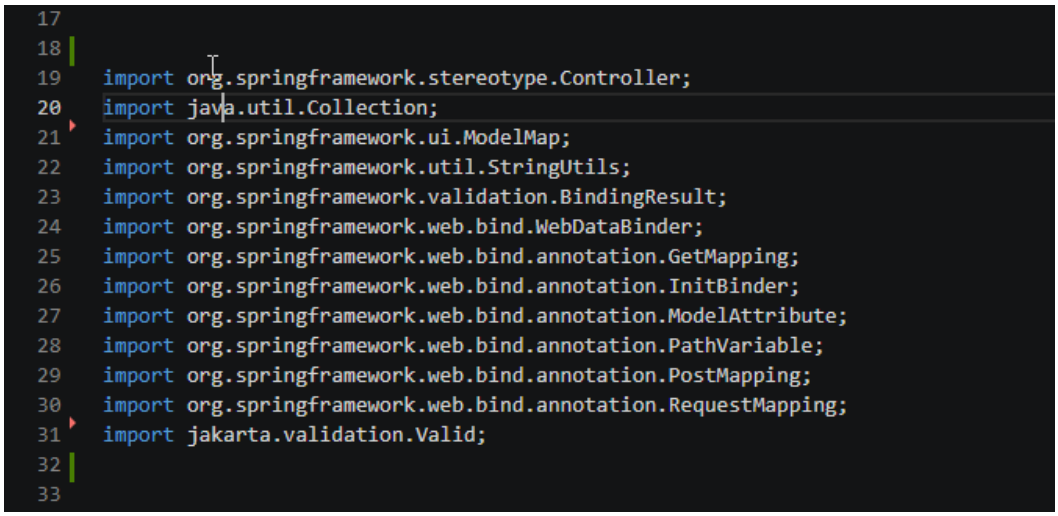
步骤3 使用快捷键“**Shift+Alt+Down**”向下复制一行。



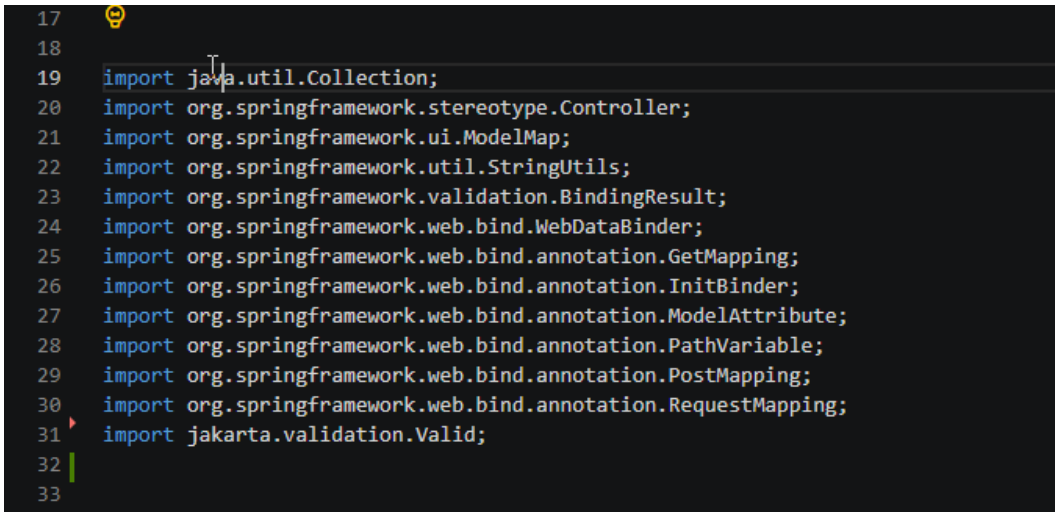
----结束

向上/向下移动行

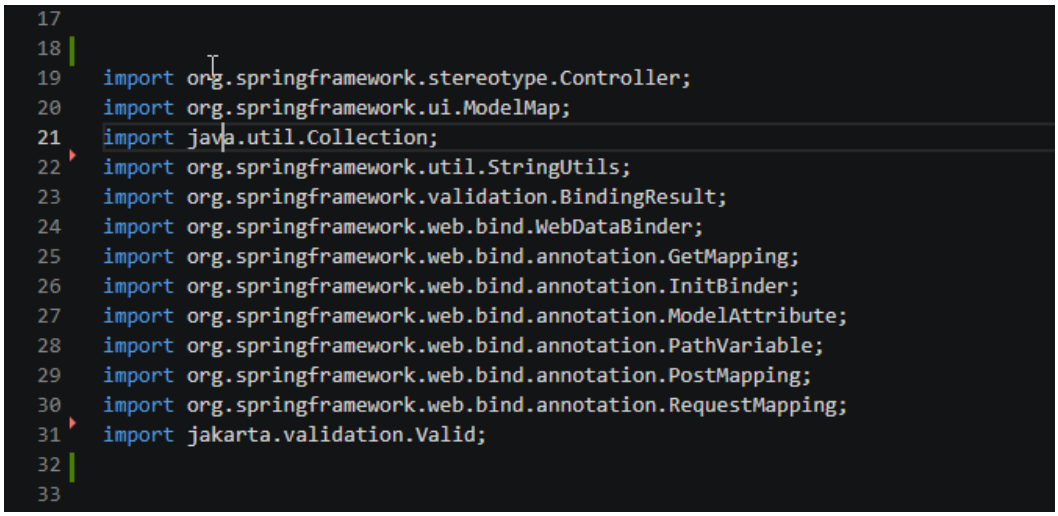
步骤1 光标定位到想要移动的行上面。



步骤2 使用快捷键“**Alt+Up**”或“**Shift+Alt+Up**”或“**Ctrl+Shift+Up**”(IDEA快捷键方案)向上移动一行。



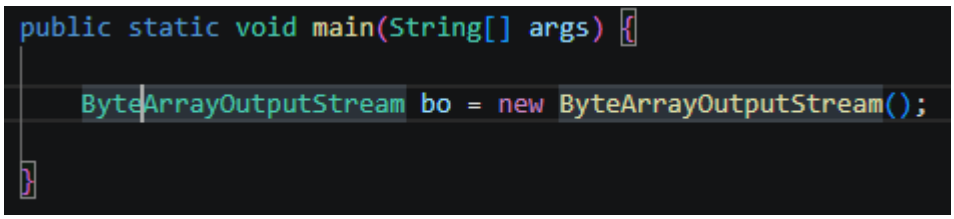
步骤3 使用快捷键“**Alt+Down**”或“**Shift+Alt+Down**”或“**Ctrl+Shift+Down**”(IDEA快捷键方案)向下移动一行。

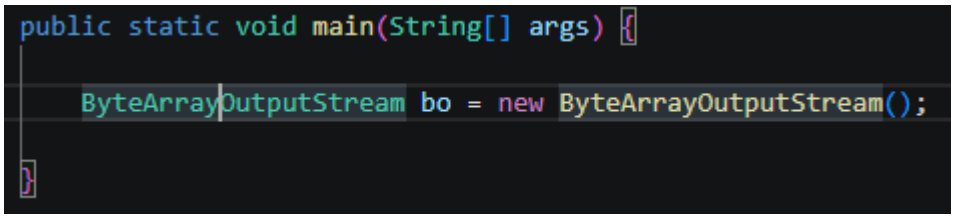


----结束

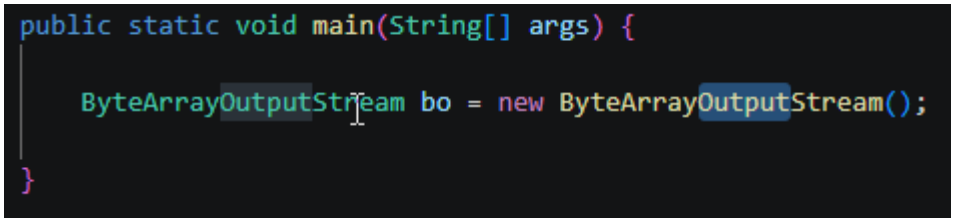
使用 CamelHumps 选择和导航代码

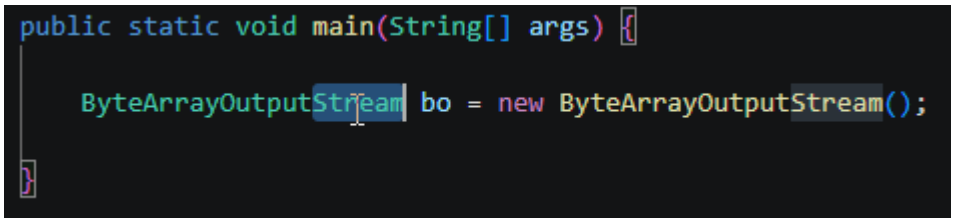
CodeArts IDE可以使用CamelHumps简化代码导航。如果**editor.useCamelHumpsWords**环境已启用，在单词之间按下“**Ctrl+Arrow**”会使光标在CamelHump单词的各个部分之间进行快速跳转，而不是在整个单词之间进行跳转。





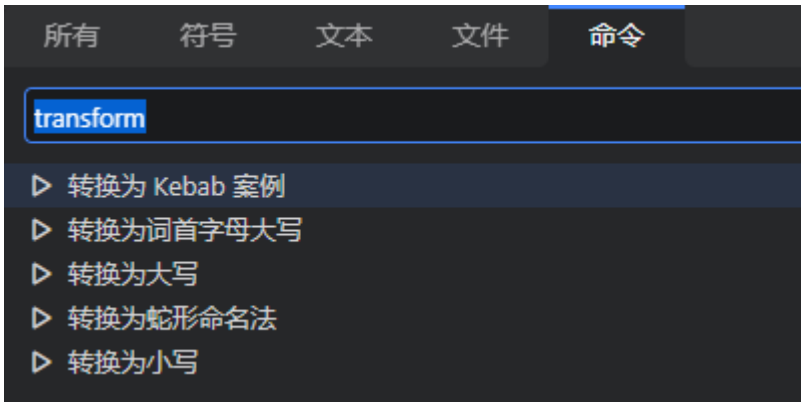
如果**editor.honorCamelHumpsWords**环境启用后，双击CamelHump单词只会选择其单击的单词部分，而不是整个单词。





转换文本

- 如下图所示，如果用户的编程语言是JAVA，使用快捷键“**Ctrl+Shift+P**”打开“命令”面板，输入“transform”执行命令，将选定的首字母自动转换为大写、小写、首字母大写、串式命名和蛇形命名。
- 或者双击“**Ctrl**”键打开“**命令**”面板执行转换命令



提示:

- Kebab Case（串式命名）：又称破折号方式(dash-case)，每个单词全小写或全大写，多单词使用中划线隔开。
- Snake Case（蛇形命名）：每个单词全小写或全大写，多单词使用下划线隔开。

保存/自动保存

默认情况下，CodeArts IDE需要手动操作来保存对磁盘的更改，键盘快捷键：“**Ctrl+S**”。

同时，用户可以打开自动保存，这将在配置的指定延迟后或焦点离开编辑器时保存更改。启用此选项后，无需手动保存文件。打开自动保存的最简单方法是单击菜单栏“**文件**”，勾选“**自动保存**”开关，打开延迟自动保存。

有关对自动保存的更多控制，请打开用户设置并查找相关设置：

- files.autoSave：可以设置为以下值：
  - off - 禁用热退出。当尝试关闭具有未保存更改的编辑器的窗口时，将显示提示。
  - afterDelay - 在配置的“Files：Auto Save Delay”之后，会自动保存具有更改的编辑器。
  - onFocusChange - 当编辑器失去焦点时，会自动保存具有更改的编辑器。
  - onWindowChange - 当窗口失去焦点时，会自动保存具有更改的编辑器。
- files.autoSaveDelay：当file.autoSave被配置为afterDelay时，配置延迟以毫秒为单位。默认值为1000毫秒。

> **说明**
>
> 用户可以通过**本地历史记录**查看文件的历史，或者如果用户的项目关联了Git存储库，可以使用CodeArts IDE内置的**Git支持**。

热退出

默认情况下退出CodeArts IDE时，CodeArts IDE将记住对文件的未保存更改。当通过“文件 > 退出”关闭应用程序或关闭最后一个窗口时，将触发热退出。

用户可以通过将files.hotExit设置为以下值来配置热退出：

- off：禁用热退出。当尝试关闭具有未保存更改的编辑器的窗口时，将显示提示。
- onExit：触发“**workbench.action.quit**” 命令(命令面板、键绑定、菜单)或在Windows/Linux上关闭最后一个窗口时，将触发热退出。所有未打开文件夹的窗口都将在下次启动时恢复。可通过“**文件**”>“**打开最近使用的文件**”>“**更多**”，访问之前打开的窗口(包含未保存的文件)列表。
- onExitAndWindowClose：触发“**workbench.action.quit**”命令(命令面板、键绑定、菜单)或在Windows/Linux上关闭最后一个窗口时将触发执退出，还将对已打开文件夹的所有窗口触发执退出(无论是否是最后一个窗口)。所有未打开文件夹的窗口将在下次启动时恢复。可通过“**文件**”>“**打开最近使用的文件**”>“**更多**”，访问之前打开的窗口(包含未保存有的文件)列表。

如果热退出出现问题，所有备份都存储在以下文件夹中，用于标准安装位置：

- Windows：**%APPDATA%\Roaming\codearts-java\Backups**（以java客户端为例）。

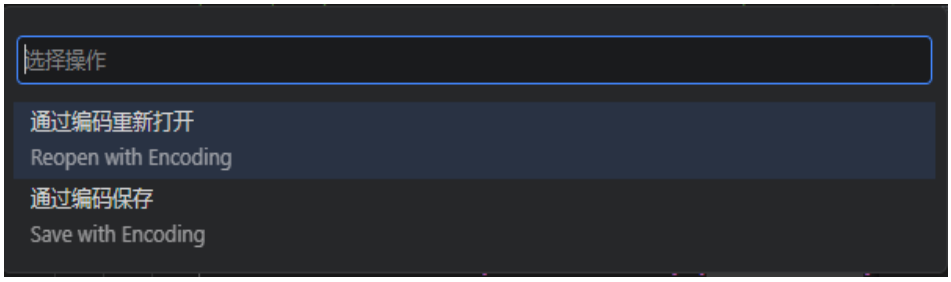
文件编码设置

步骤1 通过用户设置或工作区设置中的file.encoding设置来设置全局或每个工作区的文件编码。

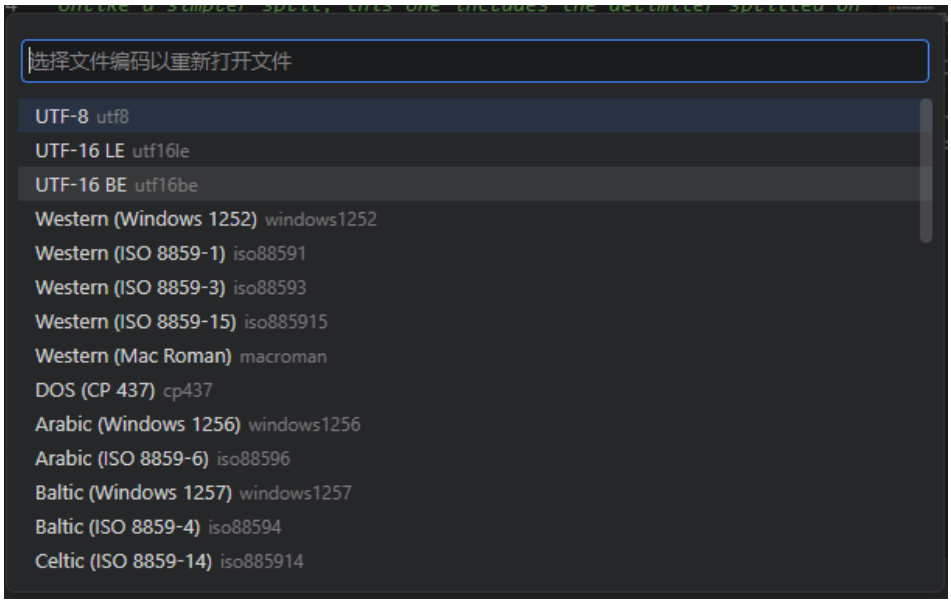
```
//-------- Files configuration --------

// The default character set encoding to use when reading and writing files.
"files.encoding": "utf8",
```

步骤2 用户可以在CodeArts IDE状态栏中查看文件编码。

步骤3 单击状态栏中的编码按钮可使用不同编码重新打开或保存活动文件。



步骤4 然后选择编码类型。



----结束

重新恢复关闭的页面

当用户关闭了一个文件时，如果想要重新打开，可以按“**Ctrl+Shift+T**”重新恢复该页面。

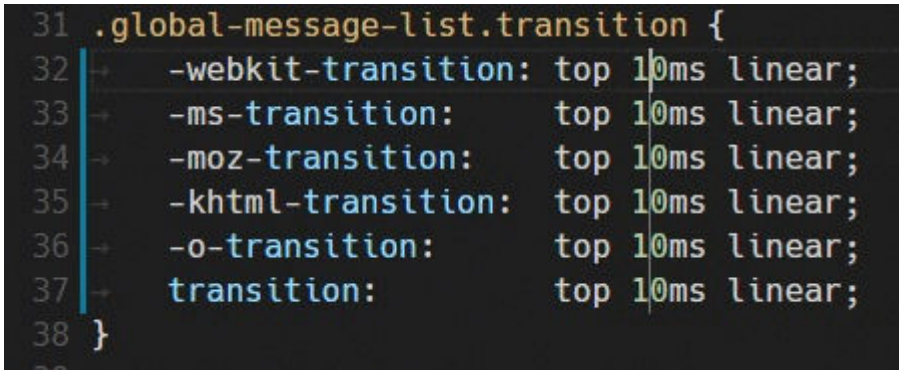
2.3.1.2 选择代码

选择当前行

使用“**Ctrl+L**”可以快速选中当前行代码。

多重选择（多光标）

CodeArts IDE支持多个光标以实现快速的同步编辑。用户可以使用“**Alt+单击**”添加二级光标。每个光标根据其所在的上下文独立运行。添加更多光标的常见方法是使用“**Ctrl+Alt+Down**”或“**Ctrl+Alt+Up**”将光标插入下方或上方。

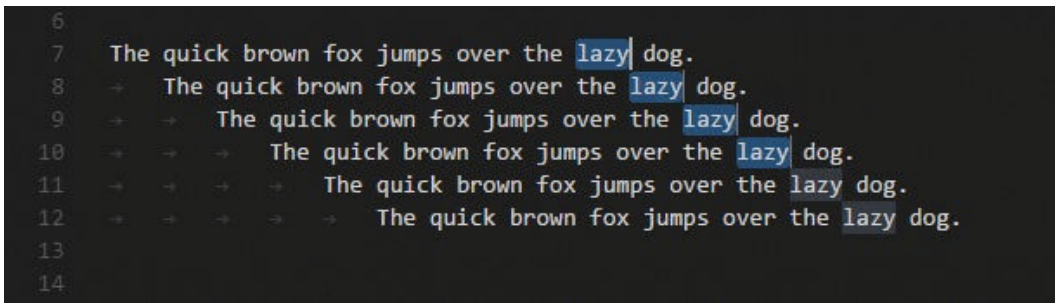


```
.global-message-list.transition {
    -webkit-transition: top 120ms linear;
    -ms-transition:     top 120ms linear;
    -moz-transition:    top 120ms linear;
    -khtml-transition:  top 120ms linear;
    -o-transition:      top 120ms linear;
    transition:         top 120ms linear;
}
```

说明

用户的显卡驱动程序可能会覆盖这些默认的快捷方式。

按“**Ctrl+D**”或“**Alt+J**”(IDEA快捷键方案)选择光标处的单词，或选择当前出现的下一个匹配的单词。



用户还可以使用“**Ctrl+Shift+L**”或“**Ctrl+Shift+Alt+J**”（IDEA快捷键方案）同时添加更多光标，这将在当前选中的文本的每一个匹配项添加一个光标。

多光标修改器

如果用户想将通过鼠标添加多个光标时使用的修改键改为“**Ctrl+单击**”，用户可以通过editor.multiCursorModifier设置项来实现。这让来自其他编辑器（如Sublime Text或Atom）的用户可以继续使用熟悉的键盘修改器。

设置项可以被设置为：

- ctrlCmd - 映射为“**Ctrl**”（Windows和Linux）或 “**Command**”(macOS)

- alt - 映射为“**Alt**”（Windows和Linux）或“**Option**”(macOS)。

“转到定义”和“打开链接”功能所需的鼠标动作将会相应调整，并且不会与多光标修改键冲突。例如，当设置为ctrlCmd时，可以使用“**Ctrl+单击**”添加多个光标，打开链接或转到定义可以使用“**Alt+单击**”调用。

缩小/扩大选区

快速缩小或扩大当前选定内容。可使用“**Shift+Alt+Left**”和“**Shift+Alt+Right**”或“**Ctrl+Shift+W**”(IDEA快捷键方案)触发。

下面是一个用“**Shift+Alt+Right**”扩大选区的例子：

```
int main(int argc,char *argv[])
{
   int decompress = 0;
   int level = 9;
   char *fn_r = NULL;
   char *fn_w = NULL;

#ifdef _WIN32
   if(BZ2DLLLoadLibrary()<0){
      fprintf(stderr,"Loading of %s failed.  Giving up.\n", BZ2_LIBNAME);
      exit(1);
   }
   printf("Loading of %s succeeded.  Library version is %s.\n",
          BZ2_LIBNAME, BZ2_bzlibVersion() );
#endif
   while(++argv,--argc){
      if(**argv =='-' || **argv=='/'){
         char *p;
         for(p=*argv+1;*p;p++){
            if(*p=='d'){
               decompress = 1;
            }else if('1'<=*p && *p<='9'){
               level = *p - '0';
            }else{
               usage();
               exit(1);
            }
         }
      }else{
         break;
      }
   }
```

```
int main(int argc,char *argv[])
{
   int decompress = 0;
   int level = 9;
   char *fn_r = NULL;
   char *fn_w = NULL;

#ifdef _WIN32
   if(BZ2DLLLoadLibrary()<0){
      fprintf(stderr,"Loading of %s failed.  Giving up.\n", BZ2_LIBNAME);
      exit(1);
   }
   printf("Loading of %s succeeded.  Library version is %s.\n",
          BZ2_LIBNAME, BZ2_bzlibVersion() );
#endif
   while(++argv,--argc){
      if(**argv =='-' || **argv=='/'){
         char *p;
         for(p=*argv+1;*p;p++){
            if(*p=='d'){
               decompress = 1;
            }else if('1'<=*p && *p<='9'){
               level = *p - '0';
            }else{
               usage();
               exit(1);
            }
         }
      }else{
         break;
      }
   }
```

列（框）选择模式

将光标放置在第一行的右上角，然后按住“**Shift+Alt**”，同时拖动光标到最后一行的右下角，即可选中所有行中的指定内容。当“**Ctrl**”设置为多光标功能（**多光标修改器**）时，请使用“**Shift+Ctrl**”。

```
s->blockNo           = 0;
s->state             = BZ_S_INPUT;
s->mode              = BZ_M_RUNNING;
s->combinedCRC       = 0;
s->blockSize100k     = blockSize100k;
s->nblockMAX         = 100000 * blockSize100k - 19;
s->verbosity         = verbosity;
s->workFactor        = workFactor;

s->block             = (UChar*)s->arr2;
s->mtfv              = (UInt16*)s->arr1;
s->zbits             = NULL;
s->ptr               = (UInt32*)s->arr1;
```

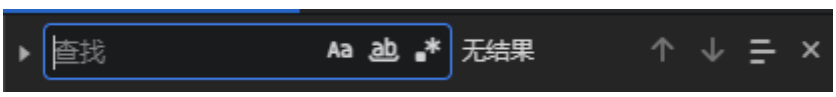
```
s->blockNo           = 0;
s->state             = BZ_S_INPUT;
s->mode              = BZ_M_RUNNING;
s->combinedCRC       = 0;
s->blockSize100k     = blockSize100k;
s->nblockMAX         = 100000 * blockSize100k - 19;
s->verbosity         = verbosity;
s->workFactor        = workFactor;

s->block             = (UChar*)s->arr2;
s->mtfv              = (UInt16*)s->arr1;
s->zbits             = NULL;
s->ptr               = (UInt32*)s->arr1;
```

2.3.1.3 代码搜索

2.3.1.3.1 查找和替换

CodeArts IDE支持用户快速查找和替换当前打开的文件文本。

- 按“**Ctrl+F**”在编辑器中打开“查找”小组件，搜索结果将在编辑器和右侧缩略图突出显示。

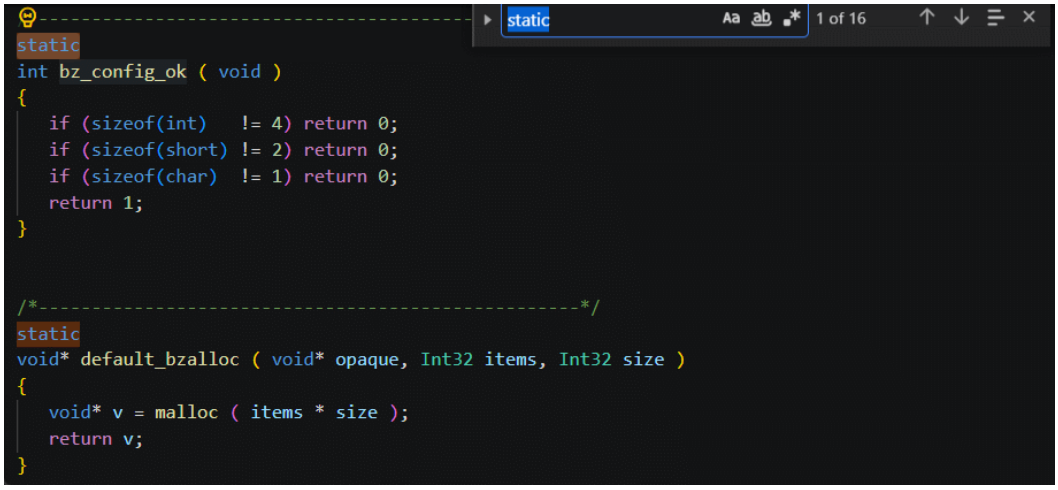


如果当前打开的文件中存在多个匹配结果，可以按“**Enter**”/“**Shift+Enter**”和“**F3**”/“**Shift+F3**”导航到下一个或上一个结果。

- 要将小组件切换到替换模式，请单击展开箭头，或按“**Ctrl+H**”或“**Ctrl+R**”（IDEA快捷键方案）。

从选定内容中搜索关键词

光标定位在任意文本或者选中内容，打开“查找”小组件时，它会自动将编辑器中选中的文本填充到查找输入框中。如果选择为空，则将插入光标下的单词。



须知

可以通过搜索字符串选择项：“editor.find.seedSearchStringFromSelection”设置为**“never”**，将此功能关闭。

在选定内容中查找

默认情况下，可以在编辑器的整个文件内查找，也可以在选定的文本上查找。用户可以通过单击“查找”小组件中的“在选定内容中查找”按钮（ ）来打开此功能。

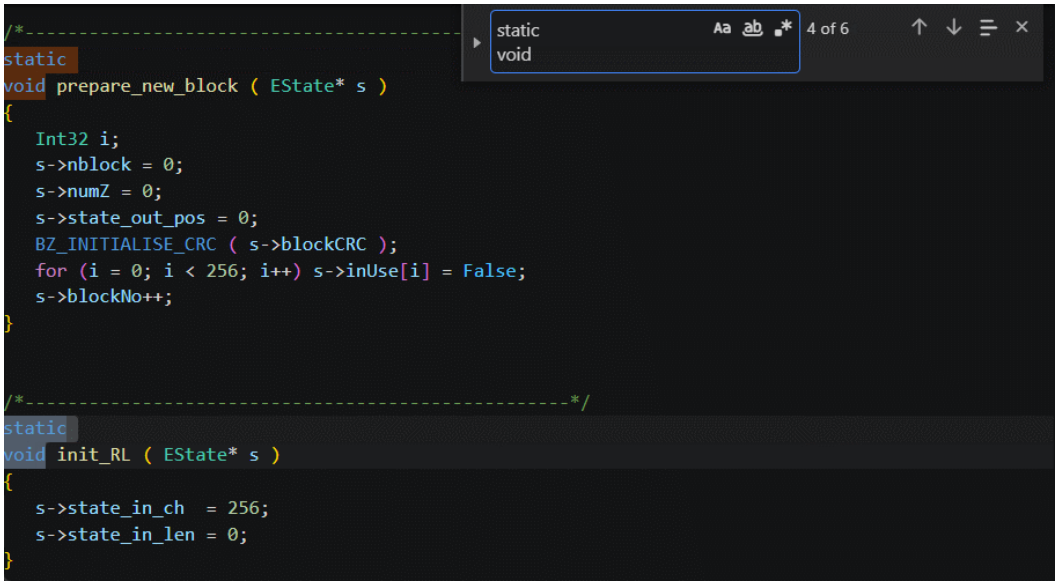
须知

如果用户希望它成为查找小组件的默认行为，则可以将“editor.find.autoFindInSelection”选择项设置为“always”，如果用户希望仅在选择的多行内容上查找，则可以将其设置为“multipleline”。

多行支持和查找小部件调整大小

用户可以通过将文本粘贴到“查找”字段和“替换”字段中来搜索多行文本。

要在字段中插入新行，请按**“Ctrl+Enter”**键。



高级查找和替换选项

除了查找和替换为纯文本外，查找小组件还有三个高级搜索选项：

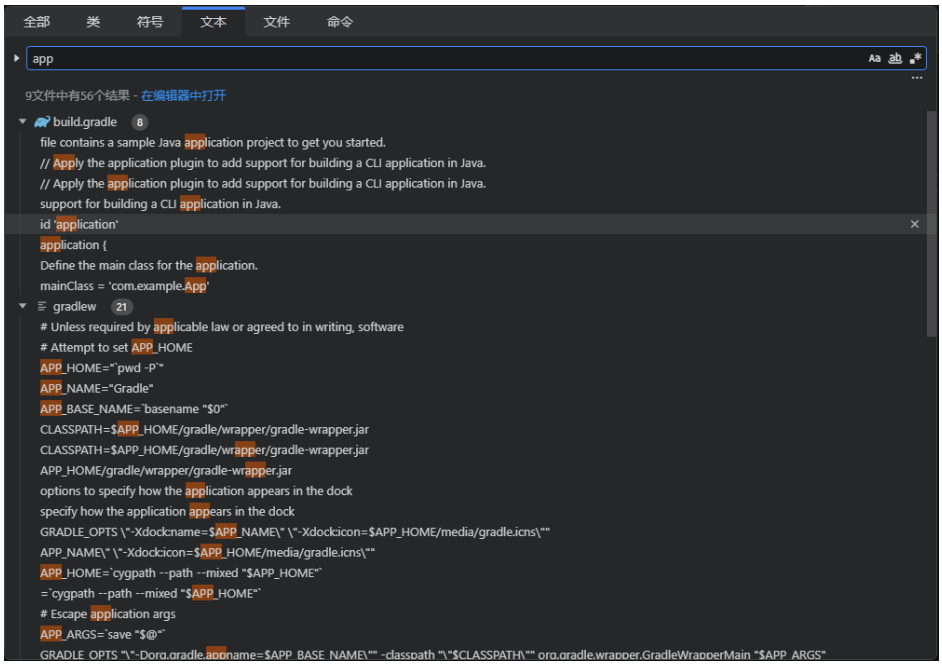
- 区分大小写(Aa)**“Alt+C”**。
- 全字匹配(ab)**“Alt+W”**。
- 使用正则表达式(.*)**“Alt+R”**。

可以保留被替换文本中的大小写（即全大写、全小写和标题大小写）。

2.3.1.3.2 跨文件搜索

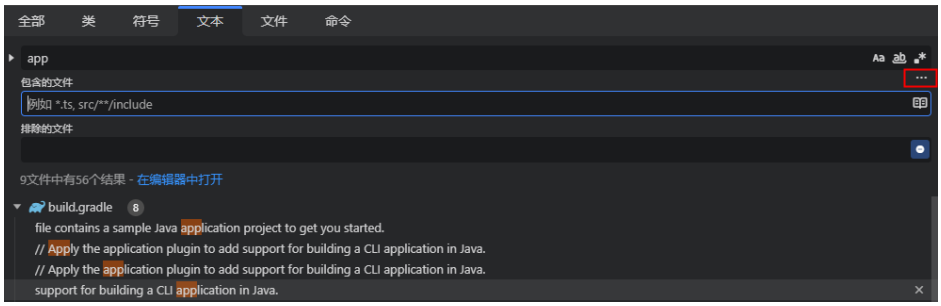
CodeArts IDE允许用户快速搜索当前打开的文件夹中的所有文件。

- 按**“Ctrl+Shift+F”**（仅IDEA快捷键方案适用）并输入搜索词。
  搜索结果被分组到包含搜索词的文件中，并指示每个文件中的匹配项及其位置。
- 展开文件可查看该文件中所有选中的预览，然后单击其中一个搜索结果可在编辑器中查看它。



高级搜索选项

用户可以通过单击搜索字段下面的“**切换搜索详细信息**”按钮（ ··· ）来提供高级搜索选项，并在“**包括的文件**”或“**排除的文件**”字段中输入要从搜索中包括或排除的模式。



如果用户输入example，这将匹配工作区中每个名为example的文件夹和文件。如果用户输入./example，这将匹配工作区顶层的文件夹example/。使用逗号（,）来分隔多个模式。路径必须使用正斜线。

用户也可以使用glob语法来提供样式：

- ***** - 匹配一个路径段中的零个或多个字符。
- **?** - 匹配一个路径段中的单个字符。
- ****** - 匹配任意数量的路径段，包括无。
- **{}** - 用于分组条件（例如，**{**/*.html，**/*.txt}**匹配所有HTML和文本文件）。
- **[]** - 用于声明要匹配的字符范围（例如，**example.[0-2]**匹配example.0，example.1，example.2）。
- **[!...]** - 否定要匹配的字符范围（例如，**example.[!0-9]**匹配example.a，example.b，但不匹配example.0）。

CodeArts IDE默认排除了一些文件夹，以减少用户可能不感兴趣的搜索结果（例如，node_modules）。用户可以打开设置，并在files.exclude和search.exclude部分改变这

些默认选项。如果要快速包含或排除被“.gitignore”文件忽略的文件或被files.exclude和search.exclude设置匹配的文件，请在**“排除的文件”**栏中单击**使用“排除设置”与“忽略文件”**按钮（ ）。

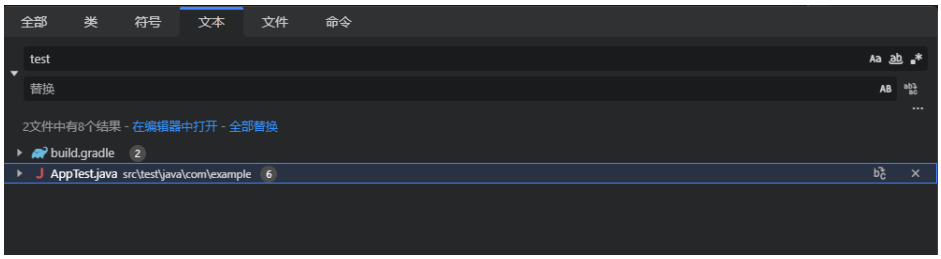
须知

搜索视图中的glob模式与files.exclude和search.exclude等设置中的工作方式不同。在设置中，glob模式总是相对于工作区文件夹的路径进行评估，用户必须使用**/example来匹配子文件夹folder1/example中名为example的文件夹。在搜索视图中，**前缀是假定的。

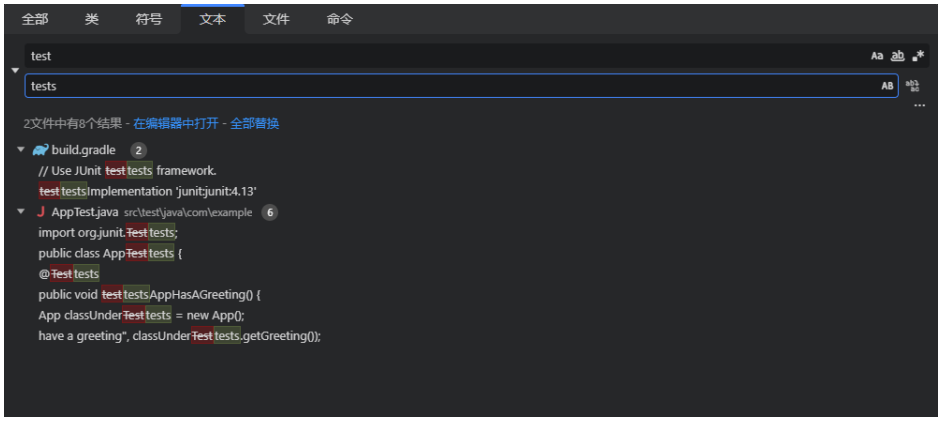
如果用户要将搜索范围限制在当前打开的文件，请在**“包含的文件”**栏中单击**“仅在打开的编辑器中搜索”**按钮（ ）。

查找和替换

用户可以跨文件搜索和替换。如果要显示替换字段，单击展开搜索部件，或按**“Ctrl+Shift+H”**或**“Ctrl+Shift+R”**（IDEA快捷键方案）。



当用户在**“替换”**输入框中键入文本时，CodeArts IDE会显示一个待定修改的差异视图。可以选择单个替换、在一个文件中全部替换或跨所有文件替换。



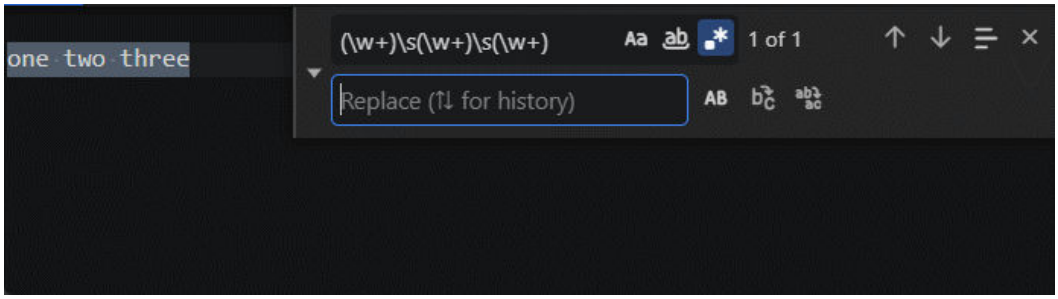
须知

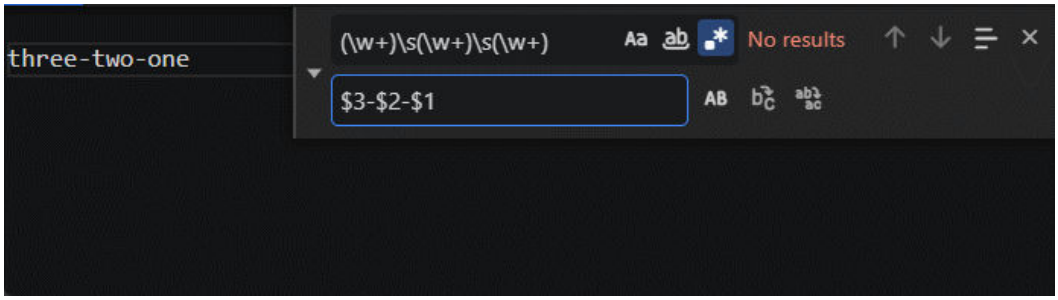
通过使用**Down**键和**Up**键可在搜索词历史记录中进行导航，用户可以快速重用以前的搜索词。

2.3.1.3.3 搜索并替换为正则表达式

除了搜索和替换表达式外，用户还可以通过将正则表达式与捕获组一起搜索和重用匹配的部分内容。通过单击**“使用正则表达式”**按钮（ .* ）或按**“Alt+R”**，在搜索框中启用正则表达式，然后编写正则表达式并使用括号定义组。

然后，用户可以通过在替换字符串中使用$n引用每个组中匹配的内容，其中n是捕获组的顺序（例如，$1、$2）。



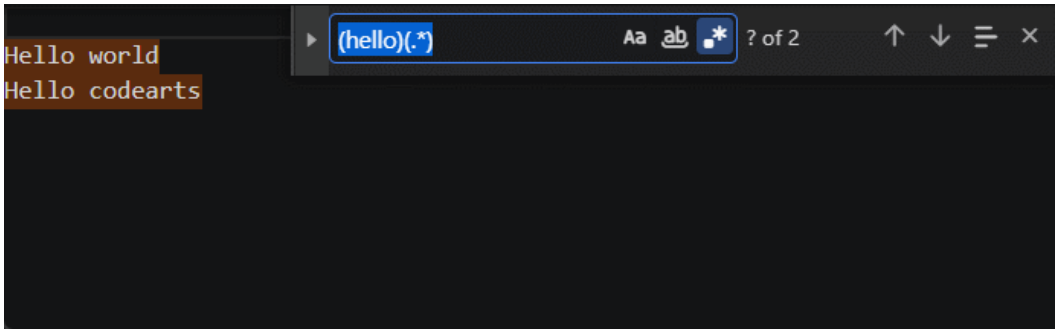


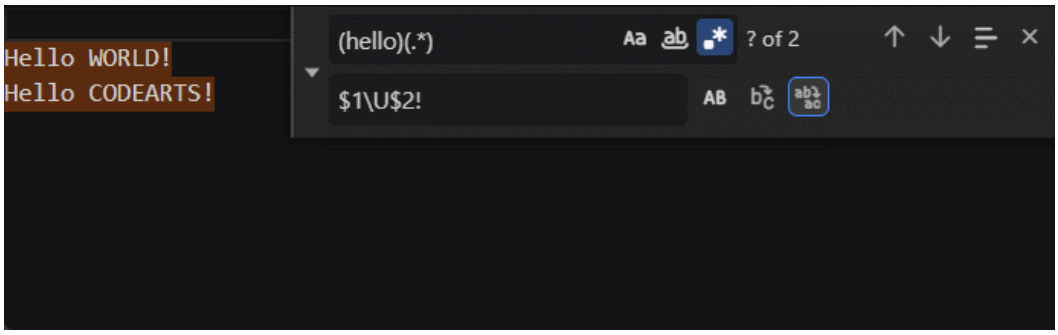
正则表达式替换中的大小写变化

CodeArts IDE支持在编辑器或全局中执行搜索和替换时改变正则表达式匹配组的大小写。

这可以通过使用修改符来实现：

- \u： 大写单个字符。
- \U： 将匹配组的其余部分大写。
- \l： 小写单个字符。
- \L： 将匹配组的其余部分小写。





修饰符可以堆叠，例如，\u\u\u$1将大写组的前三个字符，\l\U$1将小写第一个字符，大写其余字符。

2.3.1.4 格式化代码

2.3.1.4.1 概述

CodeArts IDE提供了许多代码格式化功能。编辑器有两个显式格式操作：

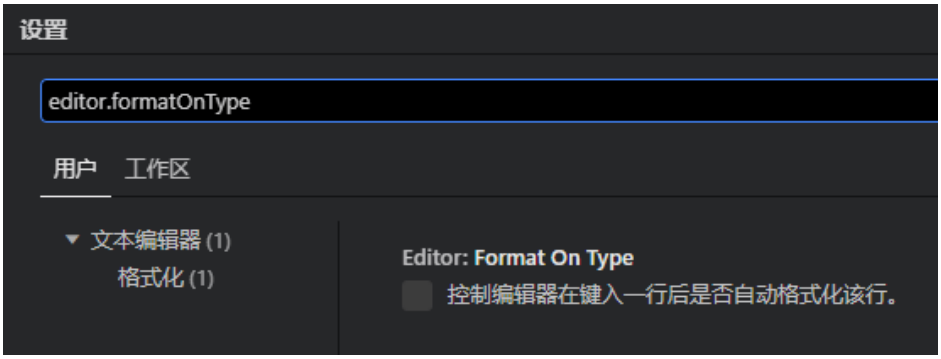
- 格式化文档（**“Shift+Alt+F”**或**“Ctrl+Alt+L”**（IDEA快捷键方案））：格式化整个活动文件。
- 格式化选定内容（**“Ctrl+K Ctrl+F”**或**“Ctrl+Alt+L”**（IDEA快捷键方案））：设置选定文本的格式。

用户可以从“命令”面板（**“Ctrl+Shift+P”**或双击**“Ctrl ”**）或编辑器上下文菜单调用这些操作。

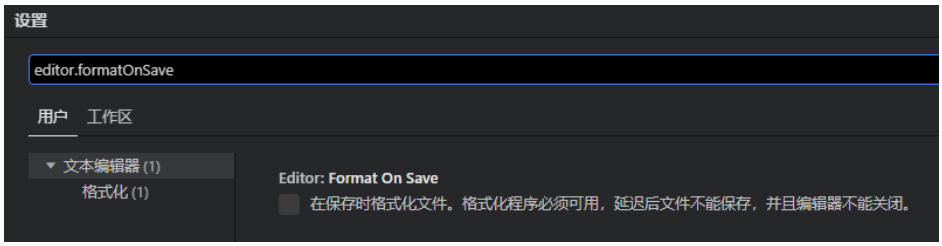
CodeArts IDE具有对JavaScript、TypeScript、JSON、HTML和Java的默认格式化程序。每种语言都有特定的格式选项（例如，html.format.indentInnerHtml），用户可以根据用户或工作区设置中的首选项进行调整。如果用户安装了另一个为同一语言提供格式化的扩展，用户也可以禁用默认语言格式化程序。

除了手动调用代码格式化外，用户还可以根据用户操作触发格式化，如键入、保存或粘贴。默认情况下，这些功能处于关闭状态，但用户可以通过以下设置启用这些功能：

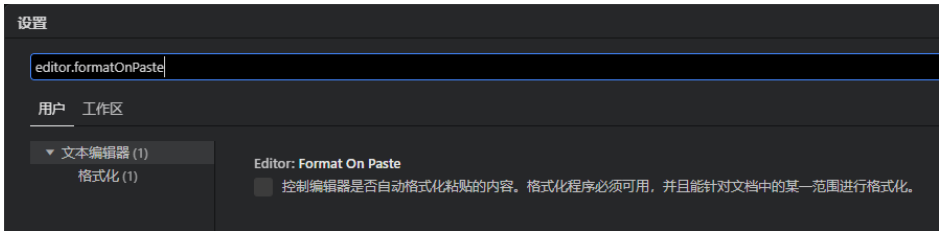
- editor.formatOnType：键入一行后自动格式化该行。



- editor.formatOnSave：保存时格式化文件。



- editor.formatOnPaste：格式化粘贴的内容。



除了默认的格式化工具外，用户也可以安装支持其他语言或格式化工具的扩展插件。

说明

并非所有格式化程序都支持粘贴时的格式，它们必须支持格式化文本的选定内容。

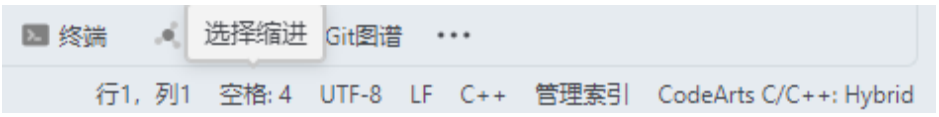
2.3.1.4.2 缩进

CodeArts IDE允许用户使用空格或制表符来控制文本缩进。默认情况下，CodeArts IDE插入空格，并且每个“**Tab**”键使用4个空格。如果要使用其他默认值，用户可以修改编辑器选项“editor.insertSpaces”和“editor.tabSize”设置。

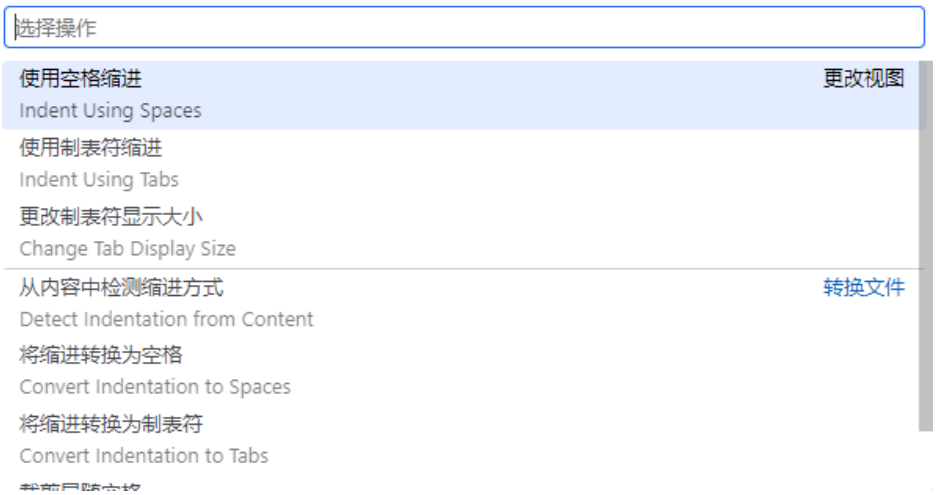
- editor.insertSpaces：按Tab键时是否插入空格。
- editor.tabSize：一个制表符等于的空格数。

自动检测

CodeArts IDE会根据用户所打开的文件类型来决定文档中使用的缩进。自动检测的缩进将覆盖默认的缩进设置。检测到的设置将显示在底部状态栏的右侧。



用户可以通过单击状态栏中的缩进设置打开缩进命令列表，然后更改打开文件的默认缩进设置或在制表符和空格之间转换。



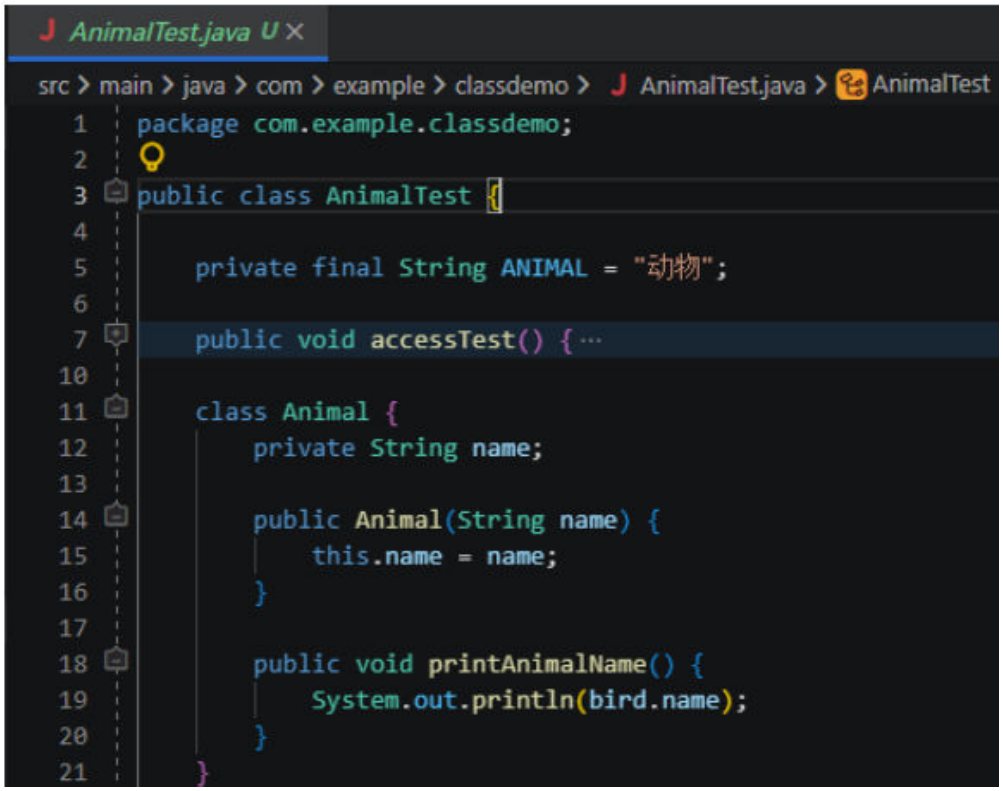
须知

CodeArts IDE自动检测检查2、4、6或8空格的缩进。如果文件使用不同数量的空格，则可能无法正确检测缩进。例如，如果用户需使用3个空格缩进，则用户需关闭“editor.detectIndentation”选项，并将“editor.tabSize”选项大小显式设置为3。

```
"editor.detectIndentation": false,
"editor.tabSize": 3,
```

2.3.1.5 折叠代码

用户可以使用行号和行开始之间的折叠图标来折叠代码区域。将鼠标移动到折叠图标上，然后单击图标实现折叠和展开区域。使用**“Shift + 单击”**折叠图标实现折叠或展开区域和内部的所有区域。



用户还可以使用以下快捷键操作：

- 折叠：折叠光标处最里面的未折叠区域。
  - IDEA快捷键方案：“**Ctrl+-**”。
  - CodeArts快捷键方案：“ **Ctrl+Shift+[**”。
- 展开：在光标处展开折叠区域。
  - IDEA快捷键方案：“**Ctrl+=**”。
  - CodeArts快捷键方案：“**Ctrl+Shift+]**”。
- 切换折叠：折叠或展开光标处的区域。
  - 先按下“**Ctrl+K**”，再按下“**Ctrl+L**”。
- 递归折叠：折叠光标处最里面的未折叠区域以及该区域内的所有区域。
  - IDEA快捷键方案：“**Ctrl+Alt+-**”。
  - CodeArts快捷键方案：先按下“**Ctrl+K**”，再按下“**Ctrl+[**”。
- 递归展开：展开光标处的区域以及该区域内的所有区域。
  - IDEA快捷键方案：“**Ctrl+Alt+=**”。
  - CodeArts快捷键方案：先按下“**Ctrl+K**”，再按下“**Ctrl+]**”。
- 全部折叠：折叠编辑器中的所有区域。
  - IDEA快捷键方案：“**Ctrl+Shift+-**”。
  - CodeArts快捷键方案：先按下“**Ctrl+K**”，再按下“**Ctrl+0**”。
- 全部展开：展开编辑器中的所有区域。
  - IDEA快捷键方案：“**Ctrl+Shift+=**”。
  - CodeArts快捷键方案：先按下“**Ctrl+K**”，再按下“**Ctrl+J**”。

- 折叠级别X：折叠级别X的所有区域，但当前光标位置的区域除外。
  - 先按下“**Ctrl+K**”，再按下“**Ctrl+{X}**”，其中**{X}**对应需要折叠的级别，例如折叠级别2，则**{X}**为2。
- 折叠所有块注释：折叠以块注释标记开头的所有区域。
  - 先按下“**Ctrl+K**”，再按下“**Ctrl+/**”。

默认情况下，使用基于缩进的折叠策略。

特定语言的折叠区域

折叠区域可以根据编辑器配置的语言的语法标记计算。以下语言已经提供了语法感知折叠：Markdown、HTML、CSS、LESS、SCSS和JSON。

如果用户想为上述一种（或所有）语言切换回基于缩进的折叠，请使用：

```
"[html]": {
    "editor.foldingStrategy": "indentation"
},
```

区域也可以由每种语言定义的标记定义。当前以下语言定义了标记：

Language	Start region	End region
Bat	::#region or REM #region	::#endregion or REM #endregion
C#	#region	#endregion
C/C++	#pragma region	#pragma endregion
CSS/Less/SCSS	/*#region*/	/*#endregion*/
Coffeescript	#region	#endregion
F#	//#region or (#region)	//#endregion or (#endregion)
Java	//#region or //<editor-fold>	// #endregion or //</editor-fold>
Markdown	<!-- #region -->	<!-- #endregion -->
Perl5	#region or =pod	#endregion or =cut
PHP	#region	#endregion
PowerShell	#region	#endregion
Python	#region or # region	#endregion or # endregion
TypeScript/JavaScript	//#region	//#endregion
Visual Basic	#Region	#End Region

仅折叠和展开标记定义的区域，请用户使用以下操作：

- 折叠所有区域：先按下快捷键“**Ctrl+K**”，再按下快捷键“**Ctrl+8** ”。
- 展开所有区域：先按下快捷键“**Ctrl+K**”，再按下快捷键“**Ctrl+9** ”。

2.3.2 代码补全

2.3.2.1 编程语言的代码补全

智能代码补全是各种代码编辑功能的总称，包括：代码补全、参数信息、快速信息和成员列表。

代码补全功能有时被称为“内容辅助”或“代码提示”。

- CodeArts IDE为JavaScript、TypeScript、JSON、HTML、CSS、SCSS和Less等编程语言提供代码补全。
- CodeArts IDE支持任何编程语言的基于单词的补全。

2.3.2.2 代码补全功能

概述

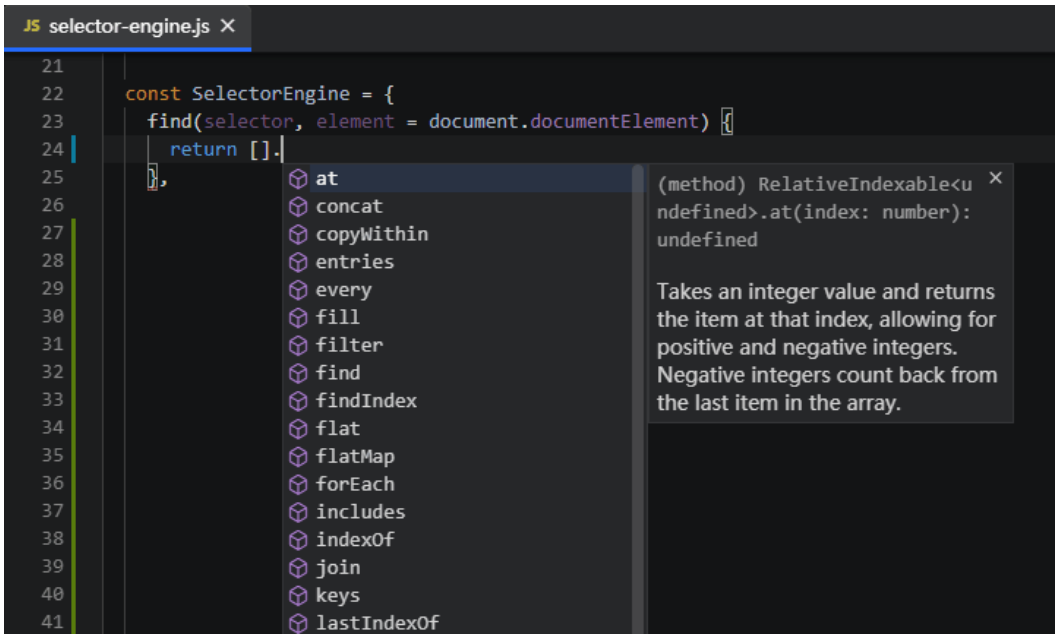
CodeArts IDE代码补全功能由语言服务提供支持，该服务基于语言语义和代码分析提供智能代码补全功能。如果语言服务识别，则在用户键入代码时自动弹出补全推荐列表。如果继续键入字符，则将过滤符号列表（变量、方法等），直到包含键入字符的符号列表。补全推荐小部件还支持驼峰过滤：如要限制补全推荐的代码数量，请在符号名称中键入大写字母。例如，cra将快速调出createApplication。

- 要手动触发代码补全，请按“**Ctrl+Shift+Space**”（IDEA快捷键方案）或键入触发字符（如JavaScript中的点字符(.)）。
- 要插入选定的符号，请按“**Enter**”键。
- 要插入选定的符号并替换当前位于光标位置的符号，请按“**Tab**”键。
- 要关闭推荐列表而不插入推荐代码，请按**退出“Escape”**键。

快速信息

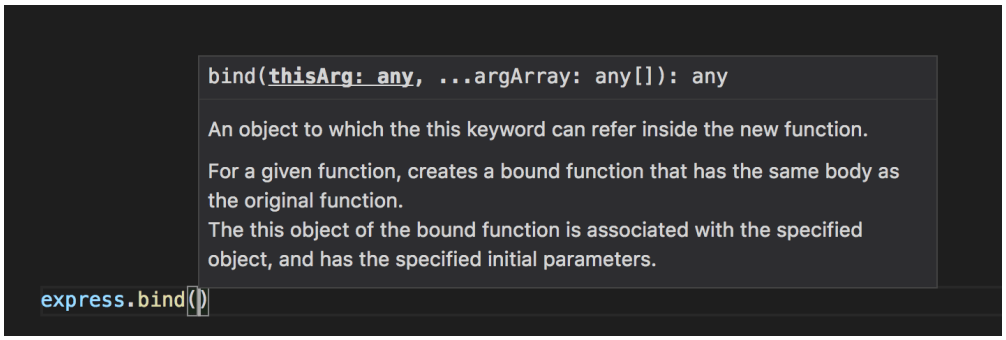
依靠语言服务提供的功能，用户可以通过在推荐列表中再次按“**Ctrl+I**”（**非Java项目**）或“**Ctrl+Space**”来查看方法的快速信息。该方法的文档弹出窗口将扩展到侧面，并在用户导航列表时自动更新。用户还可以通过按“**Ctrl+I**”（**非Java项目**）或“**Ctrl+Shift+Space**”（IDEA快捷键方案）打开插入符号处任何符号的快速信息弹窗。

要隐藏快速信息的弹窗，请再次按“**Ctrl+I**”（**非Java项目**）或“**Ctrl+Space**”键或单击关闭图标。



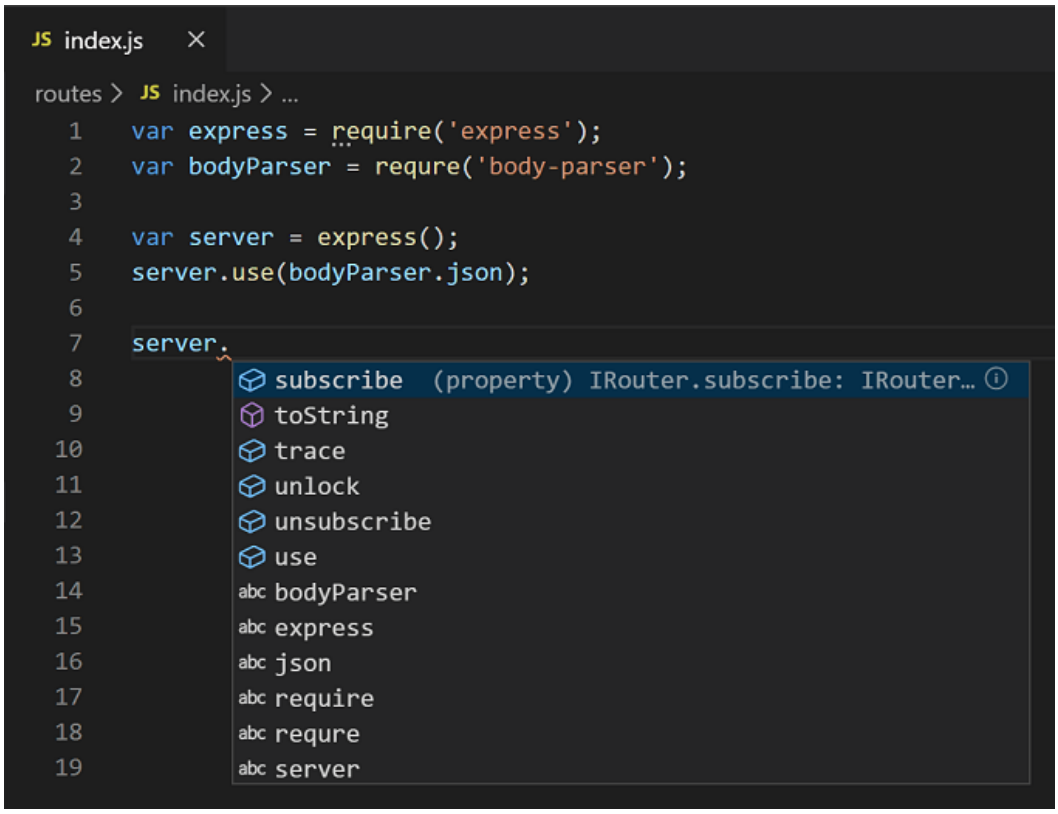
参数信息

在编辑区选择方法后，CodeArts IDE将显示参数信息。如果适用，语言服务将在快速信息和方法签名中显示基础类型。需要随时打开参数信息弹窗，请按“**Ctrl+P**”。



2.3.2.3 补全类型

代码补全可以提供补全推荐和项目的全局标识符。在代码推荐列表中，首先显示推荐的符号，然后是全局标识符（由Word图标显示）。标识符和标识符含义见表2-1。



CodeArts IDE代码补全包括基于语法语义的补全、片段推荐补全和简单的基于单词的文本补全。SmartAssist套件推荐的代码会额外标记为图标。

表 2-1 标识符列表

图标	描述
	方法与函数
	变量
	字段
	类型参数
	常量
	类
	接口
	结构
	事件
	运算符
	模块

图标	描述
	属性
	值与枚举
	引用
	关键字
	文件
	文件夹
	颜色
	单元
	代码片段前缀
abc	单词
	智能化建议

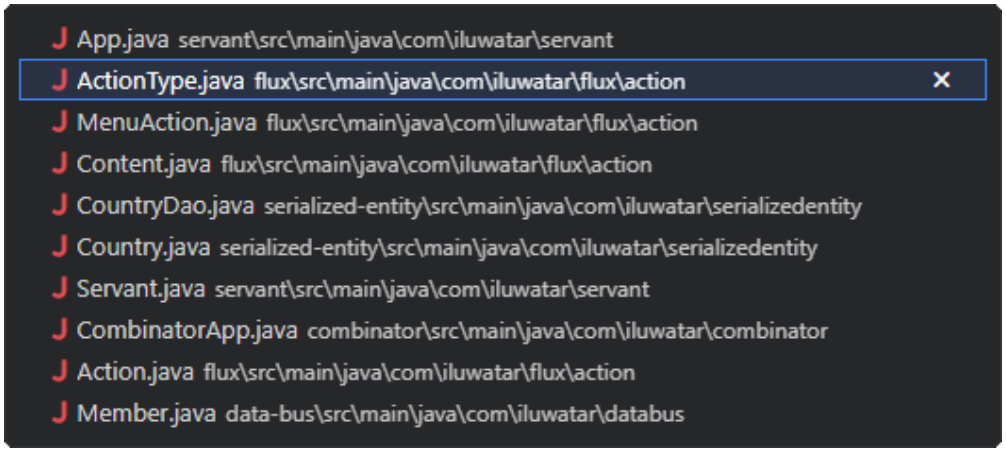
2.3.3 代码导航

2.3.3.1 快速文件导航

当浏览项目时，资源管理器支持用户在文件之间导航。CodeArts IDE提供了功能强大的命令，可以通过易于使用的键绑定在文件中导航和跨文件导航。

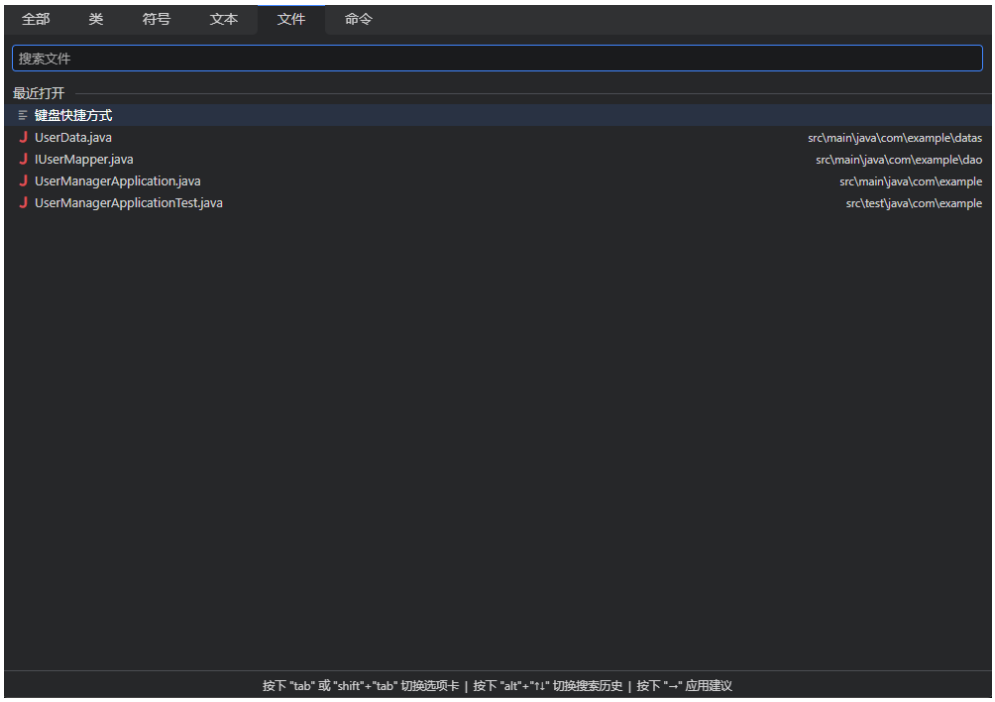
文件切换器

按“**Ctrl+Tab**”键打开文件切换器，列出**编辑器组**中当前打开的所有文件，并按“**Tab**”键循环文件。然后松开“**Ctrl**”键切换到选定的文件。



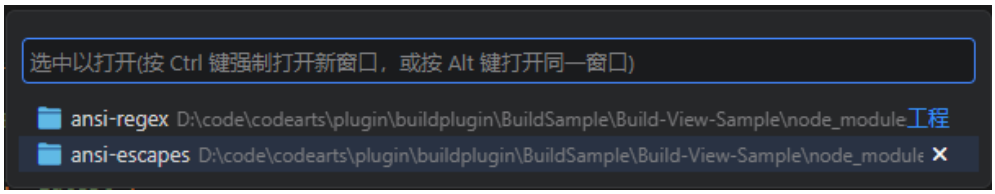
在最近打开的文件之间导航

按“**Ctrl+Shift+N**”打开“文件”窗口，其中列出了最近打开的文件。可以使用向上和向下光标键选择所需的文件，然后按“**Enter**”键打开选中的文件。要缩小列表范围，请键入要打开的文件的名称。



在最近打开的文件夹之间导航

按“**Ctrl+R**”，或在主菜单中选择“文件 > 打开最近的文件> 更多”，查看最近打开的文件夹列表。使用向上和向下光标键选择所需的文件夹，然后按“**Enter**”键打开该文件夹。



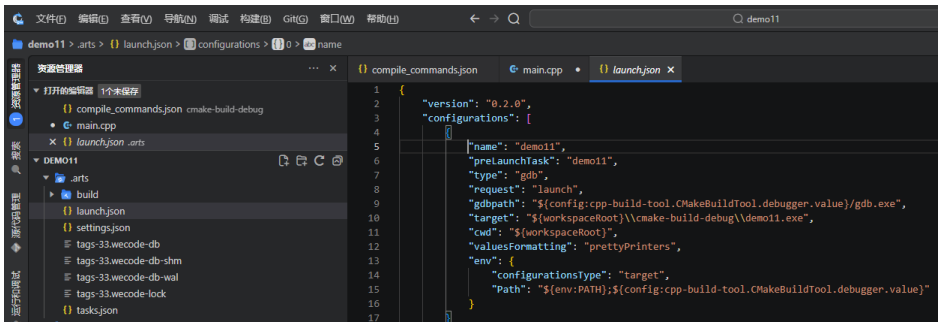
导航到文件的开头和结尾

按“**Ctrl+Home**”键导航到文件的开头，按“**Ctrl+End**”键导航到结尾。

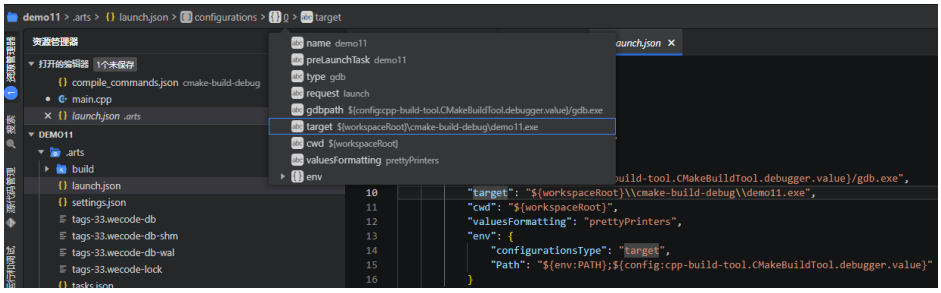
2.3.3.2 使用面包屑导航路径

编辑器内容上方有一个导航栏。它显示当前位置，并允许用户在文件夹、文件和符号之间快速导航。用户可以通过Breadcrumbs.Enabled设置，启用或禁用导航路径。

导航栏始终显示文件路径，并在语言扩展的帮助下显示光标所在位置的符号路径。显示的符号与**大纲视图**和**转到编辑器中的符号**中的相同。



在路径中选择面包屑将显示一个包含该级别同级的列表，以便用户可以快速导航到其他文件夹、文件和符号。



自定义面包屑

导航栏的外观可以自定义。如果显示路径很长，或者只对文件路径或符号路径感兴趣，则可以通过breadcrumbs.filePath和breadcrumbs.symbolPath设置项来配置。两者都支持on、off和last，它们定义了用户是否能看到路径或看到哪一部分的路径。默认情况下，导航痕迹在导航栏的左侧显示文件和符号图标，但用户可以通过将breadcrumbs.icons设置为false来删除图标。

用户可以通过breadcrumbs.symbolSortOrder设置控制“导航路径”大纲视图中符号的排序方式。

支持的排序方式为：

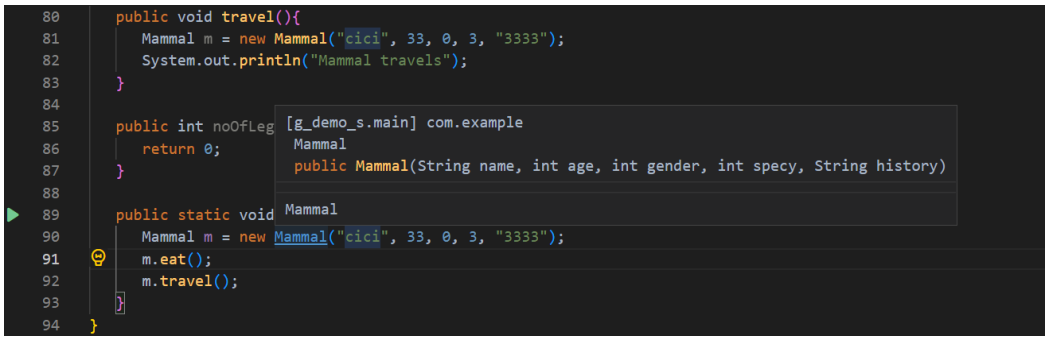
- position：以文件位置顺序显示符号大纲（默认）。
- name：以字母顺序显示符号大纲。
- type：以符号类型显示符号大纲。

面包屑键盘导航

要与面包屑进行交互，请按“**Ctrl+Shift+.**”。这将选择该元素并打开一个列表，允许用户导航到兄弟文件或符号。使用“**Ctrl+Left**”或“**Ctrl+Right**”或“**Right**”键盘快捷键，以前往当前元素之前或之后的元素。当列表打开时，开始键入元素的名称以快速选择进行导航。

2.3.3.3 转到定义

语言服务加载成功后，可以通过按“**F12**”或在主菜单中选择“导航 > 转到定义”来转到符号的定义。如果按“**Ctrl**”键并将鼠标悬停在符号上，将显示声明的预览。



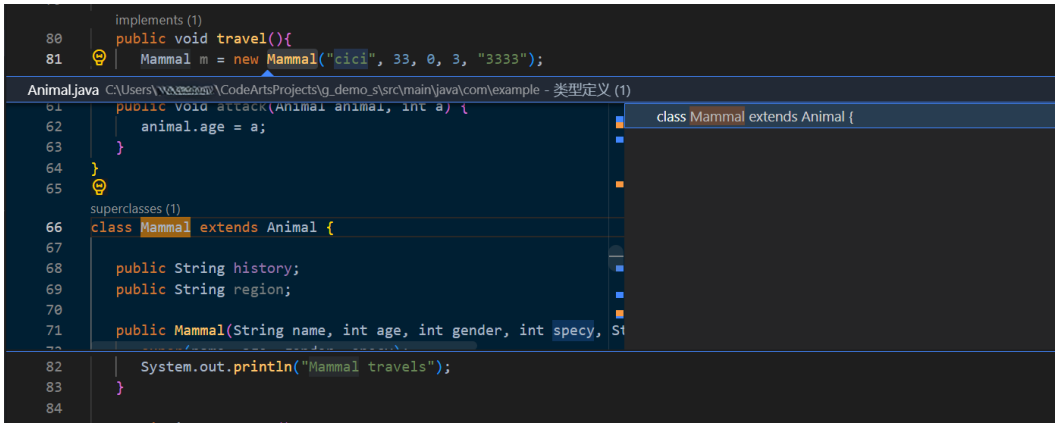
用户可以使用“**Ctrl+单击**”跳转到定义，也可以使用“**Ctrl+Alt+单击**”将定义打开在侧边打开。

还可以通过“快速查看”视图使用此功能，该视图显示在当前编辑器中，因此用户无需切换上下文。在“快速查看”视图中查看符号的定义，右键单击符号，然后在上下文菜单中选择“快速查看 > 速览定义”。

2.3.3.4 转到类型定义

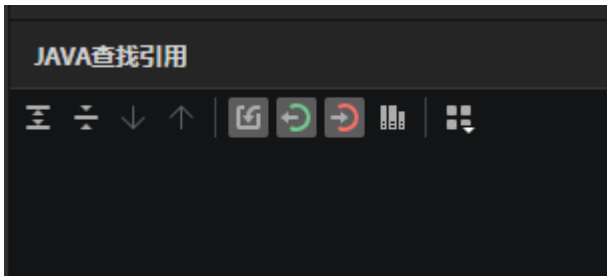
某些语言服务支持通过从编辑器上下文菜单或“**命令**”面板运行“**转到类型定义**”命令，或按“**Ctrl+Shift+B**”跳转到符号的类型定义。

还可以通过“快速查看”视图使用此功能，该视图显示在当前编辑器中，因此用户不需要切换上下文。要在“快速查看”视图中查看符号类型的定义，右键单击符号，然后在上下文菜单中选择“快速查看 > 快速查看类型定义”。



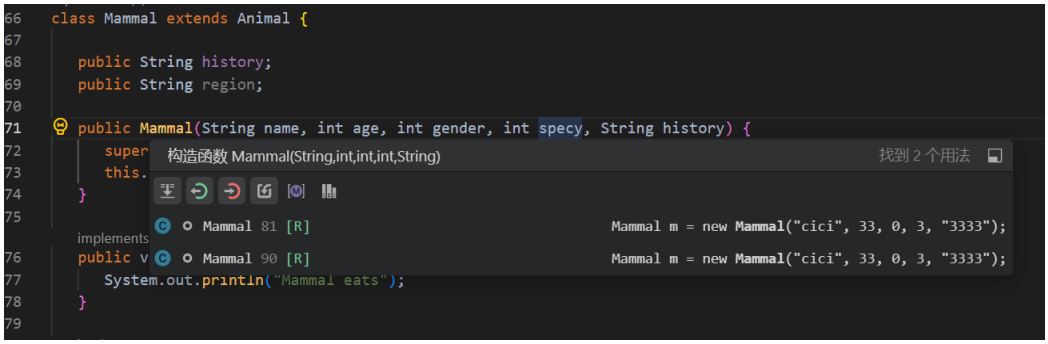
2.3.3.5 查找引用

选择一个符号，然后按“**Shift + Alt + F12**”（**Java工程不适用于该快捷键**）或“**Alt +F7**”在“**查找引用**”视图中打开对它的所有引用。



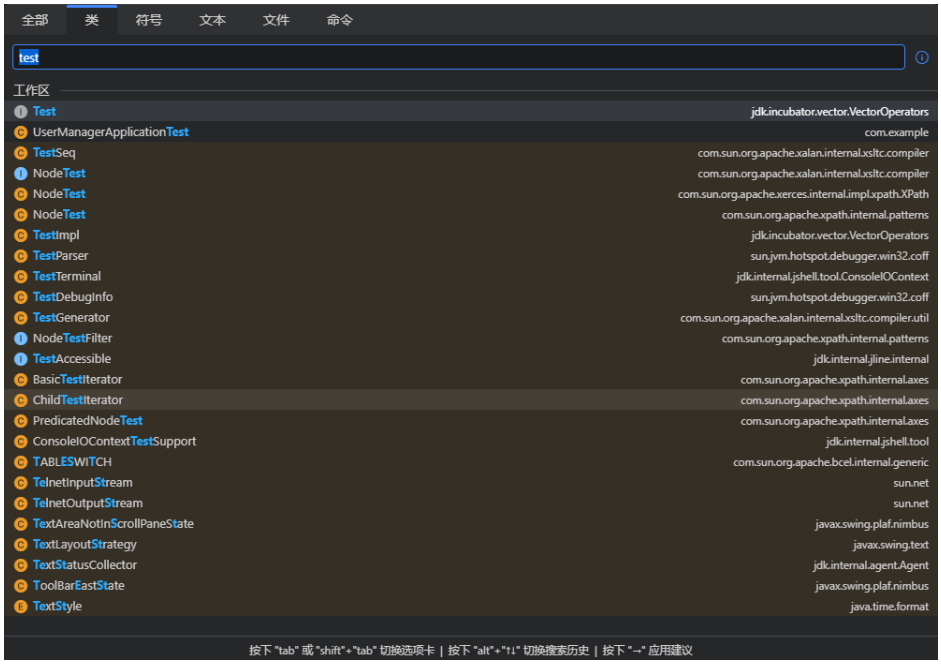
使用“**查找引用**”视图的工具栏按钮可以显示导入语句（  ）、显示读取访问权限（  ）和显示写入访问权限（  ）中的符号引用。

还可以通过“快速查看”视图快速查看引用，该视图显示在当前编辑器中，因此用户不需要切换上下文。要在“**快速查看**”视图中查看所有引用，右键单击符号，然后在上下文菜单中选择“快速查看 > 查看用法”。

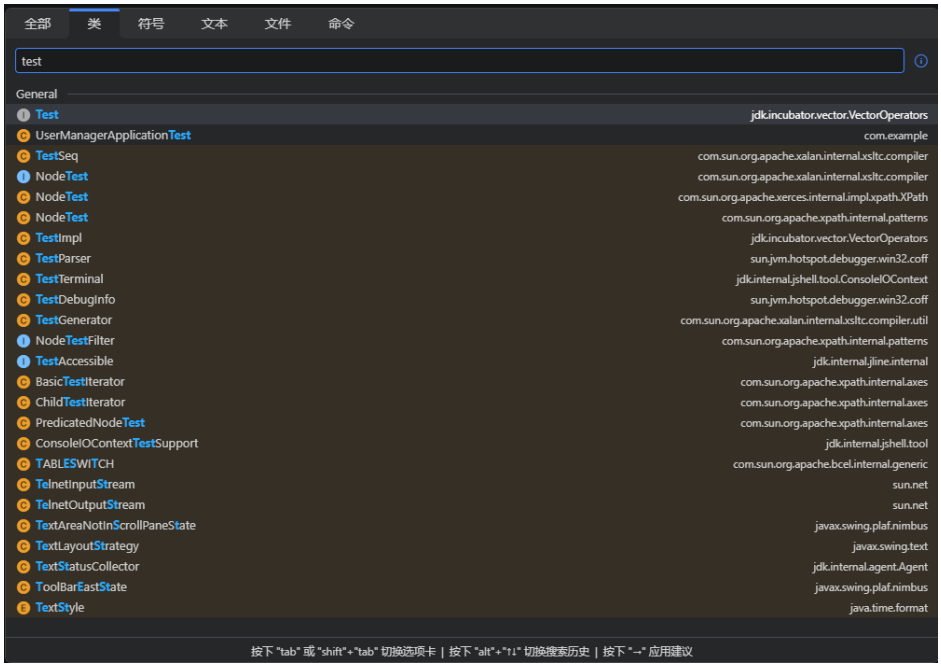


2.3.3.6 转到编辑器中的符号

使用“**Ctrl+Shift+O**”或“**Ctrl+Shift+Alt+N**”（IDEA快捷键方案）或“**Ctrl+F12**”（IDEA快捷键方案）导航到文件中的符号。按向上或向下键导航到所需的位置。



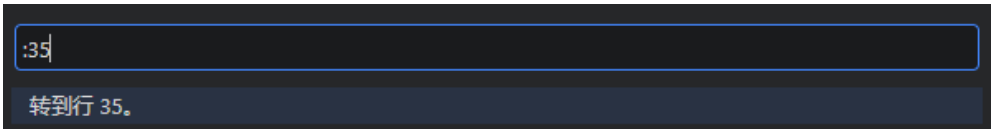
要导航到项目中的符号，请按“**Shift+Alt+T**”，开始键入符号的名称，然后使用向上或向下键导航到它。



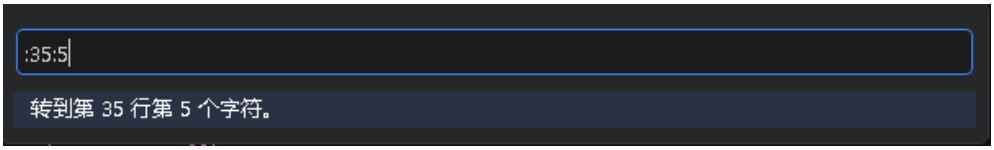
2.3.3.7 转到行

用户可以跳转到当前打开的文件中的特定行，也可以跳转到任意文件中的特定行。

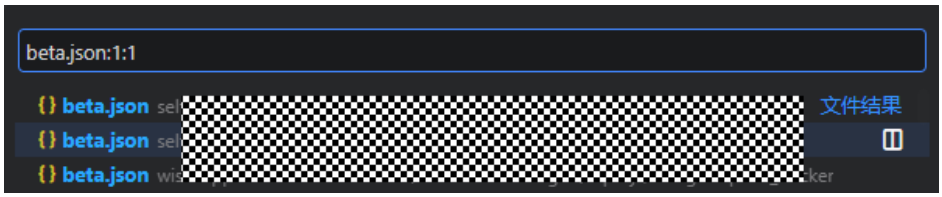
- 要跳到当前打开的文件中的特定行，请先按“**Ctrl+G**”，然后在弹出的窗口中键入行号，整个查询格式为:**行号**。

- 如果要额外指定要跳转至的列号，请再键入一个冒号(:)，最后再键入列号，整个查询格式为:**行号:列号**。

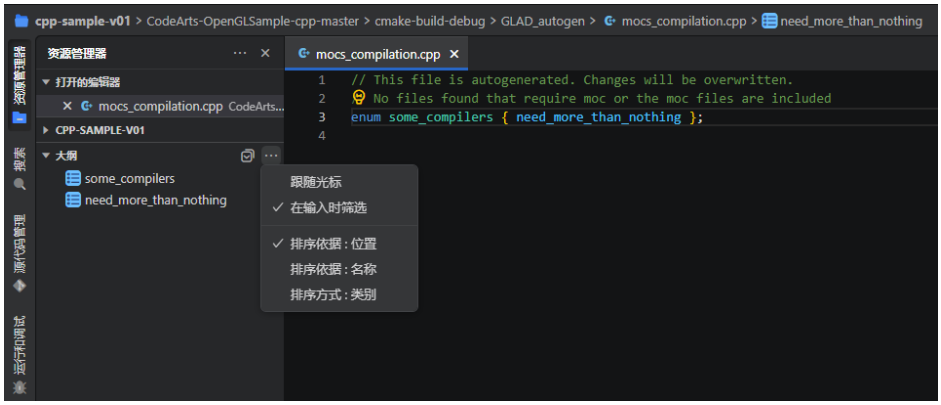
  

- 要跳转到任意文件中的行，请执行以下操作：
  a. 先按“**Ctrl+G**”，在打开的弹窗中，删除前导冒号(:)，然后开始键入所需的文件名。
  b. 文件名后键入冒号(:)，再键入需要跳转至的行号。如果要额外指定要跳转至的列号，请再键入一个冒号(:)，最后再键入列号，整个查询格式为**文件名:行号:列号**。
  c. 使用向上和向下键选择所需的文件，然后按“**Enter**”键跳转至该文件。



2.3.3.8 大纲视图

对于Cpp客户端（最新版Java客户端没有大纲视图），当“大纲”视图展开时，将显示当前活动编辑器的符号树。“大纲”视图提供了几种排序模式、可选的光标跟踪，并允许用户在键入时查找或筛选符号。错误和警告（如果有）将显示在受影响符号旁边的“大纲”视图中。



以下几项“大纲”视图设置，允许用户启用/禁用图标并控制错误和警告的显示（默认情况下都启用）：

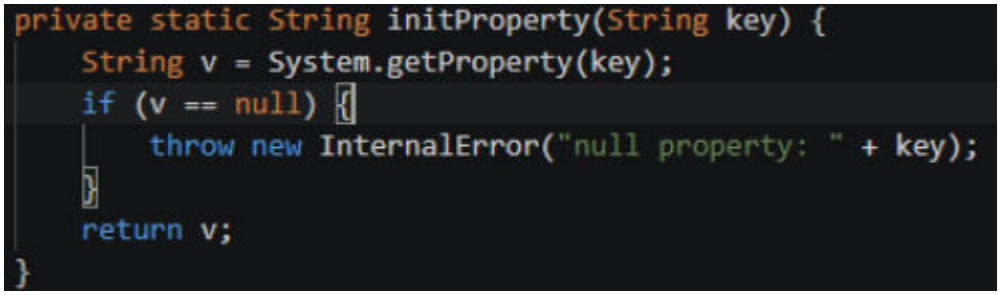
- outline.icons：显示大纲图标。
- outline.problems.enabled：显示大纲元素上的错误和警告。
- outline.problems.badges：对错误和警告使用徽章。
- outline.problems.colors：对错误和警告添加颜色。

2.3.3.9 括号匹配

当光标靠近其中一个括号，匹配的括号就会高亮显示。

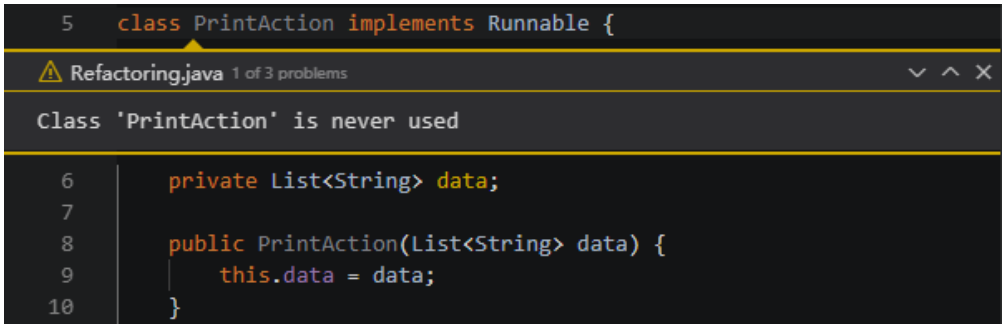
可以使用“**Ctrl+Shift+**”跳转到匹配的括号，帮助用户更好地理解和维护代码。

这个功能能够自动在代码中查找匹配的括号，括号是编程语言中非常重要的一部分，例如圆括号、方括号和大括号，它们用于组合代码块和定义控制结构。



2.3.3.10 错误和警告

要循环浏览当前文件中的错误或警告，请按“F2”或“Shift+F2”，这将显示一个内联区域，详细说明问题和可能的代码操作。



须知

有关CodeArts IDE中代码验证的更多详细信息，请参见**代码校验**。

2.3.4 代码校验

2.3.4.1 简介

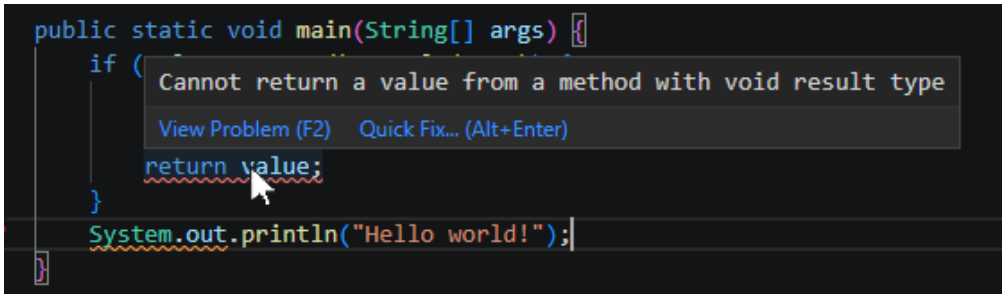
编写代码时，CodeArts IDE会根据预定义的验证规则自动在后台分析代码。CodeArts IDE可以发现各种问题，识别可能的错误、拼写问题等。这有助于用户在运行代码之前检测和更正代码中的问题。CodeArts IDE提供了快速修复功能，让用户可以迅速修复它们。

警告和错误显示在CodeArts IDE用户界面的几个位置：

- 状态栏显示所有错误和警告的摘要。
- “问题”视图列出了当前打开的文件中的所有问题。
- 打开文件时，所有错误和警告都将直接在代码编辑器中呈现，与文本内联，并在概述标尺中呈现。

2.3.4.2 在代码编辑器中查看问题

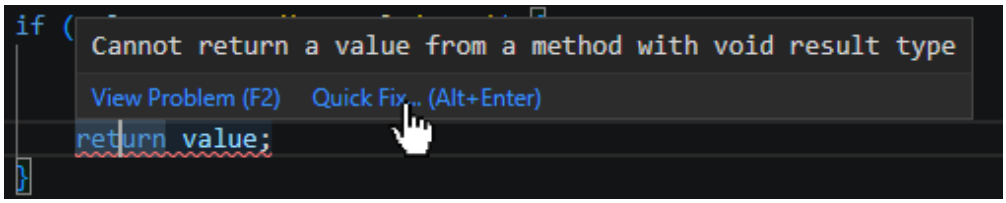
CodeArts IDE自动高亮显示代码中检测到的所有问题。要查看问题详细信息，请将鼠标悬停在代码编辑器中高亮显示的位置。



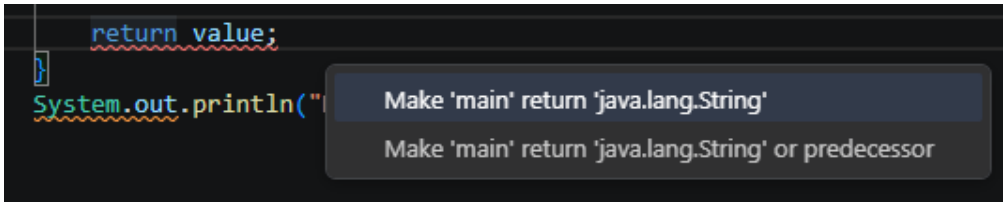
应用快速修复

如果检测到的问题有快速修复可用，用户可以即时修复它。

1. 单击问题描述中的“**快速修复**”链接。或将光标定位在高亮显示的位置，然后按“**Alt+Enter**”键。

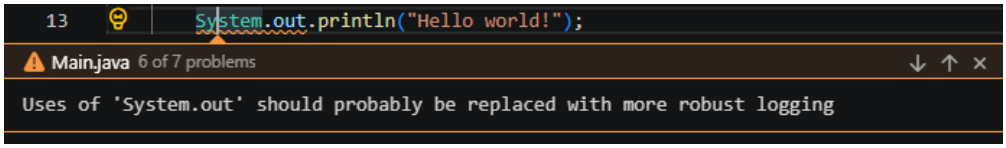


2. 在弹出菜单中，选择所需的快速修复。



查看问题详情

- 要在快速查看器中打开问题说明，请单击“**查看问题**”链接。或将光标定位在高亮显示的位置，然后按“**F2**”。

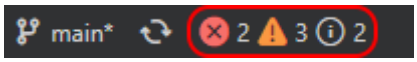


- 在快速查看器中，单击↓(转到下一个问题)和↑(转到上一个问题)，或按“**Alt+F8**”或“**F2**”(IDEA快捷键方案)或“**Shift+Alt+F8**”或“**Shift+F2**”(IDEA快捷键方案)在当前文件中的错误之间跳转。

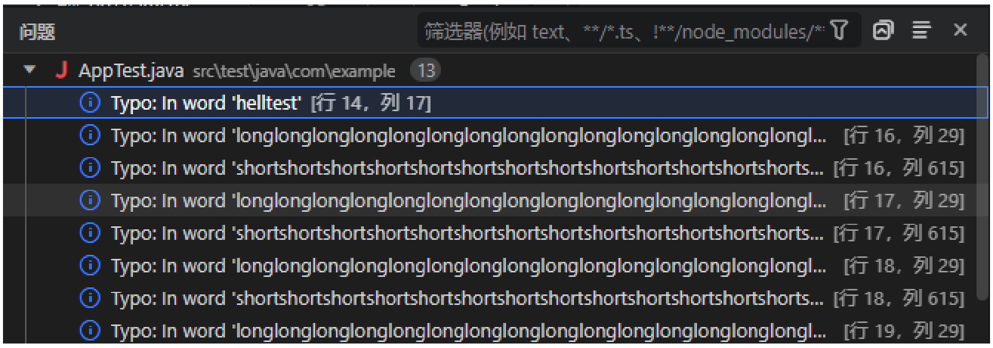
2.3.4.3 使用“问题”视图

“问题”视图列出了当前打开的文件中的所有问题。要打开它，请执行以下任何操作：

1. 单击CodeArts IDE状态栏中的问题摘要。



2. 在主菜单中，选择“查看 > 问题”。
3. 按“**Ctrl+Shift+M**”或“**Shift+Escape**”（IDEA快捷键方案）或“**Alt+0**”（IDEA快捷键方案）。

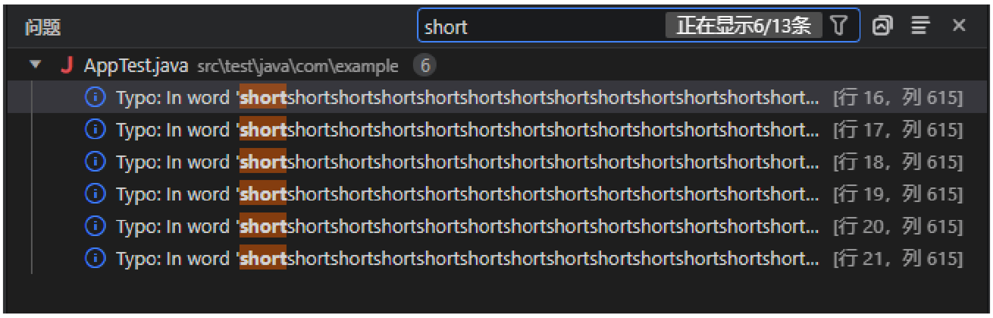


默认情况下，“**问题**”视图显示按其所属文件分组的问题。要在树视图和平面表视图之间切换，请单击“**问题**”视图标题栏中的“**以表形式查看**”（ ）或“**以树形式查看**”（ ）按钮。

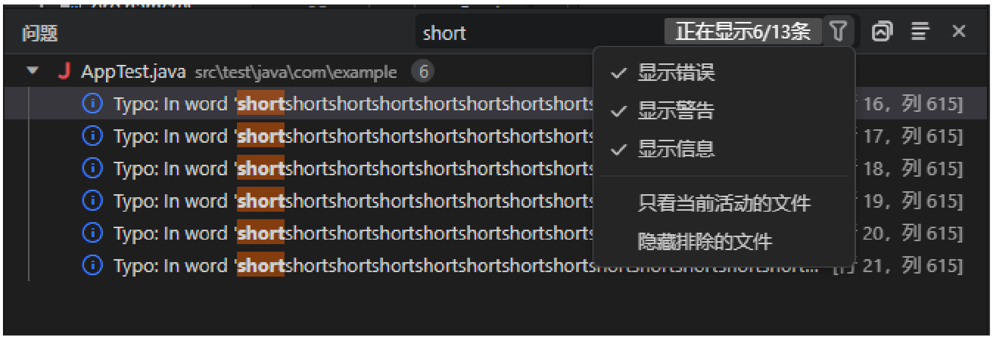
过滤问题列表

要限制显示的问题数量，用户可以过滤列表。

- 要按任意查询进行筛选，请在搜索框中键入该查询。支持“**/*.java”等通配符模式。



- 要使用预定义的过滤器，请单击搜索框中的“过滤器”按钮（ ），然后在弹出菜单中选择所需的过滤器。



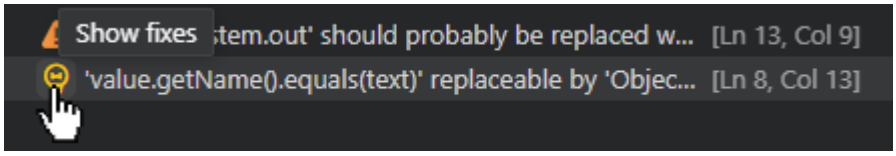
须知

隐藏排除的文件筛选器可通过file.exclude 设置配置为排除的文件。

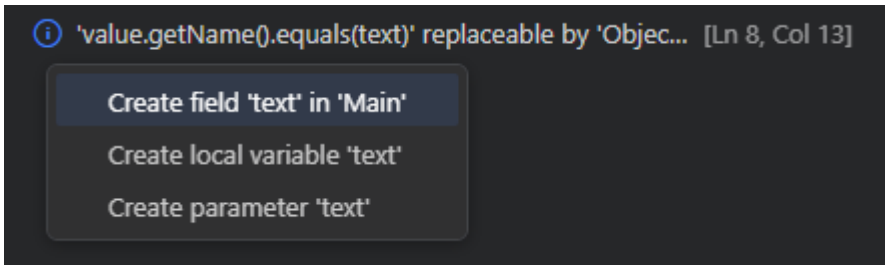
应用快速修复

如果检测到的问题有快速修复可用，用户可以直接从“**问题**”视图应用它们。

步骤1 鼠标悬停到问题图标，使其更改为，然后单击它。



步骤2 在弹出式菜单中，选择要应用的快速修复项。

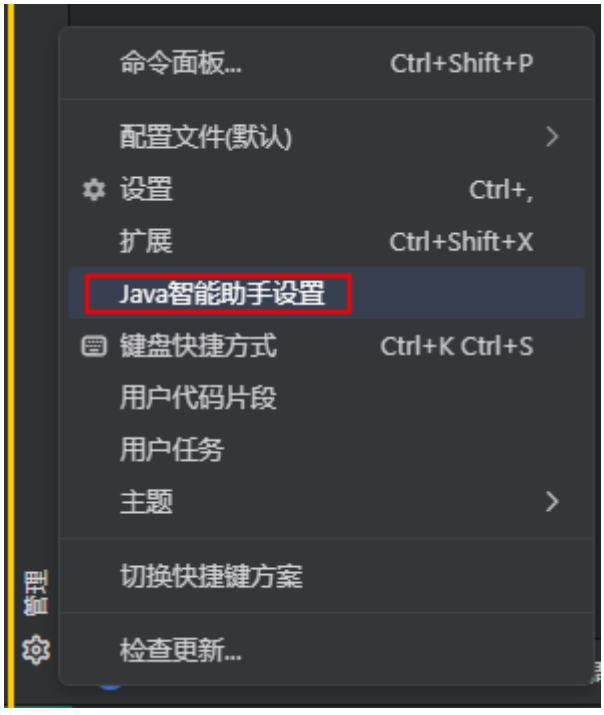


----结束

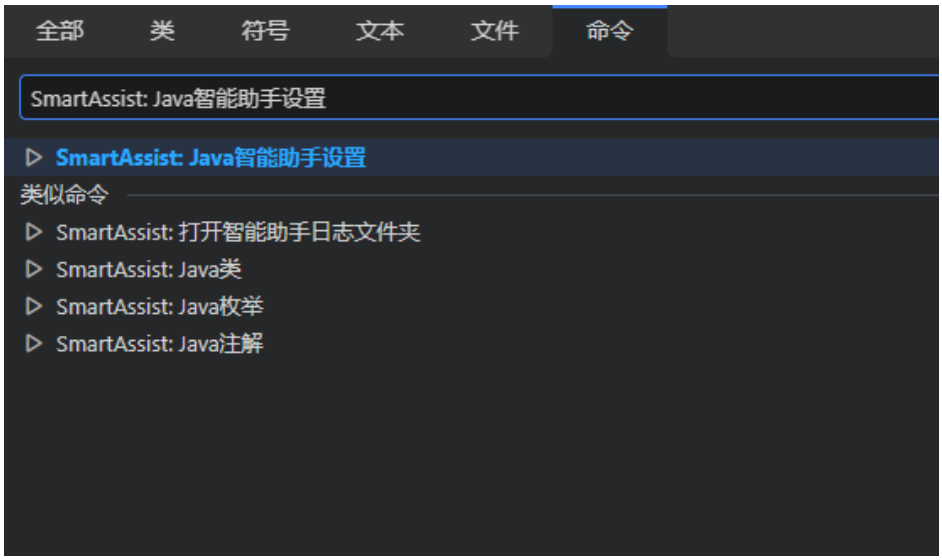
2.3.4.4 配置校验规则

用户可以自定义应用于代码的验证规则集。

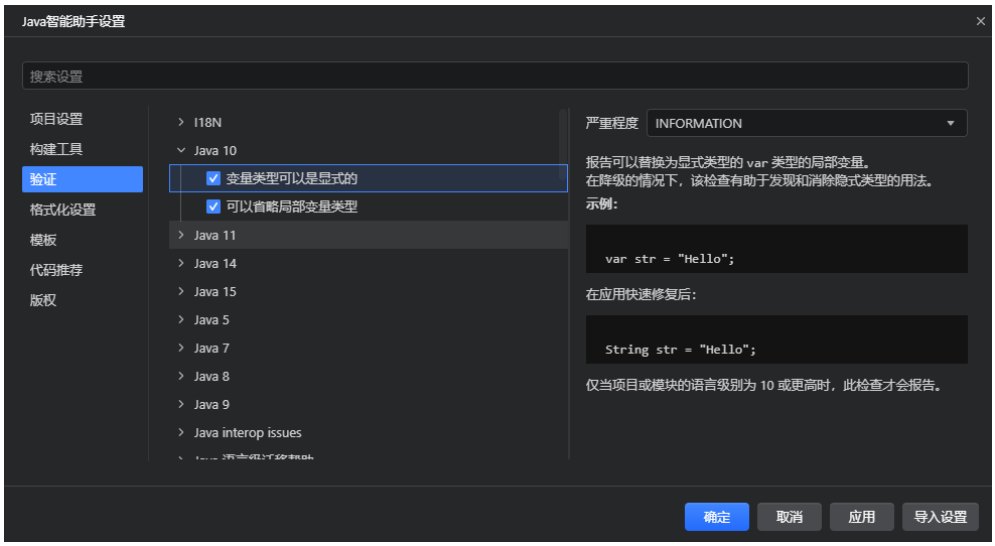
1. 通过执行以下任一操作打开**“Java智能助手设置”**对话框。
   - 通过单击左下角**“管理”**>“Java智能助手设置”



   - 通过**“Ctrl+Shift+P”**或者**“双击Ctrl”**打开“命令”面板，输入**SmartAssist: Java智能助手设置**命令。



2. 使用搜索字段快速定位验证规则。然后配置如下：
   - 要启用或禁用规则，请使用其名称旁边的复选框。
   - 要调整相应代码问题在代码编辑器中突出显示的方式，请在**“严重程度”**列表中选择所需的严重性级别。



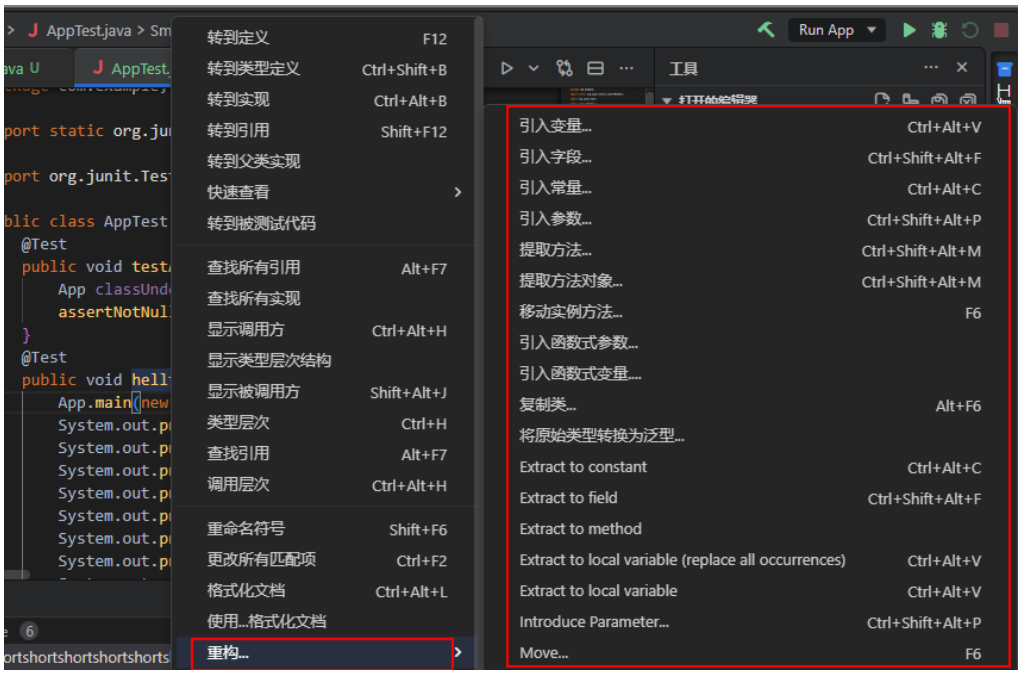
须知

有关配置Java项目的更多详细信息，请参见**配置Java项目**。

2.3.5 重构

2.3.5.1 简介

代码重构可以通过重构代码而不修改其运行行为来提高项目的质量和可维护性。CodeArts IDE支持重构操作，以在编辑器中改进代码库。



例如，用于避免重复代码的常见重构是提取方法重构，在这种重构中，用户可以将希望重用的代码拉到其自己的共享方法中。

重构由语言服务提供，CodeArts IDE内置了对TypeScript、JavaScript和Java的重构支持。

2.3.5.2 代码操作

在CodeArts IDE中，代码操作可以为检测到的问题提供重构和快速修复（以绿色曲线突出显示）。

如果代码操作可用，则当光标位于曲线或选定文本区域上时，灯泡图标将显示在代码附近。

单击代码操作灯泡图标或使用**“快速修复”**命令（**“Alt+Enter”**）将显示快速修复和重构建议。如果用户只想查看没有快速修复的重构，请使用**“重构”**命令。

说明

要禁用代码编辑器中的代码操作灯泡图标，请调整editor.lightbulb.enable设置。用户仍然可以通过快速修复命令和**“Alt+Enter”**键盘快捷键打开快速修复。

2.3.5.3 重构操作

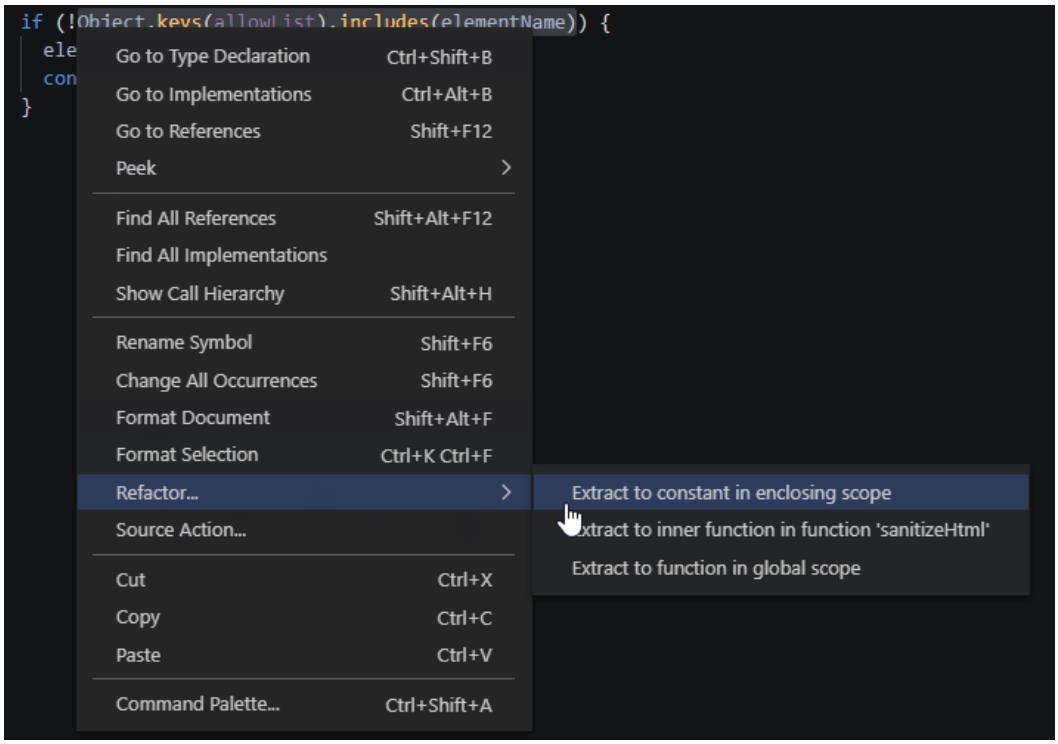
提取方法

选择要提取的代码，然后单击装订线中的灯泡图标，或按**“Alt+Enter”**键查看可用的重构（仅适用于IDEA快捷键方案）。

源代码片段可以提取到新方法中，也可以提取到不同范围的新函数中。在提取重构期间，系统将提示用户提供有意义的名称。

提取变量

TypeScript语言服务提供**Extract to constant**重构，为当前选定的表达式创建新的局部变量：



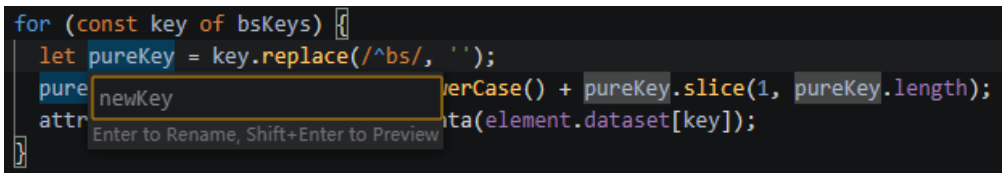
使用类时，还可以将值提取到新属性中。

2.3.5.4 重命名符号

重命名是与重构源代码相关的常见操作，CodeArts IDE有一个单独的**“重命名符号”**命令(“Shift+F6”)。

某些语言支持跨文件重命名符号。按“Shift+F6”，键入新的所需名称，然后按“Enter”键。文件中符号的所有用法都将重命名。

这个功能允许在项目中安全地更改一个变量、函数、类或其他标识符的名称，同时自动更新所有对该标识符的引用。从而确保代码的一致性和正确性，避免手动更改时可能出现的错误。



2.3.5.5 代码操作的键绑定

editor.action.codeAction命令允许用户为特定代码操作配置键绑定。例如，此键绑定触发提取函数重构代码操作：

```
{
  "key": "ctrl+shift+r ctrl+e",
  "command": "editor.action.codeAction",
```

```
  "args": {
    "kind": "refactor.extract.function"
  }
}
```

代码操作类型由扩展使用增强的CodeActionProvided API指定。种类是分层的，因此**"kind": "refactor"**将显示所有重构代码操作，而**"kind": "refactor.extract.function"**将仅显示提取函数重构。

使用上述键绑定，如果只有单个**"refactor.extract.function"**代码操作可用，则将自动应用该代码操作。如果有多个提取函数代码操作可用，用户可以从上下文菜单中选择所需的代码操作。

用户还可以使用应用参数控制自动应用代码操作的方式和时间：

```
{
  "key": "ctrl+shift+r ctrl+e",
  "command": "editor.action.codeAction",
  "args": {
    "kind": "refactor.extract.function",
    "apply": "first"
  }
}
```

“apply”的有效值：

- first：始终自动应用第一个可用的代码操作。
- ifSingle：默认情况下。如果只有一个代码操作可用，则自动应用代码操作。否则，显示上下文菜单。
- never：始终显示代码操作上下文菜单，即使只有一个代码操作可用。

当代码操作键绑定配置为**"preferred": true**时，仅显示首选的快速修复和重构。首选的快速修复解决了基础错误，而首选的重构是最常见的重构选择。例如，虽然可能存在多个refactor.extract.const重构，但每个重构都提取到文件中的不同作用域，但首选的refactor.extract.constant重构项是提取到局部变量的重构。

此键绑定使用**"preferred": true**创建始终尝试将选定源代码提取到本地作用域中的常量的重构：

```
{
  "key": "shift+ctrl+e",
  "command": "editor.action.codeAction",
  "args": {
    "kind": "refactor.extract.constant",
    "preferred": true,
    "apply": "ifSingle"
  }
}
```

2.4 C/C++

2.4.1 准备工作

Linux 环境工具安装

在Linux环境上IDE未提供内置的MinGW工具，用户需自行根据操作系统安装gcc、g++、gdb工具。以apt包管理工具安装为例：

- 安装gcc：**sudo apt install gcc**
- 安装g++： **sudo apt install g++**
- 安装gdb： **sudo apt install gdb**

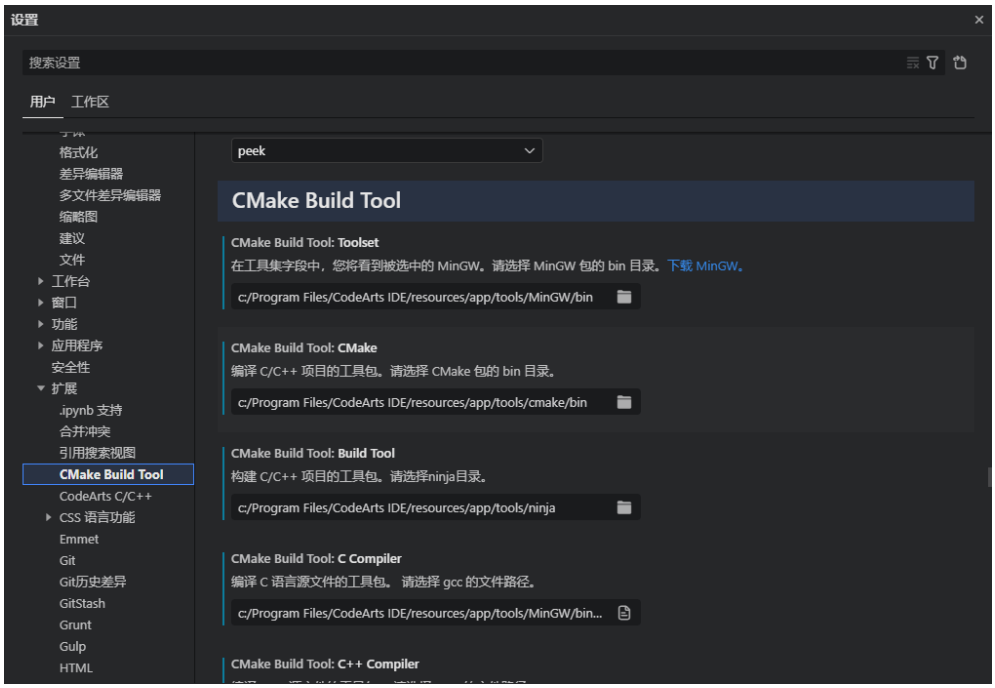
IDE 配置路径

安装好gcc、g++、gdb工具后，需在IDE的配置项中自行配置安装路径，以及配置好CMake和Ninja的安装路径。

CMake路径：“~/CodeArts IDE/resources/app/tools/cmake/bin”

Ninja路径：“~/CodeArts IDE/resources/app/tools/ninja”

配置路径：“设置 > 扩展 > CMake Build Tool”， 配置对应路径。



2.4.2 创建 C/C++工程

CodeArts IDE for Cpp提供了创建C或C++工程的能力，可参考以下步骤进行创建：

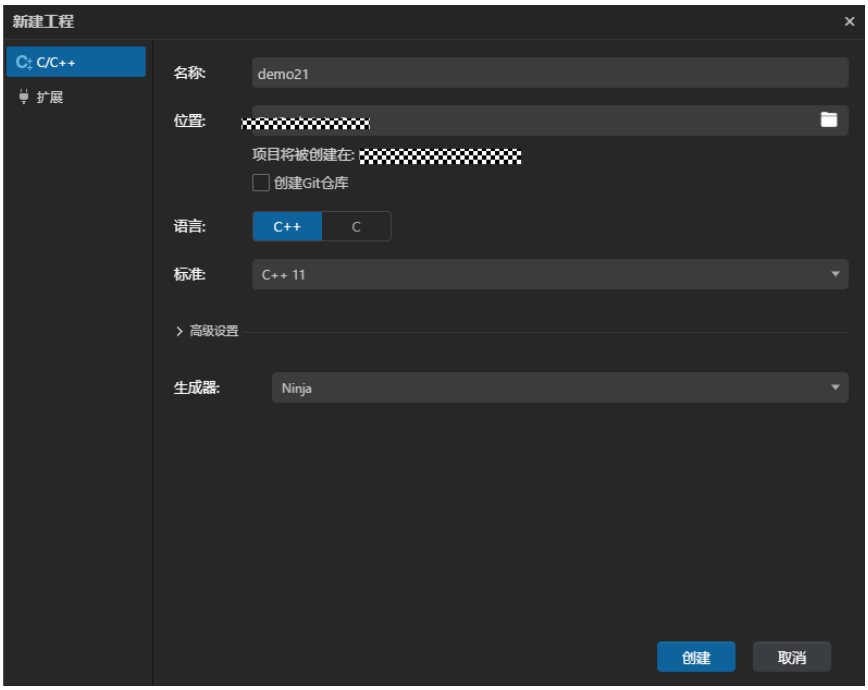
步骤1 单击顶部菜单“文件 > 新建 > 工程...”，进入新建工程页面。

步骤2 语言选择“C/C++”。

步骤3 填写表单指定的内容并单击创建按钮。

步骤4 等待工程创建完成并打开项目。

----结束

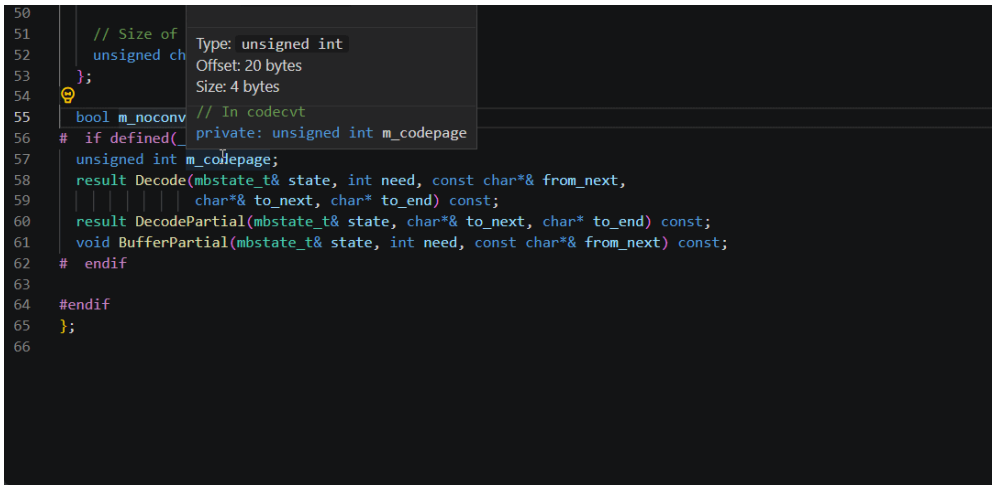


2.4.3 C/C++代码编写

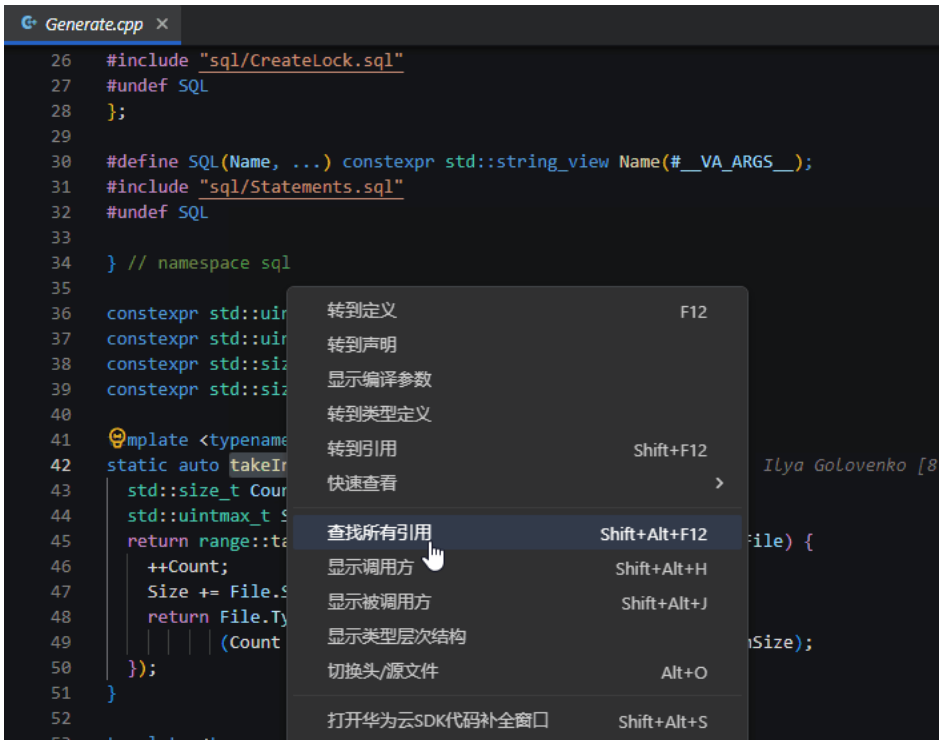
2.4.3.1 编码基础操作

CodeArts IDE for Cpp包含了内置的语法着色、定义预览、跳转定义、类继承关系图和调用关系图等一些编码基础功能。

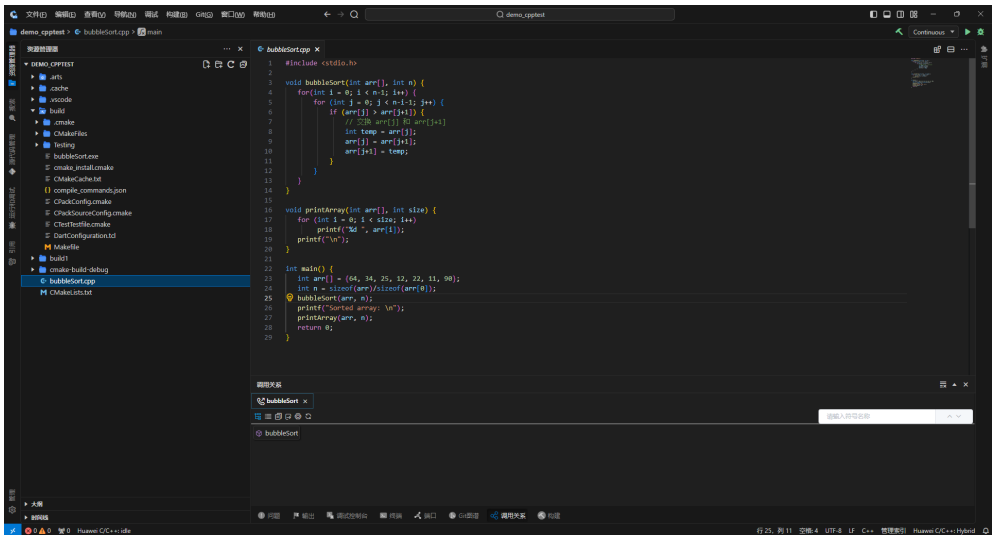
- **语法着色** - 该功能可对函数、类型、局部变量、全部变量、宏、枚举、成员变量等上色。
- **跳转定义** - 当光标放在代码处，“**Ctrl+单击**”或者“**F12**”跳转到定义，或者使用“**Ctrl+Alt+单击**”会打开定义到侧边栏。
- **定义预览** - 当光标移至符号处，则会有符号定义的悬停预览，也可以用“**Alt+F12**”的快捷键进行文件内的符号预览。



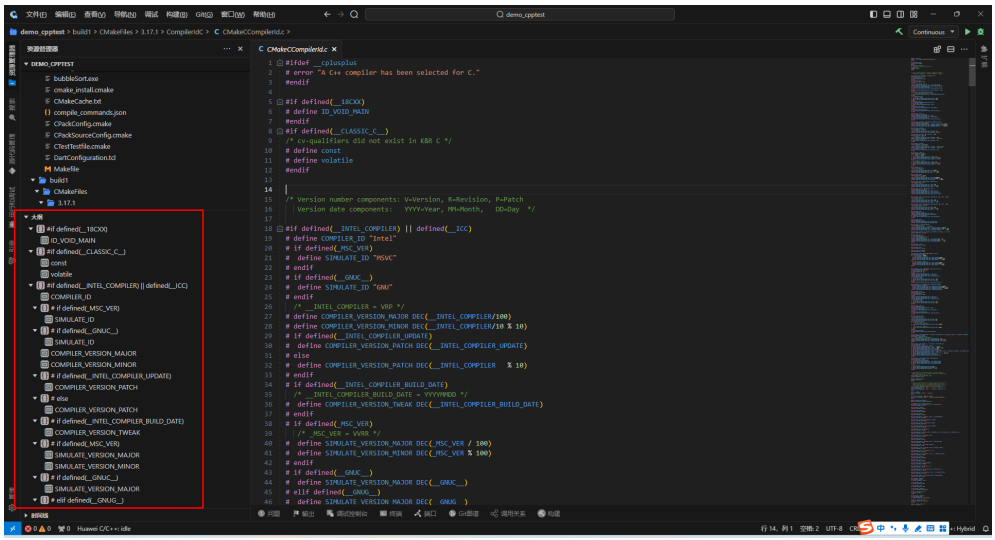
- **查找所有引用** - 当光标单击或者选择到需要查找的符号，“右键菜单 > 查找所有引用”或者使用快捷键“**Shift+Alt+F12**”会打开定义在侧边栏。



- **调用关系图** - 当光标单击或选中需要调用关系图的函数时，“右键菜单 > 调用关系图”，或可以使用快捷键“**Shift+Alt+J**”调出。在关系图中，也可以单击需要查看的函数并导航到该函数，同时也能够查看子类和基类。



- **符号大纲** - “资源管理器 > 右上角... > 大纲”，即可打开符号大纲。打开大纲后，双击函数即可到达函数定义的位置，并且当前符号大纲可跟随光标移动（此功能需要在大纲菜单栏中勾选“**跟随光标**”选项）。

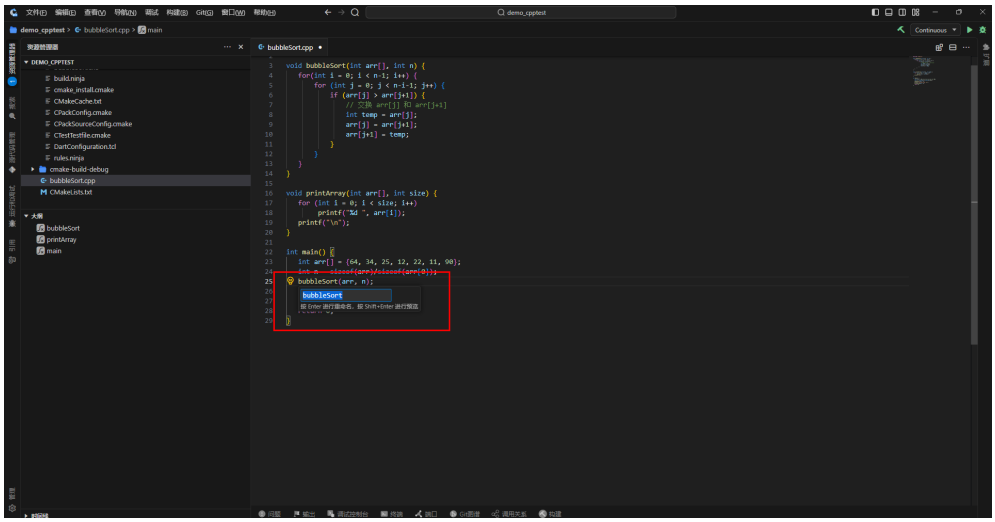


2.4.3.2 代码编写操作

CodeArts IDE for Cpp包含了内置的符号重命名，提取重构，代码补全/提示，实时语法检查等一些高级代码编写功能。

- **符号重命名（Rename Symbol）**

  最基础的重构之一，但是变量或方法名字的可读性非常重要。在光标选中某个变量或方法后，右键单击以调出编辑器上下文菜单并且选择重命名符号或直接按“**F2**”，来重命名整个C/C++项目中所有用到该命名的地方。

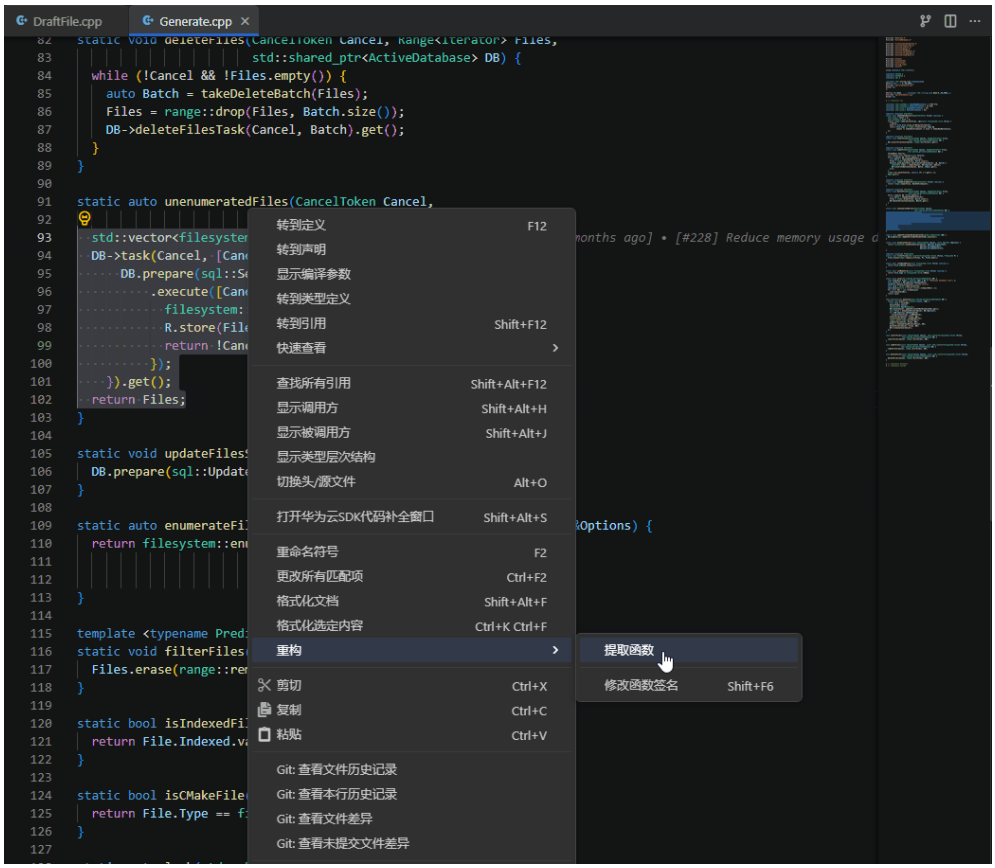


- **提取重构（Extraction refactoring）**

  CodeArts IDE for Cpp支持将字段、方法和参数提取到新类中，根据提取的内容，会提供不同的重构类型。

  可用的C/C++重构类型包括：

  提取函数/方法（Extract method）- 将选定的语句或表达式提取到文件中的新方法或新函数。



在选择**提取方法（Extract method）**重构后，输入提取的方法/函数的名称。

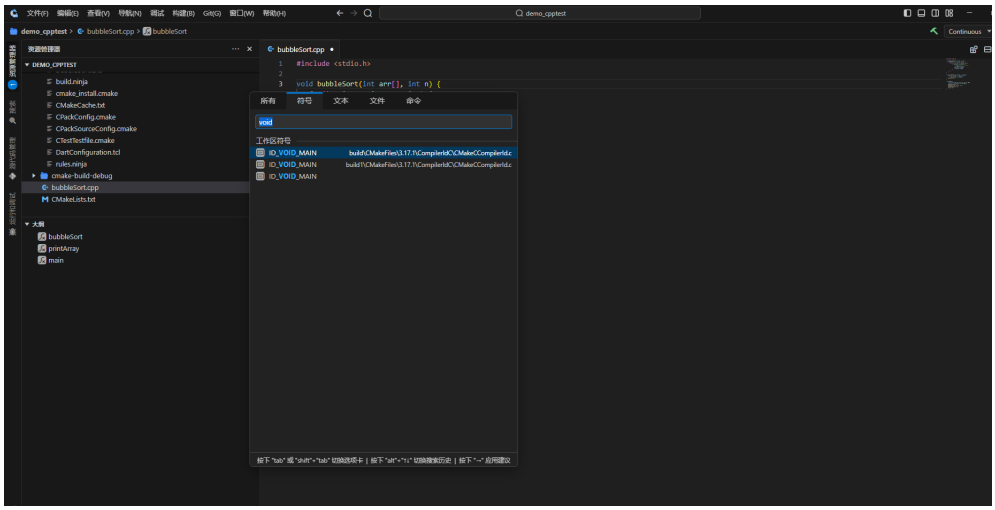
提取表达式到变量（Extract subexpression to variable）- 将选定的表达式提取为文件中的新变量。

- **代码补全/提示（Code Completion/Hinting）**

  CodeArts IDE for Cpp代码补全包含了各种代码编辑功能，包括：代码完成，快速信息，成员列表以及参数信息。当用户输入字符时，代码补全如果知道可能的补全选项，则会自动弹出成员列表。如果用户继续输入字符，成员列表（变量，方法等）将被过滤为仅包含用户输入字符的成员。用户可通过左键单击或者按**“Enter”**或**“Tab”**键插入选定的成员名称。该功能会提供各种提示信息，帮助用户更加方便快速地编辑代码。

- **全局符号搜索（Global Symbol Search）**

  按**“Ctrl+T”**，弹出搜索框，输入需要查找的符号，页面会显示出当前文件夹所有包含此符号的文件，单击即可跳转。或者按向上或向下键选择并按**“Enter”**导航到想要的位置。



- **实时检查编译错误（该功能依赖compile_commands.json文件）**

  实时检查编译错误是指解决编码错误的建议编辑，包括自动补全，实时语法检查等。

  当编译错误时，会在错误处出现波浪线。可将光标移动或单击到C/C++的代码错误上时，会显示黄色灯泡，表示可以使用快速修复按钮。单击灯泡或按“**Ctrl+.**”，会显示可用的快速修复和重构列表。

- **Compile_commands.json管理功能**

  Compiler模式功能全面，但需要“**compile_commands.json**”文件编译数据库才能正常工作，可使用三种方式获取该文件。

- 使用内置CMake Build Tool插件（推荐）。构建CMake项目，会自动生成“cmake-build-debug/compile_commands.json”文件， 并且插件会自动将该文件导入到“.arts”文件夹。
- 使用CMake生成。 如果当前工程是CMake工程，可以在CMake构建命令添加参数“**-DCMAKE_EXPORT_COMPILE_COMMANDS=1**”生成“**compile_commands.json**”，并通过“帮助 > 显示所有命令 > CodeArts C/C++:导入编译数据库文件”命令导入。
- 使用CodeArts C/C++提供的**Generate**命令。可通过“帮助 > 显示所有命令 > CodeArts C/C++:生成编译数据库文件”，并选择存放源文件的文件夹，该方法用于分析头文件生成的对应的编译数据库。

同时CodeArts C/C++也支持以下功能：

- 合并多个“**compile_commands.json**”文件。
- 移除“**compile_commands.json**”文件中重复的命令。
- 导入时为**clangd**提供额外的参数设置。
- 索引更新命令
  - 同步工程索引（“帮助 > 显示所有命令 > CodeArts C/C++:同步工程索引”）
  - 同步文件夹索引（“资源管理器选中文件夹，右键菜单 >CodeArts C/C++:同步文件夹索引”）
  - 同步文件索引（“资源管理器选中文件，右键菜单 > CodeArts C/C++:同步当前文件索引”）
  - 重置工程索引（“帮助 > 显示所有命令 > CodeArts C/C++:重建全项目索引”）

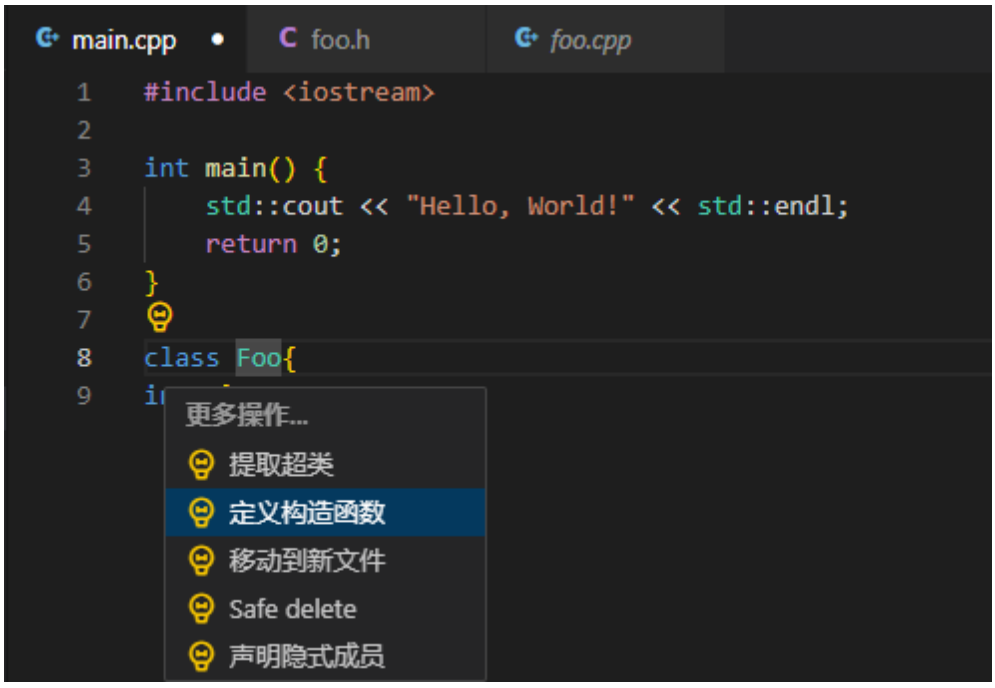
以上命令和功能在Compiler模式或Hybrid模式均有效。

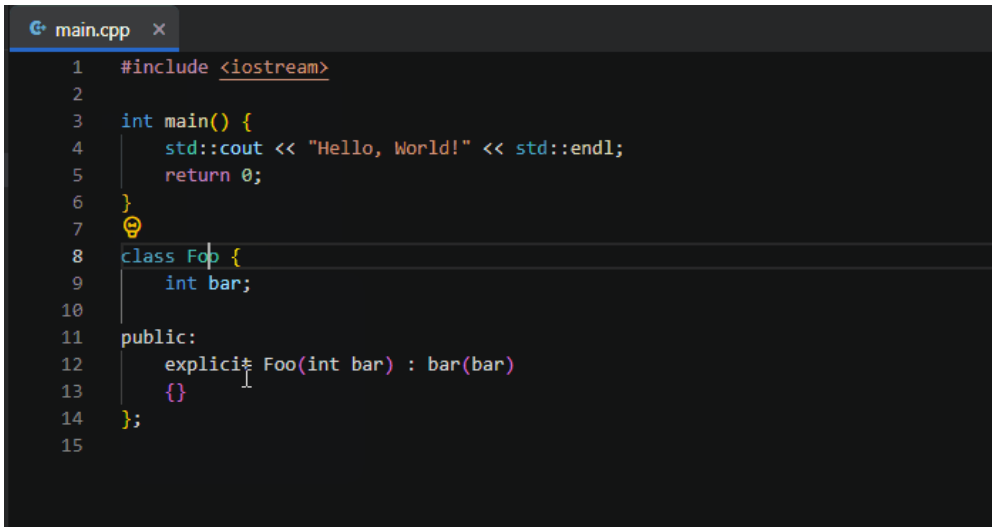
2.4.3.3 代码重构操作

重构是通过改变现有程序结构而不改变其功能和用途，来提高代码的可重用性和可维护性。CodeArts IDE支持重构操作，提供了多种重要的重构类型，来改变编辑器中的代码库。CodeArts IDE for Cpp内置了对C/C++重构的支持，本专题将展示C/C++ 语言服务的重构支持。注：重构功能需选择语言解析模式为Compiler模式。

定义构造函数（Define constructor）

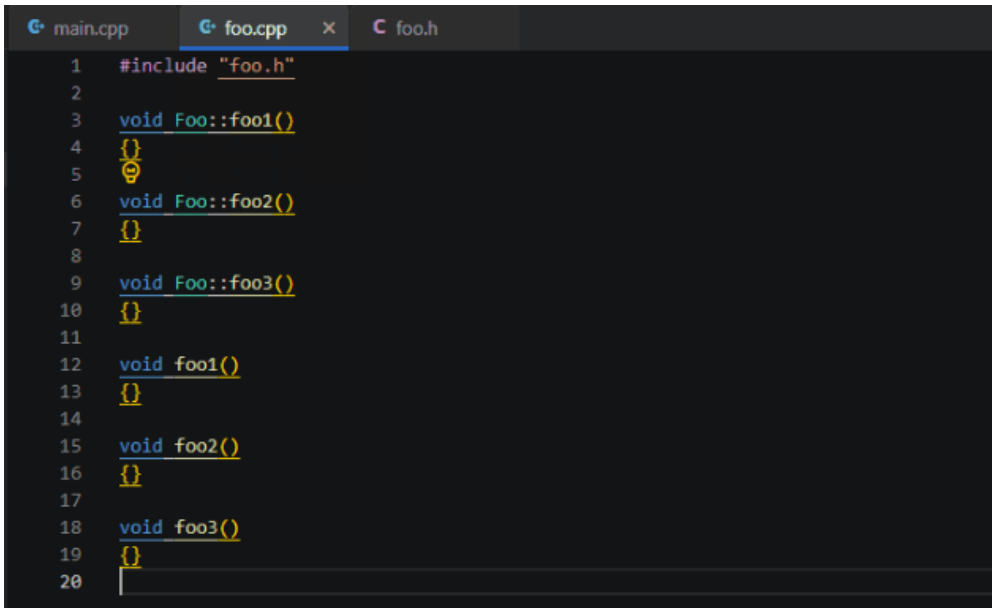
在每次创建类时，可以自动定义类的构造函数，并且初始化成员。当单击或选中类名时，可以单击左侧黄色灯泡选择定义构造函数。





根据声明顺序排序函数（Sort functions to declarations）

根据头文件中的声明顺序，排序当前定义函数/方法的顺序。当单击或选中当前函数/方法定义时，重构选项可用。



将定义添加到实现文件（Add definition to implementation file）

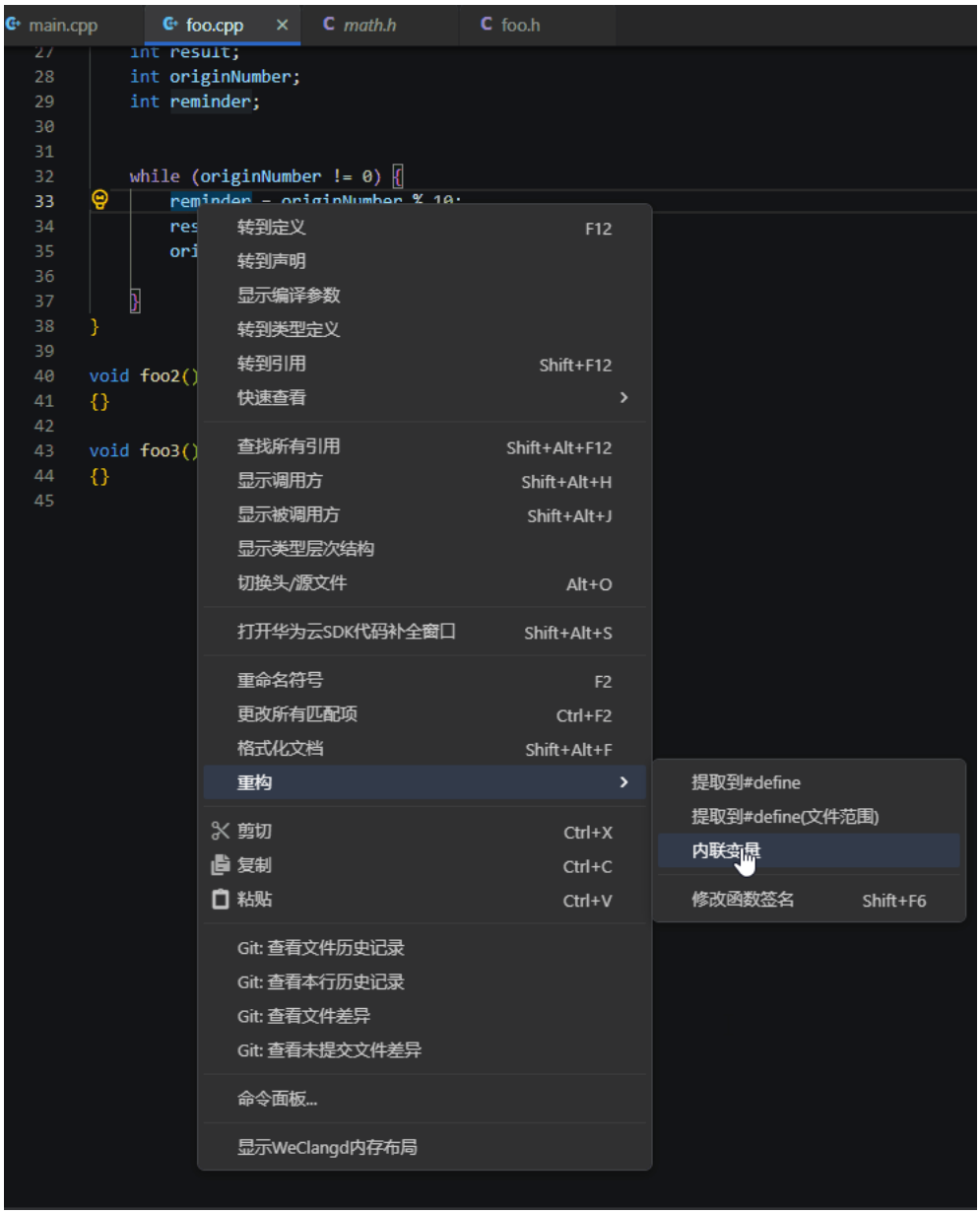
将头文件的定义添加到实现文件中。当单击或选中当前函数/方法时，重构选项可用。

交换 if 分支（Swap if branches）

如果当前条件只有if和else分支，选中代码片段后，选择**“交换 if 分支”（Swap if branches）**，可自动交换if和else分支。

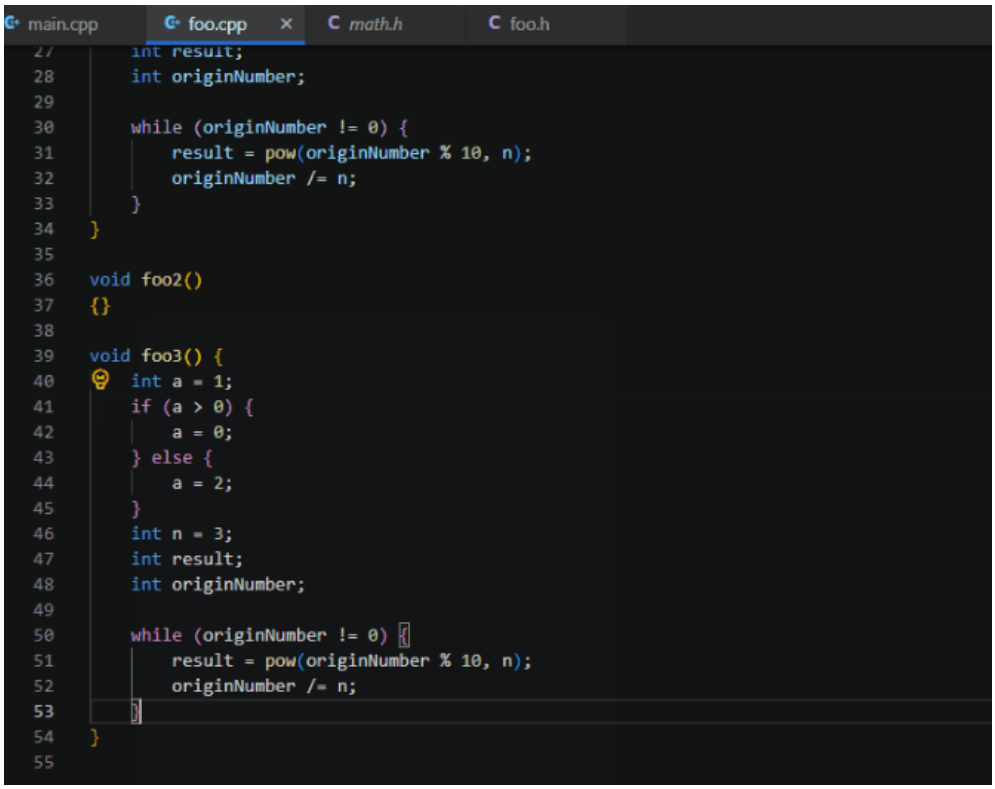
内联变量（Inline variable）

该功能可以用相应的值替换所有引用。假设计算值总是产生相同的结果。选中需要替换的内容，重构选项可用。



内联函数（Inline function）

该功能尝试使用适当的代码内联所有函数用法。它只能处理简单的功能，不支持内联方法、函数模板、主函数和在系统头文件中声明的函数。该功能可以内联所有函数引用。



生成 getter 和 setter（Generate getter and setter）

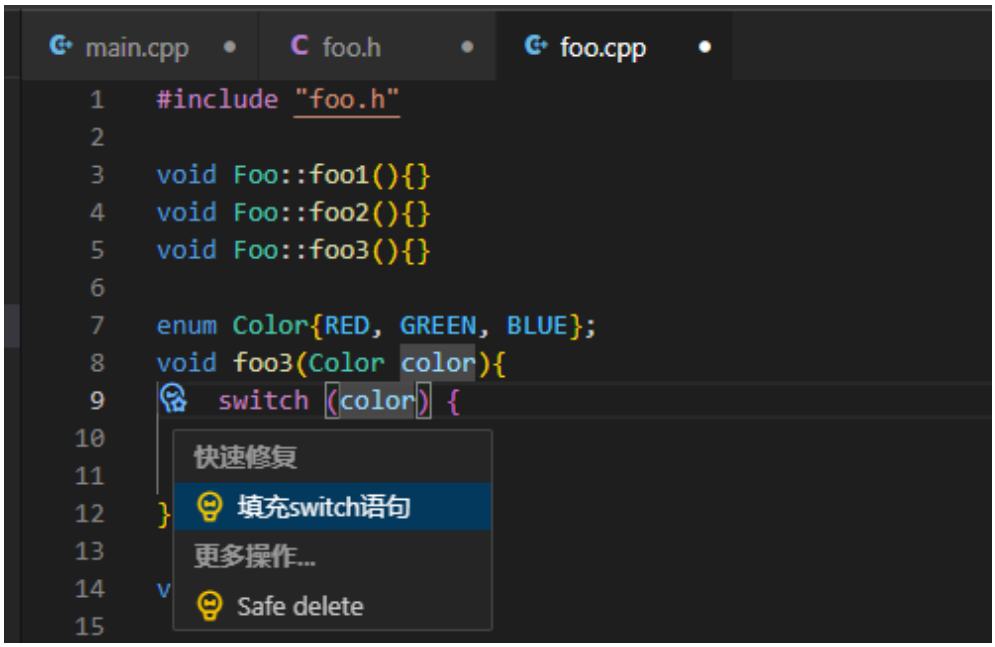
通过“**生成getter和setter**”（**Generate getter and setter**）来封装选定的类属性。同时也可以选择只“**生成getter**”（**Generate getter**）或者“**生成setter**”（**Generate setter**）选项。

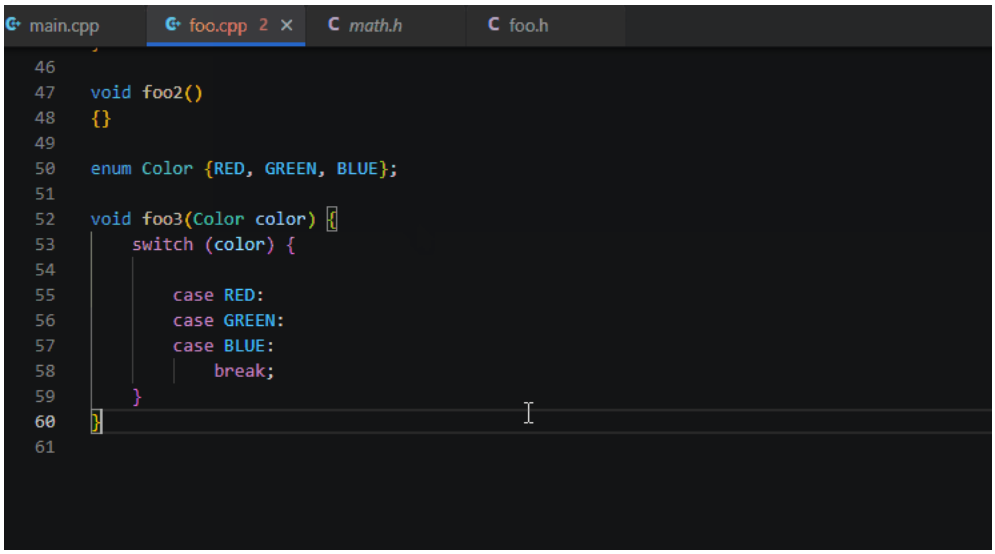
声明隐式成员（Declare implicit members）

此选项会将类的隐式成员在类中声明，当选中类名时，重构选项可用。

填充 switch 语句（Populate switch）

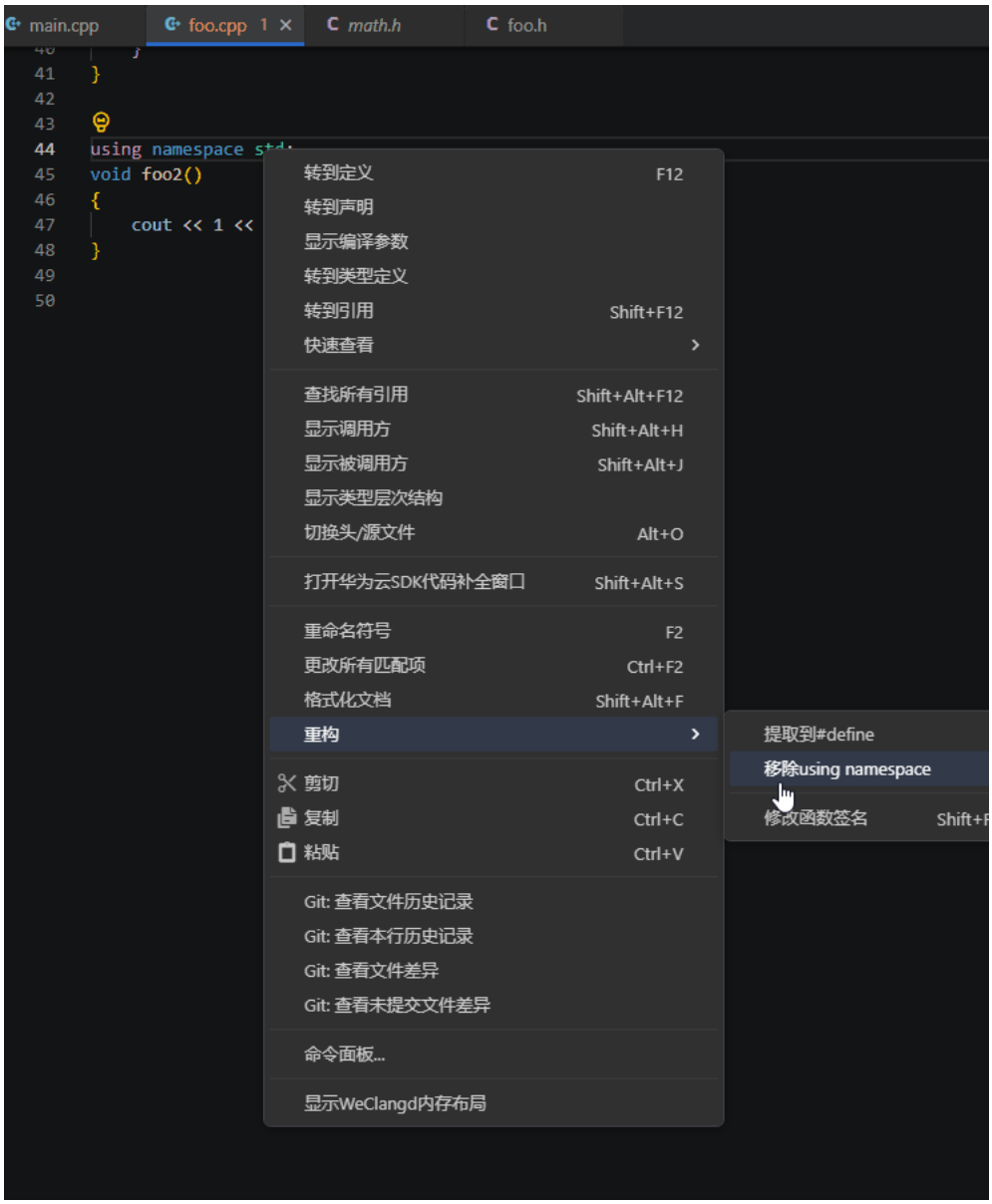
该功能可以自动填充switch语句。选中任意switch字段，并且单击黄色灯泡，选择**“填充switch语句”**。





移除 using namespace（Remove using namespace）

移除namespace功能，会自动移除所有使用到的namespace。当光标单击或选中namespace关键字时，重构选项可用。

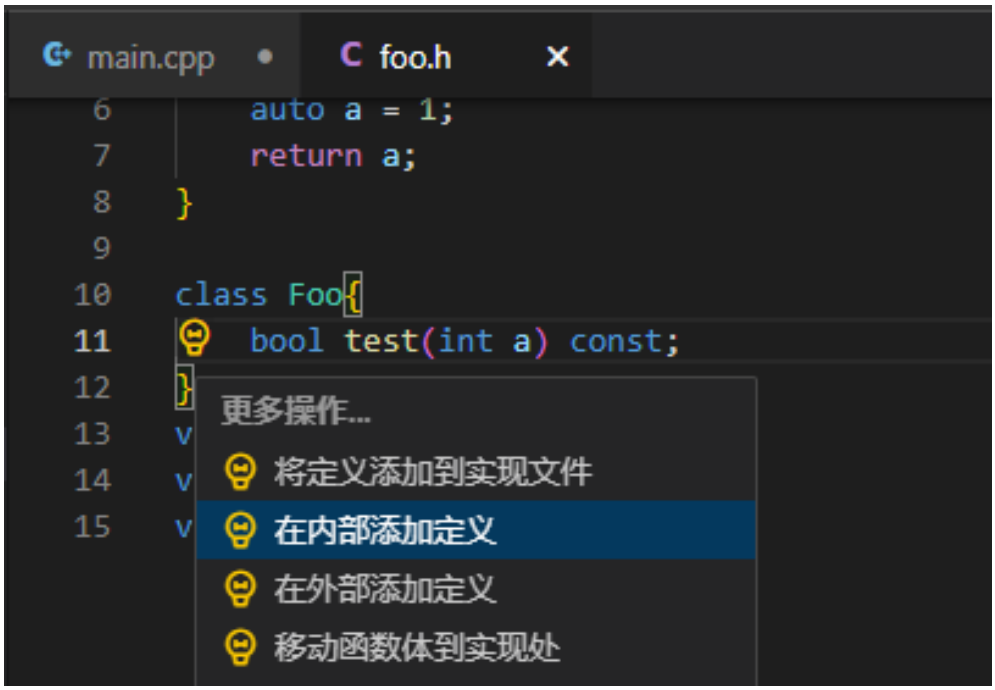


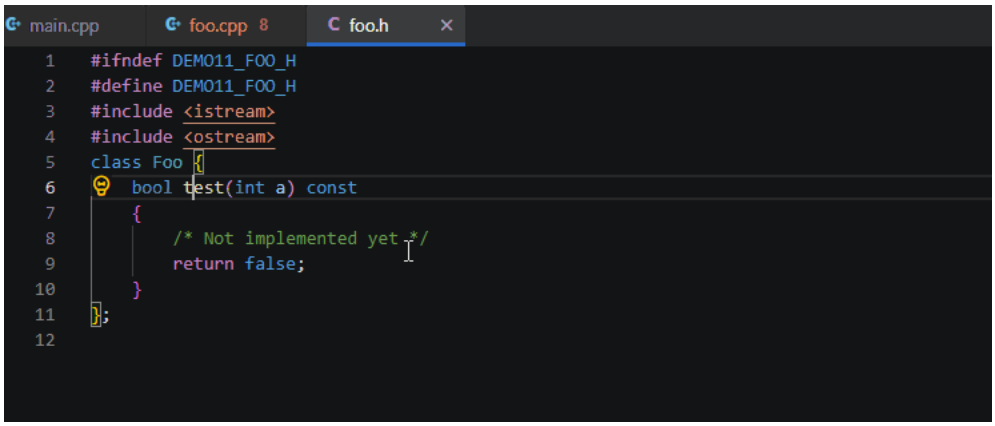
移动函数体到声明处（Move function body to declaration）

将函数/方法定义移动到它声明的位置。

在内部添加定义（Add definition in-place）

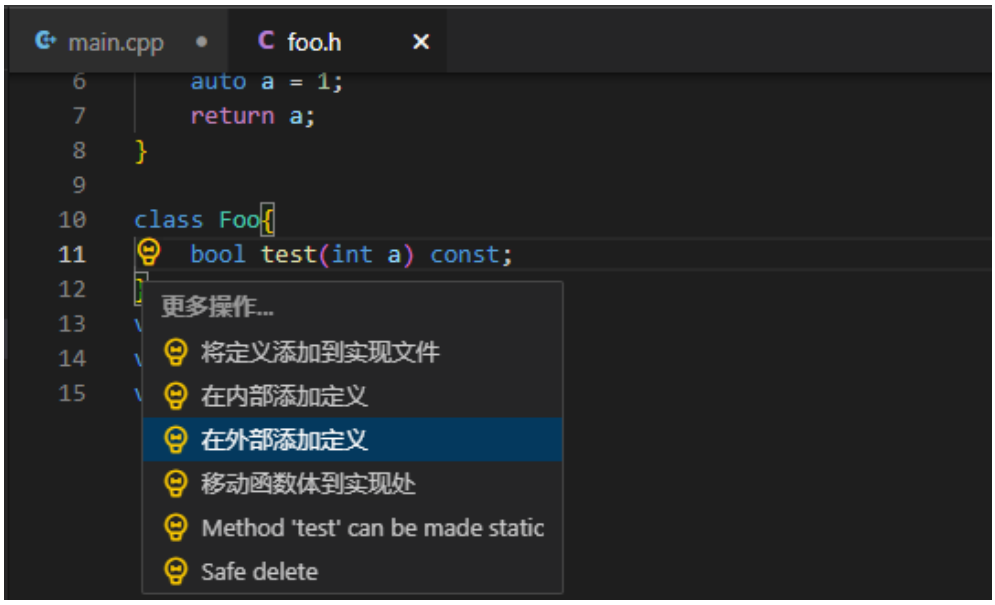
在当前函数/方法并且在类内部生成函数定义。当光标移动到函数/方法时，单击黄色灯泡，重构选项可用。

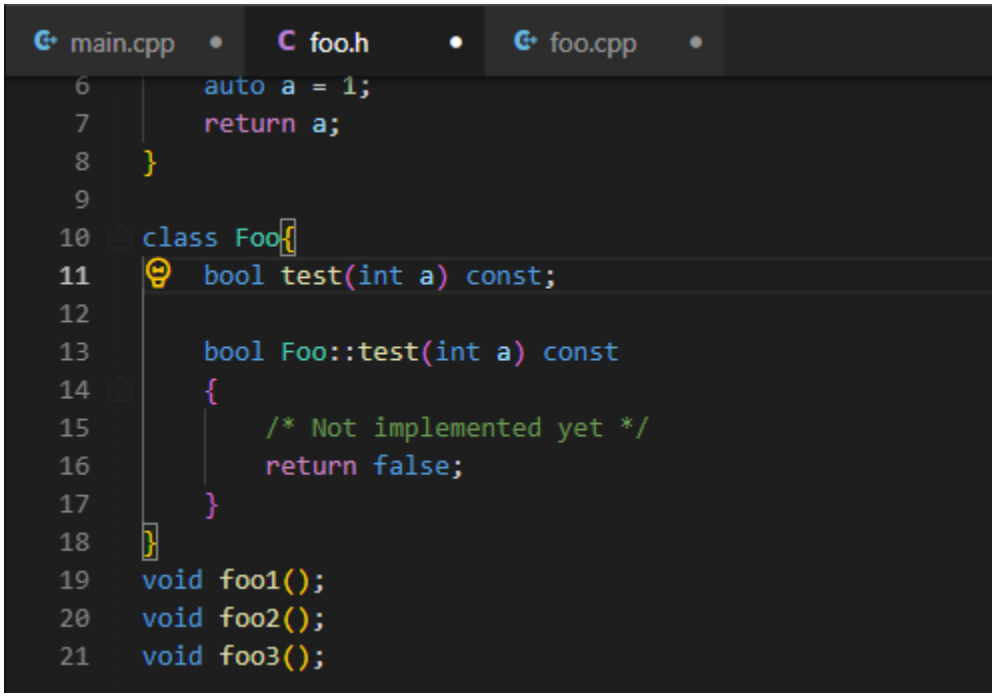




在外部添加定义（Add definition out-of-place）

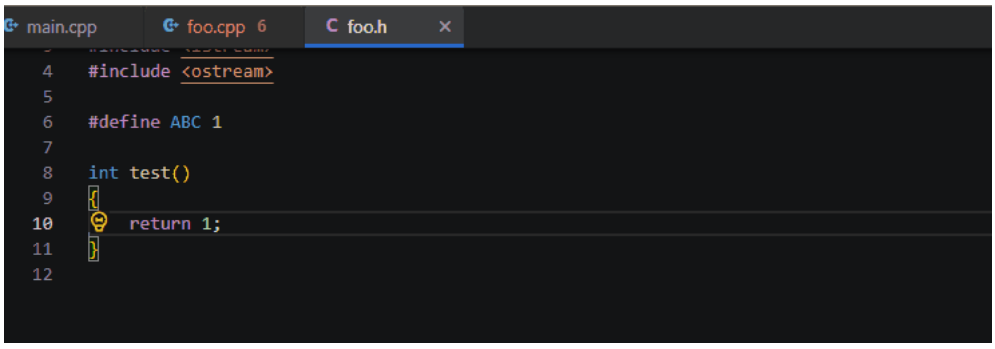
在类外部生成当前函数/方法的函数定义。当光标移动到函数/方法时，单击黄色灯泡，重构选项可用。





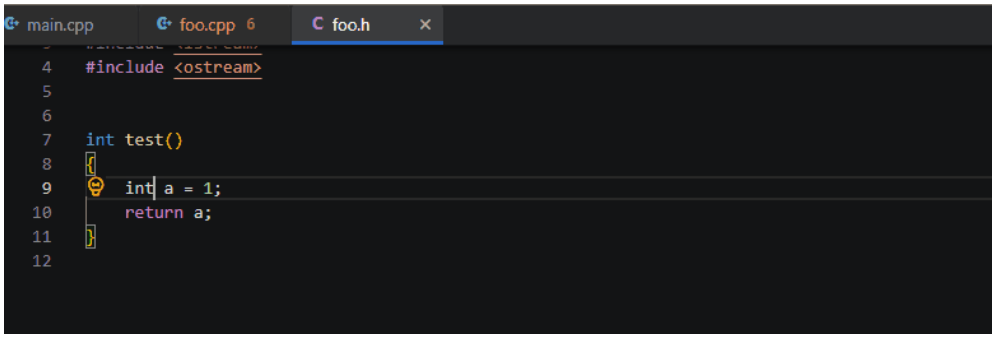
展开宏（Expand macro）

在页面上添加“**展开宏**”（**Expand macro**），以便在可扩展/可折叠的部分提供内容。



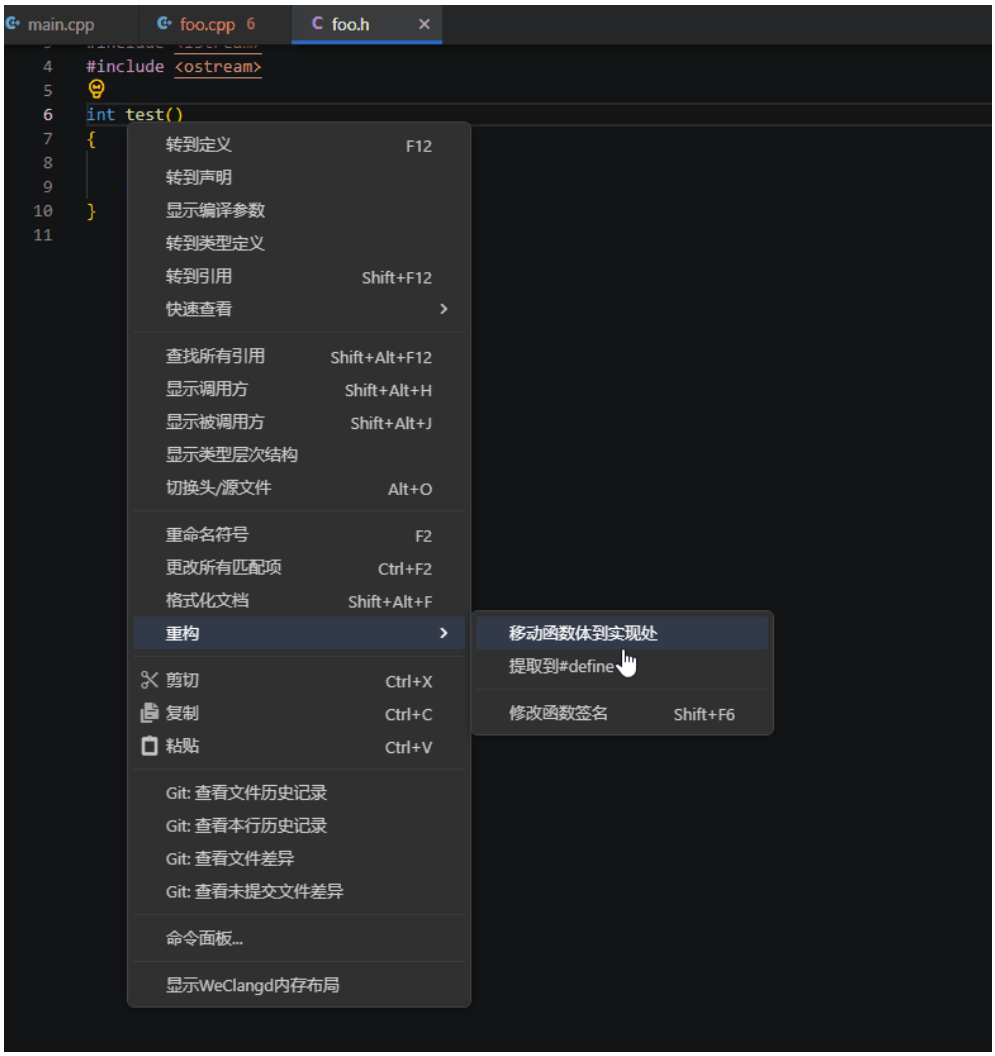
展开 auto（Expand auto type）

展开auto type所隐藏的变量类型。



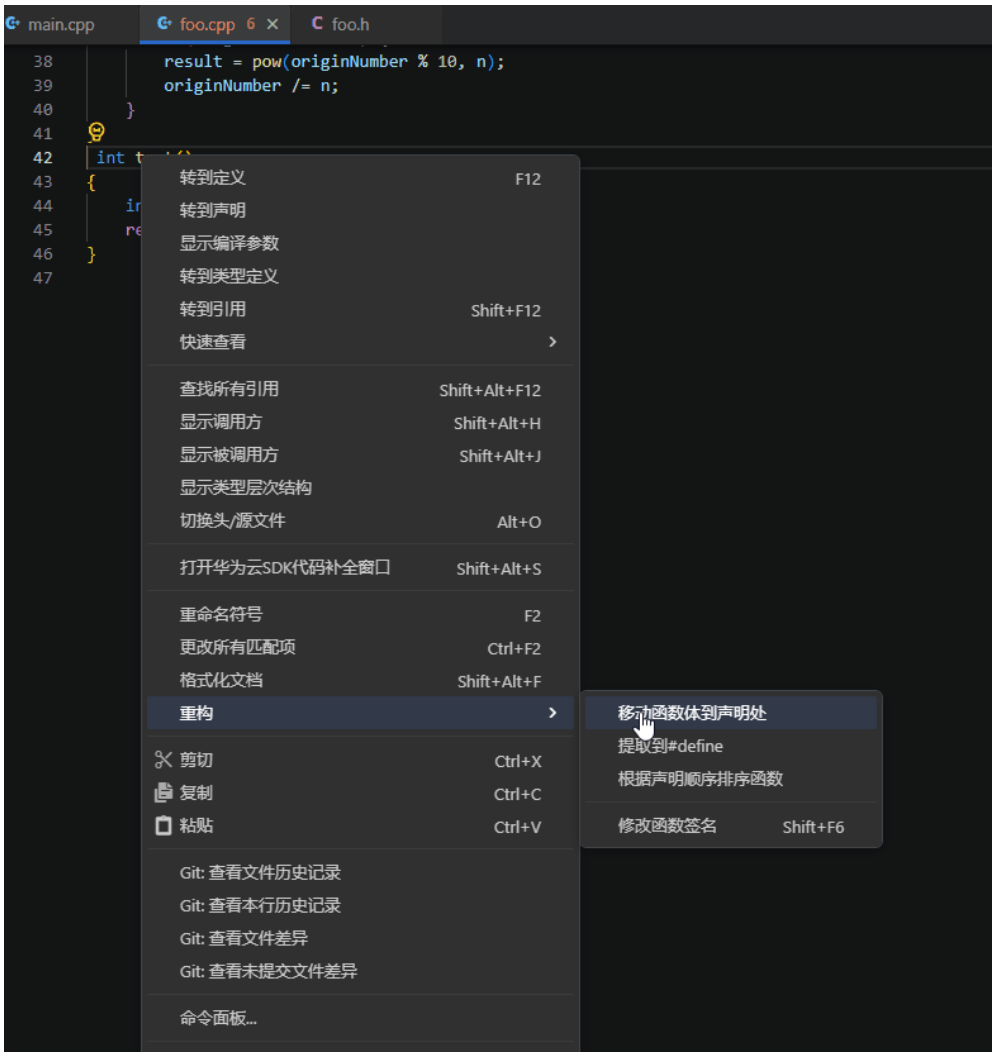
函数定义外移（Move function body to declaration）

该功能会将函数/方法的定义移动到声明的位置。



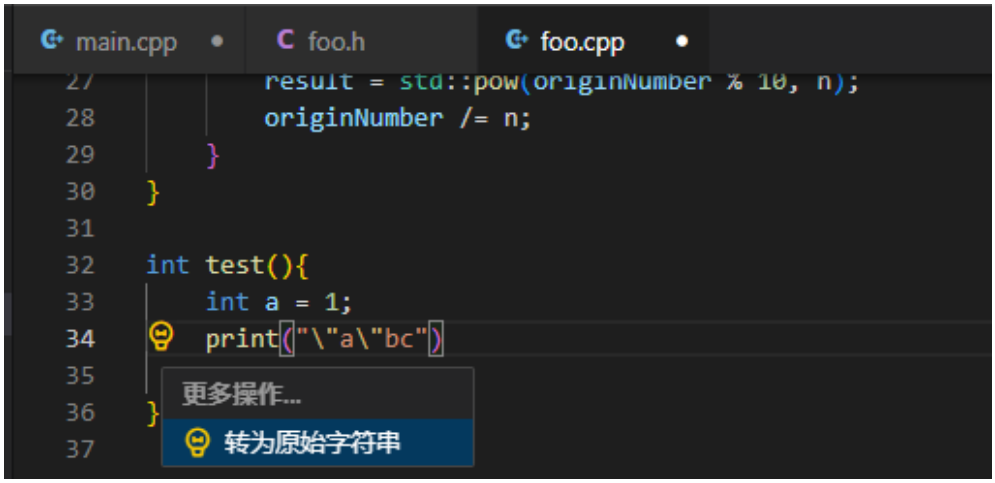
函数定义内移（Move function body to out-of-line）

该功能会将函数/方法的定义移动到对应的文件中。



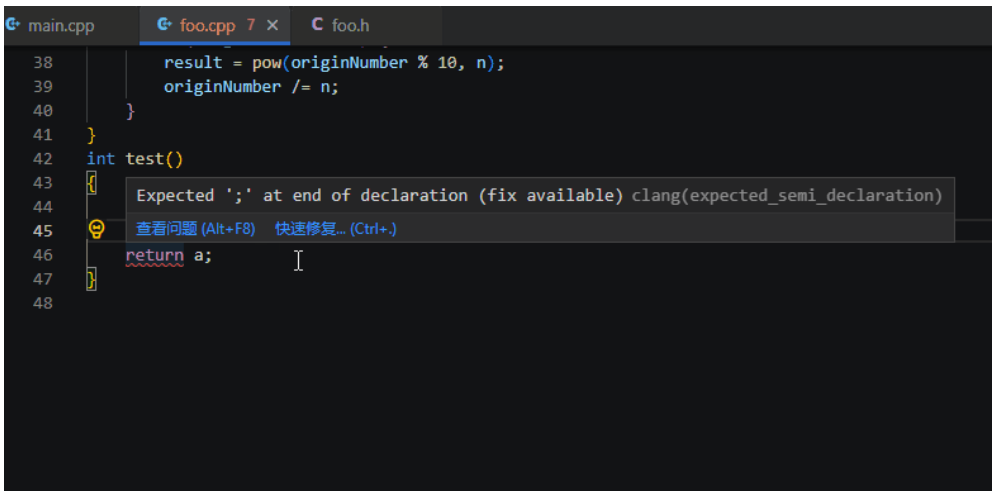
转为原始字符串（Convert to raw string）

此方法可以将转义后的字符串转换为原始的字符串。当单击或选择了当前字符串，单击黄色灯泡，重构选项可用。



快速修复（Quick fixes）

快速修复用于解决简单的编码错误，包括自动补全、实时语法检查等。当光标移动或单击到C/C++的错误代码上时，会显示黄色灯泡，表示可以使用快速修复。单击灯泡或按**Ctrl+.**会显示可用的快速修复选项和重构列表。

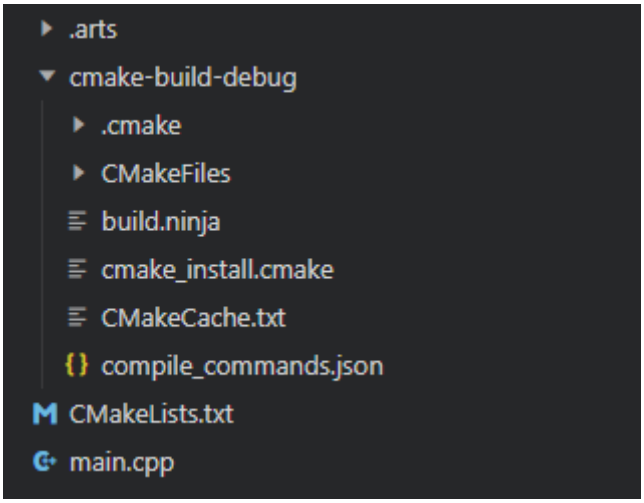


2.4.4 Cmake 工程支持

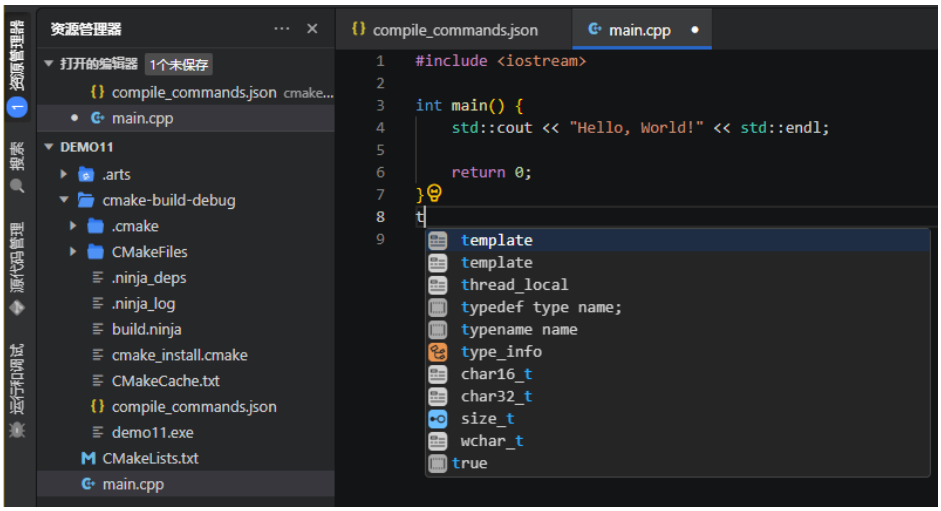
2.4.4.1 CMake 工程加载

CodeArts IDE for Cpp识别到打开的工程为CMake工程时（默认为Debug模式），将自动完成以下操作：

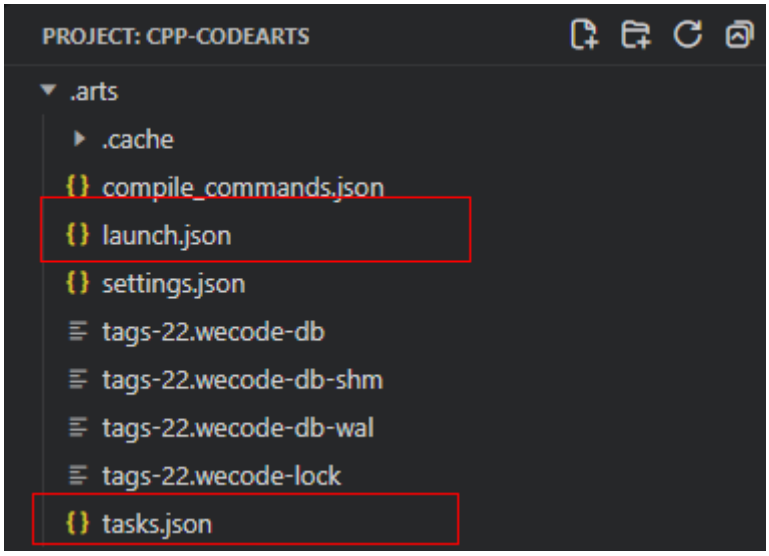
1. 生成cmake-build-debug目录，该目录中有“compile_commands.json” 文件、“.cmake”目录、“CMakeFiles”目录和“CMakeCache.txt”等文件。



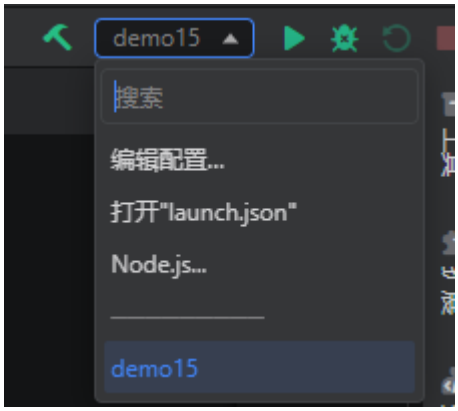
2. 生成“compile_commands.json”文件并导入，CMake工程获得**CodeArts** C/C++插件的代码检测与提示功能。



3. 生成“tasks.json”和“launch.json”文件。



- “tasks.json”是配置构建任务的文件。
- “launch.json”是启动程序的配置文件，该文件中的configurations会在右上角的“运行和调试”的下拉框展示。

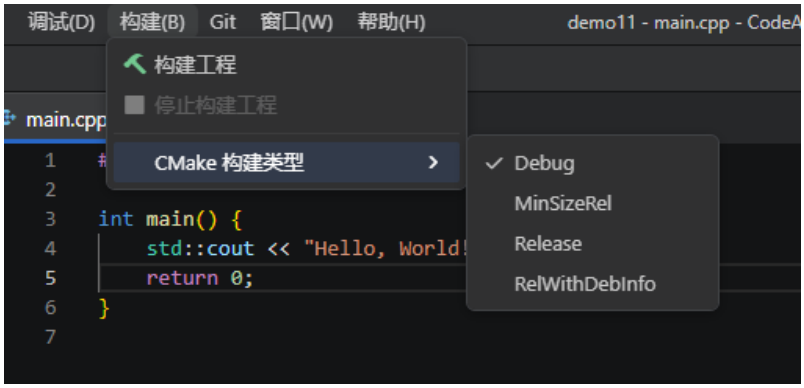


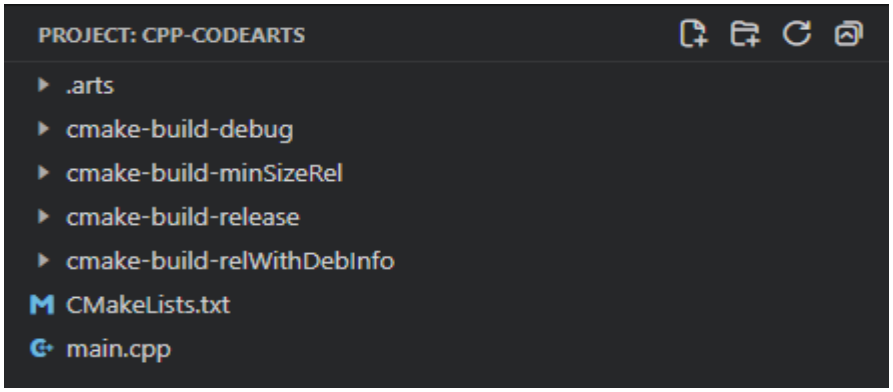
在此过程中，状态栏显示加载过程，单击可以查看具体的加载日志。

2.4.4.2 多种构建类型

CMake工程一共有四种构建类型，分别是Debug、MinSizeRel、Release、RelWithDebInfo。选择需要的构建类型，工程构建之后，生成对应的目录。

- Debug：禁用优化并包含调试信息。
- Release：包括优化但没有调试信息。
- MinSizeRel：优化大小，没有调试信息。
- RelWithDebInfo：优化速度并包含调试信息。

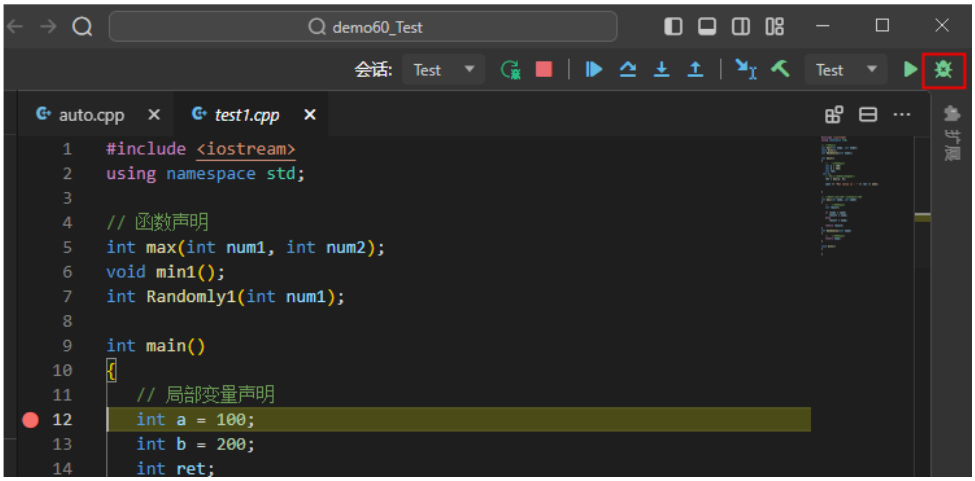




2.4.4.3 CMake 工程调试

打开“main.cpp”文件，选择所需“断点”，然后单击“调试”图标。

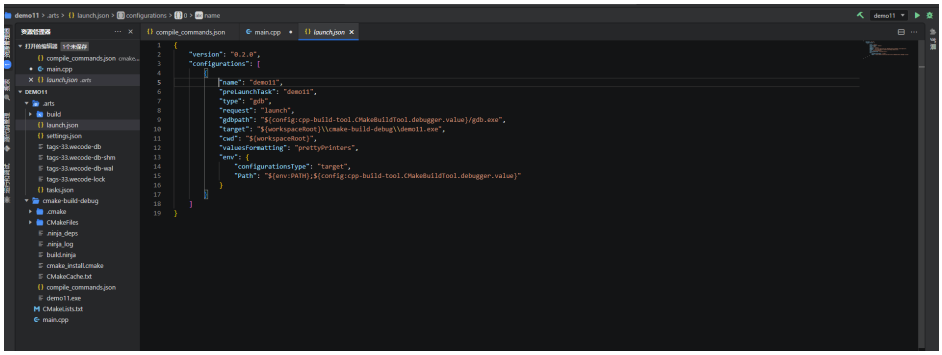
- **设置断点**：在代码中设置断点，以便在特定的代码行暂停执行，检查变量。
- **单步执行**：使用单步执行功能逐步执行代码，观察程序的运行状态。
- **查看变量**：在调试过程中，查看变量的值，确保它们符合预期。



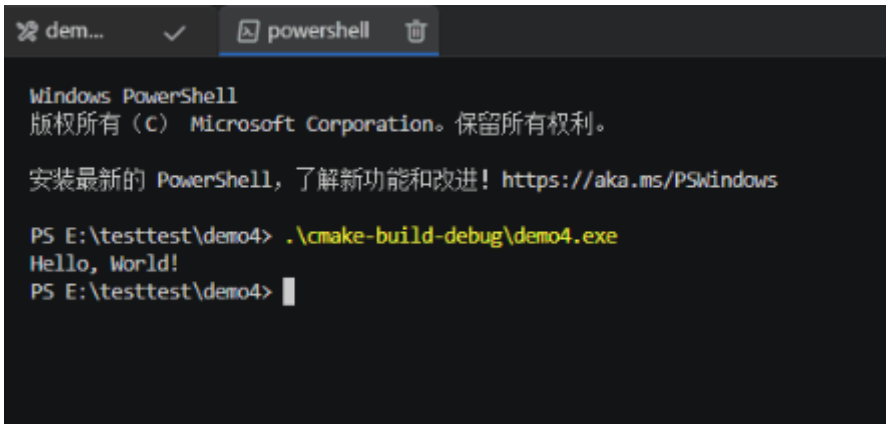
2.4.4.4 CMake 工程运行

可通过以下几种方式运行：

- 单击右上角的“开始执行”按钮，执行“launch.json”文件



- 控制台输入需要运行的文件路径

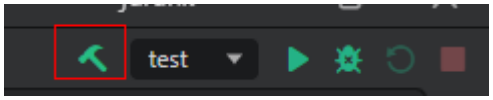


2.4.4.5 CMake 工程构建

可从以下任一渠道构建：

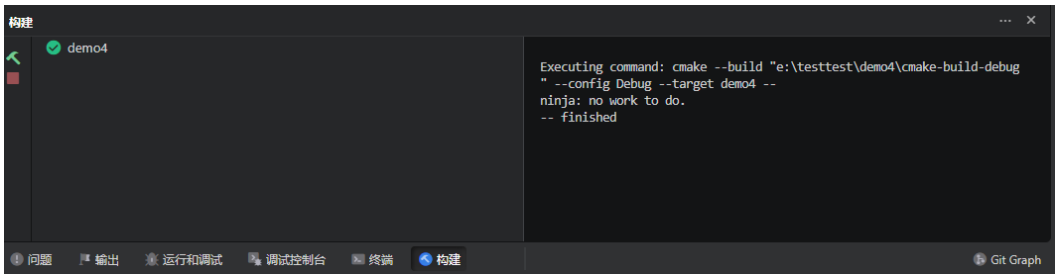
- 打开“命令”面板并运行“**CMake Build Tool: CMake Targets**”命令。
- 单击“构建”菜单。

- 单击构建图标。



- 构建面板。

构建后，构建日志展示在构建面板右侧。

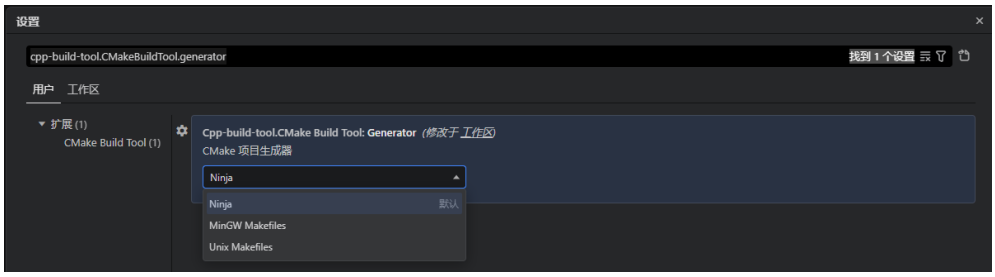


2.4.4.6 多种生成器类型

CodeArts IDE for Cpp为CMake工程提供了三种生成器类型：Ninja、MinGW Makefiles、Unix Makefiles。IDE构建项目时，默认使用Ninja生成器。

区分不同操作系统，Ninja和MinGW Makefiles适用于Windows操作系统，Ninja和Unix Makefiles适用于Linux操作系统。

打开“设置”页面，搜索“cpp-build-tool.CMakeBuildTool.generator”，在“工作区”修改生成器类型。



2.4.5 常用设置项

1. **排除或包含某些文件夹**

   Tag或Hybrid模式下：

   - “排除某些目录”设置项中搜索huawei-cpp.wecodeDb.excludePaths，默认值为：

     ```
     **/.mm/**
     **/.git/**
     **/build/**
     **/output/**
     ```

   - “包含文件夹”设置项中搜索huawei-cpp.wecodeDb.includeFolders，将文件夹绝对路径填入即可。

   Compiler模式下：

   - “排除某些目录”设置项搜索：huawei-cpp.codebase.generator.pathsExclude，使用Glob通配符排除一些路径，然后重新生成“compile_commands.json”才会生效。

2. **开启/关闭问题窗口中的诊断信息**

   设置项中搜索huawei-cpp.clangd.ignoreDiagnostics：

   - none： 显示所有诊断信息。
   - all： 隐藏所有诊断信息。
   - not_indexed：仅当当前文件有编译选项或已经索引时显示诊断信息。

3. **修改系统头文件提供方**

   Huawei C/C++默认从compile_commands.json中的编译器提取系统头文件，如果无法提取则使用自带的RTOS头文件，可通过修改设置项改变默认规则：设置项中搜索huawei-cpp.codebase.systemHeaderProvider：

   - Compiler：仅根据compile_commands.json中提取系统头文件。
   - None：从环境变量中获取系统头文件。

4. **开启内联提示/高亮不活跃代码，开启/关闭/修改语义高亮颜色**

   - 开启或关闭内联提示： huawei-cpp.clangd.enableInlayHints
   - 开启或关闭高亮不活跃代码： huawei-cpp.syntaxColor.enableInactiveCode
   - 开启或关闭语义高亮： huawei-cpp.syntaxColor.enable

5. **cmake工程构建工具的路径**

   CodeArts IDE for Cpp提供了CMake工程构建、调试所需要的相关工具， 用户可以直接构建、调试CMake工程，不必手动配置相关环境变量。用户目录下的“.codearts”文件夹内置了cmake、MinGW、ninja工具，CMake Build Tool插件默认先读取内置工具路径。

   - cpp-build-tool.CMakeBuildTool.CMake获取cmake工具的路径。

- cpp-build-tool.CMakeBuildTool.debugger获取MinGW工具的路径。
- cpp-build-tool.CMakeBuildTool.buildTool获取ninja工具的路径。

2.5 Java

2.5.1 准备工作

JDK 的安装及环境变量配置

CodeArts IDE for Java中Java语言服务的运行依赖JDK，因此在使用CodeArts IDE之前，需要安装JDK并配置环境变量。

CodeArts IDE for Java已内置JDK 11和Maven 3.8.1，详细的JDK安装及环境变量配置步骤，可以参考**JDK的安装及环境变量配置**。

Maven 镜像源检查及配置

CodeArts IDE for Java中Maven项目需要通过Maven中配置的镜像源来下载依赖，因此在使用CodeArts IDE之前，需要确认Maven镜像源是否可以正常访问。

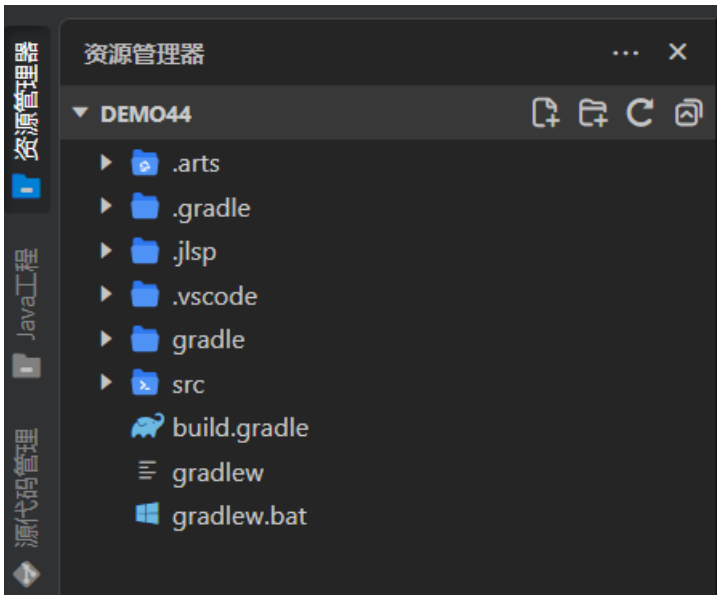
具体的Maven镜像源检查及配置步骤，可以参考**2.1.11 .2 Maven镜像源检查及配置**。

2.5.2 使用 Java 项目

2.5.2.1 简介

当用户打开包含源代码文件的任意文件或文件夹时，CodeArts IDE仅提供基本的文本编辑功能。要获得完整的Java编码帮助，CodeArts IDE会自动通过分析文件内容并生成项目的元数据来初始化项目。所有项目的元数据都存储在项目根目录下的“.jlsp”文件夹中。

在CodeArts IDE中，用户的Java项目的内容会显示在“资源管理器”视图中（**“Ctrl+Shift+E”**或**“Alt+1”**(IDEA快捷键方案)），该视图提供了常见的文件管理功能。



2.5.2.2 管理 Java 项目

2.5.2.2.1 打开文件夹或现有 CodeArts IDE 项目

步骤1 在主菜单中选择“文件 > 打开项目...”。

步骤2 在打开的“打开项目...”对话框中，找到所需的文件夹，然后单击**“选择文件夹”**。

CodeArts IDE会自动检测项目文件夹中的**pom.xml**或**build.gradle**文件，并自动加载相应的项目（Maven或Gradle）。如果两个文件都存在，CodeArts IDE会提示用户指定项目的构建系统。

如果**.jlsp**文件夹尚不存在，CodeArts IDE会创建该文件夹，并扫描项目的内容。一旦此过程完成并且Java编码辅助功能可用，CodeArts IDE会显示相应的通知。

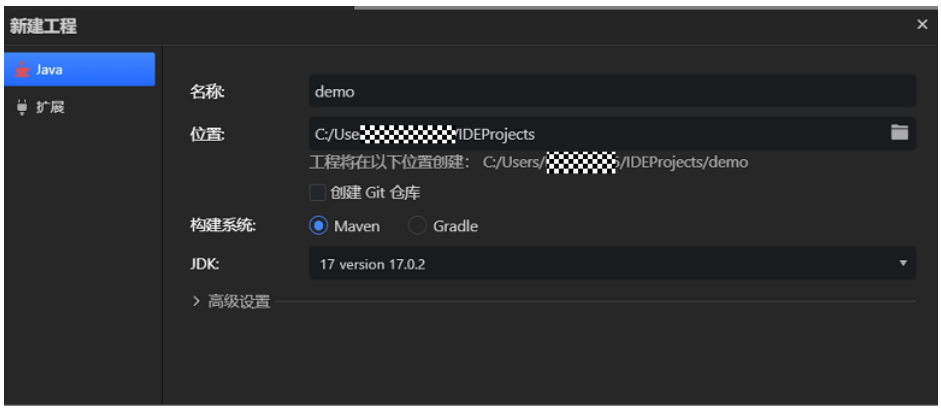
----结束

2.5.2.2.2 创建并加载项目

创建项目

步骤1 在主菜单中，选择“文件 > 新建 > 工程...”。

步骤2 在打开的“新建工程”向导中，提供常见的项目参数：项目名称、位置、使用的构建系统和项目JDK。CodeArts IDE会自动检测用户系统上安装的JDK，并在**“JDK”**列表中显示它们。要在创建的项目内自动**初始化Git存储库**，请选中**“创建Git仓库”**复选框。



步骤3 要基于Spring Boot框架创建项目，请按照以下步骤操作：

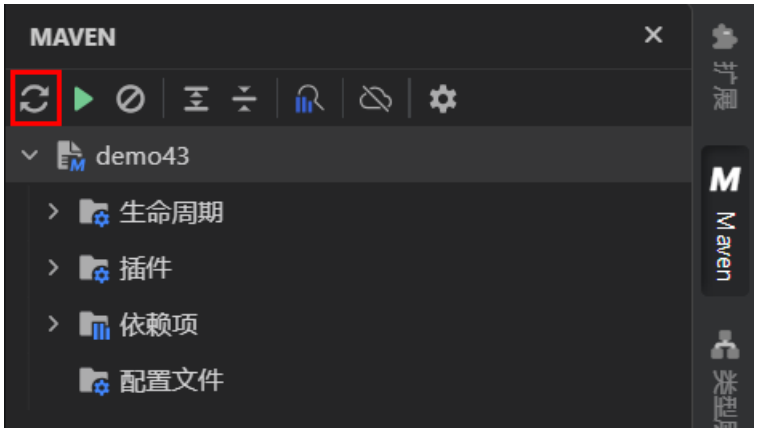
1. 展开“**高级设置**”部分，并将“**框架**”设置为“**SpringBoot**”。
2. 在“**版本**”列表中，选择Spring Boot的版本。
3. 在“**Java**”列表中，选择项目的Java语言级别。语言级别定义了代码编辑器提供的一组编码辅助功能（如代码补全或错误突出显示）。
4. 在依赖项“**Dependencies**”列表中搜索并选择所需的项目依赖项。所选的项目依赖项将在**build.gradle**（对于Gradle）或**pom.xml**（对于Maven）中注册。

步骤4 单击“**创建**”。CodeArts IDE会根据所选的模板创建项目结构。

----结束

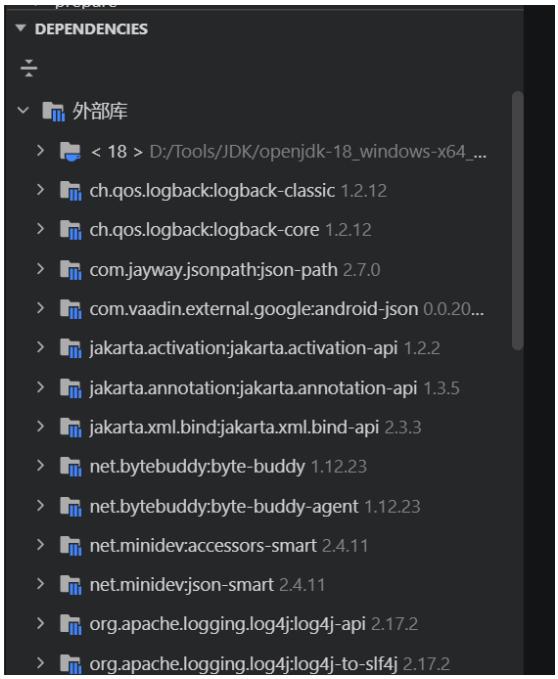
重新加载项目

如果用户修改了项目的构建脚本（Gradle的**build.gradle**或Maven的**pom.xml**），例如添加了一个依赖项，用户需要重新加载项目使CodeArts IDE解析项目的结构。下图以Maven为例，要执行此操作，请在工具栏上单击“**重新加载所有Maven工程**”按钮()。



2.5.2.2.3 查看项目依赖关系

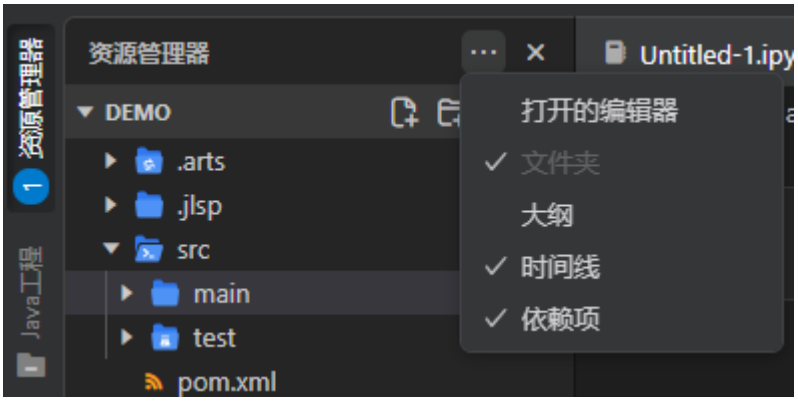
如果用户的项目在**build.gradle**（对于Gradle项目）或**pom.xml**（对于Maven项目）中声明了依赖项，CodeArts IDE会在资源管理器中的“**依赖项**”部分显示它们。配置的JDK的内容也会显示在“**依赖项**”部分中。



用户可以浏览依赖项列表，并以只读模式在代码编辑器中打开文件。

说明

如果“**依赖项**”被隐藏了，用户可以在资源管理器中单击“**视图**”和“**更多操作**”按钮(...)并在弹出菜单中启用“**依赖项**”来显示它。



2.5.2.2.4 创建文件和文件夹

CodeArts IDE支持在创建Java源文件时应用模板。当用户在资源管理器中创建一个**java**文件时，语言服务器会自动生成类的主体，并为用户填充包的信息。

创建文件夹

步骤1 在资源管理器中，在指定目录下创建新文件夹时，右键单击该目录，然后从上下文菜单中选择“新建 > 文件夹”。

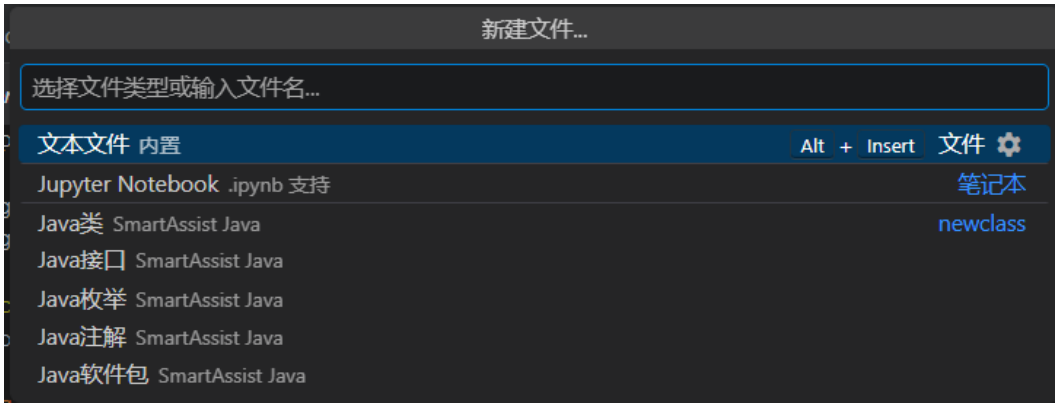
步骤2 键入文件夹的名称，然后按“Enter”键。

----结束

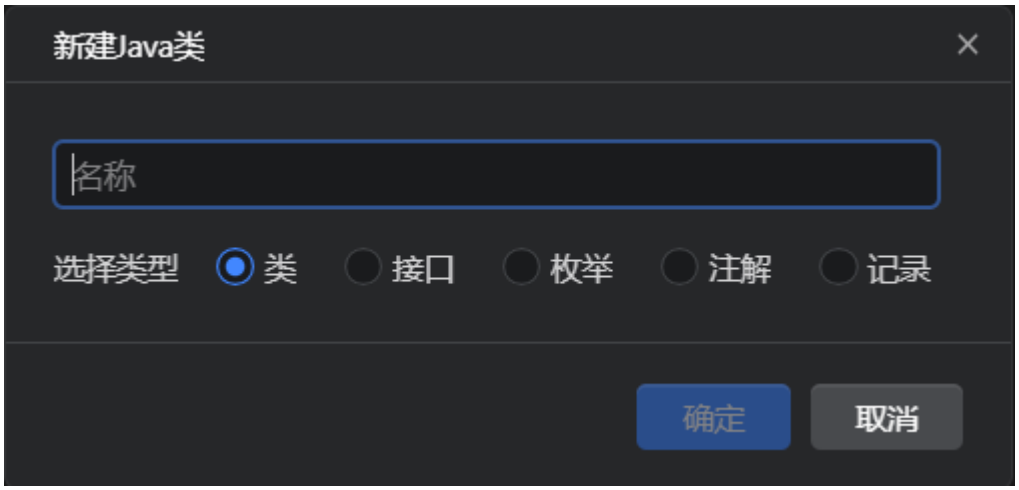
创建文件

步骤1 在资源管理器中，右键单击要在其中创建新文件的文件夹，然后从上下文菜单中选择“新建 > 文件”或按下“Ctrl+Alt+Win+N”。或者在主菜单中选择“文件 > 新建 > 文件...”。

步骤2 在打开的“新建文件...”弹出窗口中，根据需要选择相应的选项。



步骤3 假如选择Java Class，在打开的“新建Java类”对话框中，提供文件名。



步骤4 单击**“确定”**。CodeArts IDE会在指定的目录中创建文件，并根据所选的文件模板填充它的内容。

----结束

注意

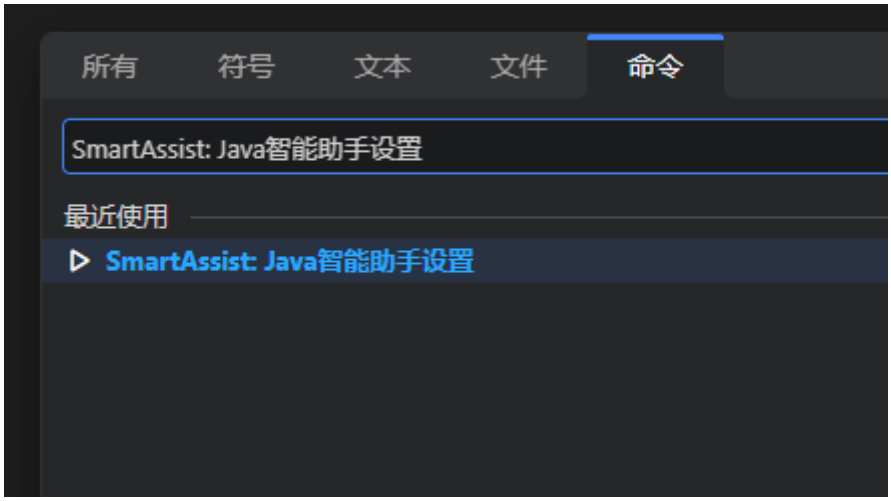
新建Java类时，类名以及文件夹路径不可包含-+.<>等特殊符号。

2.5.2.3 配置项目

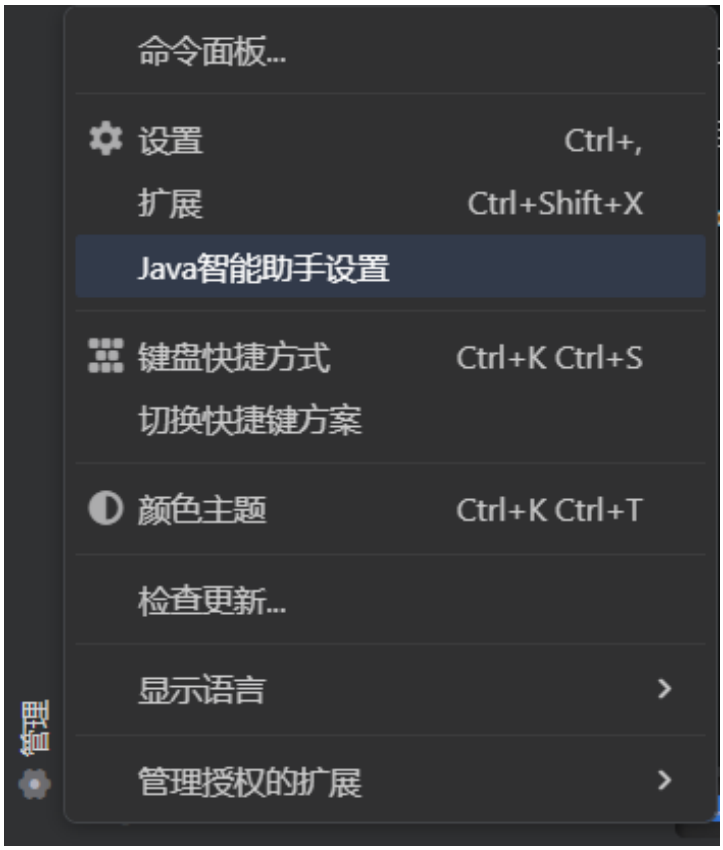
2.5.2.3.1 简介

要配置Java项目，先打开“Java智能助手设置”对话框，用户可以通过以下任意一种方式打开“Java智能助手设置”对话框：

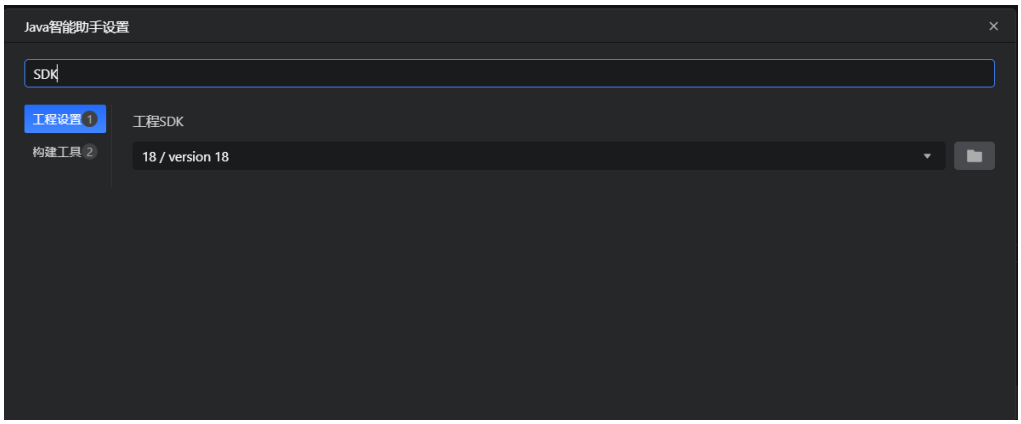
- 通过“Ctrl+Shift+P”或者“Ctrl+Ctrl”打开“命令”面板，输入**SmartAssist: Java智能助手设置**命令。



- 单击CodeArts IDE左下角**“管理”**选项卡，选择**“Java智能助手设置”**，打开“Java智能助手设置”对话框。



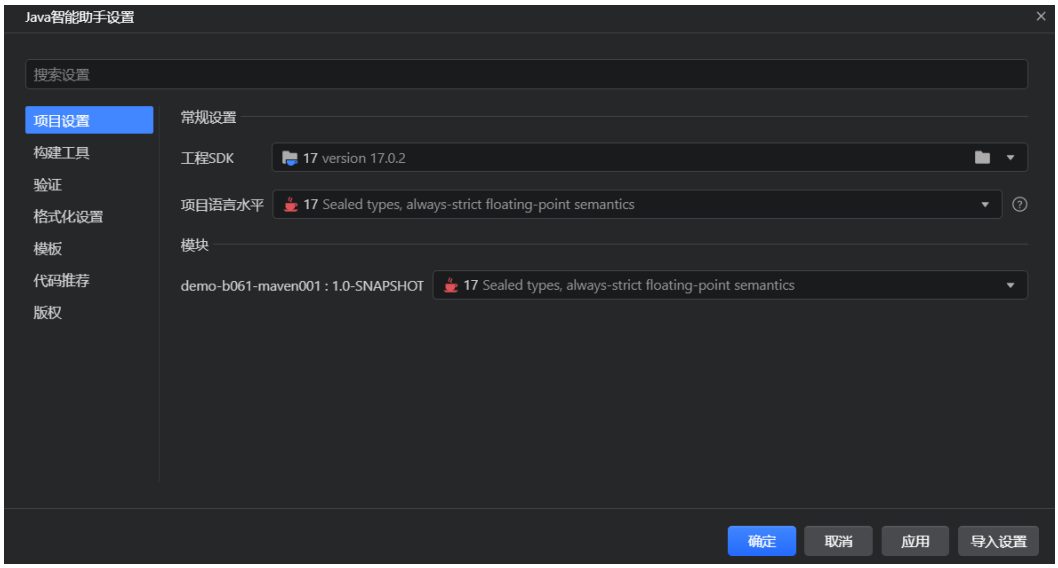
打开“Java智能助手设置”对话框之后，在对话框中，使用搜索字段快速定位所需的设置。



2.5.2.3.2 项目与模板设置

项目设置

在“**项目设置**”页面，用户可以提供Java SDK（JDK）的路径并配置项目的语言级别，该级别定义了代码编辑器提供的一组编码辅助功能（例如代码完成）。

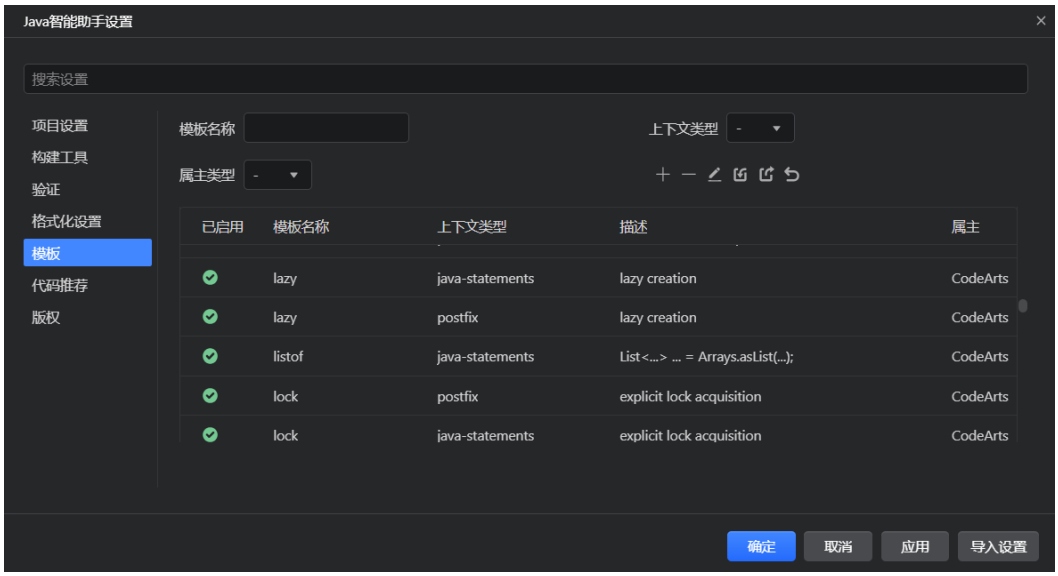


选定的语言级别将与项目生成配置文件同步。如果在设置中调整它，CodeArts IDE会相应地更新相应的值：

- 对于Gradle在**build.gradle**文件中，“**sourceCompatibility**”和“**targetCompatibility**”的值已经更新。
- 对于Maven在**pom.xml**文件中，Maven的“**source**”和“**target**”已经更新。

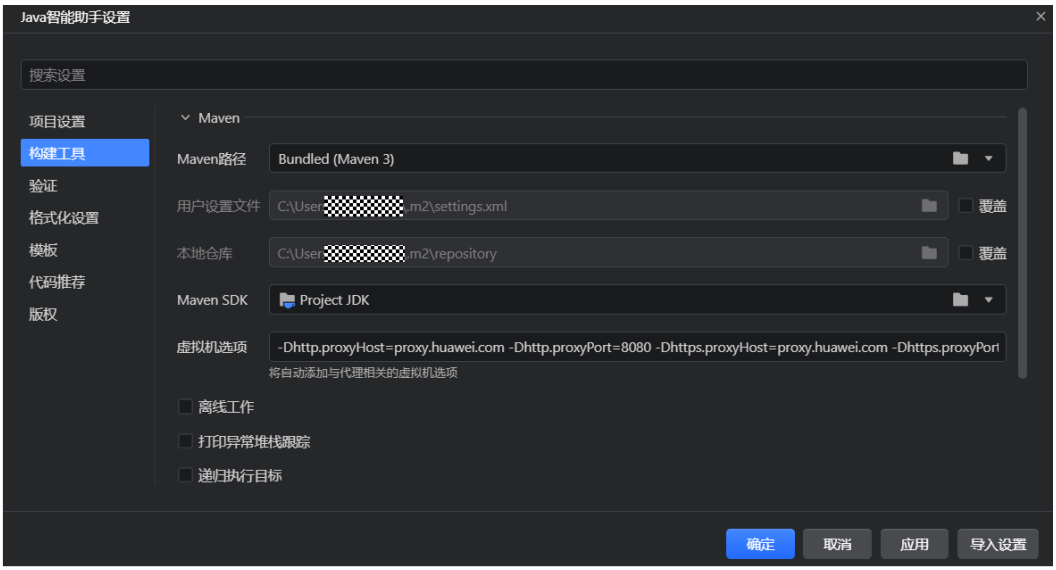
模板设置

在“**模板**”页面列出了用户项目中定义的模板。



2.5.2.3.3 构建工具设置

在“**构建工具**”页面上，用户可以配置项目使用的构建工具（Maven或Gradle）。



Maven 设置

- **Maven路径**：请使用此字段选择捆绑的Maven版本（Maven 3）或者单击浏览按钮，手动定位用户自己的Maven安装位置。
- **用户设置文件**：在此字段中，指定包含Maven用户特定配置的文件。
- **本地仓库**：在此字段中，指定用户主目录下的本地目录的路径，该目录存储下载并包含临时生成工件。
- **Maven SDK**：从此列表中，选择要与Maven一起使用的JDK：捆绑的JDK、项目级JDK或从系统变量（如**JAVA_HOME**）解析的JDK。
- **离线工作**：如果选中，Maven将在脱机模式下工作。它不连接到远程存储库，只使用本地可用的资源。此选项对应于**--offline**命令行选项。

- **打印异常堆栈跟踪**：如果选中，则生成异常堆栈跟踪。此选项对应于**--errors**命令行选项。
- **递归执行目标**：如果选中，则将递归执行生成，包括嵌套项目。此选项对应于**--non-recursive**命令行选项。

Gradle 设置

- **Gradle用户目录**：在此字段中，指定Gradle用户主目录的路径（默认主目录路径为“$USER_HOME/.gradle”），用于存储全局配置属性和初始化脚本、缓存和日志文件。默认值基于**GRADLE_USER_HOME**环境变量的值提供。要修改它，用户可以设置环境变量或单击“”按钮并手动定位所需的Gradle用户主目录。
- **Gradle SDK**：从此列表中选择要与Gradle一起使用的JDK：捆绑的JDK、项目级别的JDK或从系统变量（如**JAVA_HOME**）解析的JDK。

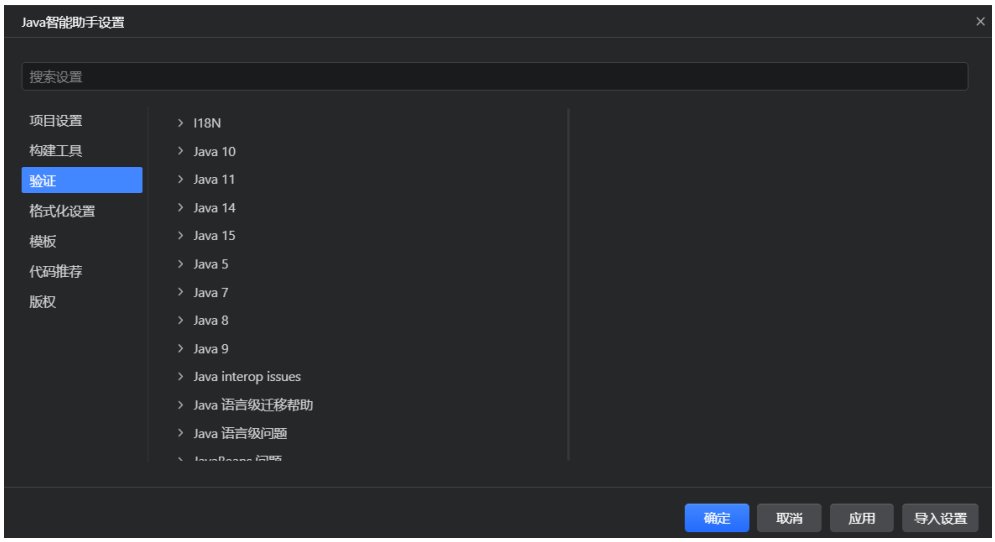
说明

CodeArts IDE暂不支持指定本地已安装的Gradle版本来解析工程。

2.5.2.3.4 代码校验规则

代码验证规则可以在编译之前检测和纠正项目中的错误代码。当用户输入代码时，检测到的问题会在代码编辑器中实时显示出来。有关CodeArts IDE中代码验证的更多详细信息，请参阅**代码校验**。

- 要启用或禁用验证规则，请使用其名称旁边的复选框。
- 要调整相应代码问题在代码编辑器中突出显示的方式，请在**“严重程度”**列表中选择所需的严重性级别。



2.5.3 代码编辑

2.5.3.1 简介

CodeArts IDE是一个具有丰富编辑功能的源代码编辑器。

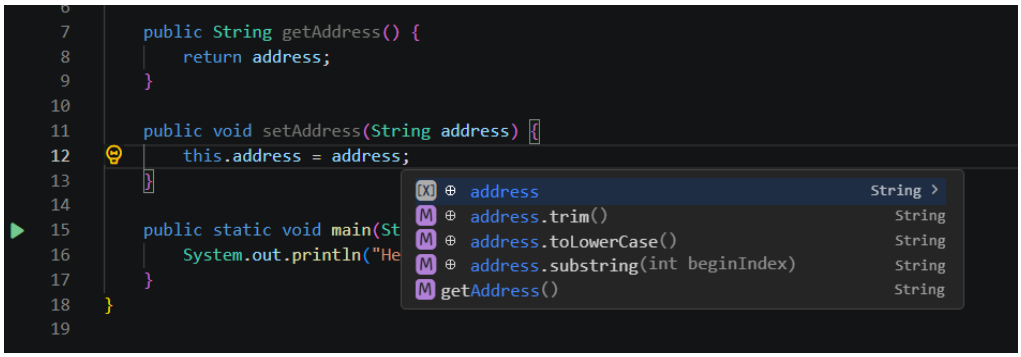
用户可以使用代码片段、各种代码操作（如生成Getter/Setter和组织导入）以及重构来优化用户的代码库。

在Java重构和Java源代码操作中了解更多信息。

2.5.3.2 代码补全

2.5.3.2.1 简介

CodeArts IDE for Java中的代码补全由SmartAssist提供支持。它的代码补全推荐基于数千个开源项目，旨在准确匹配当前代码上下文。在补全推荐列表中，SmartAssist提供的补全推荐列表会用⊕图标表示。

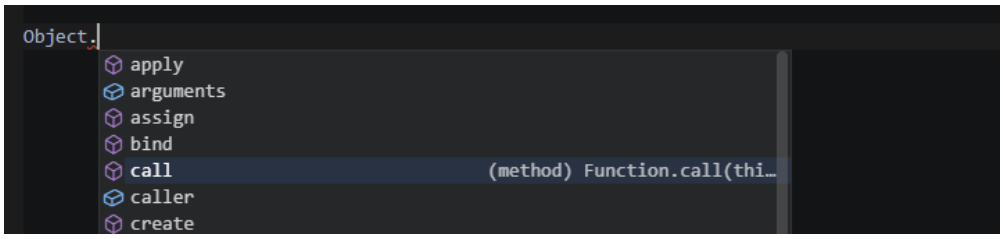


说明

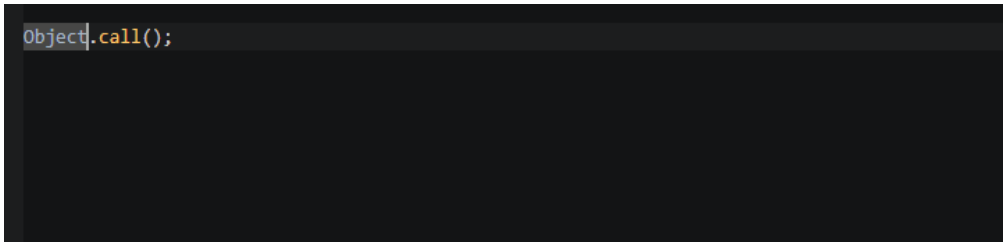
有关CodeArts IDE代码补全功能的一般信息，请参见代码补全。

2.5.3.2.2 触发代码补全

- 要手动触发代码补全，请按“**Ctrl+Shift+Space**”（IDEA快捷键方案）或键入触发字符（例如点字符“.”）。
- 要插入所选符号，请按“**Enter**”键。



- 要插入选定的符号并替换当前位于光标位置的符号，请按“Tab”键。

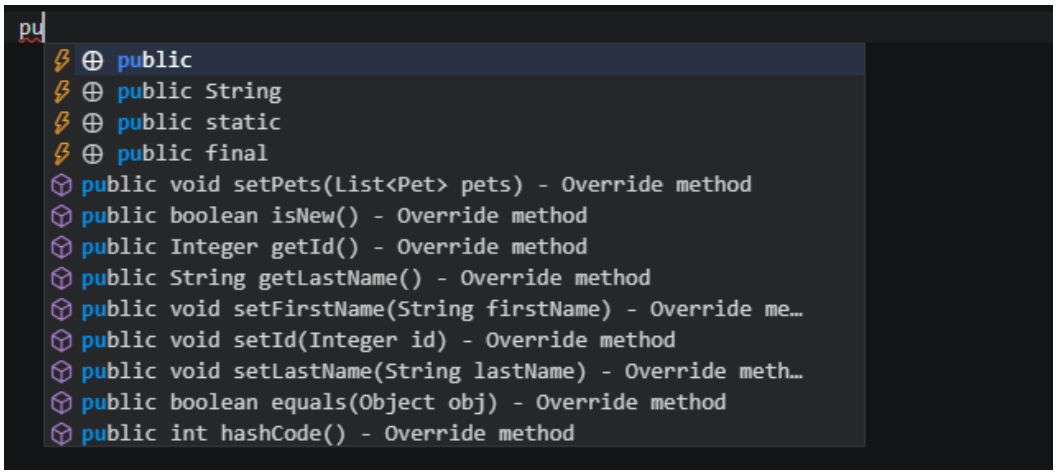


- 要关闭代码推荐列表而不插入推荐代码，请按“Escape”键。

2.5.3.2.3 关键字补全

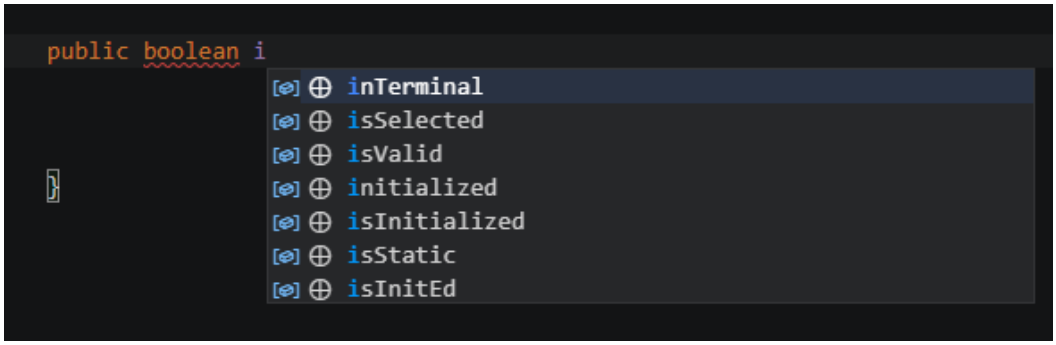
CodeArts IDE为Java保留关键字（如**public**、**void**、**if**、**while**、**boolean**等）提供代码补全推荐。

1. **打开项目**：在CodeArts IDE中打开你需要编写Java代码的项目。
2. **编写代码**：在Java代码中编写代码时，CodeArts IDE会自动提示可能的补全选项。
3. **触发补全**：使用快捷键触发补全建议。
4. **选择补全选项**：从显示的补全建议中选择合适的选项，IDE会自动补全代码。

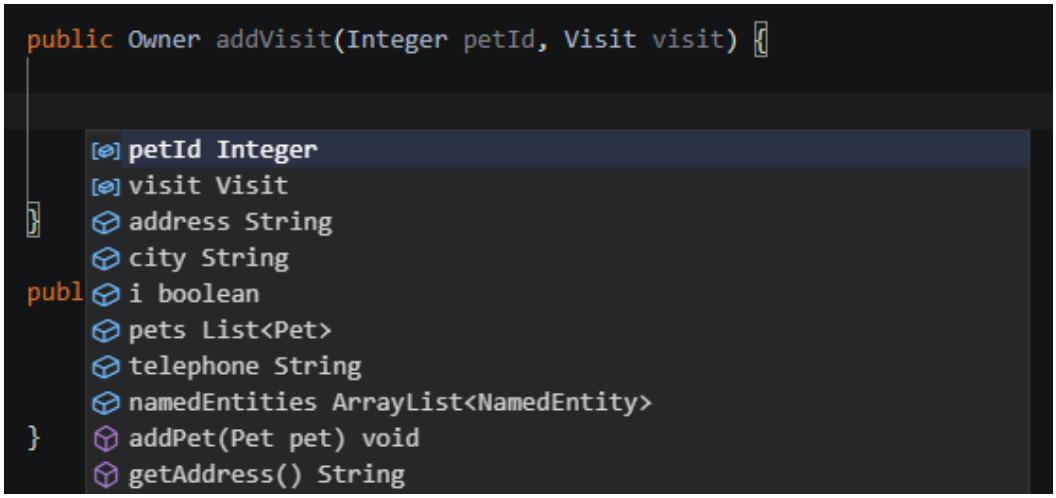


2.5.3.2.4 名字补全

基于当前上下文，CodeArts IDE为变量、参数、方法、类等提供了代码补全推荐。



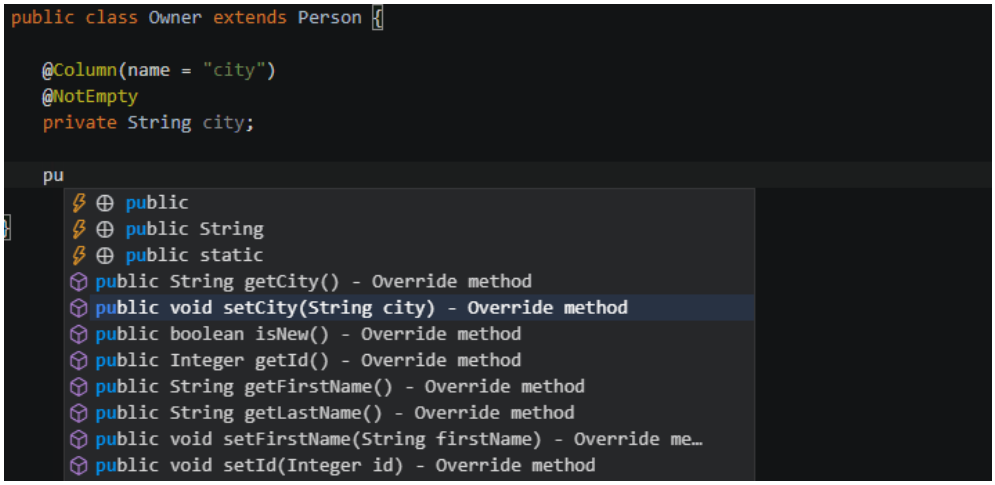
定义方法参数时，它们在代码补全推荐列表中被优先排序。



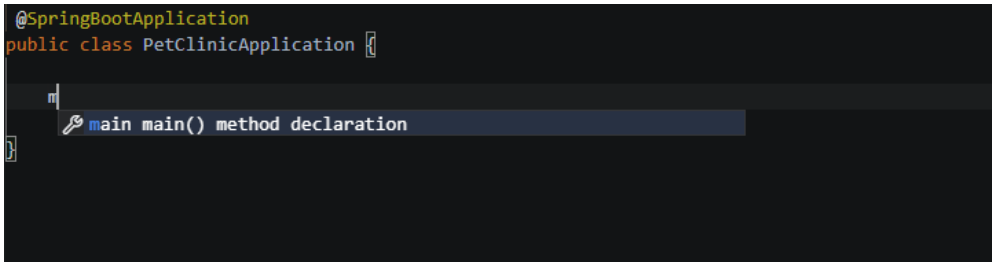
2.5.3.2.5 方法补全

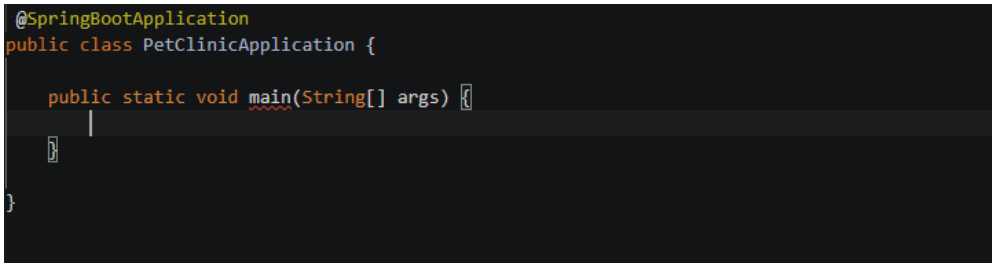
CodeArts IDE为所需方法的元素提供代码补全：方法名称、返回值类型、参数和方法体。

- 在类内部，使用代码补全会根据类变量提供与变量相关方法（即getters/setters）的声明和主体。



- 在主项目类中，键入**m**并使用代码补全快速提供**main**的声明。

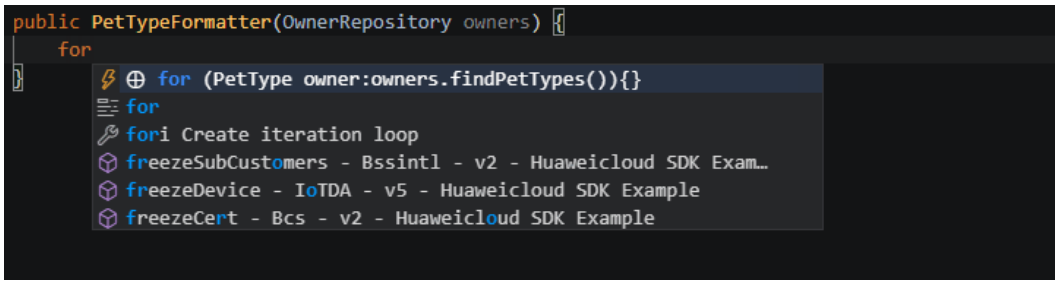




2.5.3.2.6 片段补全

代码补全功能不仅仅提供简单的单词补全，还会根据代码的上下文提供更复杂的结构建议，包括条件语句和循环、代码块折叠等。CodeArts IDE提供了多个Java代码片段补全，帮助用户提高工作效率，如**class/interface**，**syserr**，**sysout**，**if/else**，**try/catch**，静态**main**方法等。此功能使用来自Java语言服务器的信息，用户可以在选择代码期间预览代码段。

- 在代码中编写条件语句（如if/else）时，CodeArts IDE会根据上下文提供可能的条件表达式。
- 在编写循环结构（如for、while）时，CodeArts IDE会根据上下文提供可能的循环条件和迭代变量。
- CodeArtsIDE会根据代码的结构自动折叠代码块，例如函数、类、循环等，使代码更易于阅读和维护。
- CodeArtsIDE会根据编写历史和常用代码片段，提供快速插入代码片段的功能。



2.5.3.2.7 智能类型匹配补全

CodeArts IDE会自动过滤推荐列表，直到仅包括与当前上下文匹配的类型。

这在以下情况中有效：

- 在赋值语句的右侧。
- 在变量初始化中。
- 在**return**语句中。
- 在方法调用的参数列表中。
- 在对象实例化中**new**关键字后。
- 在链式表达式中。

2.5.3.3 折叠区域

折叠区域允许用户折叠或展开代码片段，以更好地查看源代码。在Java上下文中，使用以下折叠区域：

- 开始区域：**//#region**或**//<editor-fold>**。
- 结束区域：**//#endregion**或**//</editor-fold>**。

然后使用“**Ctrl+Shift+[**”或“**Ctrl+-**”或“**Ctrl+Numpad-**”（IDEA快捷键方案）来折叠光标处最内层未折叠的区域，使用“**Ctrl+Shift+]**”或“**Ctrl+=**”或“**Ctrl+Numpad+**”（IDEA快捷键方案）来展开光标处折叠的区域。有关代码折叠的更多详细信息，请参阅**代码折叠**。

2.5.3.4 智能选择

使用智能选择（语义选择），用户可以根据代码中符号插入位置的语义信息扩大或缩小选择范围。

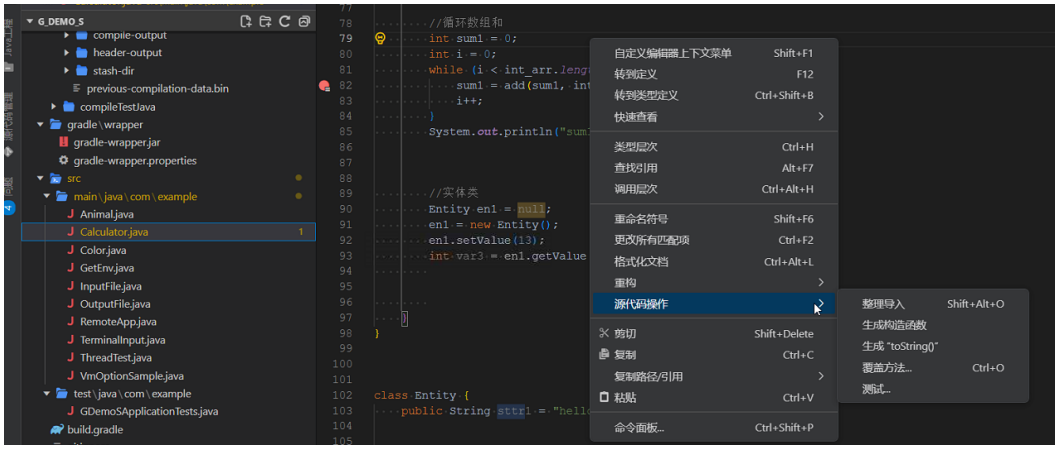
- 要扩大选区，请使用“**Shift+Alt+right**”。
- 要缩小选区，请使用“**Shift+Alt+Left**”或“**Ctrl+Shift+W**”(IDEA快捷键方案)。

2.5.4 代码生成

2.5.4.1 简介

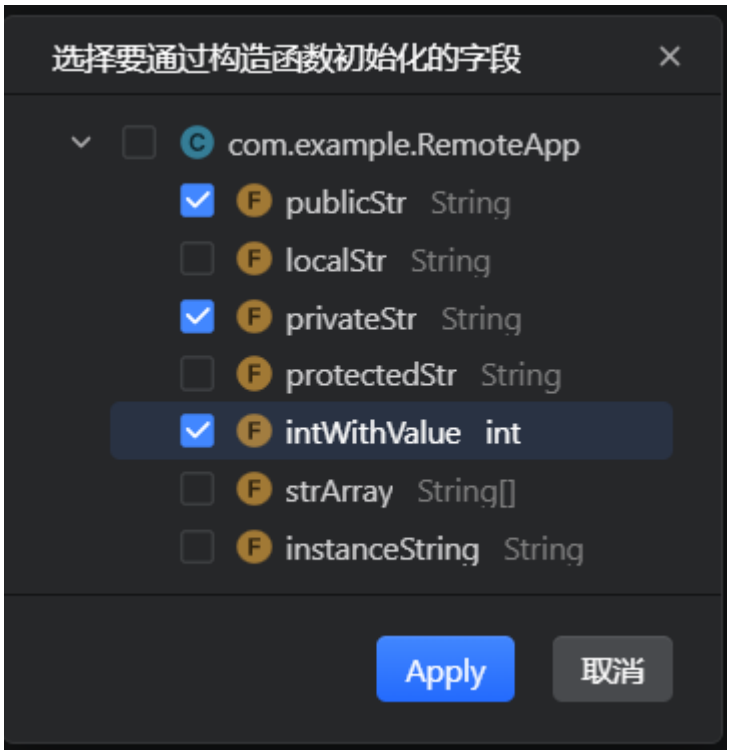
CodeArts IDE提供了多个“源代码操作”来生成公共代码结构和循环元素。要访问它们，请在代码编辑器中单击右键，然后从上下文菜单中选择“源代码操作”。

1. 在CodeArts IDE中打开你需要编写代码的项目文件
2. 在代码编辑器中，将光标放在你希望生成代码的位置，然后右键单击。
3. 在弹出的上下文菜单中，选择“源代码操作”下面的选项。

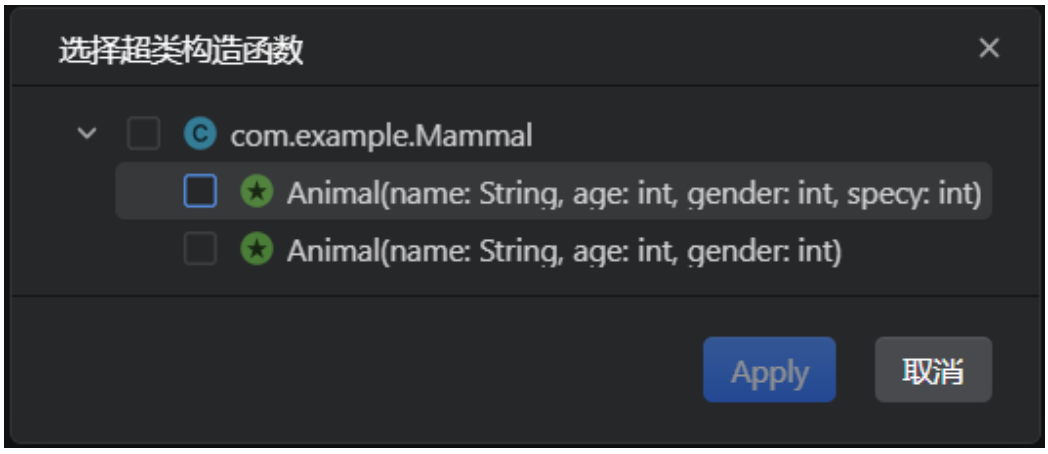


2.5.4.2 构造函数生成

使用此“源代码操作”添加初始化选定类字段的类构造函数。



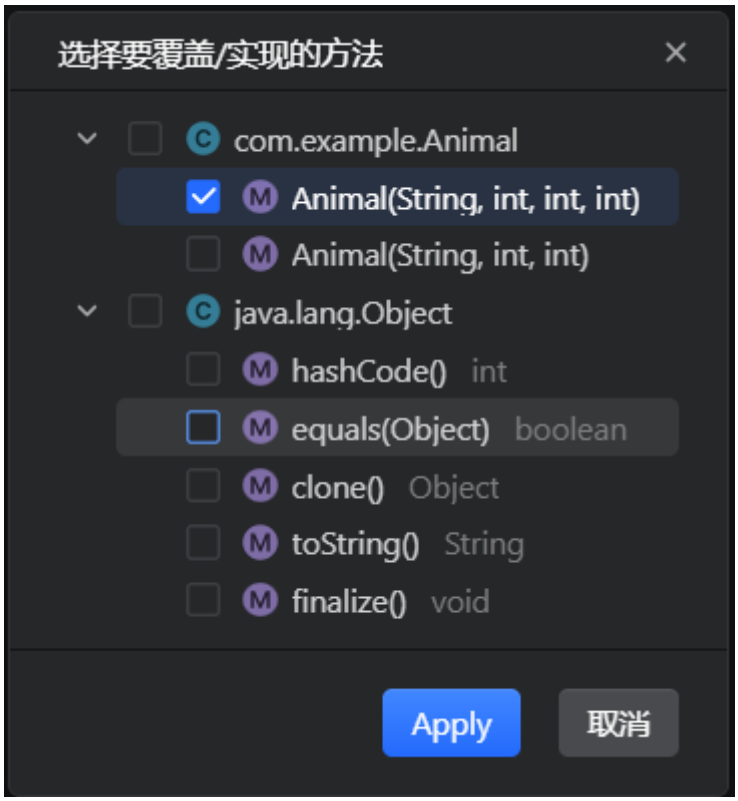
CodeArts IDE还将提示用户选择超类的构造函数（如果有），以便在生成的构造函数中调用它。超类构造函数的参数与当前类的选定字段组合。



2.5.4.3 生成方法类型

Override/implement 方法

使用这些源操作可以覆盖父类的选定方法（“Ctrl+O”），实现接口或抽象类的方法（“Ctrl+I”）。因此，CodeArts IDE为Override/implement方法生成存根。

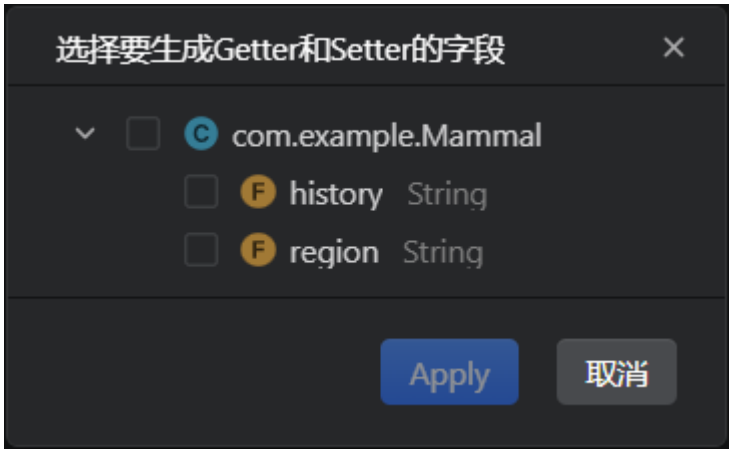


组织 imports

使用此“源代码操作”可清理未使用的导入并自动排列**import**语句。快捷键操作为：(“**Shift+Alt+O**”或“**Ctrl+Alt+O**”(IDEA快捷键方案))。

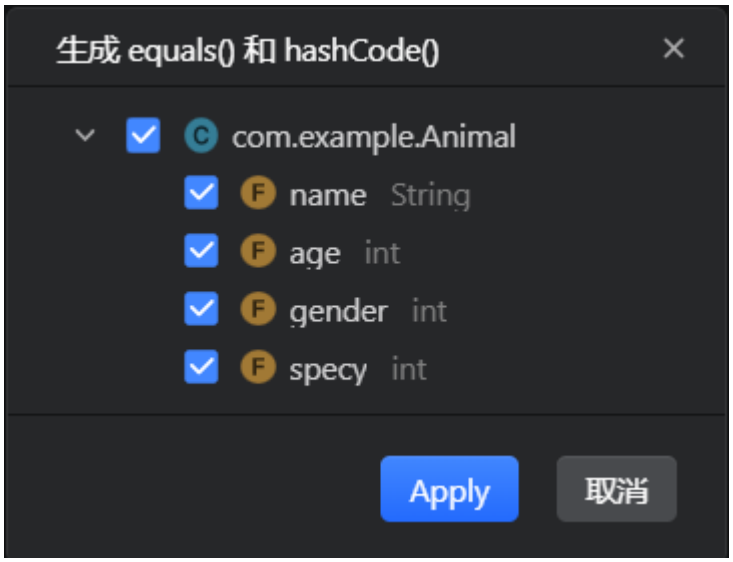
生成 getters 和 setters

使用此“源代码操作”为类的选定字段生成getter和setter方法。



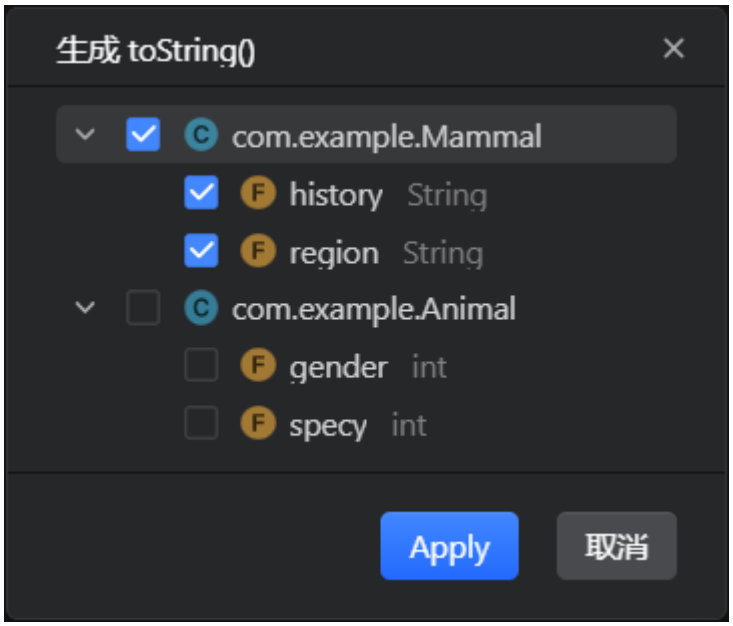
生成 hashCode()和 equals()

使用此“源代码操作”生成具有默认实现的**hashCode()**和**equals()**方法存根。CodeArts IDE提示用户选择类的非静态字段，这些字段用于确定对象相等/计算哈希代码值。



生成 toString()

使用此“源代码操作”生成返回类的字符串表示形式的**toString()**方法。在“生成 toString()”对话框中，选择要在生成的**toString()**方法中返回的字段。



2.5.4.4 生成测试类

使用此“源代码操作”为具有选定测试框架的生产类生成测试类。

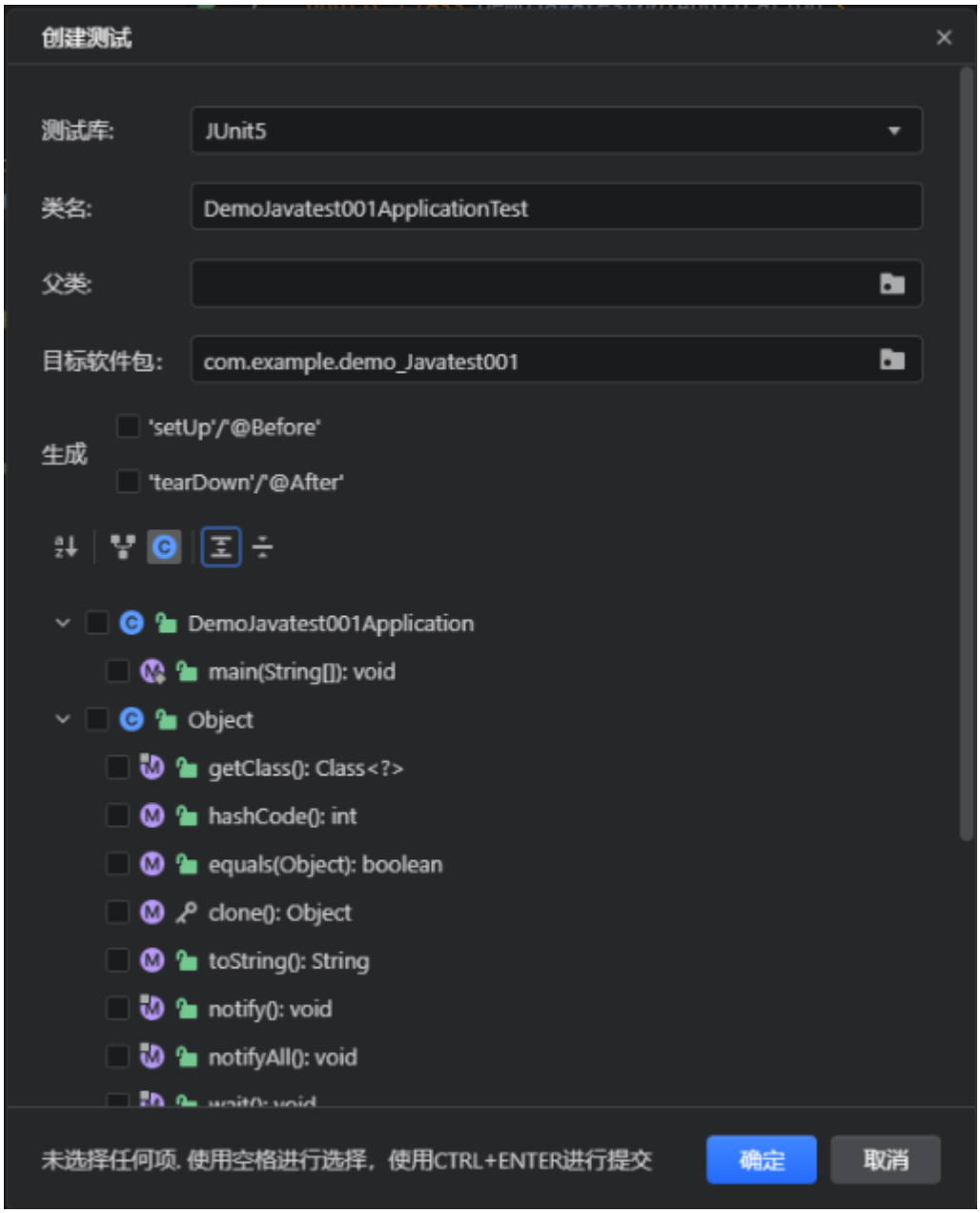
说明

有关测试Java代码的更多详细信息，请参阅**调试**。

在“创建测试”对话框中，提供测试类参数：

- **测试库**：选择要使用的测试库。

- **类名**：提供测试类的名称，用户可自定义输入。
- **父类**：为父类。
- **目标软件包:** 指定存储生成的测试类的包。
- 选中“**'setUp'/'@Before'**”或“**'tearDown'/'@After'**”复选框，让CodeArts IDE为测试夹具生成注释和存根方法。
- 在“**成员**”区域中，选择要为其生成测试方法的方法。要查看所有方法，包括继承的方法，请选中“**显示继承的方法**”复选框。



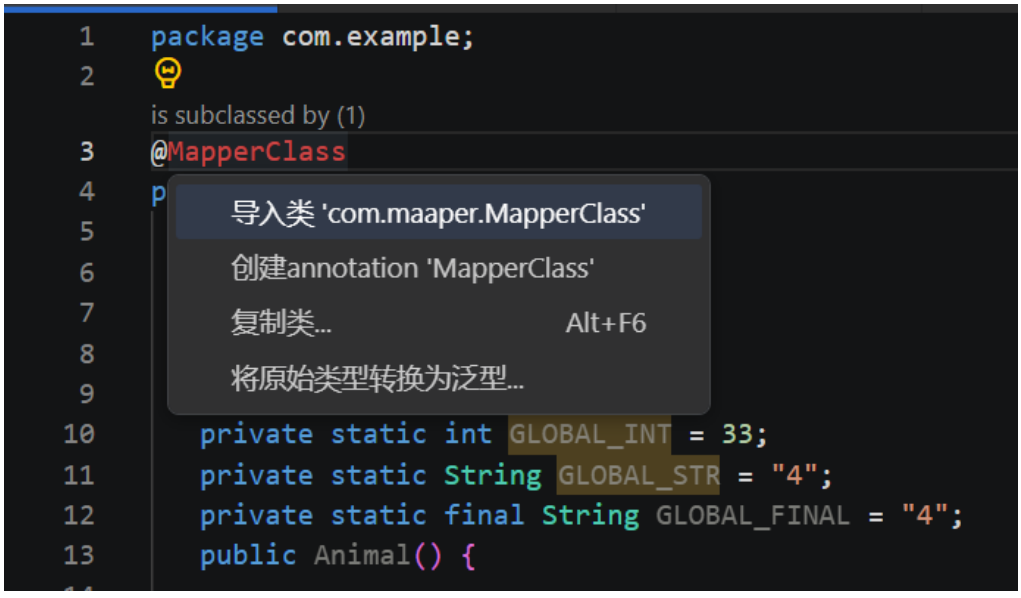
2.5.5 自动导入

2.5.5.1 添加导入

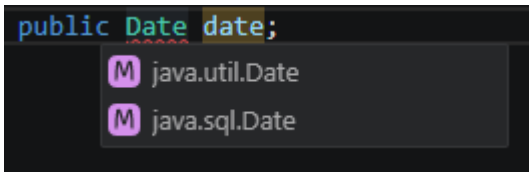
当用户键入包含对尚未导入元素的引用的代码块时，CodeArts IDE会自动插入缺少的导入语句。CodeArts IDE还突出显示当前缺少导入语句的符号，并提供了自动插入导入的**源代码操作**。

步骤1 在代码编辑器中，将光标定位在突出显示的未解析符号上。

步骤2 按“**Ctrl+.**”或“**Alt+Enter**”（适用于IDEA快捷键方案）键，然后在弹出菜单中选择**导入**。



如果有多个可能的导入声明，请选择**Import class**，然后在弹出菜单中选择要导入的所需类。



----结束

2.5.5.2 组织导入

CodeArts IDE突出显示模糊和未使用的导入，并提供**源代码操作**来自动组织它们。

步骤1 在代码编辑器中，将光标定位在突出显示的import语句处。

步骤2 按**“Ctrl+.”**或**“Alt+Enter”**（适用于IDEA快捷键方案）键，然后在弹出菜单中选择**整理导入**。CodeArts IDE删除不明确和未使用的导入，并按字母顺序对导入语句进行排序。

```
16  package org.springframework.samples.petclinic.model;
17
18  import jakarta.validation.constraints.NotEmpty;
19  port jakarta.validation.constraints.NotEmpty;
20  import org.springframework.web.bind.annotation.RestController;
21  import java.lang.annotation.Documented;
22  import java.lang.annotation.ElementType;
23  import jakarta.persistence.Column;
24  import java.lang.annotation.Retention;
25  import java.lang.annotation.RetentionPolicy;
26  import jakarta.persistence.MappedSuperclass;
27
28
```

```
16  package org.springframework.samples.petclinic.model;
17
18  import jakarta.persistence.Column;
19  port jakarta.persistence.MappedSuperclass;
20  import jakarta.validation.constraints.NotEmpty;
21  import org.springframework.web.bind.annotation.RestController;
22
23
24  /** ...
    is subclassed by (2)
29  @MappedSuperclass
30  @RestController
31  public class Person extends BaseEntity {
```

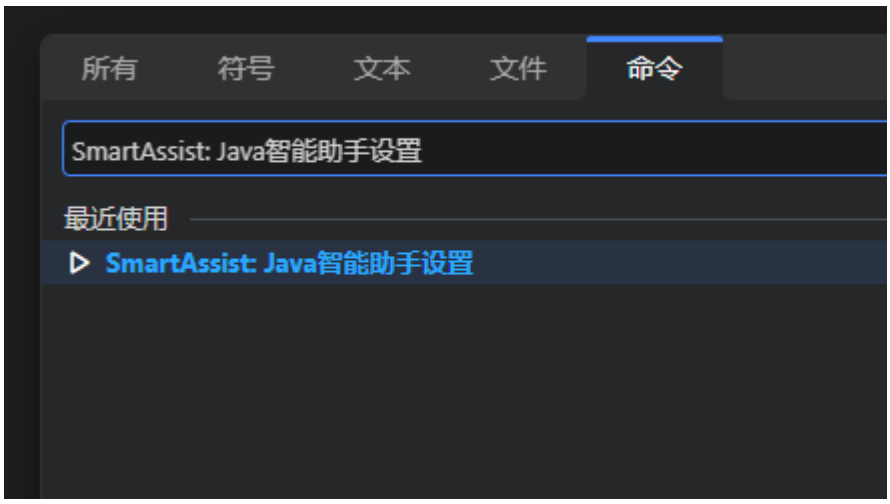
----结束

2.5.5.3 验证导入

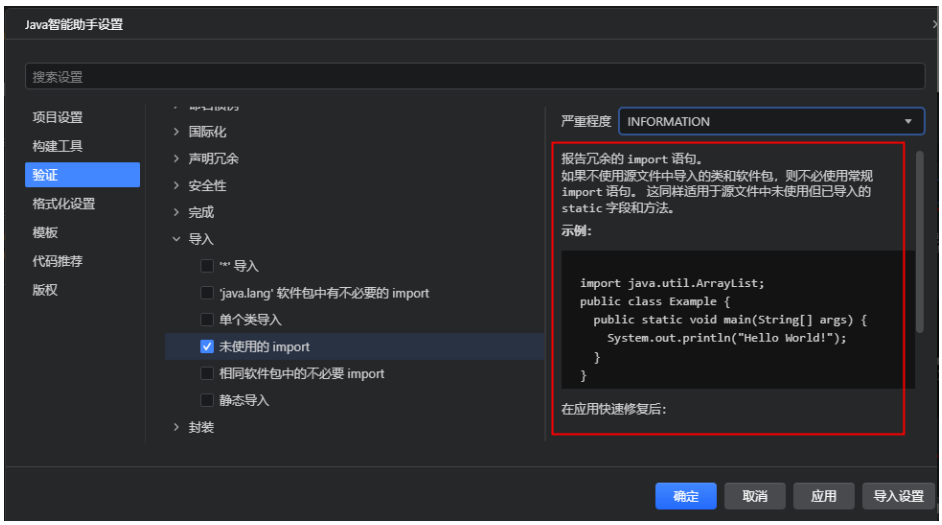
CodeArts IDE提供了几个导入验证规则，用户可以根据需要配置这些规则。

1. 通过执行以下任一操作打开“Java智能助手设置”对话框：
   - 单击CodeArts IDE状态栏中的**“Java智能助手”**。

- 通过“Ctrl+Shift+P”或者“Ctrl+Ctrl”打开“命令”面板，输入**SmartAssist: Java智能助手设置**命令。



2. 在搜索字段中键入**import**以快速找到导入验证规则。然后配置如下：
   - 要启用或禁用规则，请使用其名称旁边的复选框。
   - 要调整相应代码问题在代码编辑器中突出显示的方式，请在**“严重程度”**列表中选择所需的严重性级别。



说明

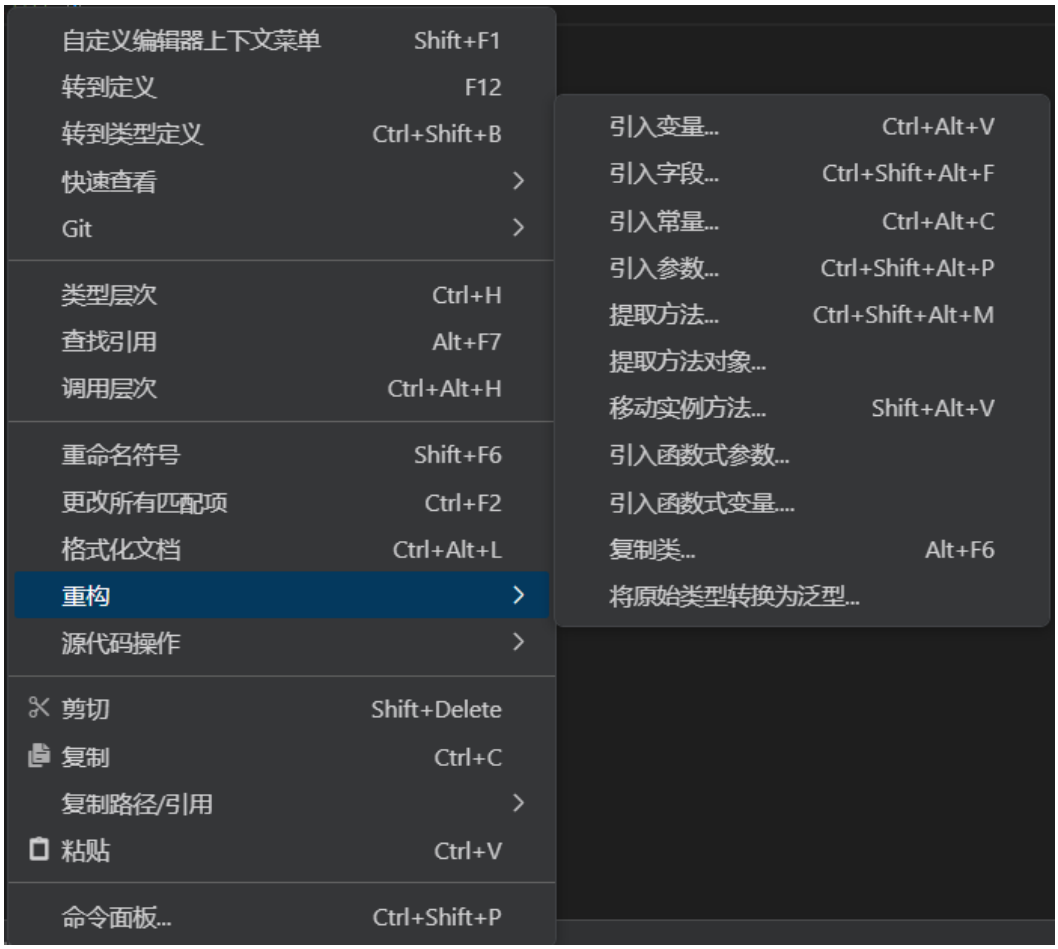
有关验证规则的更多详细信息，请参见**代码校验**。

2.5.6 重构

2.5.6.1 简介

Java程序重构的目标是在不影响程序行为的前提下进行系统范围的代码改进，SmartAssist扩展提供了许多便捷的重构选项。

重构命令存在于编辑器的右键菜单中，选择要重构的元素并单击右键，然后从上下文菜单中选择“**重构**”，或者在主菜单中选择“**重构**”。



2.5.6.2 移动重构

2.5.6.2.1 复制类

此重构支持用户在不同的包中创建类的副本，维护正确的目录结构。

执行重构

步骤1 在代码编辑器中，将光标放置在要复制的类中的任何位置，单击右键。

步骤2 在编辑器上下文菜单中，选择“重构 > 复制类” 或按“**Alt+F6**”或“**F5**”(仅适用于IDEA快捷键方案)。

步骤3 在打开的“复制类”对话框中，提供重构参数。

步骤4 单击“**重构**”以应用重构。

----结束

示例

作为一个例子，创建一个**Refactoring**类的副本，该类存储在包**com.refactoring.source**中。复制的类**RefactoringCopy**将存储在包**com.refactoring.target**中。

重构前

```
package com.refactoring.source;

public class Refactoring {
    public String testStr = "test";
    public void DoSomeThing() {
        System.out.println(testStr);
    }
}
```

重构后

```
package com.refactoring.source;

public class Refactoring {
    public String testStr = "test";
    public void DoSomeThing() {
        System.out.println(testStr);
    }
}

package com.refactoring.target;
```

```
public class RefactoringCopy {
    public String testStr = "test";
    public void DoSomeThing() {
        System.out.println(testStr);
    }
}
```

2.5.6.2.2 移动类

此重构允许用户移动不同包中的类，维护正确的目录结构。

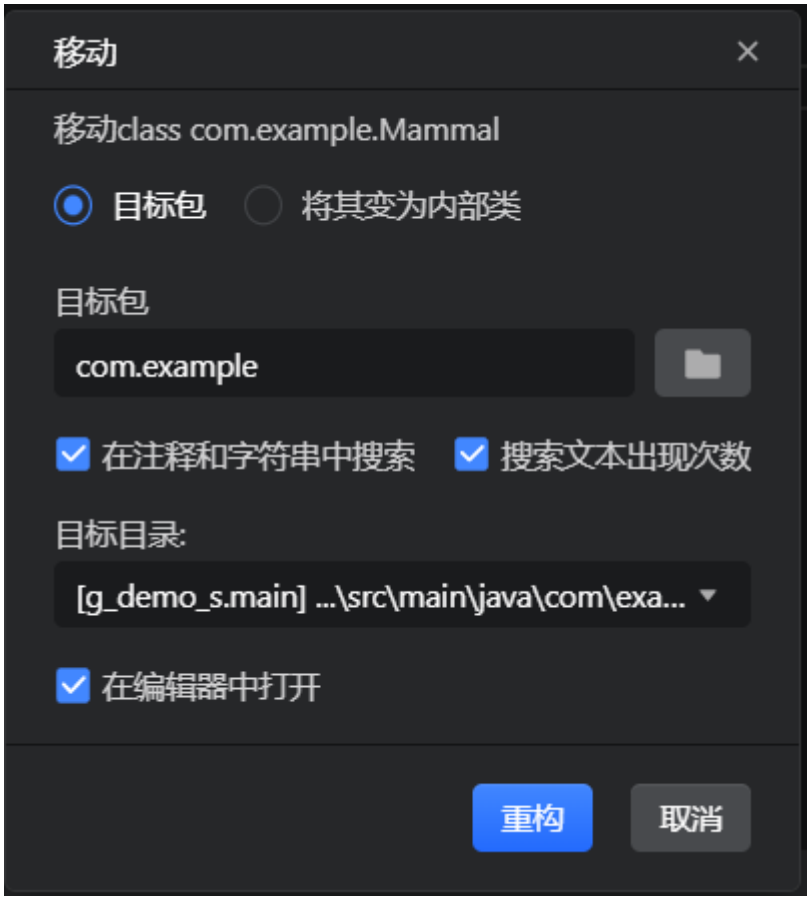
执行重构

步骤1 在代码编辑器中，将光标放在想要移动的类上。

步骤2 在编辑器的上下文菜单中，选择“重构 > 移动类...”或按“F6”。

步骤3 在打开的“移动”对话框中，提供重构参数。

- 要将类移动到不同的包中，请选择**“目标包”**并在**“目标包”**选择框中选择目标包。单击浏览按钮（ ），在打开的“选择目标包”对话框中，选择包或创建一个新包。
- 要将类移动到其他类中，使其成为内部类，请选择**“将其变为内部类”**并在**“将其变为内部类”**输入框中输入目标类的完全限定名称。
- 要在代码中搜索移动的类的出现情况，请勾选**“在注释和字符串中搜索”**和**“搜索文本出现次数”**复选框。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，将存储在包**com.refactoring.source**中的类**Refactoring**移动到包**com.refactoring.target**中。

重构前

```
package com.refactoring.source;

public class Refactoring {
    public String testStr = "test";
    public void DoSomeThing() {
        System.out.println(testStr);
    }
}
```

重构后

```
package com.refactoring.target;

public class Refactoring {
   public String testStr = "test";
   public void DoSomeThing() {
      System.out.println(testStr);
   }
}
```

2.5.6.2.3 移动包

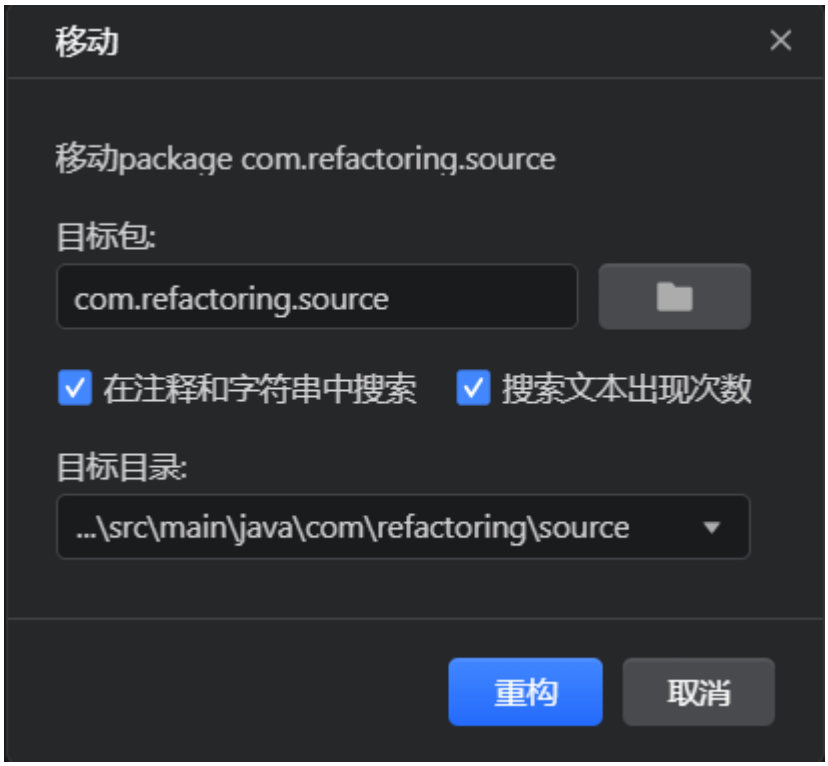
此重构允许用户将包移动到不同的包中，以保持正确的目录结构。

执行重构

步骤1 在代码编辑器中，将光标放置在要移动的包声明上。或者，在**资源管理器**中，选择与所需软件包对应的目录。

步骤2 在编辑器的上下文菜单中，选择“重构 > 移动包...” 。

步骤3 在打开的“移动”对话框中，在“**目标包**”选择框中选择目标package。单击**浏览**按钮（ ），然后在打开的“选择目标包”对话框中，选择包或创建新包。要搜索代码中移动包的引用，请选中“**在注释和字符串中搜索**”和 “**搜索文本出现次数**”复选框。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，将包**com.source.feature**移动到包**com.target**中，并替换代码中出现的**com.source.feature**包的位置。

重构前

```
package com.source.feature;

public class Refactoring {
    public String testStr = "test";
    public void DoSomeThing() {
        System.out.println(testStr);
    }
}
```

重构后

```
package com.target.feature;

public class Refactoring {
   public String testStr = "test";
   public void DoSomeThing() {
      System.out.println(testStr);
   }
}
```

2.5.6.2.4 将内部类移动到上一级

此重构支持用户将内部类移至上层，这个重构将包外的类、函数、变量、常量和命名空间移动到一个包中。

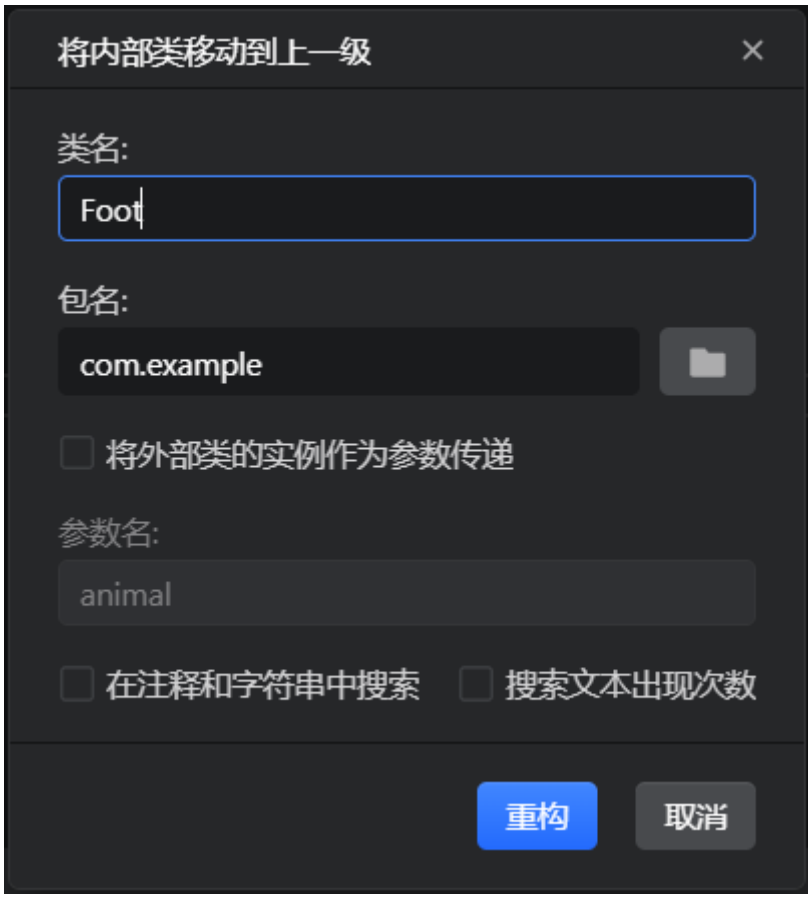
执行重构

步骤1 在代码编辑器中，将光标放在要移动到上层的类的声明位置。

步骤2 在编辑器的上下文菜单中，选择“重构 > 将内部类移动到上一级”。

步骤3 在打开的“将内部类移动到上一级”对话框中，提供移动类的名称和其他重构选项。

- 要保留移动类对其以前的外部类的访问权限，请勾选复选框**“将外部类的实例作为参数传递”**。
- 要在搜索代码中移动类的引用，请勾选**“在注释和字符串中搜索”**和**“搜索文本出现次数”**复选框 。



步骤4 单击**“重构”**以应用重构。

----结束

示例

例如，将**InnerClass**移动到上层。要保留从**InnerClass**到**OuterClass**的访问，**OuterClass**的实例将作为参数传递给**InnerClass**。

重构前

```
class OuterClass {
    String str = "test";
    public void outermethod(){
        InnerClass ic = new InnerClass();
        ic.print();
```

```
    }

    class InnerClass {
        public void print(){
            System.out.println(str);
        }
    }
}
```

重构后

```
class OuterClass {
    String str = "test";
    public void outermethod(){
        InnerClass ic = new InnerClass(this);
        ic.print();
    }
}

class InnerClass {
    private final OuterClass outerClass;

    public InnerClass(OuterClass outerClass) {
        this.outerClass = outerClass;
    }

    public void print() {
        System.out.println(outerClass.str);
    }
}
```

2.5.6.2.5 移动实例方法

如果此方法在项目中具有类型参数，则此重构允许用户将实例（非静态）方法移动到其他类。

执行重构

步骤1 在代码编辑器中，将光标放在要移动到另一个类的实例方法的声明上。

步骤2 在编辑器的上下文菜单中，选择“重构 > 移动实例方法...”。

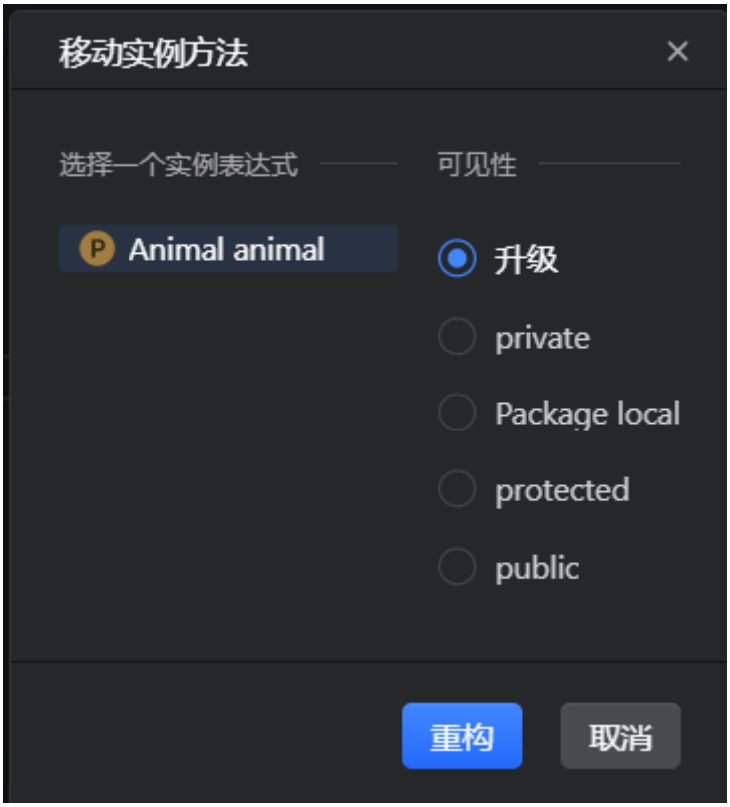
说明

如果该方法在项目中没有类型参数，则需要将其设置为静态，然后将其移动到所需的类。有关详细信息，请参见**使方法静态**和**移动静态成员**。

步骤3 在打开的“移动实例方法”对话框中，提供重构选项。

- 在**“选择一个实例表达式”**列表中，选择要将实例方法移动到的目标类。潜在移动目标的列表包括当前类中的方法参数的类和字段的类。

- 为将要移动的方法添加参数名称，并将替换对当前类所有参数的引用。
- 在**“可见性”**区域中，指定移动方法的可见性修改器，或选择**“升级”**以自动将可见性设置为所需的级别。



步骤4 单击**“重构”**以应用重构。

----结束

示例

例如，将实例方法**getName**从**Car**类移动到**MoveInstanceMethod**类方法。

重构前

```
public class MoveInstanceMethod {
    public static void main(String[] args) throws Exception {
        Car c = new Car();
        System.out.println(c.getName(new MoveInstanceMethod()));
    }
}

class Car {
    String name = "Default Car";

    String getName(MoveInstanceMethod anotherObject) {
        System.out.print(anotherObject.toString());
        return this.name;
    }
}
```

重构后

```
public class MoveInstanceMethod {
    public static void main(String[] args) throws Exception {
```

```
        Car c = new Car();
        System.out.println(new MoveInstanceMethod().getName(c));
    }

    String getName(Car car) {
        System.out.print(toString());
        return car.name;
    }
}

class Car {
    String name = "Default Car";
}
```

2.5.6.2.6 移动静态成员

此重构允许用户将类的静态成员移动到不同的类中。

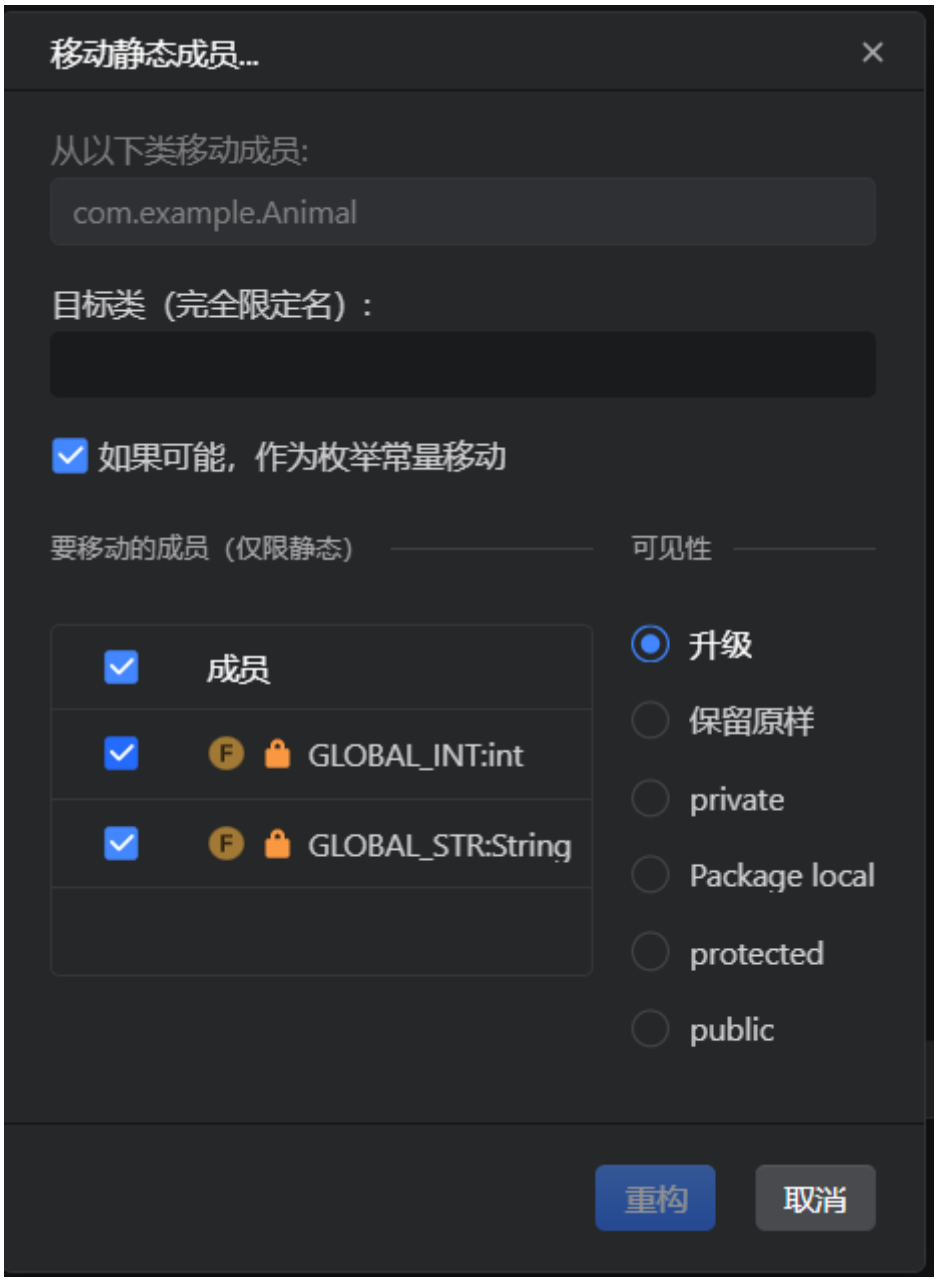
执行重构

步骤1 在代码编辑器中，将光标放置在要移动到另一个类的静态成员（字段或方法）的声明上。

步骤2 在编辑器的上下文菜单中，选择“重构 > 移动静态成员...”。

步骤3 在打开的“移动静态成员”对话框中，提供重构选项。

- 提供有效的目标类名称。
- 在**“要移动的成员”**列表中，选中要移动的静态成员复选框。
- 在**“可见性”**区域中，指定移动的静态成员的可见性修改器，或选择**“升级”**以自动将可见性设置为所需的级别。
- 选中**“如果可能，作为枚举常量移动”**复选框，将常量（即**static final**字段）作为枚举常量移动到枚举类型。如果枚举类型具有类型参数的构造函数，则这是可能的。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，将类**MoveStaticMembers**的所有静态成员移动到枚举类**MyEnum**。由于**MyEnum**有一个带有**String**类型参数的构造函数，因此可以使用**Move as enum constant if possible**选项将**STATICFINALSTR**和**staticStr**字段移动为枚举常量。

重构前

```
class MoveStaticMembers {
    Boolean isTrue;
    public static final String STATICFINALSTR = "TEST";
    static List<Integer> staticList;
```

```
    static int[] staticN;
    private static final String staticStr = "test";

    public static void staticMethod() {
        System.out.println(staticStr);
    }

    private static Boolean staticMethod2() {
        return true;
    }

    void method() {
    }
}

enum MyEnum {
    ;
    String typeName;

    MyEnum(String name) {
        typeName = name;
    }
}
```

重构后

```
class MoveStaticMembers {
    Boolean isTrue;

    void method() {
    }
}

enum MyEnum {
    STATICFINALSTR("TEST"), staticStr("test");
    static List<Integer> staticList;
    static int[] staticN;
    String typeName;

    MyEnum(String name) {
        typeName = name;
    }

    public static void staticMethod() {
        System.out.println(staticStr);
    }

    private static Boolean staticMethod2() {
        return true;
    }
}
```

2.5.6.2.7 上/下移成员

“上移成员”重构允许用户将类成员移动到超类或接口。**“下移成员”**重构的作用则相反，允许用户将类成员移动到子类。

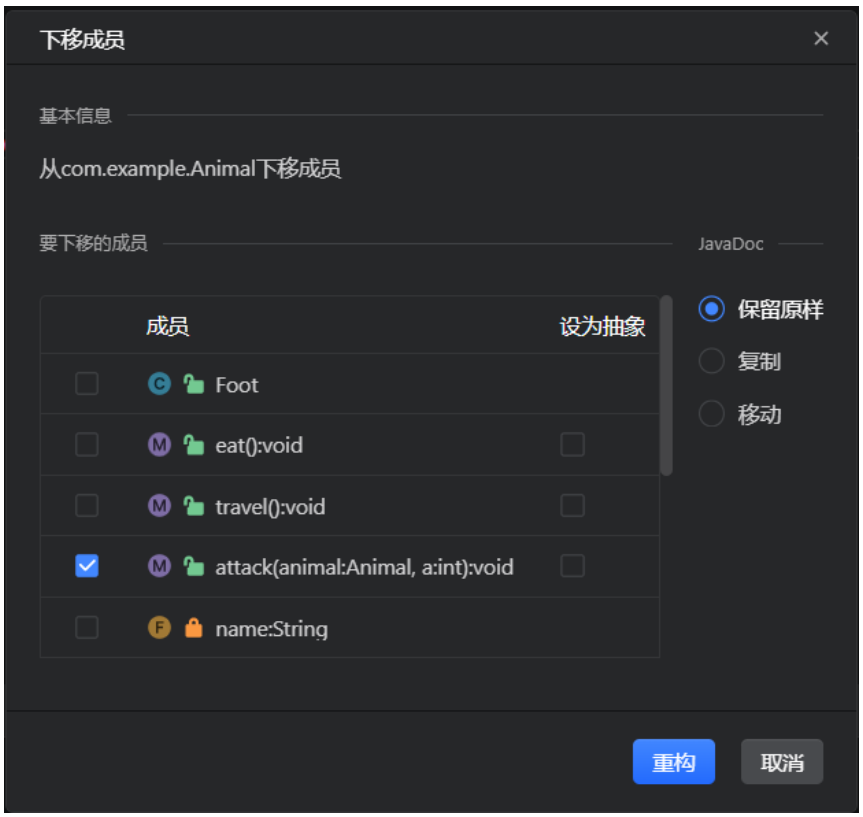
执行重构

步骤1 在代码编辑器中，将光标放置在要向上拉或向下推类层次结构的字段或方法的声明上。

步骤2 在编辑器的上下文菜单中，选择“重构 > 上移成员 / 下移成员”。

在打开的“上移成员”或“下移成员”对话框中，选择目标类并提供重构选项。

- 选中要向上（向下）移动的类成员的复选框。
- 对于方法，选中“**设为抽象**”复选框，将被移动的原始方法转换为抽象方法，并将其实现保留在原始类中。
- 在“**JavaDoc**”选项中，提供JavaDoc注释应与移动的类成员一起移动、复制还是保持原样的选择。



- 单击“**重构**”以应用重构。

----结束

示例

作为一个例子，从超类**AbstractClass**中提取字段**myField**和方法**print**的类层次结构。

重构前

```
class PullUp {

  public static void main(String[] args) {
    new InnerClass().print();
  }

  private static class InnerClass extends AbstractClass {
    public String myField;
    public void print() {
      System.out.println("Hello World");
    }
  }

  private static abstract class AbstractClass {
  }
}
```

重构后

```
class PullUp {

  public static void main(String[] args) {
    new InnerClass().print();
  }

  private static class InnerClass extends AbstractClass {
  }

  private static abstract class AbstractClass {
    public String myField;

    public void print() {
      System.out.println("Hello World");
    }
  }
}
```

2.5.6.3 提取/引入重构

2.5.6.3.1 引入变量

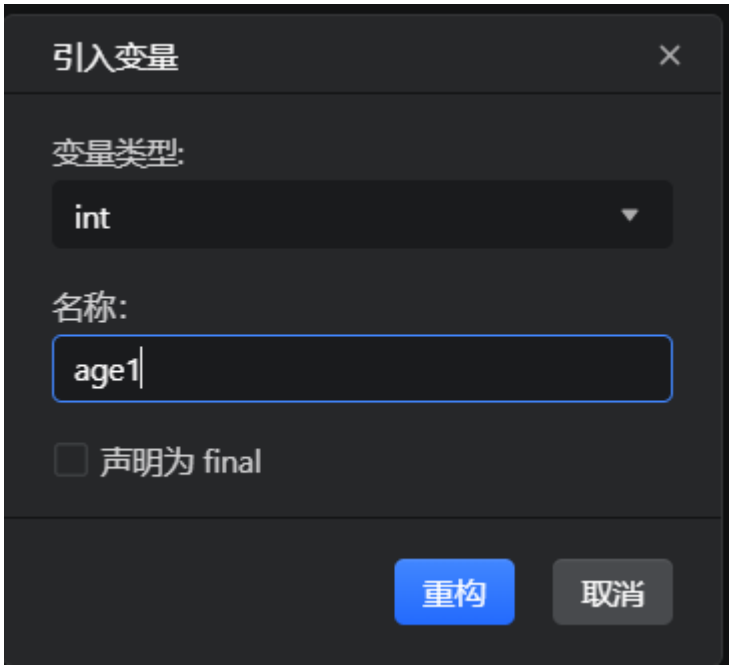
此重构允许用户创建新变量并用选定的表达式进行初始化，然后使用新变量的引用替换原始表达式，这与**内联变量**相反。

执行重构

步骤1 在代码编辑器中，将光标放置在要提取到变量的表达式上。

步骤2 在编辑器上下文菜单中，选择“重构 > 引入变量...”或者使用快捷键“Ctrl+Alt+V”。

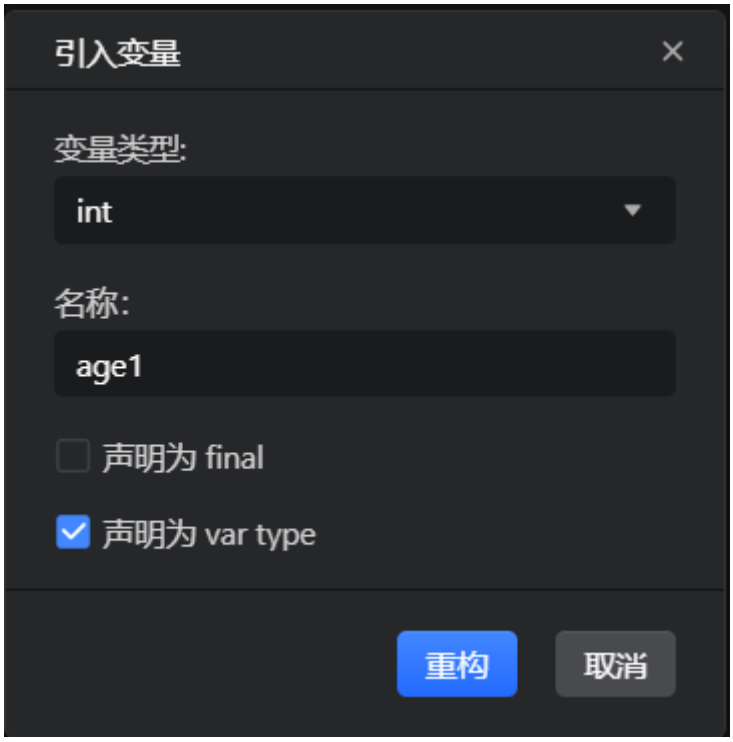
步骤3 如果多个表达式属于重构范围，请在弹窗中选择所需的表达式。



步骤4 在打开的“引入变量”对话框中，提供引入变量的类型和名称，并选择重构选项：

- 选择重构是否应用于所有找到的表达式，还是仅适用于当前表达式。

- 选择变量是否应“声明为final”。
- 如果项目的语言级别设置为Java 10及更高版本，则可以选择使用**var**标识符来声明变量，而不是显式提供其类型，这有助于提高代码的可读性。



步骤5 单击“**重构**”以应用重构。

----结束

示例

例如，将表达式“Hello”+“ ”+“World!”提取到一个新的**message**变量中。

重构前

```
class ExtractVariable {

    public static void main(String[] args) {
        sayHello();
    }

    private static void sayHello() {
        System.out.println("Hello" + " " + "World!");
    }
}
```

重构后

```
class ExtractVariable {

    public static void main(String[] args) {
        sayHello();
    }

    private static void sayHello() {
        String message = "Hello" + " " + "World!";
        System.out.println(message);
```

```
    }
}
```

2.5.6.3.2 引入参数

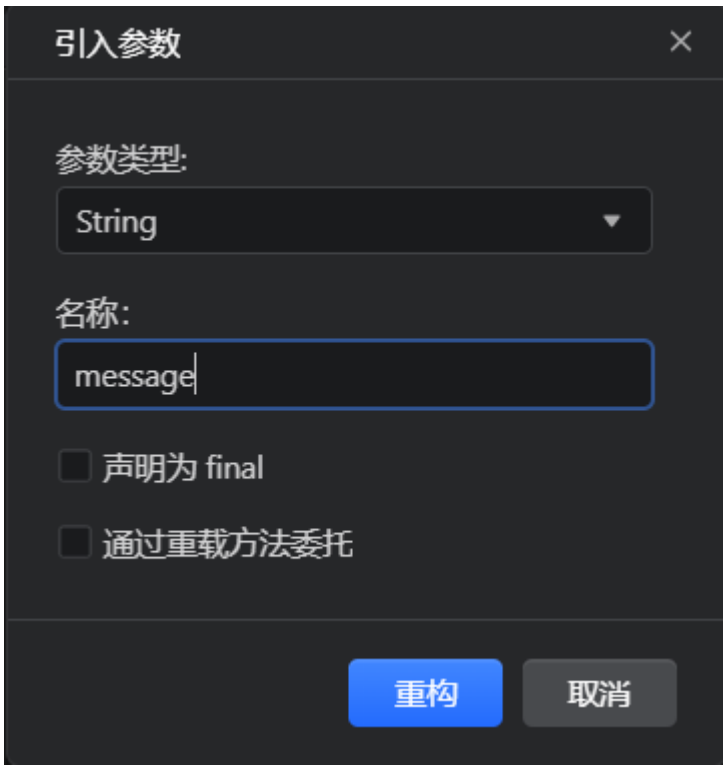
此重构允许用户为方法声明引入新参数，在方法调用时使用原始表达式作为参数。还可以选择保留原始方法或使用新参数定义一个新方法，这与**内联参数**重构相反。

执行重构

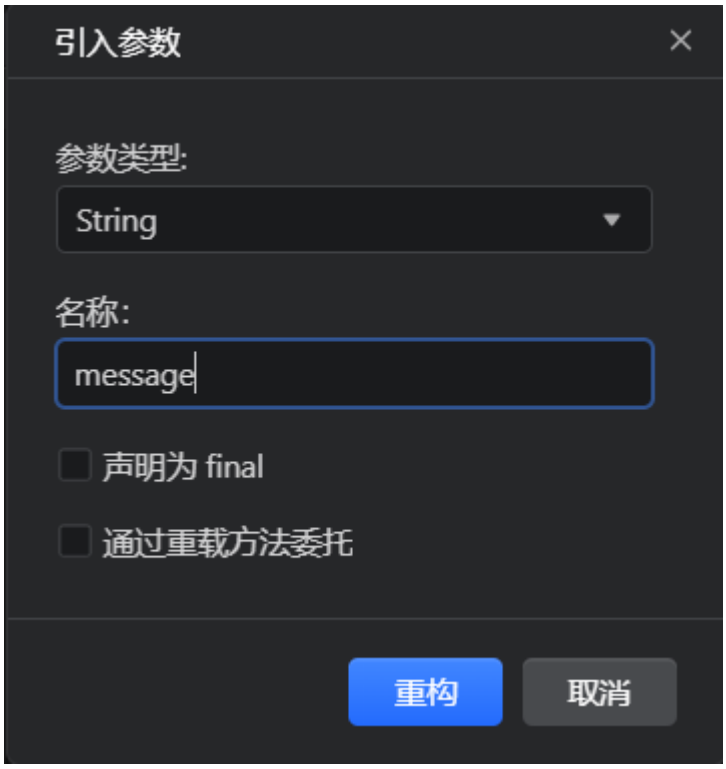
步骤1 在代码编辑器中，将光标放置在要提取到参数的表达式上。

步骤2 在编辑器的上下文菜单中，选择“重构 > 引入参数...”或者使用快捷键“Ctrl+Shift+Alt+P”。

步骤3 如果多个表达式属于重构范围，请在弹窗中选择所需的表达式。



步骤4 在打开的“引入参数”对话框中，提供引入参数的类型和名称，并选择是否应将参数“声明为final”参数。要保留原始方法并使用引入的参数定义新方法，请使用**“通过重载方法委托”**选项。



步骤5 单击“**重构**” 以应用重构。

----结束

示例

作为一个例子，将表达式 "Hello" + " " + "World!" 提取到一个新的**message**参数中，并将其委托给一个重载的方法。

重构前

```
class ExtractParameter {

    public static void main(String[] args) {
        sayHello();
    }

    private static void sayHello() {
        System.out.println("Hello" + " " + "World!");
    }
}
```

重构后

```
class ExtractParameter {

    public static void main(String[] args) {
        sayHello();
    }

    private static void sayHello() {
        sayHello("Hello" + " " + "World!");
    }

    private static void sayHello(String message) {
        System.out.println(message);
```

```
    }
}
```

2.5.6.3.3 引入字段

此重构允许用户创建一个新的类字段，使用选定的表达式进行初始化，并以新类字段的引用替换原始表达式，这与**内联字段**重构相反。

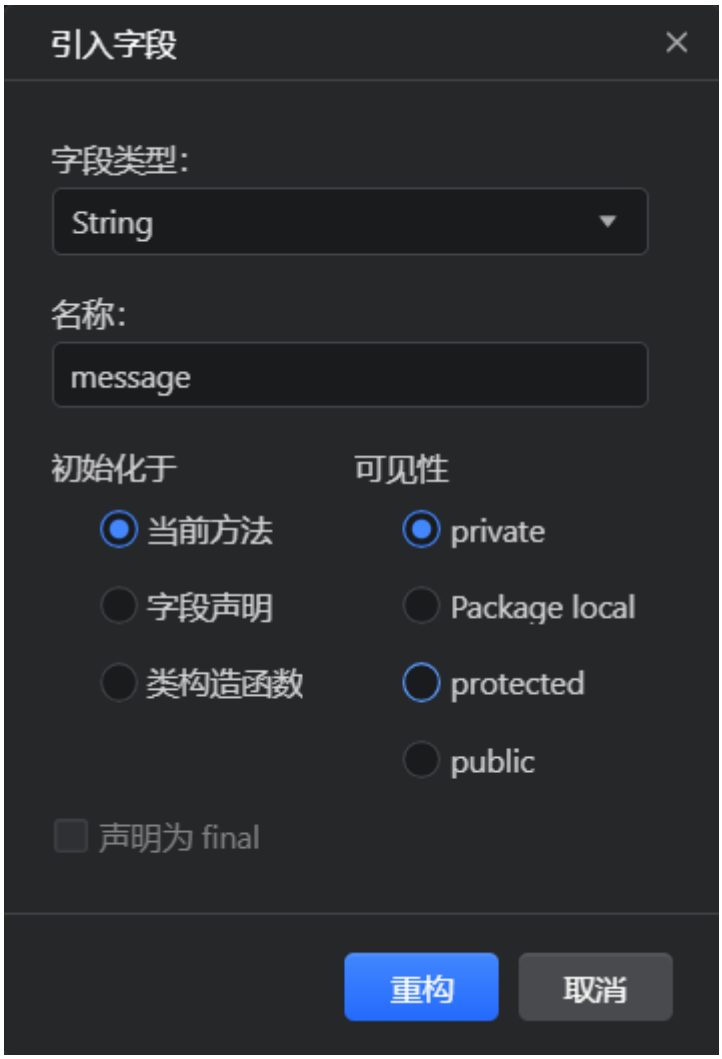
执行重构

步骤1 在代码编辑器中，将光标放置在要提取到类字段的表达式上。

步骤2 在编辑器的上下文菜单中，选择“重构 > 引入字段...”或者使用快捷键“Ctrl+Shift+Alt+F”。

步骤3 如果多个表达式属于重构范围，请在出现的弹窗中选择所需的表达式。

步骤4 在打开的“引入变量”对话框中，提供引入字段的类型和名称、其初始化和可见性选项，并选择是否应将字段声明为final字段。



步骤5 单击“**重构**”以应用重构。

----结束

示例

例如，将表达式“Hello”+“ ”+“World!”提取到在类构造函数中初始化的新**message**私有字段。

重构前

```
class ExtractField {

    public static void main(String[] args) {
        sayHello();
    }

    private static void sayHello() {
        System.out.println("Hello" + " " + "World!");
    }
}
```

重构后

```
class ExtractField {
```

```
    private static String message;

    public ExtractField() {
        message = "Hello" + " " + "World!";
    }

    public static void main(String[] args) {
        sayHello();
    }

    private static void sayHello() {
        System.out.println(message);
    }
}
```

2.5.6.3.4 引入常量

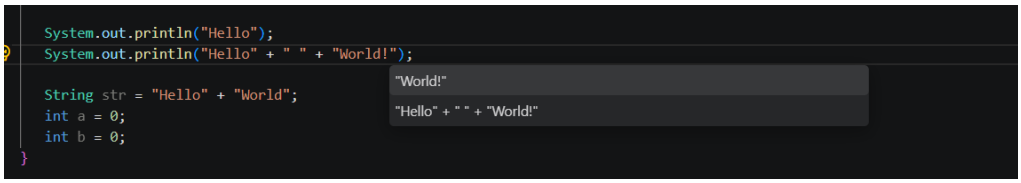
此重构允许用户创建新常量，通过使用选定的表达式进行初始化，并使用创建常量的引用替换原始表达式。

执行重构

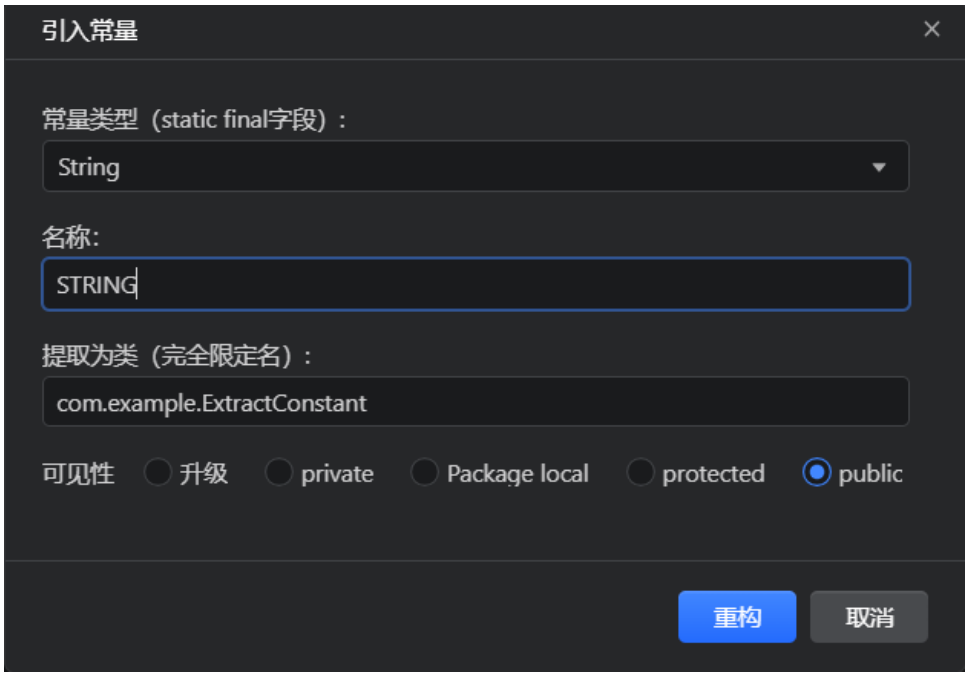
步骤1 在代码编辑器中，将光标放置在要提取到常量的表达式上。

步骤2 在编辑器的上下文菜单中，选择“重构 > 引入常量”或者使用快捷键“Ctrl+Alt+C”。

步骤3 如果多个表达式属于重构范围，请在出现的弹窗中选择所需的表达式。



步骤4 在打开的“引入常量”对话框中，提供引入常量的类型和名称，选择应声明常量的类和常量的可见性修饰符。



步骤5 单击“**重构**”以应用重构。

----结束

示例

例如，将表达式“Hello”+“ ”+“World!”提取到一个新的**MESSAGE**常量中。

重构前

```
class ExtractConstant {

    public static void main(String[] args) {
        sayHello();
    }

    private static void sayHello() {
        System.out.println("Hello" + " " + "World!");
    }
}
```

重构后

```
class ExtractConstant {

    public static final String MESSAGE = "Hello" + " " + "World!";

    public static void main(String[] args) {
        sayHello();
    }

    private static void sayHello() {
        System.out.println(MESSAGE);
    }
}
```

2.5.6.3.5 提取方法

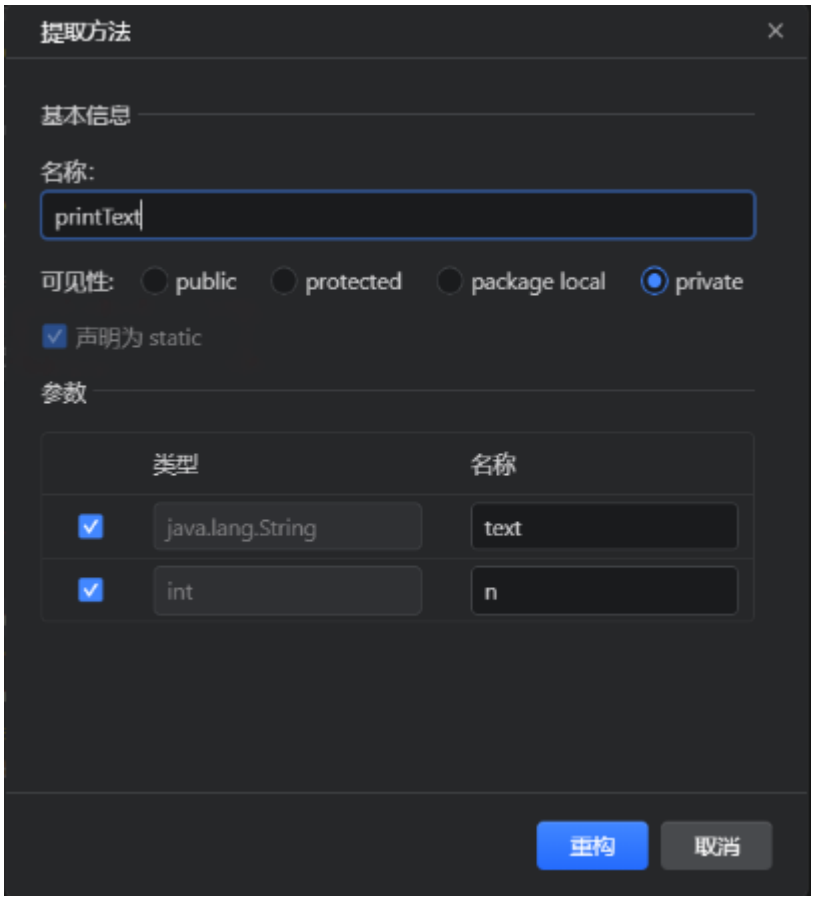
此重构允许用户将任意代码片段移动到单独的方法中，并将其替换为对此新创建的方法的调用。这与**内联方法**相反。

执行重构

步骤1 在代码编辑器中，选择要提取到新方法的代码片段。

步骤2 在的编辑器上下文菜单中，选择“重构 > 提取方法...”或使用快捷键“**Ctrl+Shift+Alt+M**”。

步骤3 在打开的“提取方法”对话框中，提供新方法的名称和可见性修饰符，并从选择范围中选择变量作为方法参数。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，将包含**println**语句的**for**循环提取到一个新的**printText**方法中，其中**text**和**n**作为方法的参数。

重构前

```
class ExtractMethod {
  public static void main(String[] args) {
    String text = "Hello World!";
    int n = 5;

    for (int i = 0; i < n; i++) {
      System.out.println(text);
    }
  }
}
```

重构后

```
class ExtractMethod {
  public static void main(String[] args) {
    String text = "Hello World!";
    int n = 5;

    printText(text, n);
  }
```

```
  public static void printText(String text, int n) {
    for (int i = 0; i < n; i++) {
      System.out.println(text);
    }
  }
}
```

2.5.6.3.6 提取接口

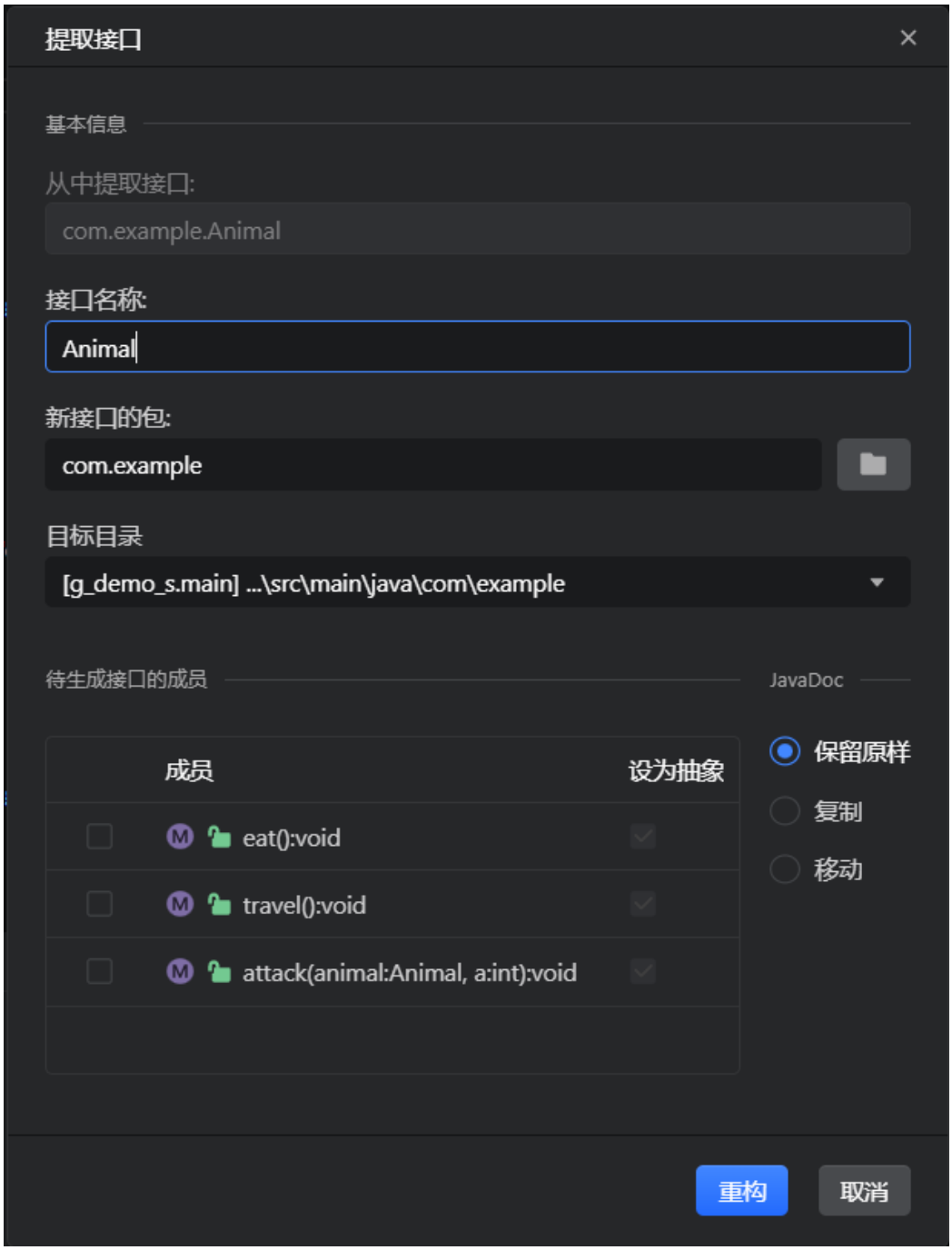
此重构允许用户选定现有类的成员来创建接口，以使它们可以被其他类继承。

执行重构

步骤1 在代码编辑器中，将光标放置在要将其成员提取到接口的类中的任何位置。

步骤2 在编辑器上下文菜单中，选择“重构 > 提取接口...”。

步骤3 在打开的“提取接口”对话框中，提供提取接口的名称和包，选择要提取的类成员。在“**JavaDoc**”选项中，选择是将JavaDoc注释移动或复制到提取的接口，还是保持原样。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，基于提取**ExtractImpl**类的**print**方法创建一个新的提取**ImplInterface**接口。

重构前

```
class ExtractImpl {
    public static void main(String[] args) {
        new ExtractImpl().print();
```

```
  }

  public void print() {
      System.out.println("Hello World!");
  }
}
```

重构后

```
class ExtractImpl implements ExtractImplInterface {
  public static void main(String[] args) {
    new ExtractImpl().print();
  }

  @Override
  public void print() {
    System.out.println("Hello World!");
  }
}

public interface ExtractImplInterface {
  void print();
}
```

2.5.6.3.7 提取超类

此重构允许用户选定现有类的成员创建新的超类。这与**内联超类**相反。

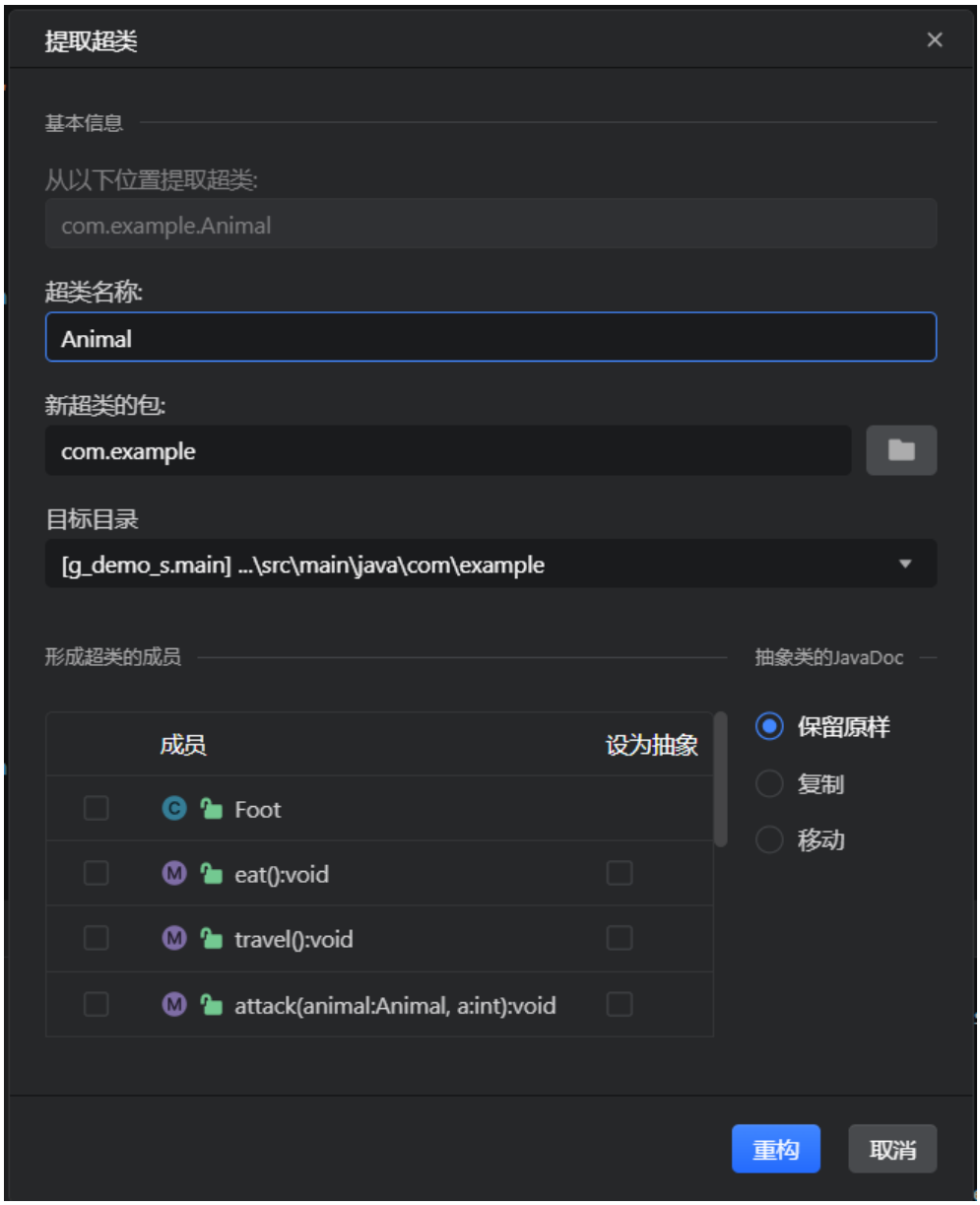
执行重构

步骤1 在代码编辑器中，将光标放置在要将其成员提取到超类中的任何位置。

步骤2 在编辑器上下文菜单中，选择“重构 > 提取超类...”。

步骤3 在打开的“提取超类”对话框中，提供重构参数。

- 提供提取的超类名称和包。
- 在**“形成超类的成员”**区域中，选择要提取的类成员。对于方法，选中**“设为抽象”**复选框，将提取的方法声明为超类中的**abstract**方法，并将其实现保留在原始类中。
- 在**“抽象类的JavaDoc”**选项中，选择是将JavaDoc注释移动或复制到提取的超类，还是保持原样。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，基于提取**ExtractImpl**类的**print**方法和**myField**字段创建一个新的**ExtractAbstract**超类。在创建的超类中，**print**方法将被声明为**abstract**。

重构前

```
class ExtractImpl {
  public int myField;

  public static void main(String[] args) {
    new ExtractImpl().print();
  }

  public void print() {
```

```
    System.out.println("Hello World!");
  }
}
```

重构后

```
class ExtractImpl extends ExtractAbstract {
  public static void main(String[] args) {
    new ExtractImpl().print();
  }

  @Override
  public void print() {
    System.out.println("Hello World!");
  }
}

public abstract class ExtractAbstract {
  public int myField;

  public abstract void print();
}
```

2.5.6.3.8 提取委托

此重构允许用户基于现有类的成员选定来创建新类。

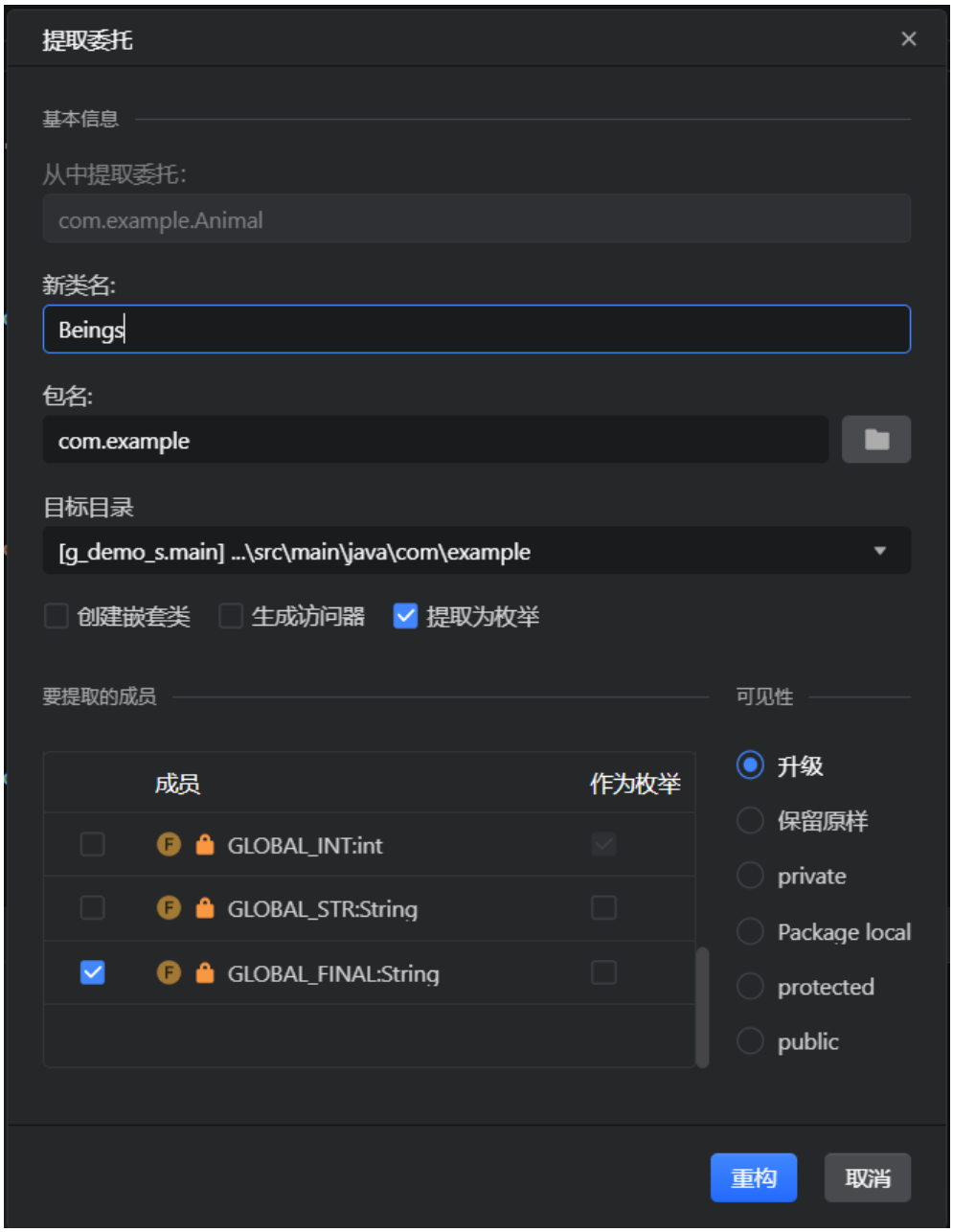
执行重构

步骤1 在代码编辑器中，将光标放置在要将其成员提取到新类的类中的任何位置。

步骤2 在编辑器上下文菜单中，选择“重构 > 提取委托...”。

步骤3 在打开的“提取委托”对话框中，提供重构参数。

- 提供提取类的名称、包和目标目录。
- 选中**“创建嵌套类”**复选框以在当前类中创建新类。
- 选中**“生成访问器”**复选框，为提取的字段生成getter方法。
- 选中**“提取为枚举”**复选框，将提取的类创建为枚举类。如果源类包含静态最终字段**static final fields**，这是可能的。
- 在**“要提取的成员”**区域中，选择要提取的类成员。如果选择了**“提取为枚举”**，则还可以选择要作为枚举常量提取的字段旁边的**“作为枚举”**复选框。
- 在**“可见性”**选项中，选择要应用于提取的类成员的可见性修改器。要自动应用所需的可见性修改器，以便访问类成员，请选择**“升级”**选项。



步骤4 单击“**重构**”以应用重构。

----结束

示例

作为一个例子，将**ExtractDelegate**类中的**print**方法提取到一个嵌套的**Printer**枚举类中。

重构前

```
class ExtractDelegate {
    public static void main(String[] args) {
        new ExtractDelegate().print();
    }
```

```
    private static final String message = "Hello World!";

    private void print() {
        System.out.println(message);
    }
}
```

重构后

```
class ExtractDelegate {
    private final Printer printer = new Printer();

    public static void main(String[] args) {
        new ExtractDelegate().print();
    }

    private void print() {
        printer.print();
    }

    public enum Printer {
        message("Hello World!");
        private String value;

        public String getValue() {
            return value;
        }

        Printer(String value) {
            this.value = value;
        }

        private void print() {
            System.out.println(message.getValue());
        }
    }
}
```

2.5.6.3.9 引入函数式参数

此重构允许用户基于适当的函数接口使用匿名类（或函数表达式）包装代码片段，并将其用作方法的参数。

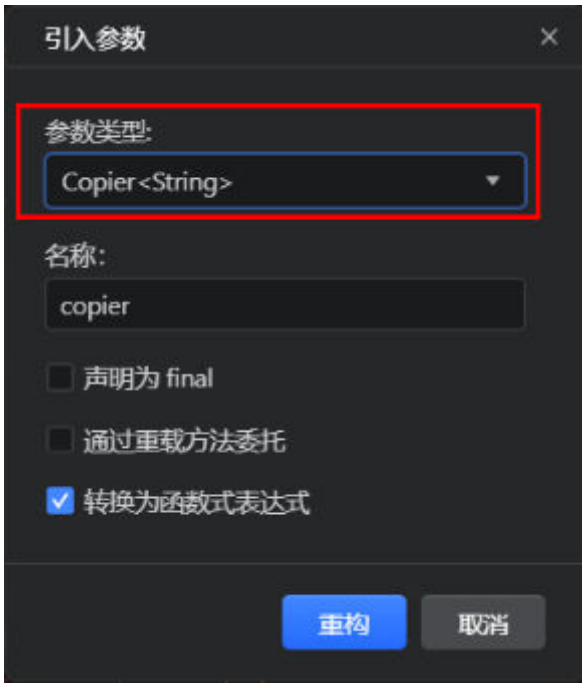
执行重构

步骤1 在代码编辑器中，选择要转换为函数参数的表达式。

步骤2 在编辑器上下文菜单中，选择“重构 > 引入函数式参数...”。

步骤3 在打开的“引入参数”对话框中，提供引入参数的名称和其他重构选项。

- 在**“参数类型”**列表中，为提取的参数选择其中一种的类型。
- 选择是否应将提取的参数“声明为final”参数。
- 要保留原始方法并使用引入的参数定义新方法，请选中**“通过重载方法委托”**选项。
- 要让CodeArts IDE生成函数表达式而不是匿名类，请选中**“转换为函数表达式”**复选框。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，提取表达式**"Hello World!".toUpperCase()**作为**generateText**方法的参数。可以将光标置于"Hello World!".toUpperCase()处，右键唤出重构菜单。通过使用勾选“**转换为函数表达式**”选项，可以将其转换为函数表达式，如果不勾选此选项，则转化为匿名类。

重构前

```
class IntroduceFunctionalParameter {
    public static void main(String[] args) {
        System.out.println(generateText());
    }

    private static String generateText() {
        return "Hello World!".toUpperCase();
    }
}
```

重构后（函数表达式）

```
import java.util.function.Supplier;

class IntroduceFuncParameter {
    public static void main(String[] args) {
        System.out.println(generateText(() -> "Hello World!"));
    }

    private static String generateText(Supplier<String> supplier) {
        return supplier.get().toUpperCase();
    }
}
```

重构后（匿名类）

```
import java.util.function.Supplier;

class IntroduceFunctionalParameter {
    public static void main(String[] args) {
        System.out.println(generateText(new Supplier<String>() {
            public String get() {
                return "Hello World!".toUpperCase();
            }
        }));
    }

    private static String generateText(Supplier<String> supplier) {
        return supplier.get();
    }
}
```

2.5.6.3.10 引入函数式变量

此重构允许将选定的表达式转换为新的函数类型变量或匿名类。

执行重构

步骤1 在代码编辑器中，选择要转换为函数变量的表达式。

步骤2 在编辑器上下文菜单中，选择“重构 > 引入函数式变量...”。

步骤3 在打开的引入函数式变量对话框中，选择**“将字段作为参数传递”**，以使实例字段作为参数传递到创建的函数表达式。

步骤4 单击**“重构”**以应用重构。

----结束

示例

例如，将表达式“**Data**”**+ data.toString()**提取到函数变量中。

重构前

```
import java.util.List;

class PrintAction implements Runnable {
    private List<String> data;

    public PrintAction(List<String> data) {
        this.data = data;
    }

    @Override
    public void run() {
        System.out.println("Data" + data.toString());
    }
}
```

重构后

```
import java.util.List;
import java.util.function.Function;

class PrintAction implements Runnable {
    private List<String> data;

    public PrintAction(List<String> data) {
        this.data = data;
    }

    @Override
    public void run() {
        Function<List<String>, String> listStringFunction = data -> "Data" + data.toString();
        System.out.println(listStringFunction.apply(data));
    }
}
```

2.5.6.3.11 提取方法对象

此重构允许用户将任意代码片段单独移动到新类的方法中，以便用户可以进一步将该方法分解为同一对象上的其他方法。

执行重构

步骤1 在代码编辑器中，选择要提取到包装类的新方法的代码片段。

步骤2 在编辑器上下文菜单中，选择“重构 > 提取方法对象...”。

步骤3 在打开的“提取方法对象”对话框中，提供重构选项。

- **“创建内部类”**：选择创建新的内部类，所有局部变量将转换为该类的字段，并提供类的名称及其可见性修饰符。
- **“创建匿名类”**：选择以创建新对象并提供要创建的方法的名称。
- 在**“参数”**区域中，选择变量作为方法参数。

步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，将包含**println**语句的**for**循环提取到**Printer**包装类的新方法中。

重构前

```
class ExtractMethodObject {
  public static void main(String[] args) {
```

```
    String text = "Hello World!";
    int n = 5;

    for (int i = 0; i < n; i++) {
      System.out.println(text);
    }
  }
}
```

重构后

```
class ExtractMethodObject {
  public static void main(String[] args) {
    String text = "Hello World!";
    int n = 5;

    new Printer(text, n).invoke();
  }

  private static class Printer {
    private String text;
    private int n;

    public Printer(String text, int n) {
      this.text = text;
      this.n = n;
    }

    public void invoke() {
      for (int i = 0; i < n; i++) {
        System.out.println(text);
      }
    }
  }
}
```

2.5.6.3.12 引入参数对象

此重构允许用户将方法的参数移动到新的包装类或某些现有的包装类。所有参数的用法都将替换为对包装类的相应调用。

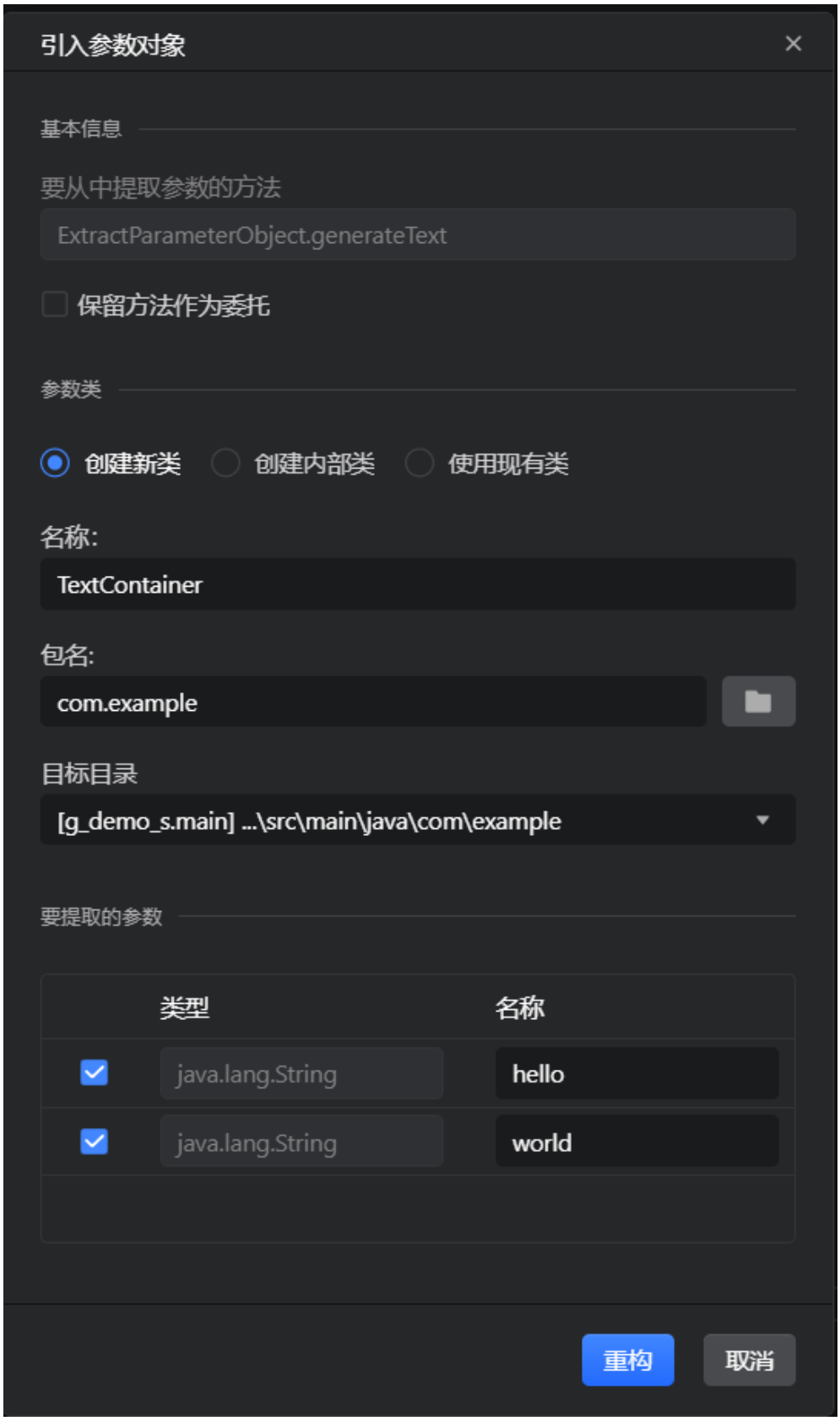
执行重构

步骤1 在代码编辑器中，将光标放置在要提取到包装类的参数上。

步骤2 在编辑器上下文菜单中，选择“重构 > 引入参数对象...”。

步骤3 在打开的“引入参数对象”对话框中，提供重构选项。

- **“保留方法作为委托”**：选择将原始方法保留为新创建方法的委托。
- **“参数类”**： 在此区域中，选择是要创建新的包装参数类、在当前包装参数类中创建内部类，还是使用某些现有类。
- **“名称”**：输入包装后参数的名称。
- **“要提取的参数”**： 在此区域中，选中要提取到包装参数类的参数旁边的复选框。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，将**hello**和**world**参数提取到**TextContainer**内部类，以使生成的**generateText**方法调用将包含对**TextContainer**对象的调用。

重构前

```
class ExtractParameterObject {
    public static void main(String[] args) {
        System.out.println(generateText("Hello", "World!"));
    }

    private static String generateText(String hello, String world) {
        return hello.toUpperCase() + world.toUpperCase();
    }
}
```

重构后

```
class ExtractParameter {
    public static void main(String[] args) {
        System.out.println(generateText(new TextContainer("Hello", "World!")));
    }

    private static String generateText(TextContainer textContainer) {
        return textContainer.getHello().toUpperCase() + textContainer.getWorld().toUpperCase();
    }

    private static class TextContainer {
        private final String hello;
        private final String world;

        private TextContainer(String hello, String world) {
            this.hello = hello;
            this.world = world;
        }

        public String getHello() {
            return hello;
        }

        public String getWorld() {
            return world;
        }
    }
}
```

2.5.6.4 内联重构

2.5.6.4.1 内联变量

此重构允许用户用变量的初始化器替换变量。这与**引入变量**相反。

执行重构

步骤1 在代码编辑器中，将光标放置在要内联其值的变量的使用位置上。

步骤2 在编辑器上下文菜单中，选择“重构 > 内联变量...”，或按“Ctrl+Alt+N”。

步骤3 在打开的“内联变量”对话框中，选择是内联所有变量的引用，还是仅内联当前引用。

说明

如果在代码中修改了变量的初始值，则仅内联修改之前的用法。

步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，内联变量**number**，用其初始化器**test.intValue()**替换它。请注意，由于变量在代码中被进一步修改，因此只有它在修改之前的一次出现会受到重构的影响。

重构前

```
class InlineVariable {
    private int a;
    private Byte test;
    private int b;

    public void InlineVariable() {
        int number = test.intValue();
        int b = a + number;
        number = 42;
    }
}
```

重构后

```
class InlineVariable {
    private int a;
    private Byte test;
    private int b;

    public void InlineVariable() {
        int number;
        int b = a + test.intValue();
        number = 42;
    }
}
```

2.5.6.4.2 内联参数

此重构允许用户使用方法调用中相应参数的值替换方法的参数。这与**引入参数**相反。

执行重构

步骤1 在代码编辑器中，将光标放置在要内联其值的方法参数的声明或调用上。

步骤2 在编辑器上下文菜单中，选择“重构 > 内联参数...”。

----结束

示例

例如，内联参数**pi**，将其替换为参数的值**Math.PI**。

重构前

```
class InlineParameter {
    private double InlineParameter(double rad, double pi) {
        return pi * rad * rad;
    }

    public void Test() {
        double area = InlineParameter(10, Math.PI);
    }
}
```

重构后

```
class InlineParameter {
    private double InlineParameter(double rad) {
        return Math.PI * rad * rad;
    }

    public void Test() {
        double area = InlineParameter(10);
    }
}
```

2.5.6.4.3 内联方法

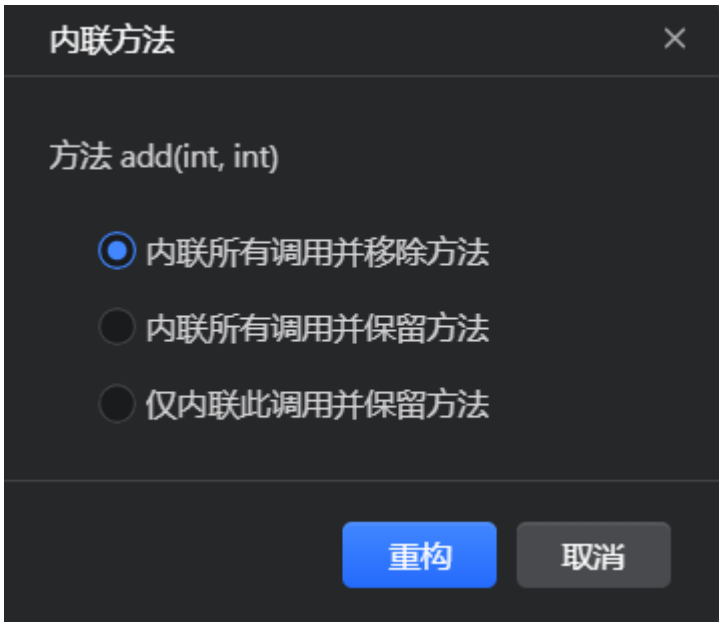
此重构允许用户用方法的主体替换方法的用法。这与**提取方法**相反。

执行重构

步骤1 在代码编辑器中，将光标放置在要内联的方法的声明或调用上。

步骤2 在编辑器上下文菜单中，选择“重构 > 内联方法...”，或按“**Ctrl+Shift+Alt+L**”。

步骤3 在打开的“内联方法”对话框中，选择是否在方法的所有引用都内联后保留该方法。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，内联方法**add**，用方法的主体替换其调用。

重构前

```
class InlineMethod {
  private int a;
  private int b;

  public void InlineMethod() {
    int c = add(a, b);
    int d = add(a, c);
  }

  private int add(int a, int b) {
    return a + b;
  }
}
```

重构后

```
class InlineMethod {
   private int a;
   private int b;

   public void InlineMethod() {
      int c = a + b;
      int d = a + c;
   }
}
```

2.5.6.4.4 内联字段

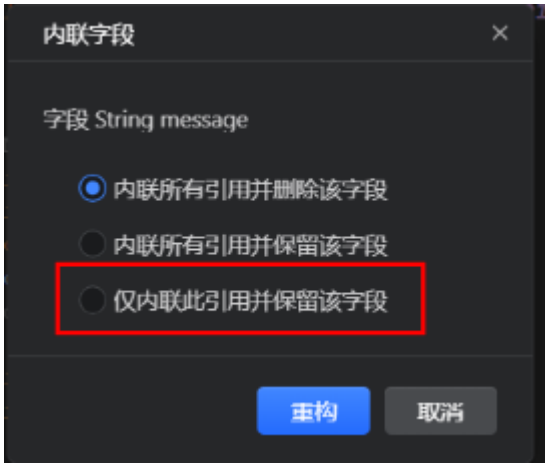
这个重构操作允许用户将字段的使用替换为其值，并删除字段的声明。这与**引入字段**相反。

执行重构

步骤1 在代码编辑器中，将光标放在想要内联其值的字段的声明或使用位置。

步骤2 在编辑器上下文菜单中，选择“重构 > 内联字段...”。

步骤3 在打开的“内联字段”对话框中，选择是要内联所有字段的出现还是只内联当前位置的字段。



步骤4 单击“**重构**”以应用重构。

----结束

示例

举个例子，将字段**message**内联，将其使用位置替换为其初始化值“**Hello World!**”。

重构前

```
class InlineField {
    private String message = "Hello World!";

    private void InlineField() {
        System.out.println(message);
    }
}
```

重构后

```
class InlineField {
    private void InlineField() {
        System.out.println("Hello World!");
    }
}
```

2.5.6.4.5 内联超类

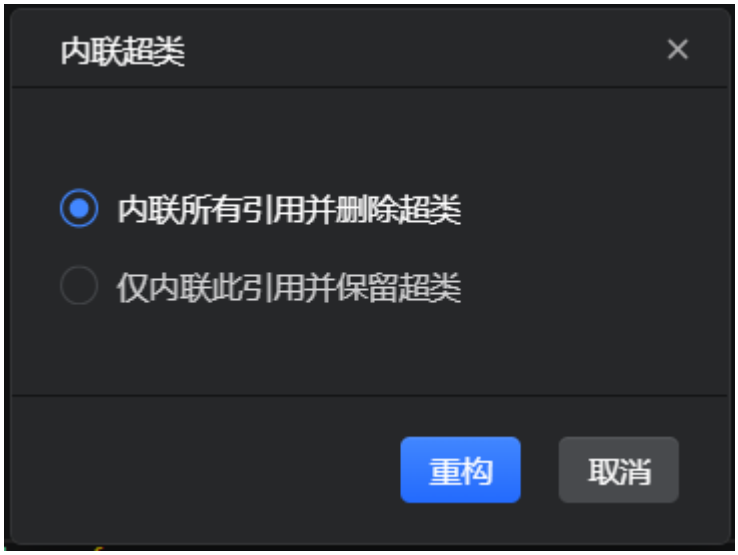
这个重构操作允许用户将超类的成员移动到子类中，并删除超类。这与**提取超类**相反。

执行重构

步骤1 在代码编辑器中，将光标放在想要内联的超类的声明或引用位置。

步骤2 在编辑器上下文菜单中，选择“重构 > 内联超类...”。

步骤3 在打开的“内联超类”对话框中，选择是否在所有内联位置完成后保留超类。



步骤4 单击“**重构**”以应用重构操作。

----结束

示例

举个例子，将类**InlineSuperClass**内联并删除，将其成员移动到**SubClass**中。

重构前

```
class InlineSuperClass {
    public int returnValue() { ... }
    public int returnNewValue() { ... }
}

class SubClass extends InlineSuperClass {
    private int myValue;

    int someMethod() {
        if (myValue > returnValue()) {
            return returnNewValue();
        }
        return 0;
    }
}
```

重构后

```
class SubClass {
    private int myValue;

    int someMethod() {
        if (myValue > returnValue()) {
            return returnNewValue();
        }
```

```
        return 0;
    }

    public int returnValue() { ... }
    public int returnNewValue() { ... }
}
```

2.5.6.4.6 内联为匿名类

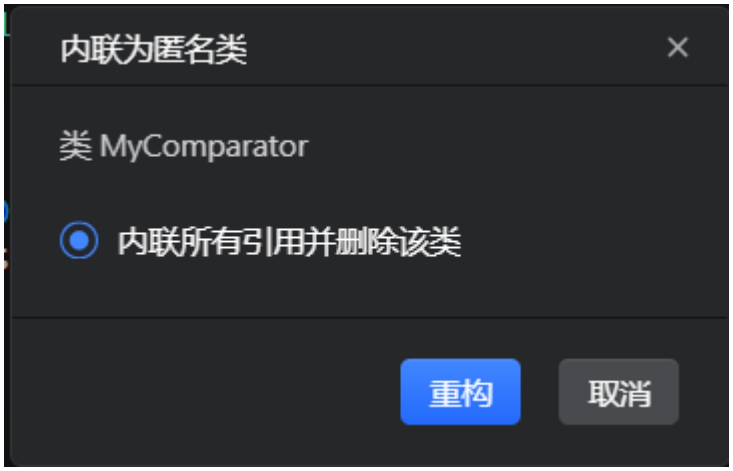
这个重构操作允许用户用其内容替换多余的类。从Java 8开始，内联的匿名类可以自动转换为lambda表达式。

执行重构

步骤1 在代码编辑器中，将光标放在想要内联为匿名类的声明位置。

步骤2 在编辑器上下文菜单中，选择“重构 > 内联为匿名类...”。

步骤3 在打开的“内联为匿名类”对话框中，选择是否在所有内联位置完成后保留该类。



步骤4 单击“**重构**”以应用重构操作。

----结束

示例

举个例子，将类**MyComparator**内联并删除。生成的匿名类将自动转换为lambda表达式。

重构前

```
class InlineAnonymousClazz {
    public class MyComparator implements Comparator<String> {
        @Override
        public int compare(String s1, String s2) {
            return 0;
        }
    }

    void sort(List<String> scores) {
        scores.sort(new MyComparator());
    }
}
```

重构后

```
class InlineAnonymousClazz {
    void sort(List<String> scores) {
        scores.sort((s1, s2) -> 0);
    }
}
```

2.5.6.5 使方法静态

此重构允许用户将内部类转换为嵌套的静态类，或将实例方法转换为静态方法。

执行重构

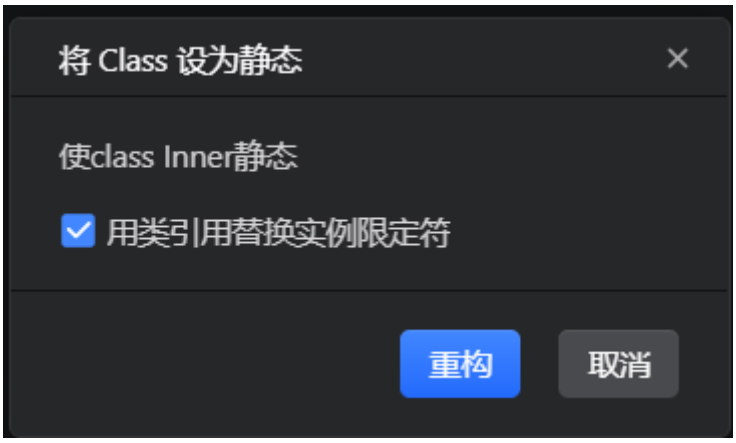
步骤1 在代码编辑器中，将光标放在要转换为静态的类或方法的声明上。

步骤2 在编辑器上下文菜单中，选择“重构 > 使静态...”。

步骤3 在打开的“将Class设为静态”对话框中，提供重构参数。

- 如果类或方法包含对外部类字段的引用，则可以将被引用的对象作为参数传递给类构造函数，或者将被引用的字段作为方法的参数传递给类构造函数。

- 否则，如果类或方法不包含对外部类字段的引用，则可以用类引用替换实例限定符。



步骤4 单击“**重构**”以应用重构。

----结束

示例

作为一个例子，将**Inner**内部类转换为嵌套的静态类。由于**Inner**类包含对**Outer**类的**message**字段的引用，可以将**Outer**对象和**message**字段作为**Inner**类构造函数的参数添加进去。

重构前

```
class Outer {
  public String message;
  public static void main(String[] args) {

  }

  class Inner{
    public void print() {
      System.out.println(message);
    }
  }
}
```

重构后

```
class Outer {
  public String message;
  public static void main(String[] args) {

  }

  static class Inner {
    private Outer outer;
    private String message;

    public Inner(Outer outer, String message) {
      this.outer = outer;
      this.message = message;
    }

    public void print() {
      System.out.println(message);
    }
  }
}
```

2.5.6.6 反转布尔值

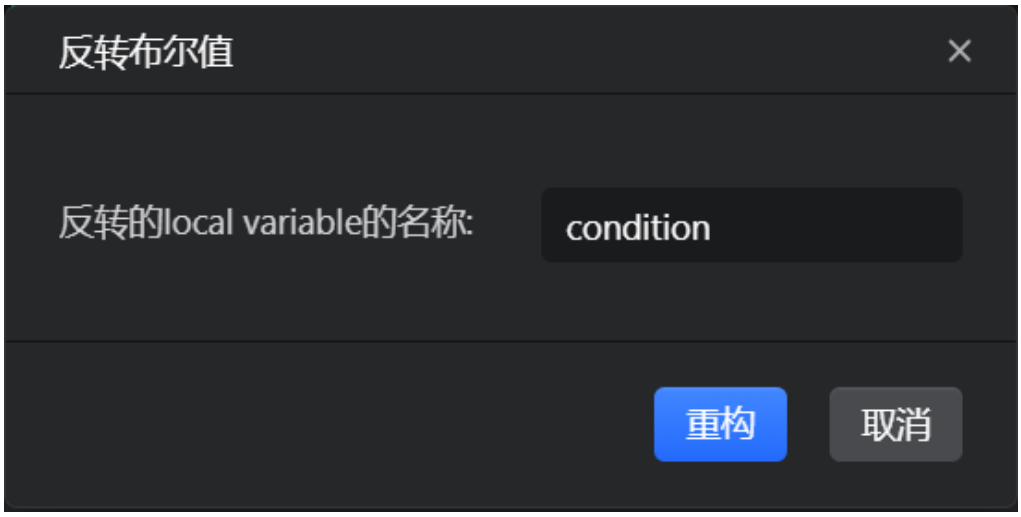
通过此重构，用户可以反转布尔变量的值或方法的返回值。

执行重构

步骤1 在代码编辑器中，将光标放置在布尔变量或具有布尔返回值的方法的声明上。

步骤2 在编辑器上下文菜单中，选择“重构 > 反转布尔值”。

步骤3 在打开的“反转布尔值”对话框中，为反转变量或方法提供新名称。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，反转**condition**变量和**checkCondition**方法的值。

重构前

```
class Invert {
    private static double a;
    public static void main(String[] args) {
        boolean condition = true;
        if (condition) {
            System.out.println("Hello World!");
        }
    }
    public static boolean checkCondition() {
        if (a > 15 && a < 100) {
            a = 5;
            return true;
        }
        return false;
    }
}
```

重构后

```
class Invert {
    private static double a;
    public static void main(String[] args) {
```

```
        boolean condition = false;
        if (!condition) {
            System.out.println("Hello World!");
        }
    }
    public static boolean checkCondition() {
        if (a > 15 && a < 100) {
            a = 5;
            return false;
        }
        return true;
    }
}
```

2.5.6.7 用委托替换继承

通过这种重构，用户可以从继承层次结构中删除类，同时保留父类的功能。在重构过程中，会创建一个私有内部类来继承以前的超类或接口。通过新创建的内部类调用父类的选定方法。

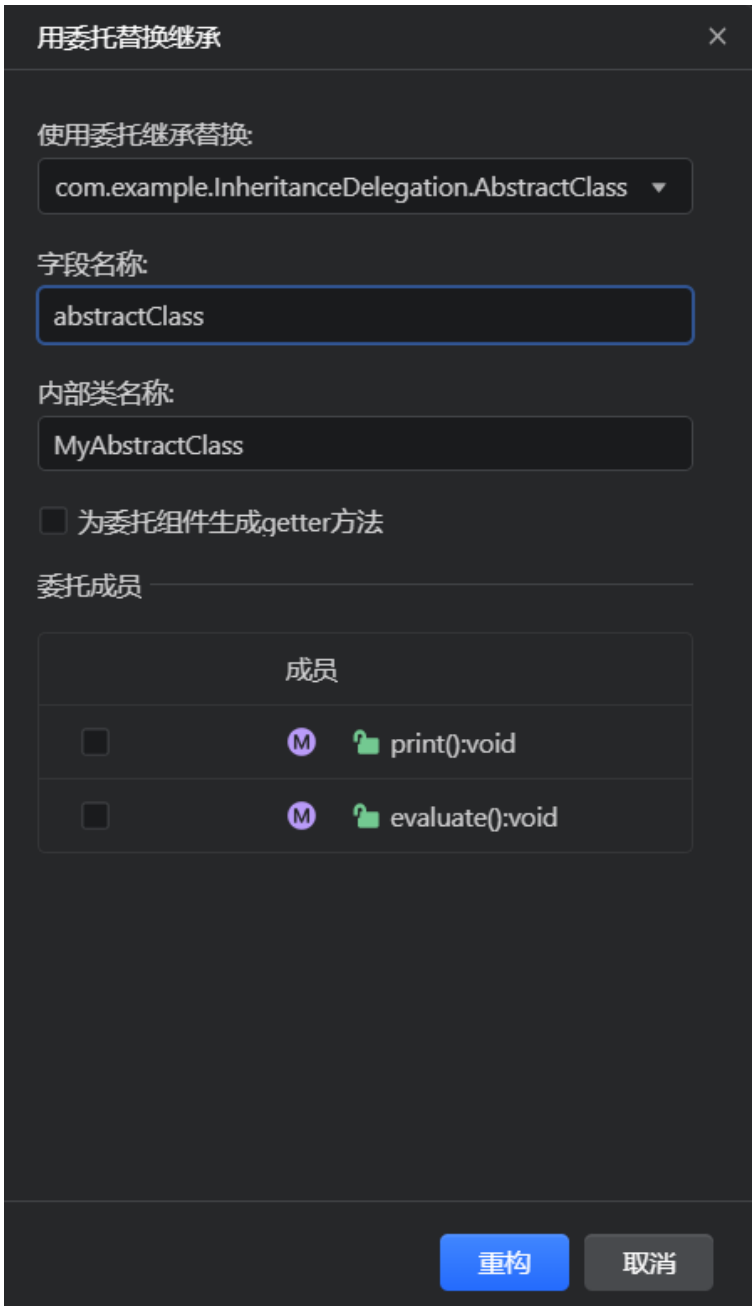
执行重构

步骤1 在代码编辑器中，选择要重构的类，并将光标放置在要从其继承层次结构中删除继承的类中。

步骤2 在编辑器上下文菜单中，选择“重构 > 用委托替换继承”。

步骤3 在打开的“用委托替换继承”对话框中，选择类，从中的继承将通过内部类替换为委托。提供重构选项：

- 在**“字段名称”**字段中，指定新创建的内部类的字段名称。
- 在**“内部类名称”**字段中，指定新创建的内部类的名称。
- 选中为**“委托组件生成Getter方法”**复选框，为新创建的内部类创建getter方法。
- 在**“委托成员”**选项中，选择要通过新创建的内部类委托的父类的成员。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，从**AbstractClass**中删除**InnerClass**的继承。因此，将创建一个新的内部**MyAbstractClass**类，并通过**MyAbstractClass**调用**print**和**evaluate**方法。

重构前

```
class InheritanceDelegation {

    public static void main(String[] args) {
        InnerClass innerClass = new InnerClass();
```

```
        print(innerClass);
    }

    private static void print(InnerClass innerClass) {
        innerClass.print();
    }

    private static class InnerClass extends AbstractClass {
        public void evaluate() {
        }
    }

    private abstract static class AbstractClass {
        public void print() {
            System.out.println("Hello World!");
        }

        public abstract void evaluate();
    }
}
```

重构后

```
class InheritanceDelegation {

    public static void main(String[] args) {
        InnerClass innerClass = new InnerClass();
        print(innerClass);
    }

    private static void print(InnerClass innerClass) {
        innerClass.print();
    }

    private static class InnerClass {
        private final MyAbstractClass abstractClass = new MyAbstractClass();

        public AbstractClass getAbstractClass() {
            return abstractClass;
        }

        public void print() {
            abstractClass.print();
        }

        public void evaluate() {
            abstractClass.evaluate();
        }

        private class MyAbstractClass extends AbstractClass {
            public void evaluate() {
            }
        }
    }

    private abstract static class AbstractClass {
        public void print() {
            System.out.println("Hello World!");
        }

        public abstract void evaluate();
    }
}
```

2.5.6.8 用工厂方法替换构造函数

此重构允许用户用返回类实例的工厂方法替换类构造函数。

执行重构

步骤1 在代码编辑器中，将光标放置在要用工厂方法替换的类构造函数上。

步骤2 在编辑器上下文菜单中，选择“重构 > 用工厂方法替换构造函数”。

步骤3 在打开的“用工厂方法替换构造函数”对话框中，提供要创建的工厂方法的名称及其包含类。

步骤4 单击“**重构**”以应用重构。

----结束

示例

作为示例，将**InnerClass**类构造函数替换为带有**newInnerClass**的工厂方法。

重构前

```
class ReplaceConstructor {
  public static void main(String[] args) {
    new InnerClass("Hello", "World").print();
  }

  private static class InnerClass {
    private String message;

    public InnerClass(String hello, String world) {
      message = hello + ", " + world;
    }

    public void print() {
      System.out.println(message);
    }
  }
}
```

重构后

```
class ReplaceConstructor {
  public static void main(String[] args) {
    InnerClass.newInnerClass("Hello", "World").print();
  }

  private static class InnerClass {
    private String message;

    private InnerClass(String hello, String world) {
      message = hello + ", " + world;
    }

    public static InnerClass newInnerClass(String hello, String world) {
      return new InnerClass(hello, world);
    }

    public void print() {
      System.out.println(message);
    }
  }
}
```

2.5.6.9 用构建器替换构造函数

通过此重构，用户可以将类构造函数的用法替换为对构建器类的引用。

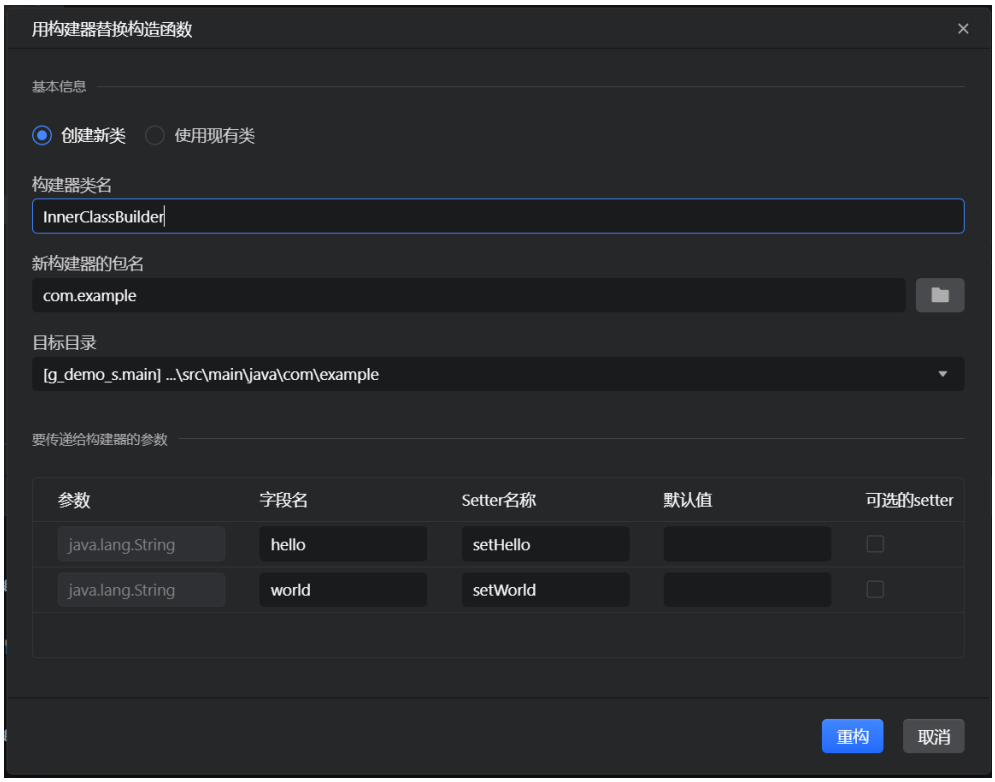
执行重构

步骤1 在代码编辑器中，将光标放置在要将其调用替换为对构建器类的引用的类构造器的声明上。

步骤2 在编辑器上下文菜单中，选择“重构 > 用构建器替换构造函数”。

步骤3 在打开的“用构建器替换构造函数”对话框中，提供重构参数。

- 选择是创建新的生成器类还是使用现有的生成器类。
- 提供生成器类名，如果创建新类，则提供其包和目标目录。
- 在**“要传递给构建器的参数”**选项，配置作为生成器类字段传递的构造函数参数。如果指定的字段的默认值与作为构造函数参数传递的值匹配，则可以选中**可选的Setter**复选框跳过此类参数的setter方法。否则，如果未选中复选框，则始终调用字段设置器。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，用对新创建的**InnerClass**构建器类的引用替换**InnerClassBuilder**调用。请注意，由于将“**Hello**”和“**World**”作为**Hello**和**World**生成器类字段的默认值，因此可以使用**可选的Setter**选项，以便在**InnerClassBuilder**调用中跳过对**setHello**和**setWorld**的调用。

重构前

```
class ReplaceConstructor {
  public static void main(String[] args) {
    new InnerClass("Hello", "World").print();
  }

  private static class InnerClass {
    private String message;

    public InnerClass(String hello, String world) {
      message = hello + ", " + world;
    }

    public void print() {
      System.out.println(message);
    }
  }
}
```

重构后

```
class ReplaceConstructor {
  public static void main(String[] args) {
```

```
    new InnerClassBuilder().createInnerClass().print();
  }

  static class InnerClass {
    private String message;

    public InnerClass(String hello, String world) {
      message = hello + ", " + world;
    }

    public void print() {
      System.out.println(message);
    }
  }
}

public class InnerClassBuilder {
  private String hello = "Hello";
  private String world = "World";

  public InnerClassBuilder setHello(String hello) {
    this.hello = hello;
    return this;
  }

  public InnerClassBuilder setWorld(String world) {
    this.world = world;
    return this;
  }

  public ReplaceConstructor.InnerClass createInnerClass() {
    return new ReplaceConstructor.InnerClass(hello, world);
  }
}
```

2.5.6.10 封装字段

此重构允许用户限制类字段的可见性，并提供用于访问它们的getter和setter方法。

执行重构

步骤1 在代码编辑器中，将光标放置在要封装字段的声明上。

步骤2 在编辑器上下文菜单中，选择“重构 > 封装字段”。

步骤3 在打开的“封装字段”对话框中，提供重构选项。

- 选择要封装的字段，并（可选）自定义getter和setter方法的名称。
- 在封装区域中，选择是同时创建或单独创建getter、setter。选中**“即使字段可访问，也使用访问器”**复选框，以将所有字段引用替换为对相应访问器方法的调用。
- 指定封装字段和访问器的可见性。

步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，封装**name**变量，并为其生成getter和setter方法。

重构前

```
class Person {
    public String name;

    public static void main(String[] args) {
        Person person = new Person();
        person.name = "John";
        System.out.println(person.name);
    }
}
```

重构后

```
class Person {
    private String name;
```

```
    public static void main(String[] args) {
        Person person = new Person();
        person.setName("John");
        System.out.println(person.getName());
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

2.5.6.11 更改方法签名

此重构允许用户修改方法：重命名方法、添加异常以及添加、删除、重新排序和重命名方法的参数。

执行重构

步骤1 在代码编辑器中，将光标放置在要更改其签名的方法的声明上。

步骤2 在编辑器上下文菜单中，选择“重构 > 更改方法签名”或按“Ctrl+F6”。

步骤3 在打开的“更改方法签名”对话框中，提供重构选项。

- 指定方法的可见性、名称和返回类型。
- 在**“参数”**选项卡上，配置方法的参数：指定参数的名称和类型，并使用工具栏按钮添加、删除和重新排序参数。

  说明

  目前CodeArts IDE不支持为参数提供默认值。如果添加参数，则可能需要手动更新方法调用。
- 在**“抛出”**选项卡上，配置方法抛出的异常列表。使用工具栏按钮添加、删除和重新排序功能例外。
- 在**“方法调用”**区域中，选择是修改现有方法调用还是保持原样，还是通过新创建的重载方法进行委托。

步骤4 单击“**重构**”以应用重构。

----结束

示例

作为示例，对**myMethod**方法通过添加参数**price**来更改方法的签名，使方法抛出异常**Exception**，并通过重载方法委托它。

重构前

```
public class Example {
  public void myMethod(int value) {
  }

  public class AntherClass {
    public void myMethodCall(Example example) {
      example.myMethod(1);
    }
  }
}
```

重构后

```
public class Example {
  public void myMethod(int value) {
    myMethod(value);
  }

  public void myMethod(int value, double price) throws Exception {}

  public class AntherClass {
    public void myMethodCall(Example example) {
      example.myMethod(1);
    }
  }
}
```

2.5.6.12 更改类签名

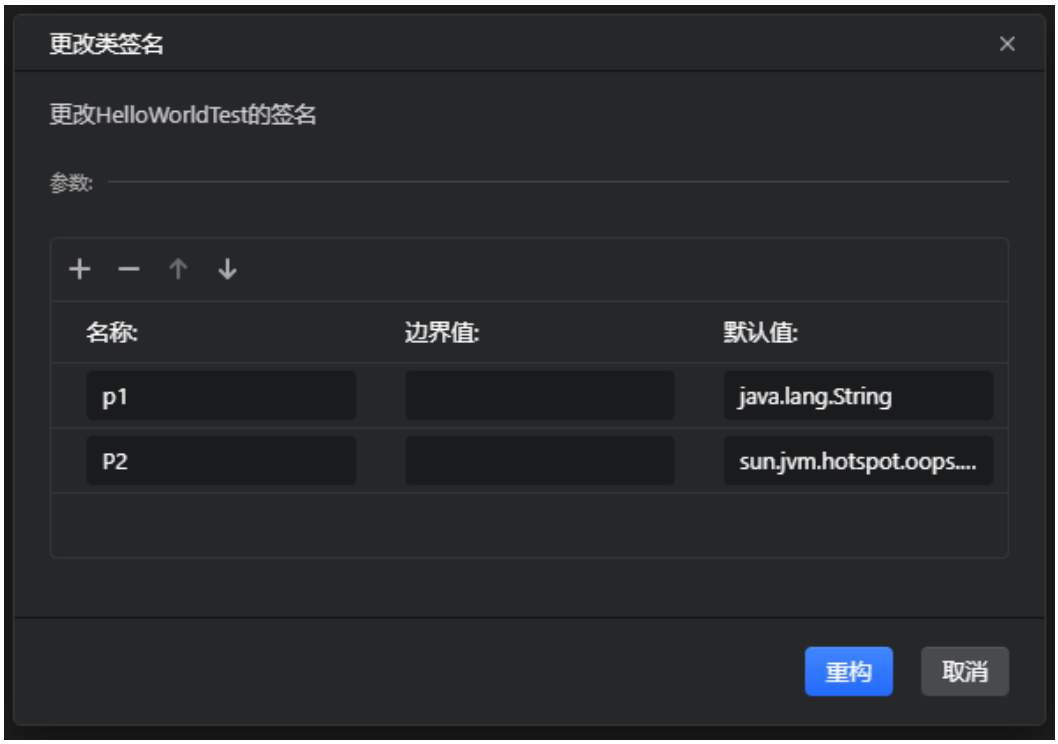
此重构允许用户将类转换为泛型并操作其类型参数。

执行重构

步骤1 在代码编辑器中，将光标放置在要更改其签名的类的声明上。

步骤2 在编辑器上下文菜单中，选择“重构 > 更改类签名”或按“Ctrl+F6”。

步骤3 在打开的“更改类签名”对话框中，配置类参数。使用工具栏按钮添加、删除和重新排序参数。对于每个参数，指定其名称和默认类型。在**“边界值”**字段中，可以选择提供一个有界值，以限制传递给类型参数的值。



步骤4 单击**“重构”**以应用重构。

----结束

示例

例如，通过添加三个类型参数来更改类**ChangeClassSignature**的签名：**P1(String)**，**P2(Integer)**，和**P3(LinkedList)**，其边界为**List**。

重构前

```
class ChangeClassSignature {

   public class MyOtherClass {
      ChangeClassSignature myClass;

      void myMethod(ChangeClassSignature myClass) {
      }
   }
}
```

重构后

```
class ChangeClassSignature<P1, P2, P3 extends List> {

   public class MyOtherClass {
      ChangeClassSignature<String, Integer, LinkedList> myClass;

      void myMethod(ChangeClassSignature<String, Integer, LinkedList> myClass) {
      }
   }
}
```

2.5.6.13 将匿名类转换为内部类

此重构允许用户将匿名类转换为重命名的内部类。

执行重构

步骤1 在编辑器中，将光标放置在要转换为内部类的匿名类表达式中的任何位置。

步骤2 在编辑器上下文菜单中，选择“重构 > 转换匿名类为内部类”。

步骤3 在打开的“转换匿名类为内部类”对话框中，提供重构参数。

- 提供内部类的名称并选择是否将其创建为静态类。
- 在**“构造函数参数”**区域中，提供要用作类构造函数参数的变量的名称。使用工具栏按钮对参数重新排序。

步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，将用作**方法**返回值的匿名类表达式转换为内部静态类**MyTestClass**。

重构前

```
class AnonymousToInner {
    public TestClass method() {
        final int i = 0;
        final String str = "string";
        return new TestClass() {
            public String str () {
                return str;
            }
            public int publicMethod() {
                return i;
            }
        };
    }

}

interface TestClass {
}
```

重构后

```
class AnonymousToInner {
  public TestClass method() {
    final int i = 0;
    final String str = "string";
    return new MyTestClass(str, i);
  }

  private static class MyTestClass extends TestClass {
    private final String str;
    private final int i;

    public MyTestClass(String str, int i) {
      this.str = str;
      this.i = i;
    }

    public String str () {
      return str;
    }

    public int publicMethod() {
      return i;
    }
  }
}
interface TestClass {
}
```

2.5.6.14 尽可能使用 Interface

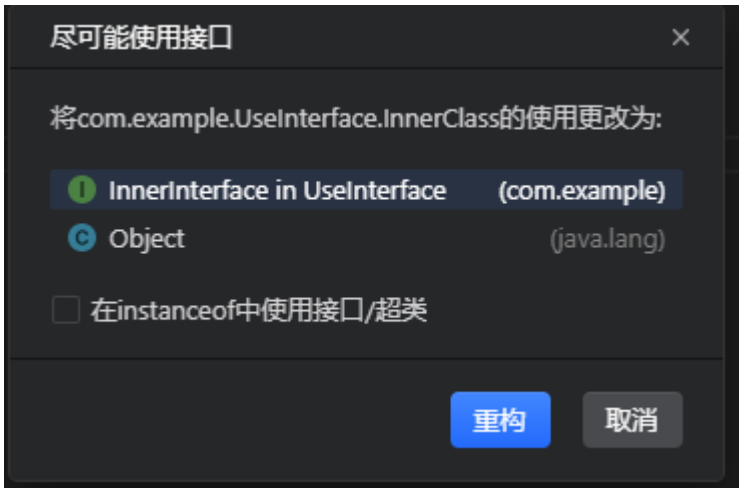
此重构允许用户将从基类/接口派生的指定方法的执行委托给实现同一接口的父类或内部类的实例。

执行重构

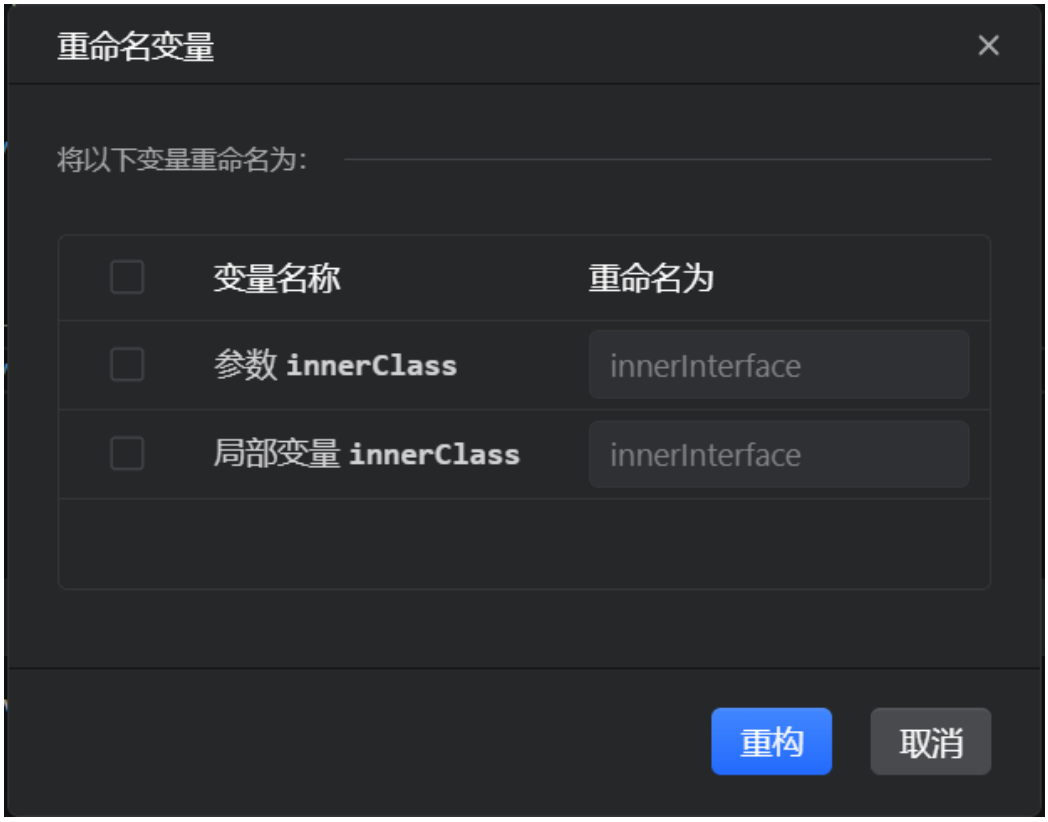
步骤1 在代码编辑器中，将光标放在应通过父类/接口委托其方法的类的声明上。

步骤2 在编辑器上下文菜单中，选择“重构 > 尽可能使用接口”。

步骤3 在打开的“尽可能使用接口”对话框中，选择应替换当前类用法的父类/接口。想要同时替换当前类在instanceof语句中的用法，请选中在**“在instanceof中使用接口/超类”**复选框。



CodeArts IDE将自动重命名变量，以匹配重构引入的更改。如有必要，请在“重命名变量”对话框中提供替代名称。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，将**print**方法的使用从类**InnerClass**委托给它实现的接口**InnerInterface**。

重构前

```
class UseInterface {
```

```
    public static void main(String[] args) {
        InnerClass innerClass = new InnerClass();
        print(innerClass);
    }

    private static void print(InnerClass innerClass) {
        innerClass.print();
    }

    private static class InnerClass implements InnerInterface {
        @Override
        public void print() {
            System.out.println("Hello World!");
        }
    }

    private static interface InnerInterface{
        void print();
    }
}
```

重构后

```
class UseInterface {

    public static void main(String[] args) {
        InnerInterface innerInterface = new InnerClass();
        print(innerInterface);
    }

    private static void print(InnerInterface innerInterface) {
        innerInterface.print();
    }

    private static class InnerClass implements InnerInterface {
        @Override
        public void print() {
            System.out.println("Hello World!");
        }
    }

    private static interface InnerInterface{
        void print();
    }
}
```

2.5.6.15 类型迁移

此重构允许用户更改类成员、局部变量、参数、方法返回值等类型。用户还可以在数组、集合之间转换变量或方法返回值的类型。

执行重构

步骤1 在代码编辑器中，将光标放置在要迁移的类型上。

步骤2 在编辑器上下文菜单中，选择“重构 > 类型迁移”。

步骤3 在打开的“类型迁移”对话框中，提供要迁移到的类型。在**“选择可能发生变更签名的范围”**列表中，指定用于查找使用实例的新类型和范围：整个项目或仅根目录文件（即不包括库和SDK）。

步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，将**myResult**字段的类型从集合**ArrayList<String>**迁移到数组**String[]**。

重构前

```
class TypeMigration {

    private ArrayList<String> myResult;

    public String[] getResult() {
        return myResult.toArray(new String[myResult.size()]);
    }
}
```

重构后

```
class TypeMigration {

    private String[] myResult;

    public String[] getResult() {
        return myResult;
    }
}
```

2.5.6.16 将原始类型转换为泛型

此重构允许用户为每个原始类型创建安全且一致的参数类型，将不使用的泛型代码转换为泛型感知代码。

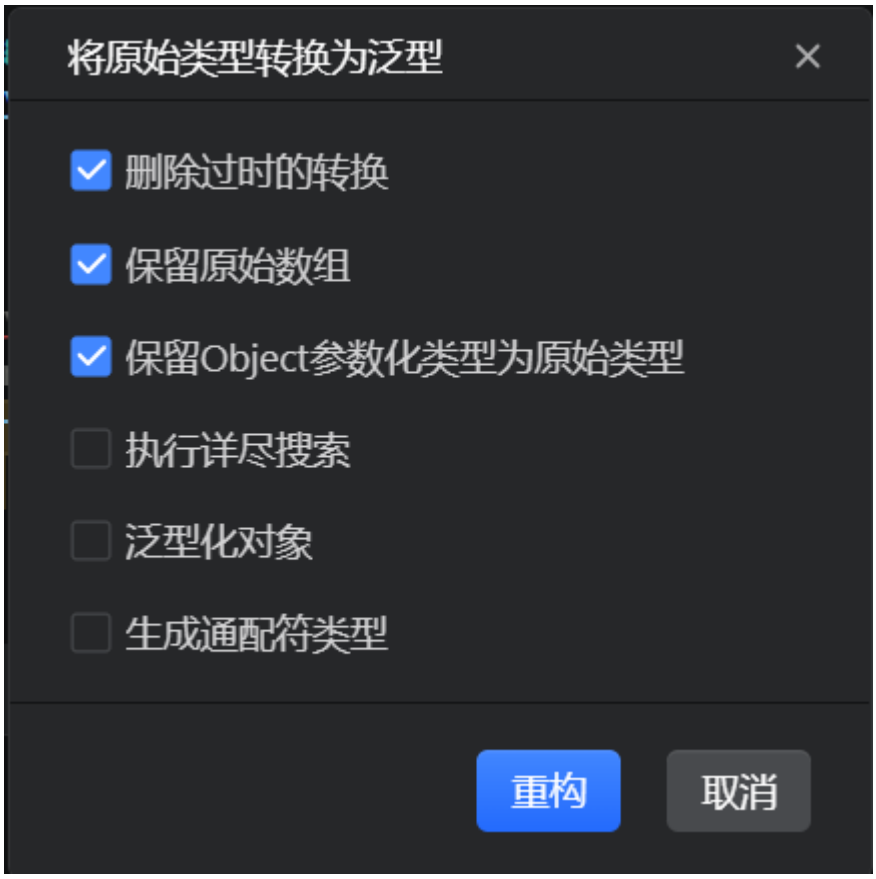
执行重构

步骤1 选择要应用重构的实体（资源管理器中的文件或文件夹）。

步骤2 在编辑器上下文菜单中，选择“重构 > 将原始类型转换为泛型”。

步骤3 在打开的“将原始类型转换为泛型”对话框中，提供重构选项。

- **“删除过时的转换”**：如果选中，CodeArts IDE将分析参数强制转换案例是否会被重构而更改。如果生成的参数类型与过期的参数类型相似，则将删除强制转换语句。
- **“保留原始数组”**：如果选中，数组不会更改为具有参数化类型的数组。否则，数组将转换为参数化类型。清除此复选框可能会有风险，并导致无法编译的代码。
- **“保留Object参数化类型为原始类型”**： 如果选择了具有**java.lang.Object**作为参数的对象，它们将被设置为原始类型。
- **“执行详尽搜索”**： 如果选中，则在所有节点上执行搜索。
- **“泛型化对象”**： 如果选中，**java.lang.Object**对象将转换为它们实际使用的类型。
- **“生成通配符类型”**：如果选择此选项，则尽可能生成通配符类型（即**List<? extends String>**等表达式）。



步骤4 单击**“重构”**以应用重构。

----结束

示例

例如，生成**List**和**LinkedList**类型。

重构前

```
public class ConvertTypes {
    public void method() {
```

```
        List list = new LinkedList();
        list.add("string");
    }
}
```

重构后

```
public class ConvertTypes {
    public void method() {
        List<String> list = new LinkedList<>();
        list.add("string");
    }
}
```

2.5.6.17 转换为实例方法

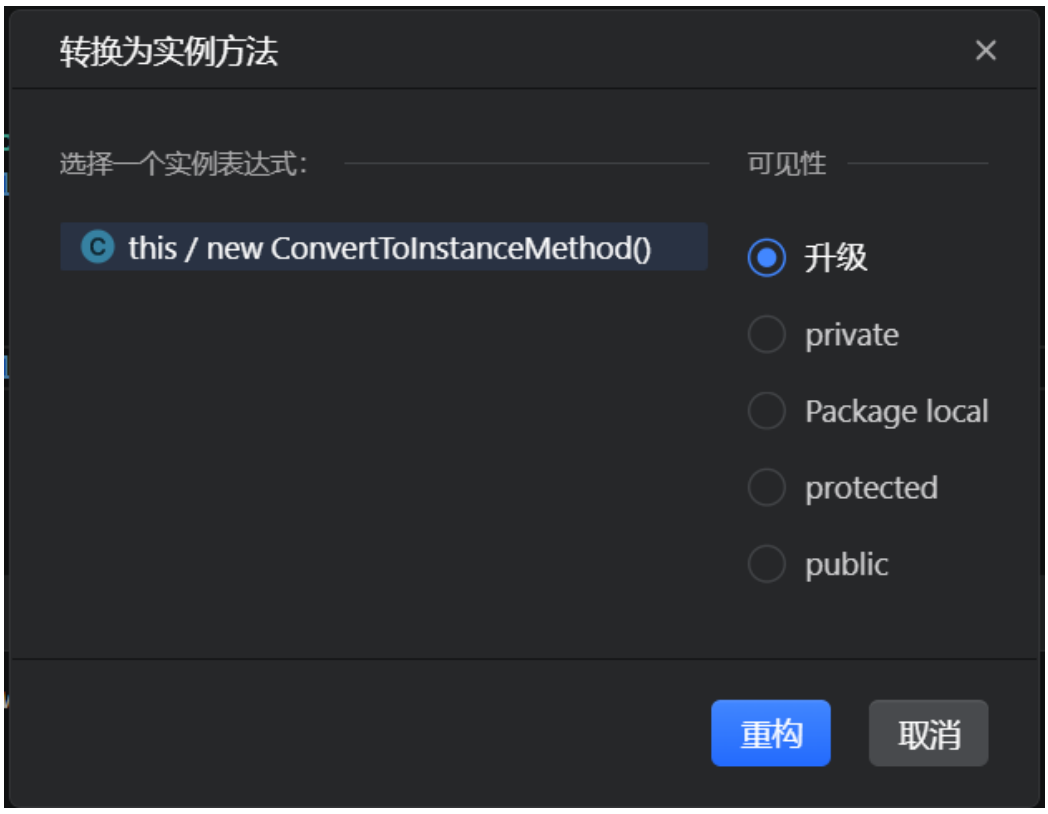
此重构允许用户将类的静态方法转换为类实例的非静态方法。

执行重构

步骤1 在代码编辑器中，将光标放置在要转换为实例方法的静态方法的声明上。

步骤2 在编辑器上下文菜单中，选择“重构 > 转换为实例方法”。

步骤3 在打开的“转换为实例方法”对话框中，选择方法所属的类。方法中该类的所有用法都将替换为this。如有必要，请更改转换方法的可见性修饰符。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，将静态方法**sayHello**转换为实例方法。

重构前

```
class ConvertToInstanceMethod {
    public static void main(String[] args) {
        sayHello();
    }

    public static void sayHello() {
        System.out.println("Hello World");
    }
}
```

重构后

```
class ConvertToInstanceMethod {
    public static void main(String[] args) {
        new ConvertToInstanceMethod().sayHello();
    }

    public void sayHello() {
        System.out.println("Hello World");
    }
}
```

2.5.6.18 移除中间人

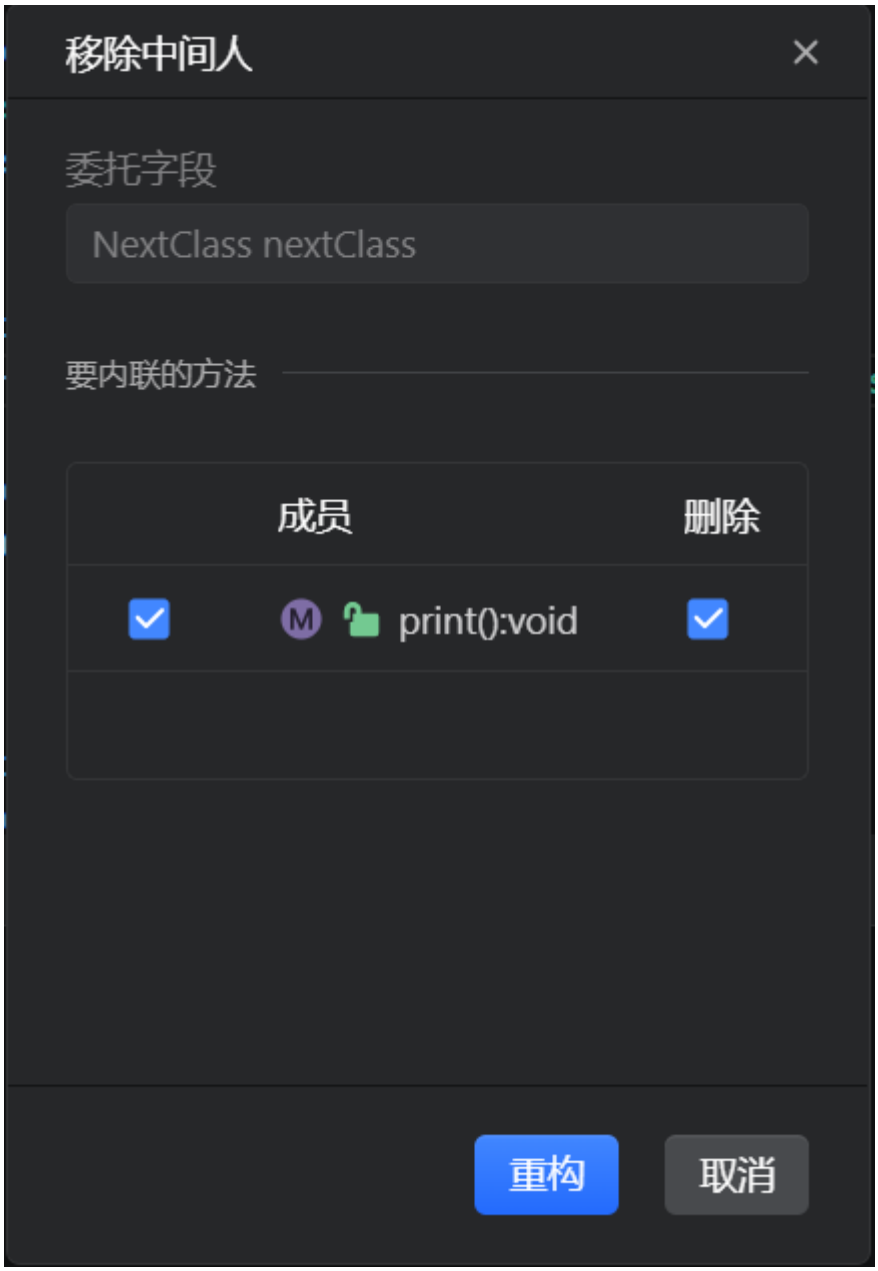
通过此重构，用户可以将对类中的委托方法的调用替换为直接对委托字段的等效调用。用户还可以删除委托方法，这些方法在重构后将不再使用。

执行重构

步骤1 在代码编辑器中，将光标放置在其声明中委托字段的名称上。

步骤2 在编辑器上下文菜单中，选择“重构 > 移除中间人”。

步骤3 在打开的“移除中间人”对话框中，选择要内联的委托方法。要删除方法，请选中该方法旁边的复选框。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，从类**InnerClass**中删除**print**委托方法，将其替换为对委托字段**nextClass**的调用。

重构前

```
class Middleman {

    public static void main(String[] args) {
        InnerClass innerClass = new InnerClass();
        innerClass.print();
    }
```

```
    private static class InnerClass {
        private final NextClass nextClass = new NextClass();

        public void print() {
            nextClass.print();
        }
    }

    private static class NextClass {
        public void print() {
            System.out.println("Hello World!");
        }
    }
}
```

重构后

```
class Middleman {

    public static void main(String[] args) {
        InnerClass innerClass = new InnerClass();
        innerClass.getNextClass().print();
    }

    private static class InnerClass {
        private final NextClass nextClass = new NextClass();

        public NextClass getNextClass() {
            return nextClass;
        }
    }

    private static class NextClass {
        public void print() {
            System.out.println("Hello World!");
        }
    }
}
```

2.5.6.19 安全删除

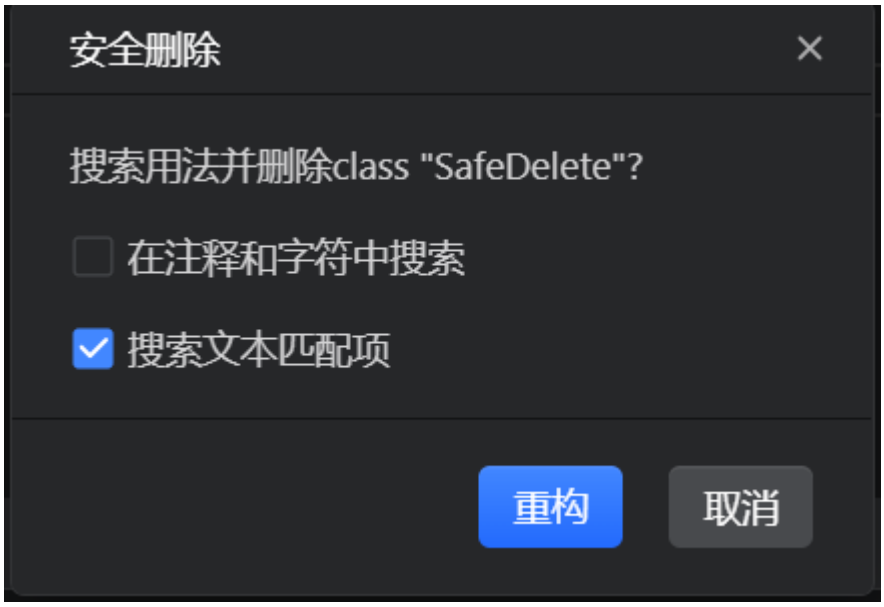
此重构允许用户安全地删除文件和代码符号。CodeArts IDE将验证受影响实体的所有用法，并允许用户相应地调整代码。

执行重构

步骤1 选择要应用重构的实体（资源管理器中的文件如下图所示或代码编辑器中的符号）。

步骤2 在编辑器上下文菜单中，选择“重构 > 安全删除”。

步骤3 在打开的“安全删除”对话框中，选择CodeArts IDE是否要搜索代码中被选定符号的引用。



步骤4 单击“**重构**”以应用重构。

----结束

示例

例如，在整个方法调用层次结构中删除未使用的参数i。

重构前

```
class SafeDelete {
    private void foo(int i) { bar(i);}
    private void bar(int i) { baz(i);}
    private void baz(int i) { }
}
```

重构后

```
class SafeDelete {
    private void foo() { bar();}
    private void bar() { baz();}
    private void baz() { }
}
```

2.5.7 导航代码

2.5.7.1 调用层次结构

“调用层次结构”视图显示了从某个方法到其他方法的所有调用，并允许用户深入到调用者和被调用者。要打开“调用层次结构”视图，可以在底部的活动栏中，单击“调用层次结构”视图。

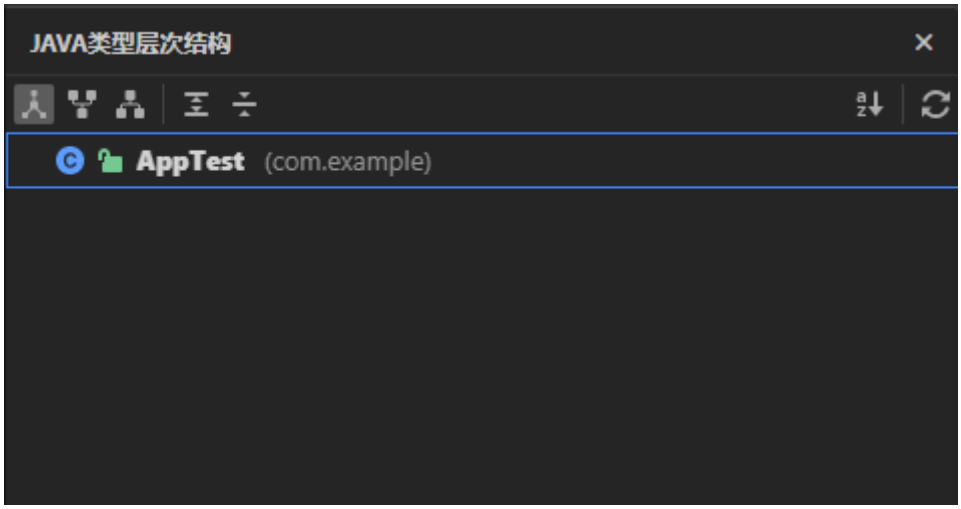
- 右键单击一个方法，选择“调用层次”，或按“**Shift+Alt+H**”或“**Ctrl+Alt+H**”（IDEA快捷键方案）。使用“**调用者方法层次结构**”（  ）和“**被调用方法层次结构**”（  ）工具栏按钮在调用者和被调用者列表之间切换。

在“调用层次结构”视图中，用户可以右键单击一个方法，从上下文菜单中选择“基于此项成员”重建层次结构，以基于所选方法重新构建层次结构。

2.5.7.2 类型层次结构

“类型层次结构”视图显示了继承关系，允许用户查看所选类的父类和子类。要打开“类型层次结构”视图，在右侧的活动栏中，单击“**类型层次结构**”视图。

- 右键单击一个类型，选择“类型层次”。



使用“类型层次结构”视图工具栏按钮在不同的查看模式之间切换。

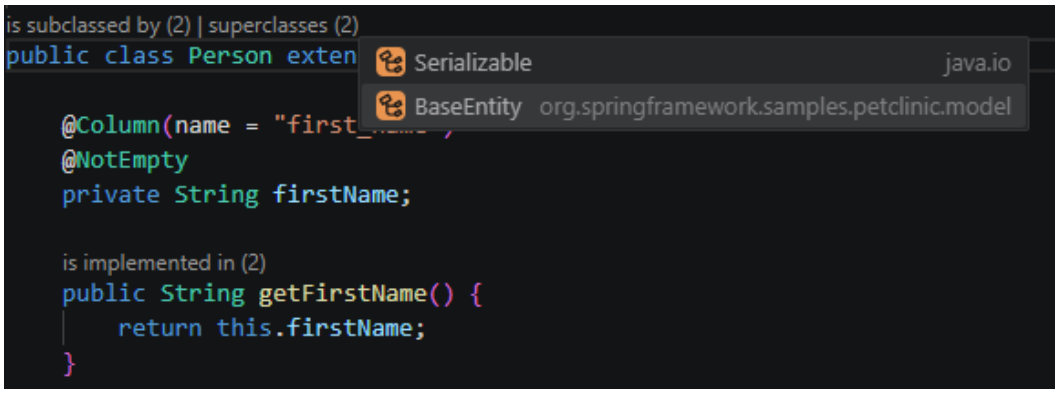
- 类层次结构  ：查看父类和子类。
- 超类型层次结构  ：查看父类。
- 子类层次结构  ：查看子类。

在“类型层次结构”视图中，用户可以右键单击一个类，并从上下文菜单中选择“基于此项类型”重建层次结构，以基于所选类重新构建层次结构。

2.5.7.3 CodeLens

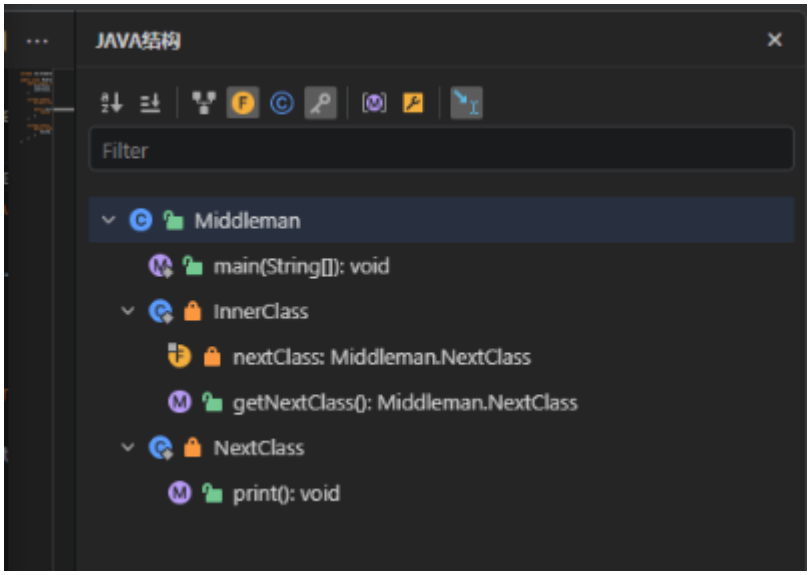
Java引用CodeLens显示当前类的超类/子类的内联计数，以及当前方法的重写。

单击CodeLens，在弹出的窗口中选择要导航到的实体。



2.5.7.4 结构

“结构”视图显示当前活动的**Java**文件的符号树，并提供了几种排序、分组和过滤功能。要打开“结构”视图，在右侧活动栏中单击**“结构”**视图。



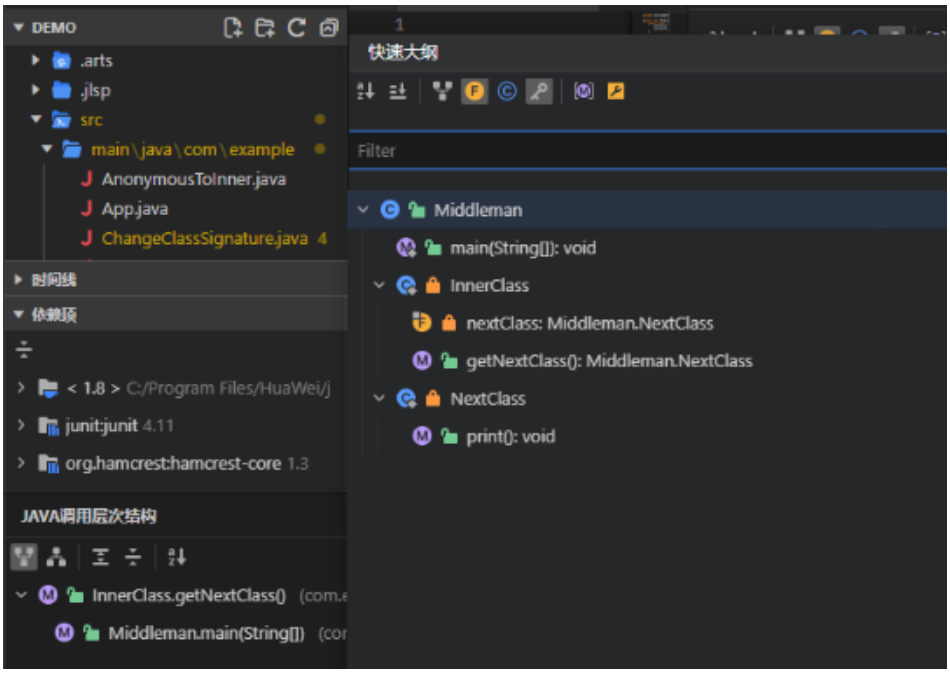
要快速导航到一个符号，在“结构”视图列表中单击相应的项。使用“结构”视图工具栏按钮对显示的符号进行排序、过滤和分组。

- : 单击后按字母顺序对列表进行排序。
- : 单击后按成员的可见性对列表进行排序。
- : 单击后显示继承的成员。
- : 单击后显示字段。
- : 单击后显示匿名类。

- : 单击后显示非公共类成员。
- : 单击后按它们所定义的类型对方法进行分组。
- : 单击后按属性分组方法。
- : 单击后追随光标。

说明

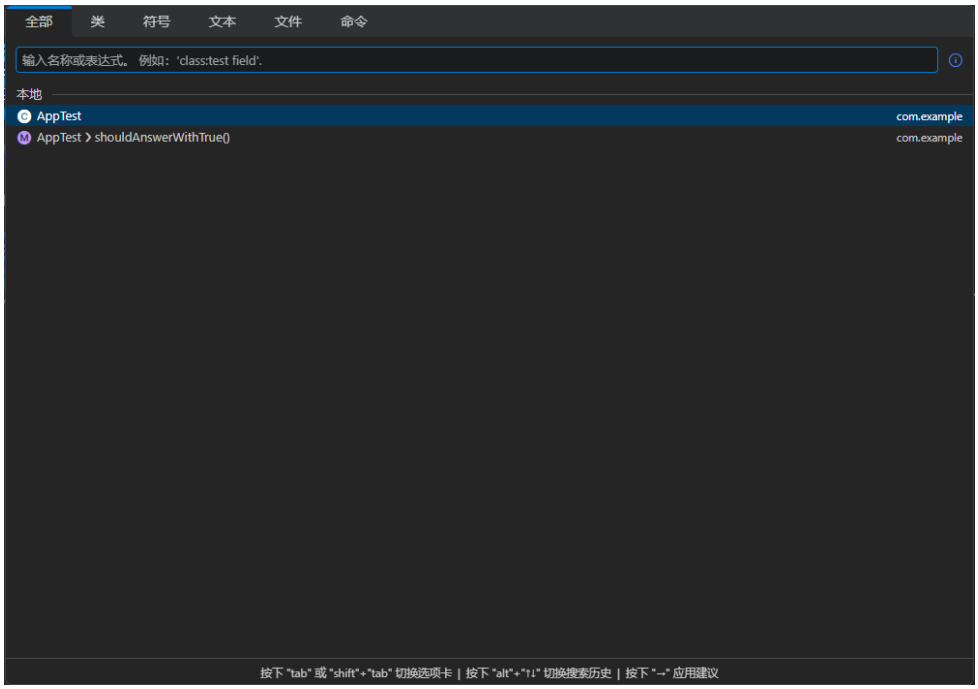
这个功能也可以通过“快速大纲”视图来实现，它会在当前编辑器中显示，这样用户就不需要切换上下文。要在“快速大纲”视图中查看文件结构，请在主菜单中“导航 > 快速大纲”，或按“**Ctrl+F10**”/“**Ctrl+F12**”。



2.5.8 通过代码搜索

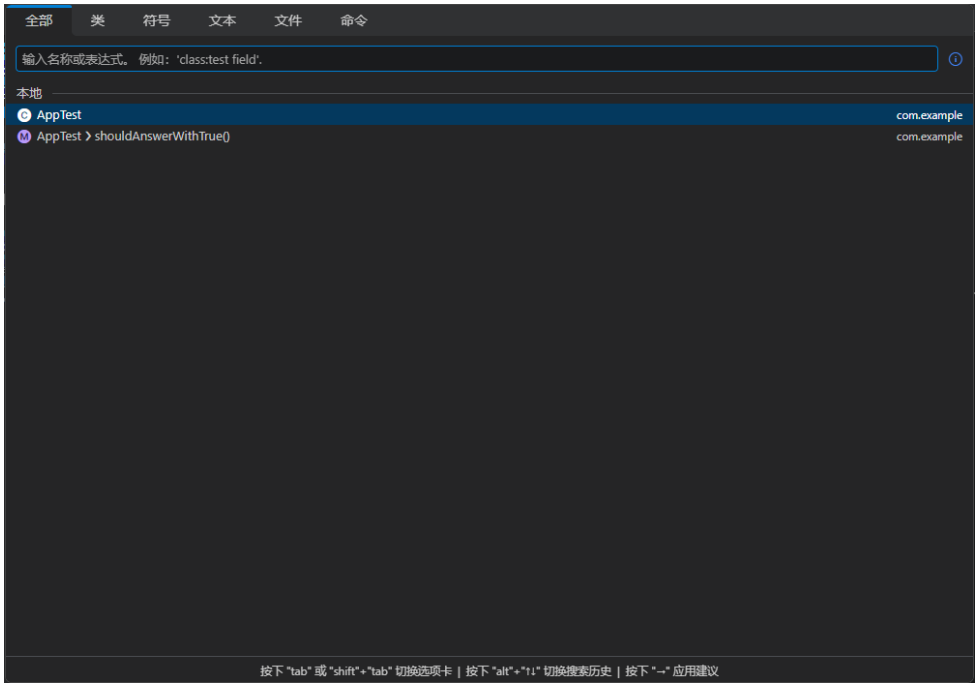
2.5.8.1 基本用法

步骤1 按下“Ctrl+Shift+A”/“Shift Shift”启动**“智能搜索”**窗口。

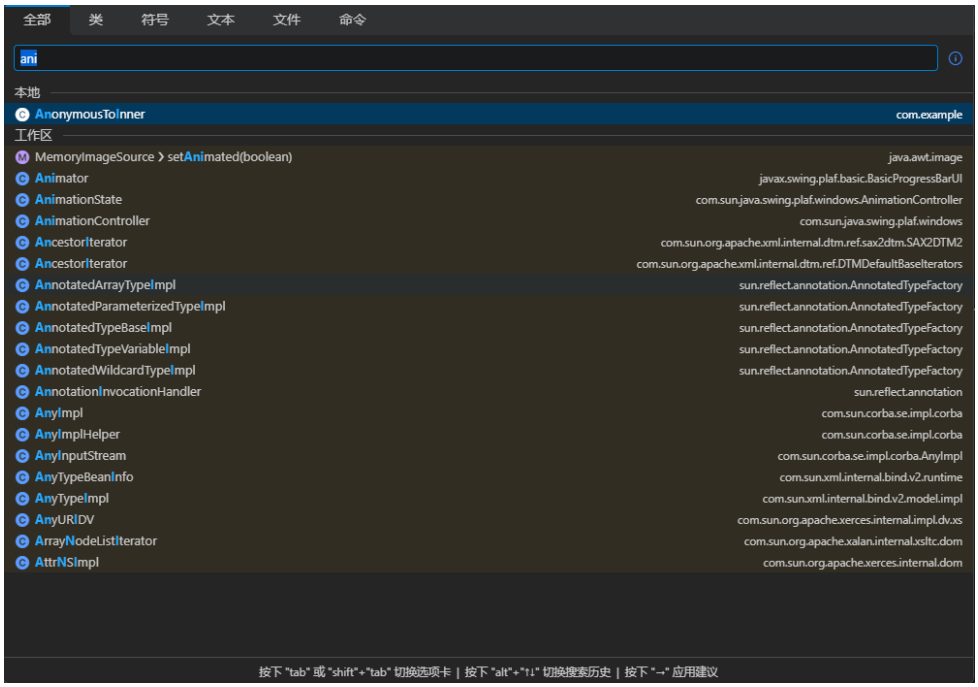


如果用户当前在代码编辑器中打开了一个Java文件，**“智能搜索”**窗口将自动显示其大纲，让用户在代码条目（例如类成员）之间导航。

步骤2 输入搜索请求。为了缩小搜索范围，可以在**“智能搜索”**窗口中切换选项卡，或使用**搜索查询语法**。



步骤3 在**“智能搜索”**窗口中，当前打开文件的结果显示在**“本地”**区域；项目中其他位置的结果显示在**“general”**区域。



使用光标键在条目之间导航，按“Enter”键转到相应位置或执行命令。或者双击所需的条目。要关闭**“智能搜索”**窗口，按“Escape”键，或者鼠标悬浮**“智能搜索”**窗口以外的区域。

----结束

2.5.8.2 搜索查询语法

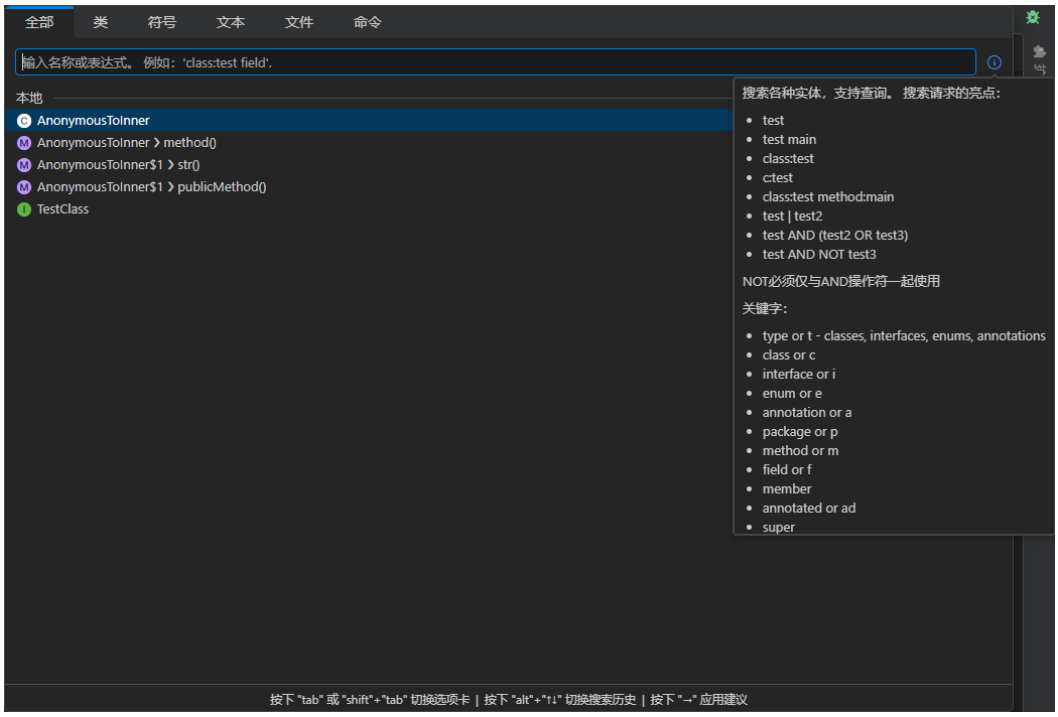
搜索查询是由**dataSource:stringToMatch**对组成的字符串，可以通过空格或运算符连接。如果查询中省略了**dataSource**，将在所有可用的数据源中进行搜索。也可以使用反向模式，即**stringToMatch:dataSource**。可用数据源见**表2-2**。

表 2-2 可用数据源列表

数据源名称	数据源简码	描述
local	l	当前文件实体。
class	c	类实体。
interface	i	接口实体。
enum	e	枚举实体。
annotation	a	注解实体。
annotated	ad	带注解实体。
method	m	方法实体。
field	f	字段实体。
super	--	超类/接口实体。
sub	--	子类/接口实体。

数据源名称	数据源简码	描述
type	--	类型化实体，即类、接口、枚举或注解实体。
member	--	成员实体，即类方法或类字段实体。
text	--	文本实体。请注意，只有文本文件会被文本搜索处理；jar文件会被忽略。
command	--	CodeArts IDE命令实体。

要快速了解“**智能搜索**”查询语法，请将鼠标悬浮在“**智能搜索**”窗口右上角的按钮。



搜索运算符

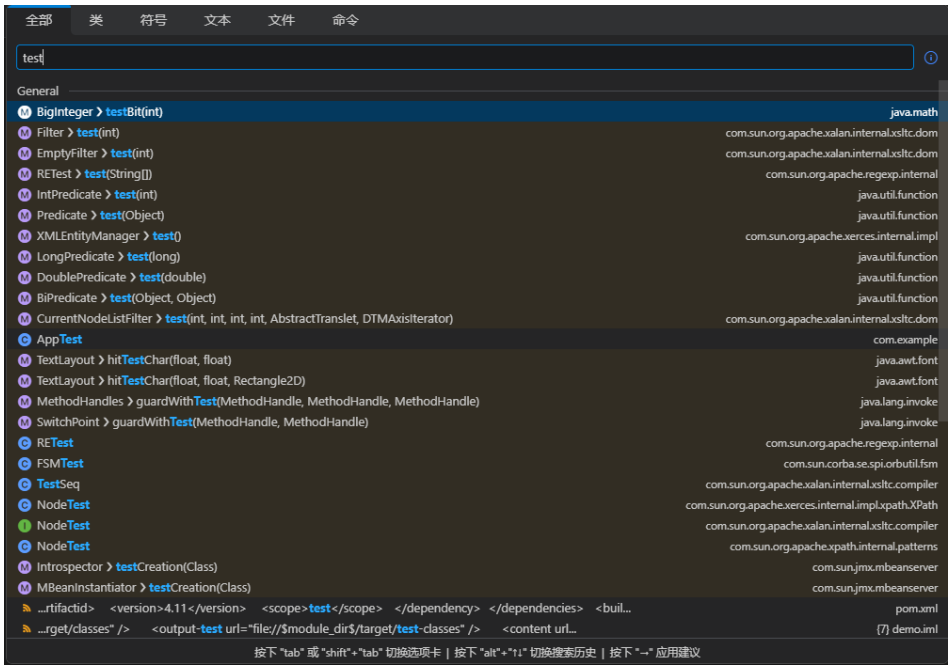
用户可以通过使用**AND**和**OR**运算符，或它们的组合，来构建复杂的搜索查询，例如**class:foo AND(method:bar OR method:baz)**。

运算符	语法	描述
AND	AND，&，&&，(space character)	**智能搜索**将定位与每个查询匹配的条目，并仅返回与彼此相关的条目。
OR	OR，|，||	**智能搜索**将返回与任何提供的查询匹配的所有条目。

2.5.8.3 定位示例

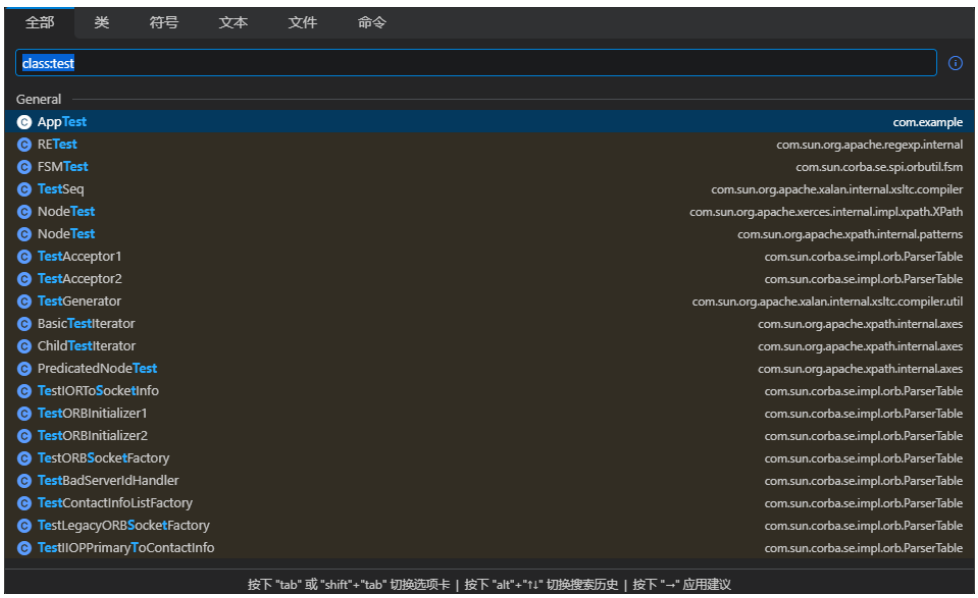
定位任意实体

一个搜索查询**test**匹配所有名称中带有**test**一词的实体。



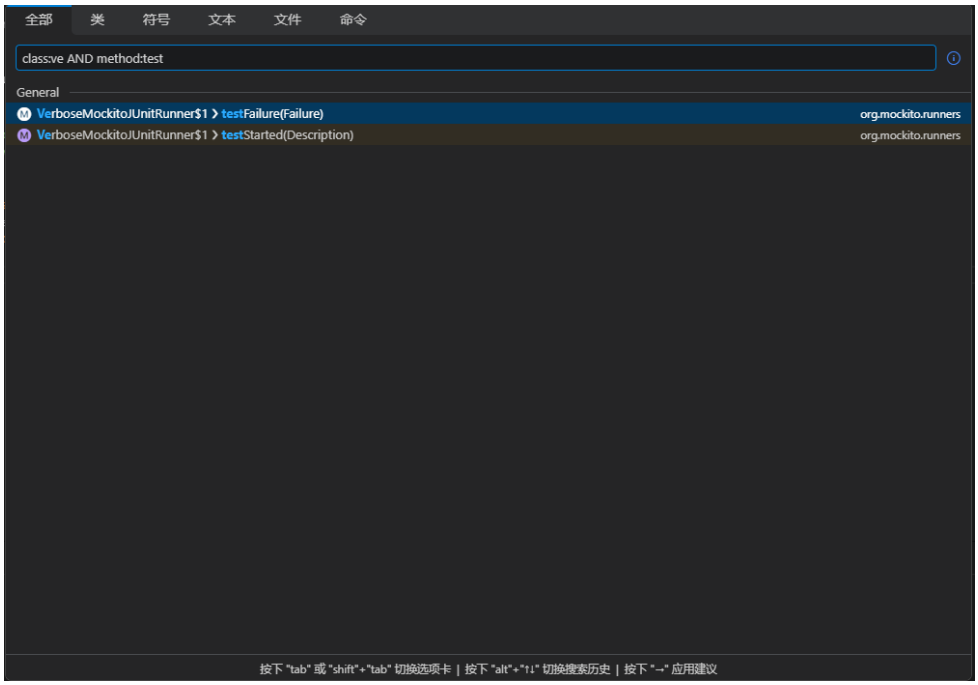
定位类

一个搜索查询**class:test**匹配所有名称中包含**test**的类。使用替代语法，这个查询也可以写作**c:test**、**test:class**或**test:c**。

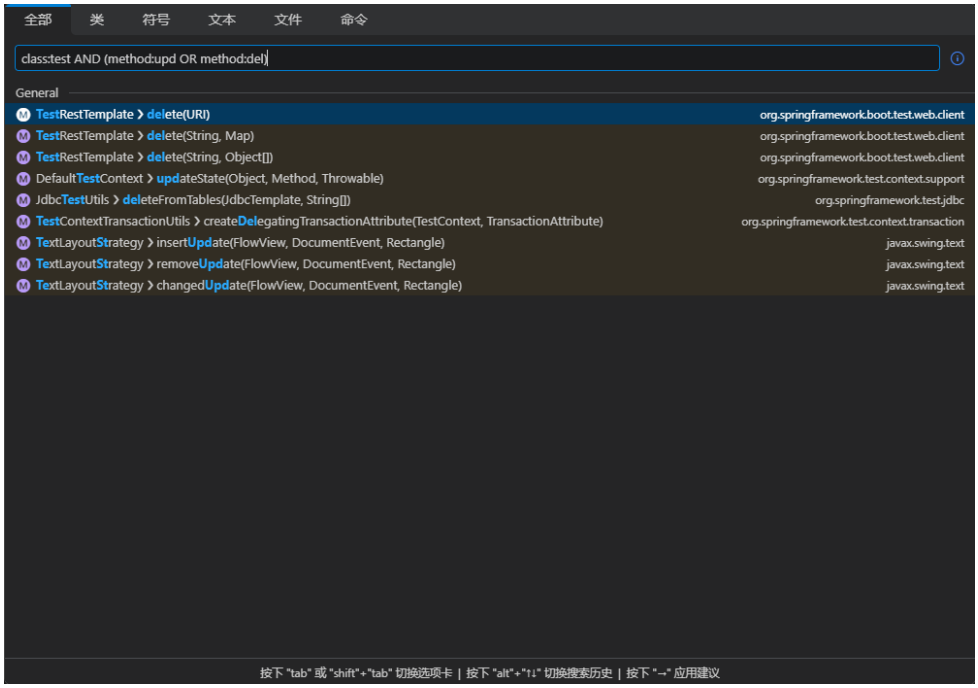


定位类中的方法

一个搜索查询**class:ve AND method:test**匹配所有名称中带有**test**的方法，并且属于名称中带有**ve**的类。



一个搜索查询**class:test AND (method:upd OR method:del)**匹配所有名称中带有**upd**或**del**的方法，并且属于名称中带有**test**的类。



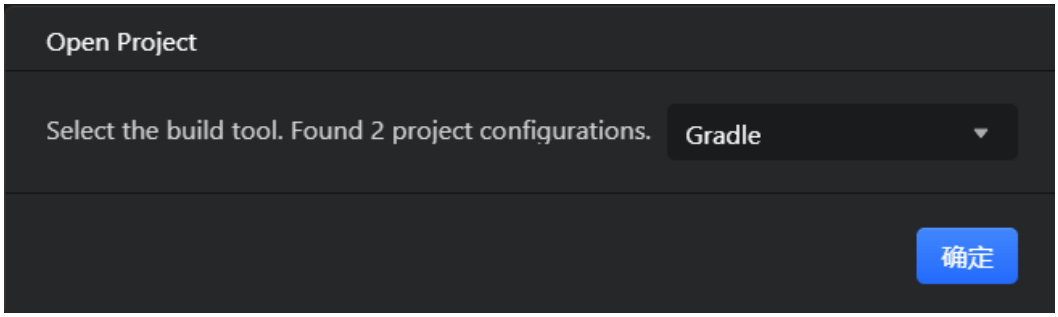
2.5.9 构建工具

2.5.9.1 简介

CodeArts IDE提供了Java构建工具的内置集成。开箱即用，支持以下构建工具：

- Gradle
- Maven

当用户首次打开包含Java源代码文件的文件夹时，CodeArts IDE会自动检测项目文件夹中的**pom.xml**或**build.gradle**文件，并加载相应的项目（Maven或Gradle）。如果同时存在这两个文件，CodeArts IDE会提示用户选择构建系统。



选择了构建工具后，CodeArts IDE会自动加载项目。

2.5.9.2 Gradle

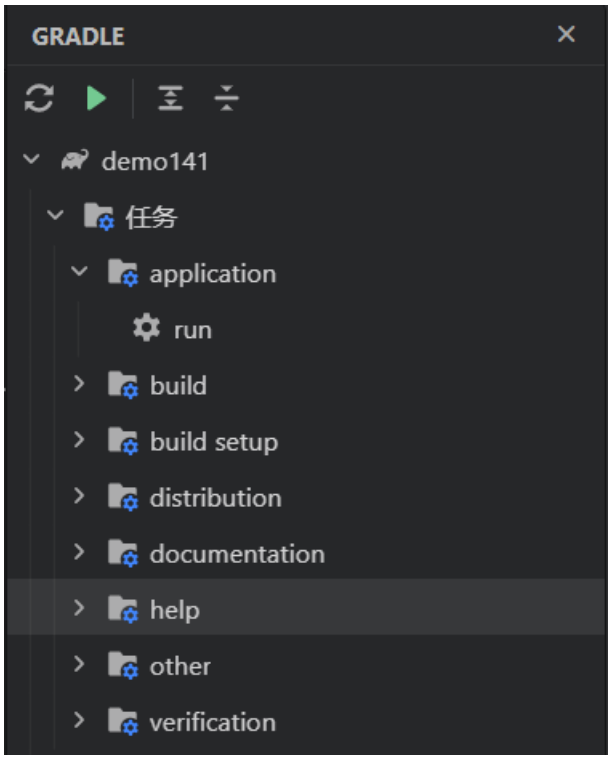
Gradle是一种用于管理Java项目和自动化应用程序构建的工具。CodeArts IDE提供了对Gradle Java项目的内置支持。专用的**Gradle**视图与build.gradle同步，并提供了一个可视化界面，用于查看项目依赖项或运行Gradle任务。

要打开**Gradle**视图，请在右侧的活动栏中选择**Gradle**视图。

在修改build.gradle文件后，例如添加新的依赖项，单击**Gradle**视图工具栏上的“**重新加载所有Gradle项目**”按钮（ ）以使CodeArts IDE应用用户的更改。

使用 Gradle 任务进行工作

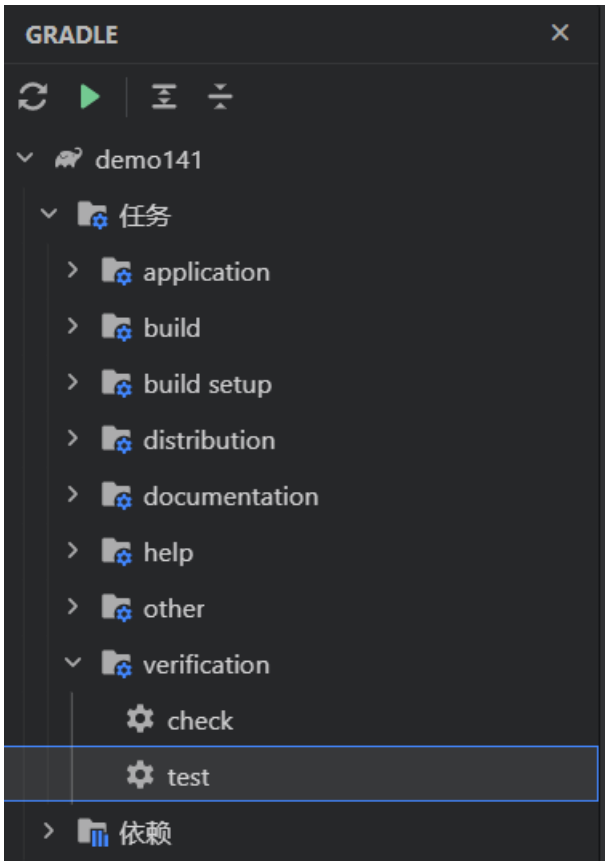
当用户在CodeArts IDE中打开一个Gradle项目时，用户可以在Gradle视图中找到列出的**Gradle**任务。



要运行任务，请执行以下任一操作：

- 双击任务列表中的任务。
- 在**Gradle**视图工具栏上，单击**“执行Gradle任务”**按钮（ ）然后在打开的“执行Gradle任务”窗口中选择所需的任务。

以同样的方式，用户可以运行在**build.gradle**的**test**任务中定义的测试。在这种情况下，CodeArts IDE将使用Gradle测试运行器。



说明

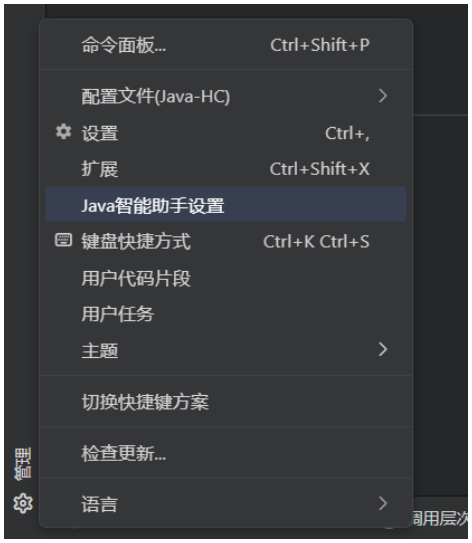
用户还可以通过**专用的Gradle启动配置**来运行Gradle任务。

有关在CodeArts IDE中测试应用程序的更多信息，请参阅**测试**。

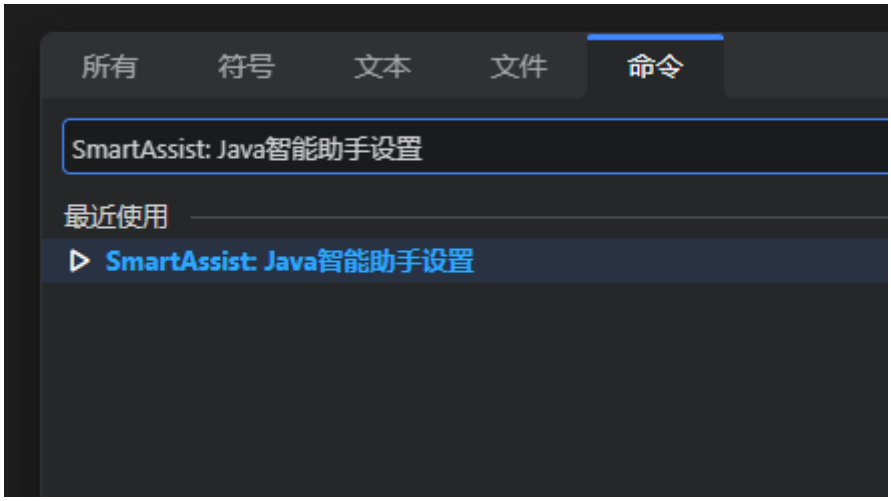
配置 Gradle 集成

步骤1 通过执行以下任一操作打开“Java智能助手设置”对话框：

- 单击左下角“管理 -> Java智能助手设置”菜单。



- 通过“Ctrl+Shift+P”或者“Ctrl+Ctrl”打开“命令”面板，输入**SmartAssist: Java智能助手设置**命令。



步骤2 切换到“构建工具”页面，在**Gradle**部分提供配置选项。

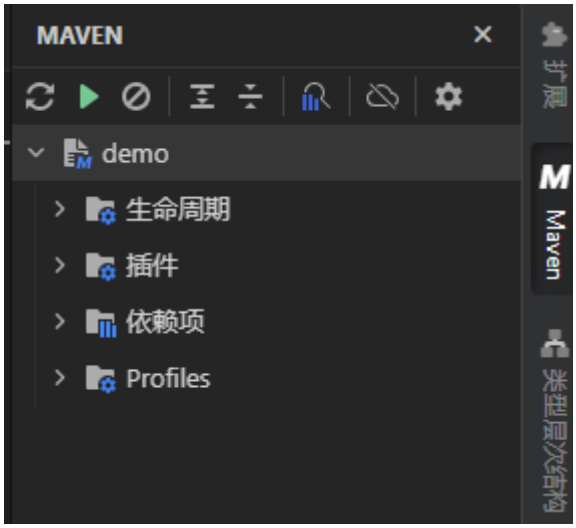
- **Gradle用户目录**：在此字段中，指定Gradle用户主目录的路径（默认为“$USER_HOME/.gradle”），该目录用于存储全局配置属性和初始化脚本、缓存和日志文件。默认值是基于**GRADLE_USER_HOME**环境变量的值提供的。要修改它，用户可以设置环境变量或单击**“浏览”**按钮，并手动定位所需的Gradle用户主目录。
- **Gradle SDK**：从这个列表中选择一个与Gradle一起使用的JDK：捆绑的JDK、项目级别的JDK或从系统变量（如JAVA_HOME）解析的JDK。

----结束

2.5.9.3 Maven

Maven是一个用于管理Java项目和自动化应用程序构建的工具。CodeArts IDE提供了集成的Maven支持。专用的**Maven**视图与**pom.xml**同步，并提供了一个可视化界面，用于查看项目依赖项或运行Maven任务。CodeArts IDE for Java已内置Maven 3.8.1。

要打开**Maven**视图，请在右侧的活动栏中选择**“Maven”**视图。

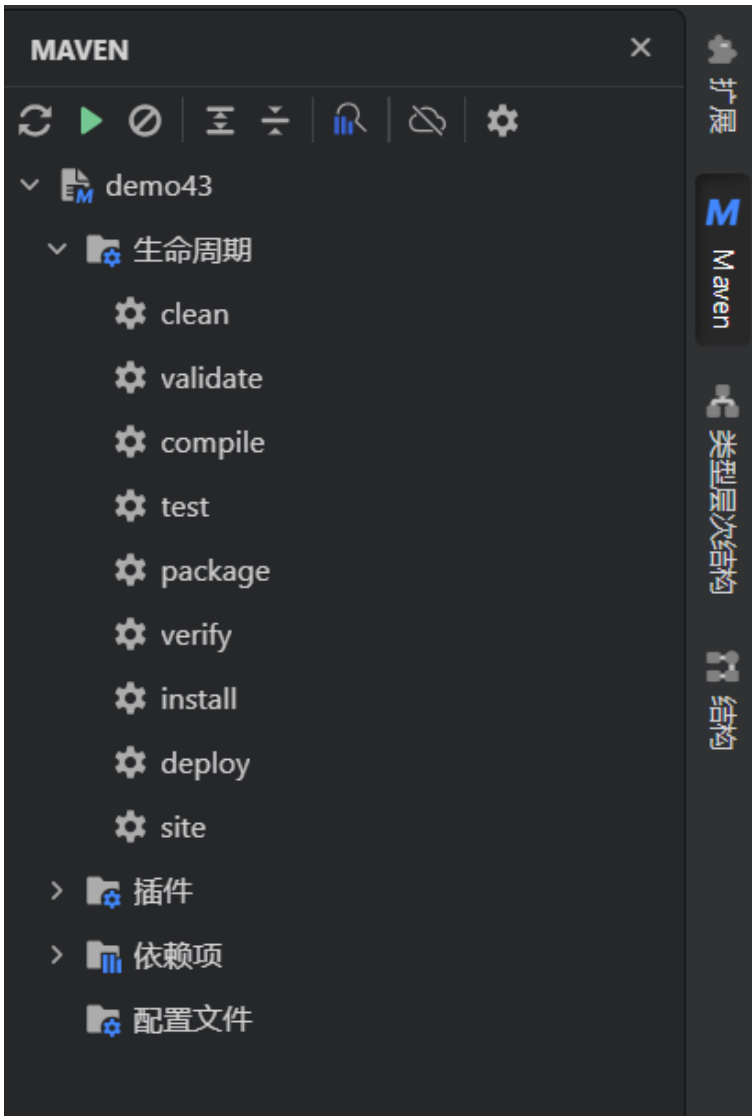


说明

在修改**pom.xml**后，例如添加新的依赖项，单击**Maven**视图工具栏上的**“重新加载所有Maven工程按钮”**（）以使CodeArts IDE应用用户的更改。

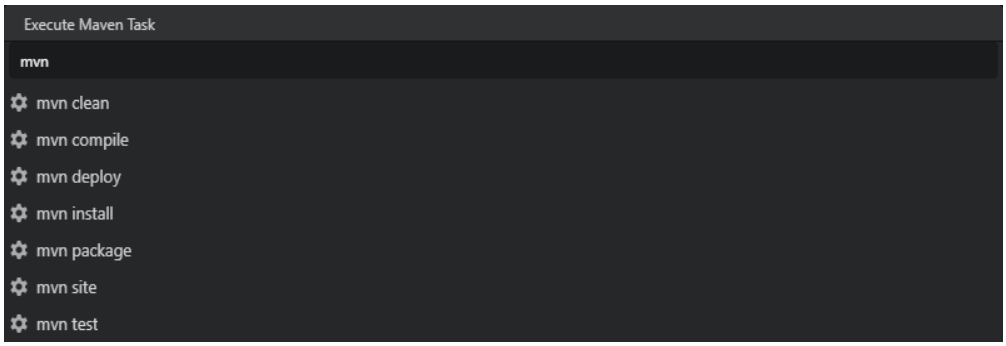
使用 Maven 任务进行工作

使用Maven任务在CodeArts IDE中打开一个Maven项目后，用户可以在**“Maven”**视图中找到Maven任务列表。



要运行任务，请执行以下任一操作：

- 双击任务列表中的任务。
- 在**“Maven”**视图工具栏上，单击**“执行Maven任务”**按钮（）然后在打开的“执行Maven任务”窗口中选择所需的任务。



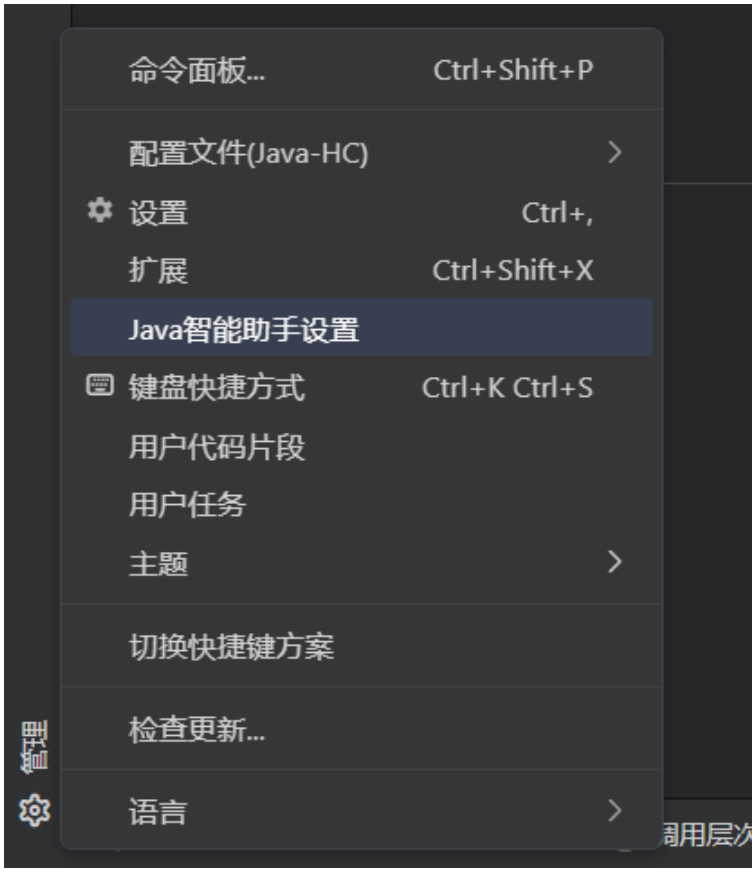
说明

用户还可以通过**专用的Maven启动配置**来运行Maven任务。

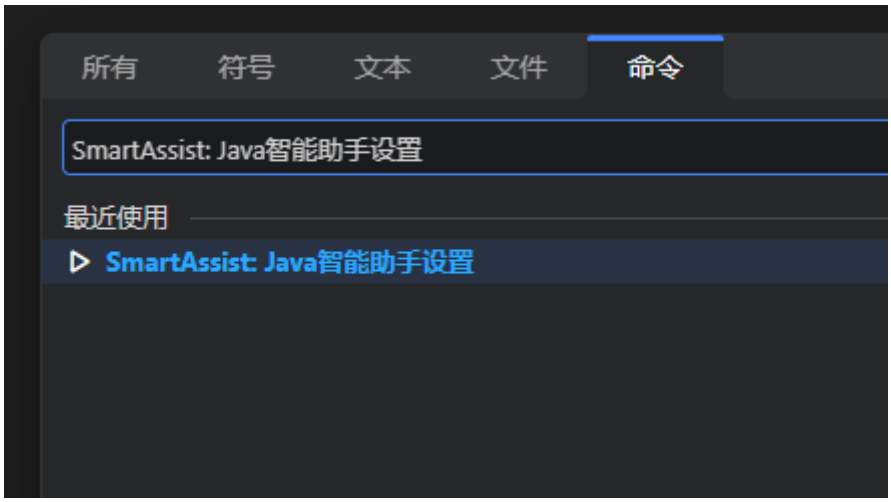
配置 Maven 集成

步骤1 通过以下任一方式打开“Java智能助手设置”对话框：

- 单击左下角“管理 -> Java智能助手设置”菜单。



- 通过“Ctrl+Shift+P”或者“Ctrl+Ctrl”打开“命令”面板，输入**SmartAssist: Java智能助手设置**命令。



步骤2 切换到“**构建工具**”页面，在**Maven**部分提供配置选项。

- **Maven路径**：使用此字段选择捆绑的Maven版本（Maven 3），或单击浏览按钮（）手动定位用户自己的**Maven**安装。
- **用户设置文件**：在此字段中指定包含用户特定配置的文件。
- **本地仓库**：在此字段中指定存储下载内容和临时构建产物的用户主目录下的本地目录路径。
- **Maven SDK**：从此列表中选择要与Maven一起使用的JDK：捆绑的JDK、项目级别的JDK或从系统变量（如**JAVA_HOME**）解析的JDK。
- **离线工作**：如果选择此项，Maven将在离线模式下工作。它不连接到远程仓库，只使用本地可用的资源。此选项对应于**--offline**命令行选项。
- **打印异常堆栈跟踪**：如果选择此项，将生成异常堆栈跟踪。此选项对应于**--errors**命令行选项。
- **递归执行目标**：如果选择此项，将递归执行构建，包括嵌套项目。此选项对应于**--non-recursive**命令行选项。

----结束

2.5.10 调试

2.5.10.1 通用调试步骤

CodeArts IDE内置调试器有助于加快编辑、编译、运行和调试循环的速度。

1. 在代码中设置断点以定义程序应停止的位置。
2. 在调试模式下运行程序。
3. 当程序挂起时，在“运行和调试”视图中检查其输出。
4. 找到错误，更正错误，然后重新运行程序。在Java上下文中，用户可以使用热代码替换功能在不停止调试会话的情况下动态修改和重新加载类。

2.5.10.2 断点

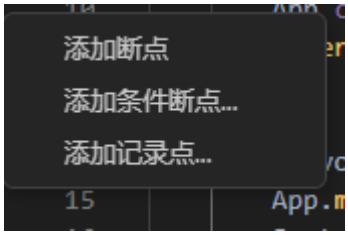
2.5.10.2.1 设置断点

行断点

行断点是常规断点，可在设置的行上停止程序的执行。

执行以下任一操作：

- 单击编辑器装订线中所需的行。
- 右键单击编辑器装订线中所需的行，然后从上下文菜单中选择“**添加断点**”。

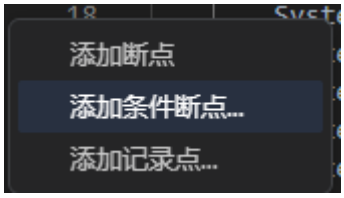


行断点由编辑器装订线中的圆形图标（ ）表示：

条件断点

CodeArts IDE调试器允许用户根据任意表达式或命中计数设置条件断点。

在代码编辑器中，右键单击所需的行，然后从上下文菜单中选择“添加条件断点...”。

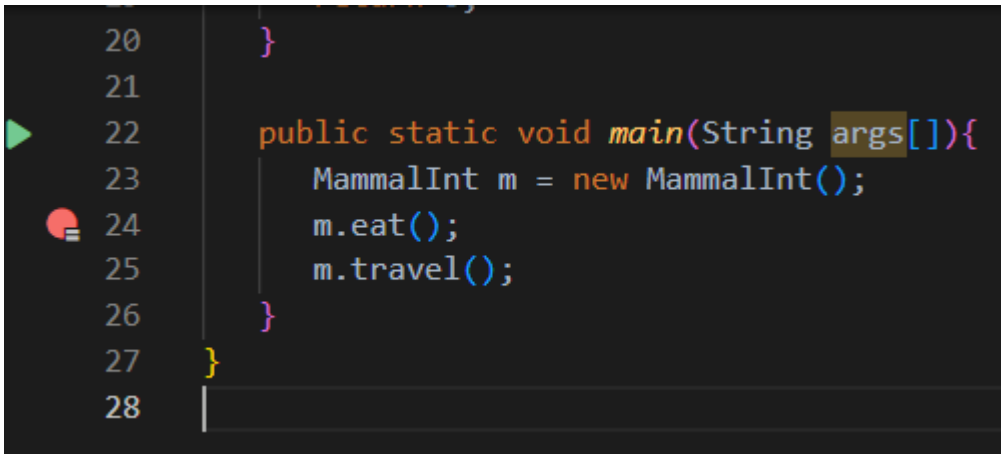


或者，在主菜单中，选择“调试 > 新建断点 > 条件断点”。在打开的查看编辑器中，从列表中选择条件类型。

- **表达式**：只要表达式的计算结果为true ，就会命中断点。
- **命中次数**：断点需要命中定义的次数才能停止程序执行。

输入条件并按“**Enter**”完成条件断点添加。

条件断点由编辑器装订线中的等号图标（ ）表示：



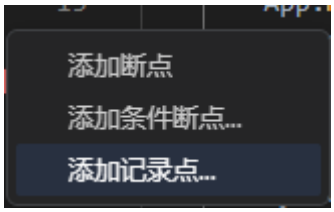
说明

用户也可以向常规行断点添加条件：表达式或命中计数。右键单击编辑器装订线中的断点，然后从上下文菜单中选择所需的操作。

记录点

记录点是一个断点，在命中时不会停止程序执行，而是将消息记录到控制台。

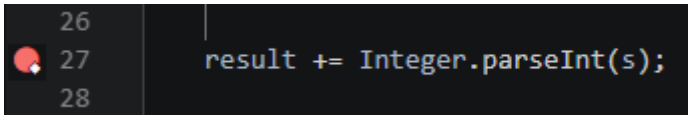
1. 在代码编辑器中，右键单击所需的行，然后从上下文菜单中选择**“添加记录点...”**。



或者，在主菜单中，选择“调试 > 新建断点 > 添加记录点...”。

2. 在打开的窗口中，键入命中记录点时应记录的消息。日志消息可以是纯文本，也可以包括要在大括号（**{}**）中计算的表达式。

记录点由编辑器**装订线**中的菱形图标（ ）表示：



说明

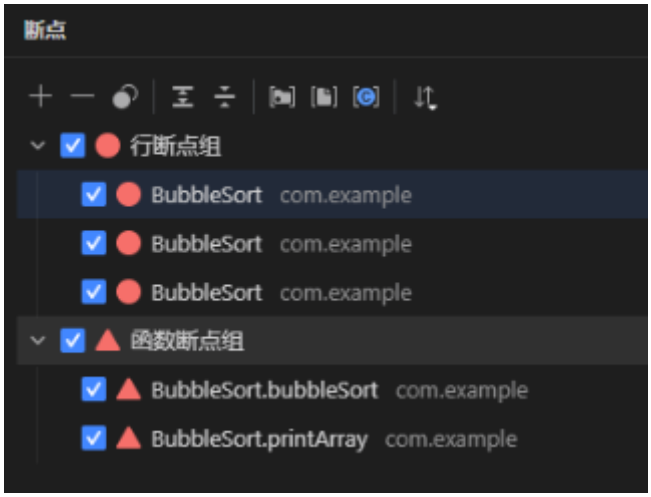
就像常规断点一样，记录点可以启用或禁用，也可以由表达式或命中次数控制。如果设置了表达式或命中次数，则仅在表达式为真或达到命中次数时记录消息。

函数断点

除了直接在源代码中放置断点外，用户还可以通过指定函数/方法名称来创建断点。程序执行在进入指定的函数时停止。

1. 单击底部活动栏中的**“断点”**按钮（ ），打开“断点”视图。
2. 在“断点”部分中，单击**“添加函数断点”**工具栏按钮（ ），或者在主菜单中选择“调试 > 新建断点 > 函数断点...”。

3. 输入所需函数的完全限定名称，然后按“**Enter**”键。

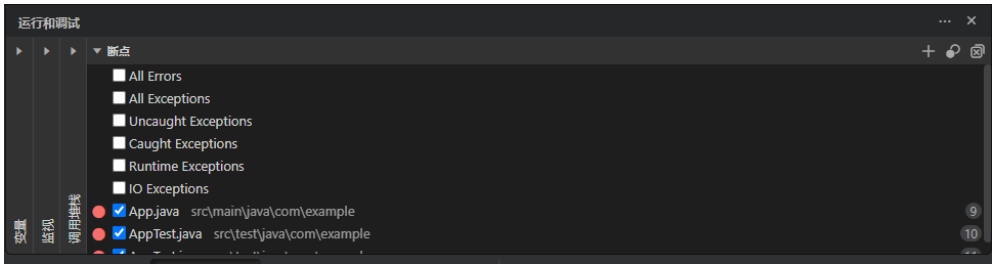


函数断点在“断点”视图中使用三角形图标（ ）表示。

异常断点

CodeArts IDE调试器支持异常断点，每当抛出Throwable或其子类时，断点就会挂起程序。异常断点是全局应用的，不需要特定的源代码引用。

1. 单击右侧活动栏中的“**运行和调试**”按钮（ ）或按“Ctrl+Shift+D”或“Shift+Alt+F9”或“Alt+5”或“Ctrl+Shift+F8”打开“运行和调试”视图。
2. 展开“断点”部分，然后选中要设置的异常断点旁边的复选框。



CodeArts IDE提供了几种类型的异常断点。这些定义了在抛出时暂停程序执行的特定异常类。

- **All Errors**： Error及其子类。
- **All Exceptions**： Exception及其子类。
- **Uncaught Exceptions**： 未捕获的Exception及其子类。
- **Caught Exceptions**： 捕获的Exception及其子类。
- **Runtime Exceptions**： RuntimeException及其子类。
- **IO Exceptions**： IOException及其子类。

2.5.10.2.2 启用和禁用断点

用户可以启用或禁用单个断点，或同时启用或禁用多个断点，或禁用所有断点。

要禁用单个断点，请执行以下任一操作：

- 在编辑器装订线中，右键单击断点，然后从上下文菜单中选择“**禁用断点**”。

- 在“断点”视图中，清除要禁用的断点旁边的复选框。

要同时禁用所有断点，请执行以下操作：

- 在“断点”视图中，单击工具栏按钮。

要删除断点，请执行以下操作：

删除单个断点，请单击编辑器装订线中的断点。

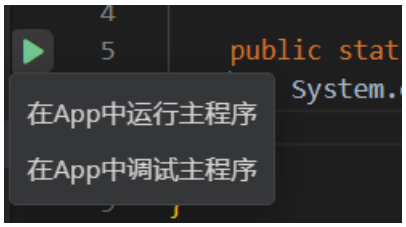
2.5.10.3 在调试模式下运行程序

CodeArts IDE允许用户直接从代码编辑器或通过启动配置启动调试会话。

从代码编辑器启动调试会话

如果用户不打算向程序传递任何参数，则可以直接从代码编辑器启动调试会话。

在具有**main()**方法的类的代码编辑器中，单击或右键**main()**方法左方**运行/调试 xxx.main**按钮，唤出下拉列表，进一步单击**在xxx中运行主程序**或**在xxx中调试主程序**项。运行应用**程序启动配置**将自动创建并运行。



说明

创建的启动配置会自动保存，用户可以随时从CodeArts IDE主工具栏上的配置列表中选择它。

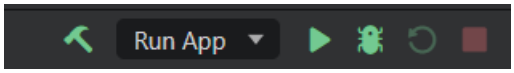
从启动配置启动调试会话

启动配置允许用户配置和保存各种调试方案的调试设置详细信息。有关使用启动配置的详细信息，请参见**启动配置**。

步骤1 从CodeArts IDE主工具栏上的配置列表中选择所需的启动配置，然后按“**F5**”或者“**Shift+F9**”。

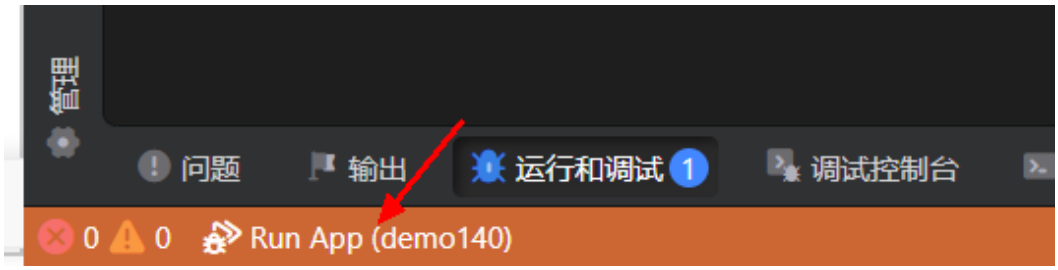
步骤2 执行以下任一操作：

- 在主菜单中选择“调试 > 启动调试”，或按“F5”或者“Shift+F9”。
- 在调试工具栏上，确保在启动配置列表中选择了所需的启动配置，然后单击“开始执行(不调试)”按钮（ ）或者“开始调试”按钮（ ）。



----结束

调试会话启动后，将立即显示**终端**或者**调试控制台**面板并显示调试输出，状态栏将更改颜色（默认颜色主题为橙色）。**调试的状态**显示在显示活动启动配置的状态栏中。单击调试状态以更改活动启动配置并重新启动调试，而无需重新打开**运行和调试**视图。



2.5.10.4 控制程序执行

启动调试会话后，可以使用调试工具栏操作控制程序的执行。

图标	动作	快捷键	描述
	暂停	F9	暂停调试会话。
	单步跳过	F8	跳过当前代码行到下一行。 如果当前行中有方法调用，则将跳过它们的实现，以便用户移动到调用者方法的下一行。

图标	动作	快捷键	描述
	单步调试	F7	进入方法以显示其实现。
	单步跳出	Shift +F8	退出当前方法，跳转到调用者方法。
	运行到光标处	Alt +F9	运行到光标所在的代码行处。
	重启	Ctrl +Shift +F5	重新启动调试会话。
	停止	Ctrl +F2	停止调试会话。
	编译并替换	--	在调试期间，用户可以编辑程序的代码并动态重新加载更改。单击此按钮可重新编译受影响的类，并将正在运行的字节码替换为新的字节码。因此，用户不需要重建整个程序和重新启动调试会话。 目前支持以下修改： • 更改任何方法的主体。 • 添加/删除私有方法。 • 更改私有方法的签名和非访问修饰符。 • 在任何方法中添加/删除/更改lambda。

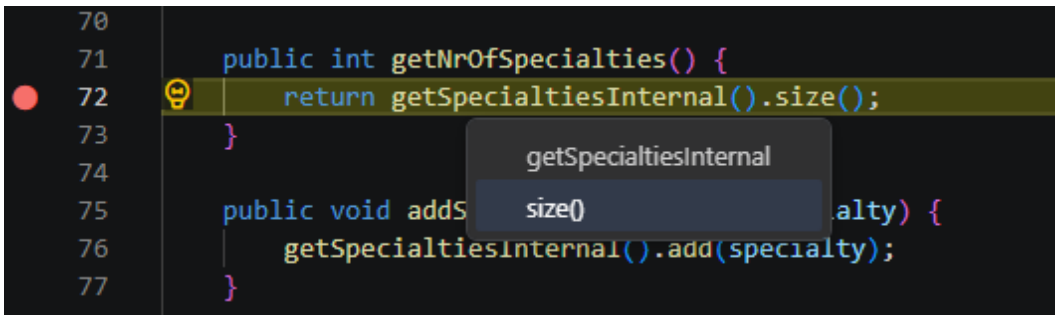
运行到光标处

当程序挂起时，用户可以继续执行，直到到达光标位置。在代码编辑器中，将光标放置在所需的行，鼠标右键唤出上下文菜单并单击**“运行到光标处”**，或者单击右上角**“运行到光标处”**（），或按“Alt+F9”。

步入目标

当一行上有多个方法调用时，**“单步执行目标”**功能允许用户选择要逐步进入的方法调用。

在代码编辑器中，鼠标右键调试会话所暂停的行，然后从上下文菜单中选择**“单步执行目标”**。在打开的弹出菜单中，选择要进入的方法。



重新加载修改后的类

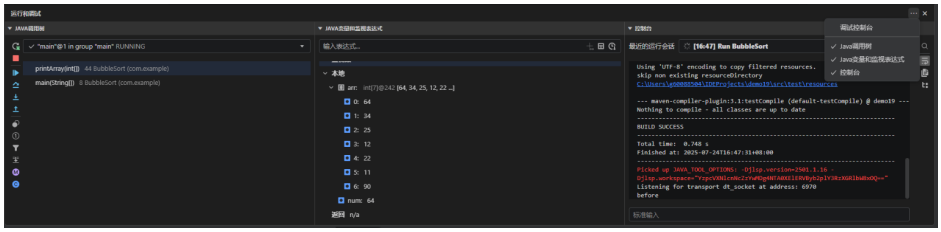
CodeArts IDE调试器提供了热代码替换功能，允许用户在调试过程中编辑程序的代码，并即时重新加载用户的更改。因此，用户无需重新构建整个程序并重新启动调试会话。

1. 按照以**调试模式运行程序**的说明开始调试会话。
2. 当程序在断点处停止时，对代码进行必要的编辑。请注意，热代码替换支持以下修改：
   - 更改任何方法的主体。
   - 添加/删除、更改私有方法的签名和非访问修饰符。
   - 在任何方法中添加/删除/更改lambda表达式。
3. 在CodeArts IDE工具栏上，单击**“编译并替换”**按钮（）以重新编译受影响的类，并用新的字节码替换正在运行的字节码。

2.5.10.5 检查暂停的程序

2.5.10.5.1 简介

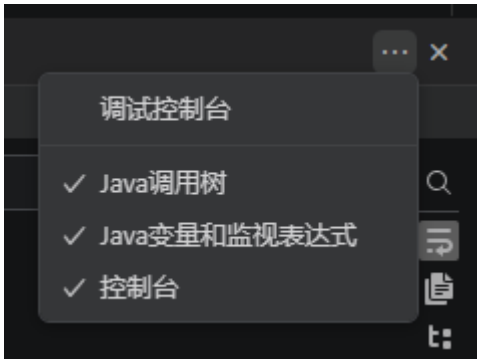
启动调试会话时，“运行和调试”视图将打开，以显示与运行和调试相关的所有信息。手动打开“运行和调试”视图，单击右侧活动栏中的**“运行和调试”**按钮（），或按“Ctrl+Shift+D”/“Shift+Alt+F9”/“Alt+5”/“Ctrl+Shift+F8”快捷键。



“运行和调试”视图包括以下部分：

- **Java调用树**
- **Java变量和监视表达式**
- **控制台**

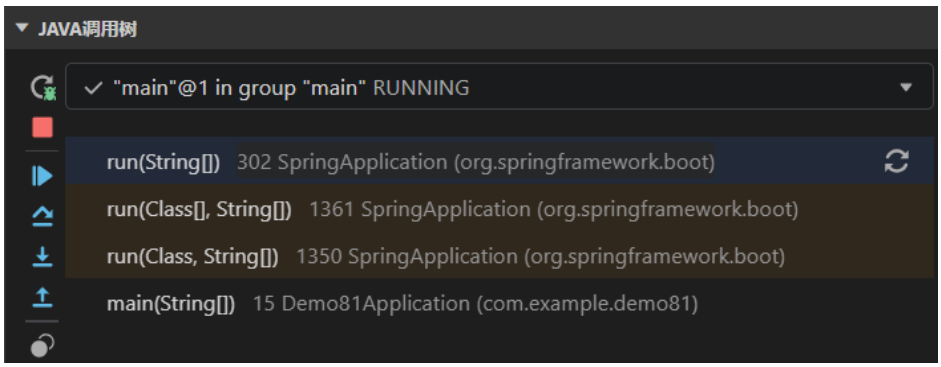
要自定义“运行和调试”视图内容，请单击右上角**“更多操作”**（...）然后选中要显示的部分旁边的复选框。



2.5.10.5.2 JAVA 调用树

“JAVA调用树”部分列出了当前活动的堆栈帧，方法的调用堆栈分组在每个帧下。如下图所示：

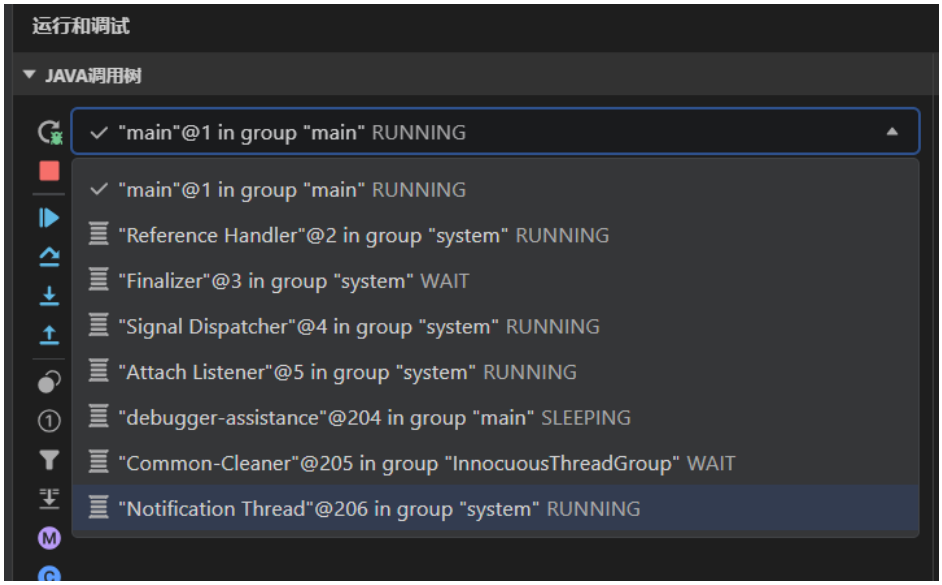
图 2-5 默认展示的主线程对应的堆栈帧



当前堆栈帧内可访问的元素列在**“JAVA变量和监视表达式”**部分中。

- 默认展示主线程的堆栈帧。要切换到其他线程，请在**“JAVA调用树”**的下拉框中选择对应的线程。如下图所示：

  图 2-6 切换不同线程的堆栈帧入口

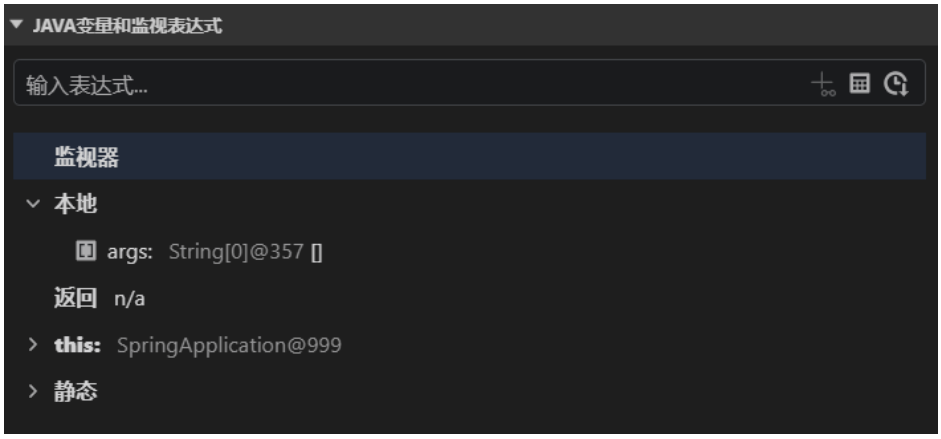
  

- 要快速导航到代码编辑器中的方法调用处，请单击**“JAVA调用树”**中的堆栈帧。

2.5.10.5.3 JAVA 变量和监视表达式

“JAVA变量和监视表达式”部分显示当前堆栈帧（即在“JAVA调用树”部分中选择的堆栈帧）中可访问的元素。如下图所示：

图 2-7 JAVA 变量和监视表达式视图



该视图有以下几部分：

- **“监视器”**：显示进行“将表达式添加到监视”操作之后的表达式和表达式的值。如下图所示：

  图 2-8 输入相关表达式并添加到监视示例

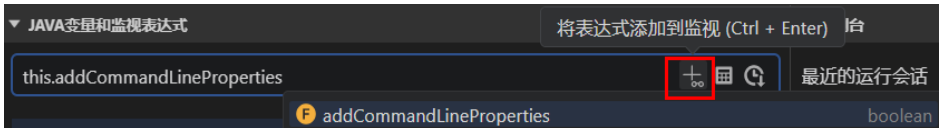
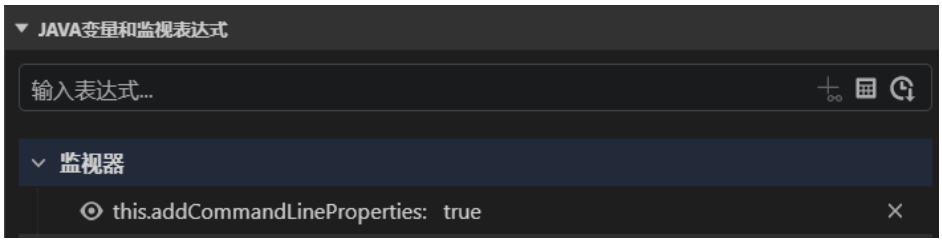
  

  图 2-9 监视器显示已添加的表达式

- **“本地”**：显示被调用方法作用域内的局部变量。
- “返回 n/a”：当一个方法在调试会话期间被多次调用时，本节显示该方法在上一步返回的值。这使用户可以观察值在方法调用之间的变化。
- **“this”**：显示正在调用其方法的对象的内容。
- “静态”：列出静态类字段。

用户可以在变量上通过右键唤出上下文菜单，使用**“设置值”**操作来修改变量的值。此外，用户可以使用**“复制值”**操作复制变量的值。

说明

用户还可以直接在CodeArts IDE代码编辑器中查看变量或表达式的值。为此，请在挂起的调试程序中将鼠标悬停在所需的变量、表达式上。

2.5.11 测试

2.5.11.1 将测试框架集成到项目中

CodeArts IDE提供了集成JUnit和TestNG测试框架的能力，让用户轻松运行和调试Java测试用例。在开始之前，请确保为项目定义了JDK，如**使用Java项目**中所述。

用户可以通过在**pom.xml**（对于Maven）或**build.gradle**（对于Gradle）中声明相应的依赖来在项目中启用测试框架集成。

JUnit 3/4

对于Maven项目，请在**pom.xml**中添加以下配置。

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
</dependency>
```

对于Gradle项目，请将以下行添加到**build.gradle**中：

```
dependencies {
    testImplementation 'junit:junit:4.11'
}
test {
    useJUnitPlatform()
}
```

JUnit 5

对于Maven项目，请在**pom.xml**中添加以下配置。

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>RELEASE</version>
    <scope>test</scope>
</dependency>
```

对于Gradle项目，请将以下行添加到**build.gradle**中：

```
dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter:5.8.1'
}
test {
    useJUnitPlatform()
}
```

TestNG

对于Maven项目，请在**pom.xml**中添加以下配置。

```
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>RELEASE</version>
    <scope>test</scope>
</dependency>
```

对于Gradle项目，请将以下行添加到**build.gradle**中：

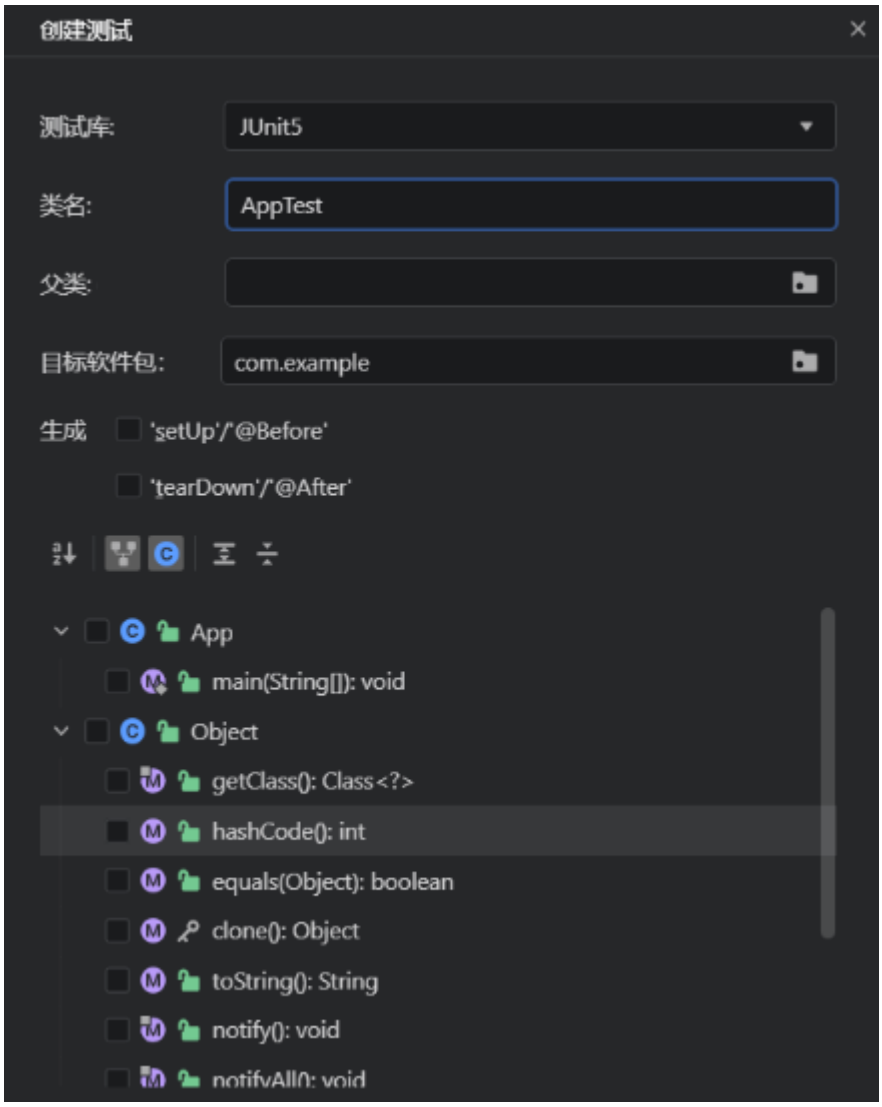
```
dependencies {
    testImplementation 'org.testng:testng:7.7.0'
}
```

2.5.11.2 创建测试

CodeArts IDE提供了专门的**源操作**，帮助用户搭建测试用例。

步骤1 在代码编辑器中，右键单击要为其创建测试的类的声明，然后从右键菜单中选择“源代码操作 > 测试”。或者，在“命令面板”（“Ctrl+Shift+P”或双击“Ctrl”）中搜索并运行**源代码操作**命令。

步骤2 在打开的“创建测试”对话框中，提供要创建的测试类的详细信息。



- 选择要使用的测试框架。
- 提供测试类的名称，或保留默认值。
- 对于JUnit3，在“**父类**”字段中提供**junit.framework.TestCase**。对于其他框架，请将该字段留空。
- 在“目标软件包”中，选择测试类存储在其中的包。
- 如有必要，请选择“**'setUp'/'@Before'**”或“**'tearDown'/'@After'**”复选框，以将测试装置和注释的存根方法包括到生成的测试类中。
- 为了能够查看并选择从超类中继承的方法，请选中“**显示继承的方法**”复选框。

- 在“**成员**”区域中，选中要为其创建测试方法的方法旁边的复选框。

步骤3 单击“**确定**”。

----结束

2.5.11.3 运行测试

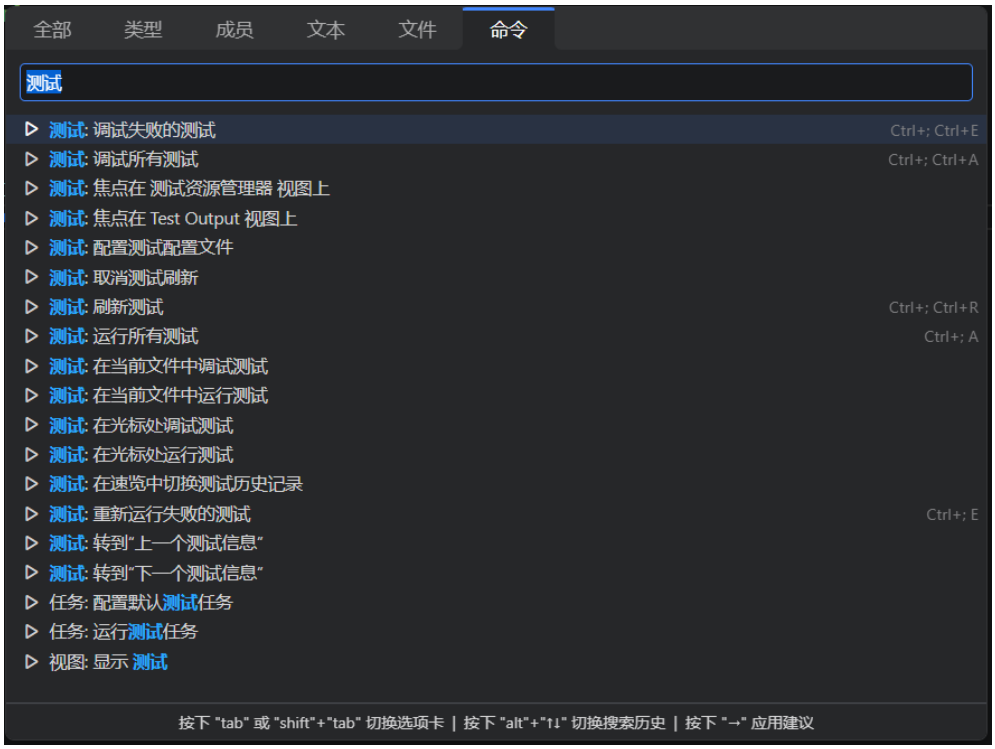
2.5.11.3.1 简介

CodeArts IDE提供了多个选项来运行和调试测试：

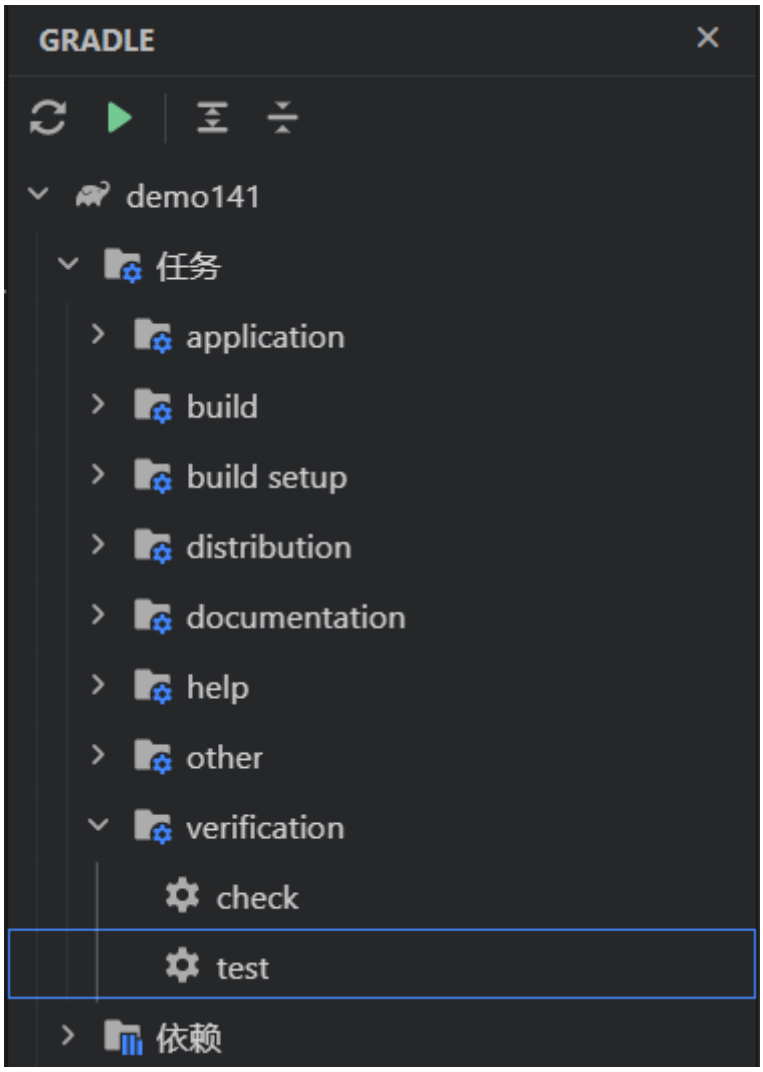
- 在测试类的代码编辑器中，单击测试类声明旁边的“**运行**”按钮( )（运行类中的所有测试）或者单个测试方法（仅运行单个测试）。如果需要调试测试，请右键单击“**运行**”按钮( )，然后从右键菜单中选择“**调试测试**”。

```
10  public class AppTest {
11      /**
12       * Rigorous Test :-)
13       */
14      @Test
15      public void shouldAnswerWithTrue() {
16          assertTrue(true);
17      }
18  }
```

- 使用**测试视图**管理和运行测试。
- 使用**测试启动配置**：**Run All Tests（JUnit）**和**Debug Tests（JUnit）**。
- 在“命令”面板（按“Ctrl+Shift+P”或双击“Ctrl”）中，搜索**测试**并使用与测试相关的命令，如“**在当前文件中运行测试**”或“**在光标处运行测试**”。



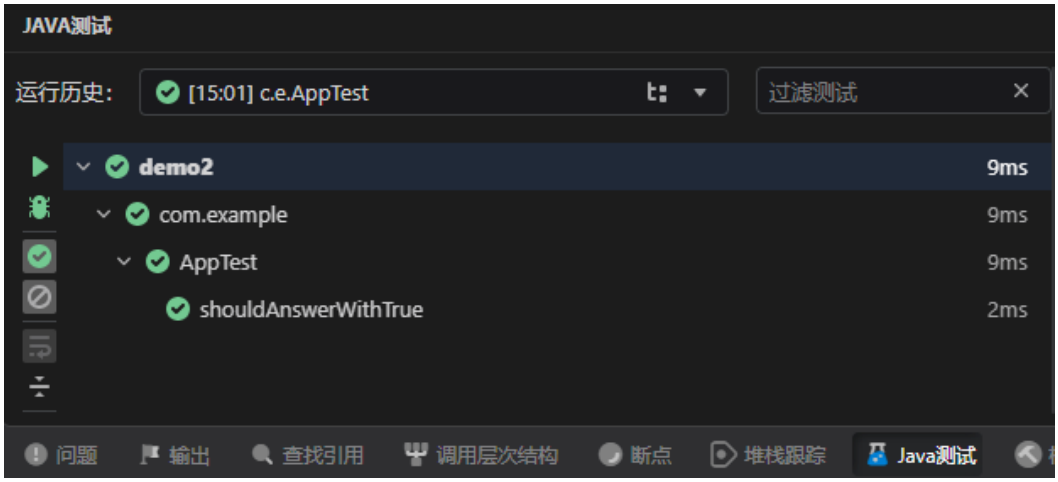
- 在Gradle项目中，在Gradle视图中，通过双击“**test**”来执行该任务。有关CodeArts IDE中**Gradle**集成的详细信息，请参阅**Gradle**。



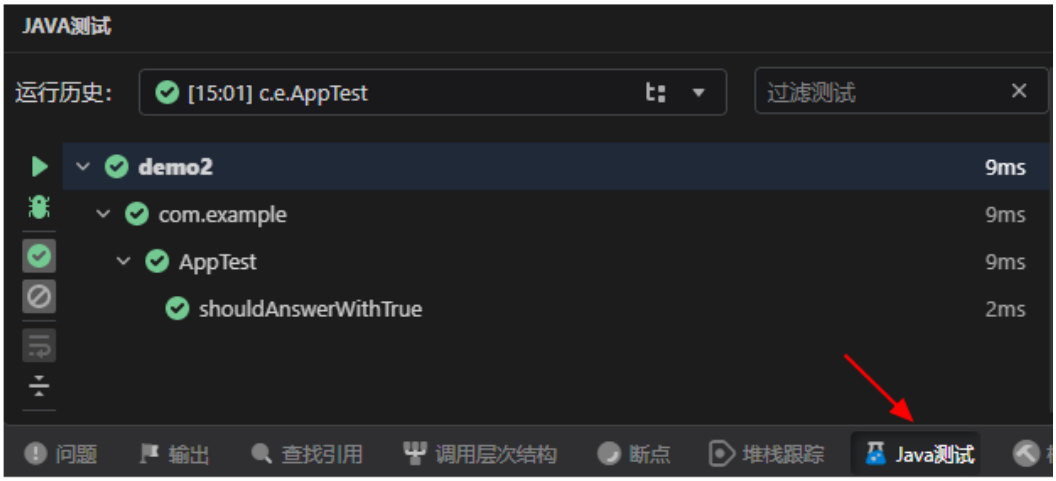
2.5.11.3.2 测试视图

简介

“测试”视图列出了项目中的所有测试用例，让用户可以运行它们并检查结果。



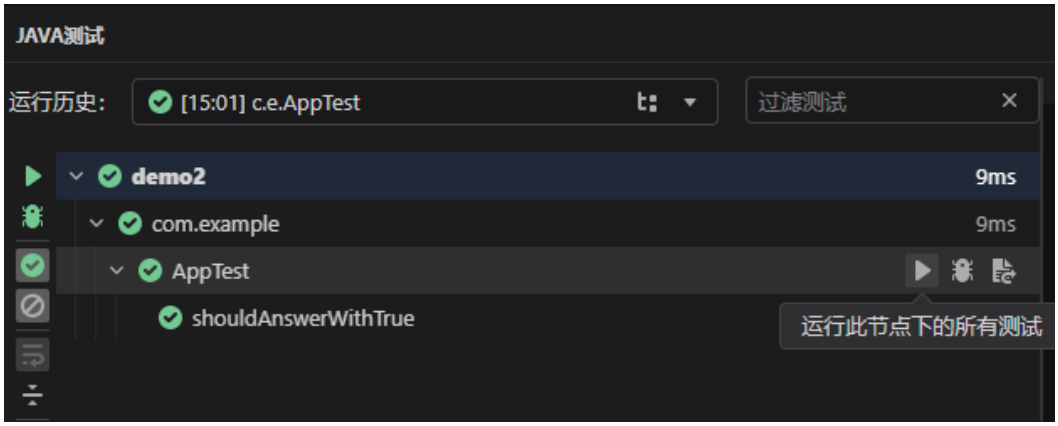
要打开“测试”视图，请单击CodeArts IDE底部面板中的**“Java测试”**按钮（ ）。



运行和调试测试

步骤1 将鼠标悬停在与包含要运行的测试的包、类或方法对应的树节点上。

步骤2 单击**“运行”**按钮（ ）运行测试，或单击**“调试”**按钮（ ）调试测试。



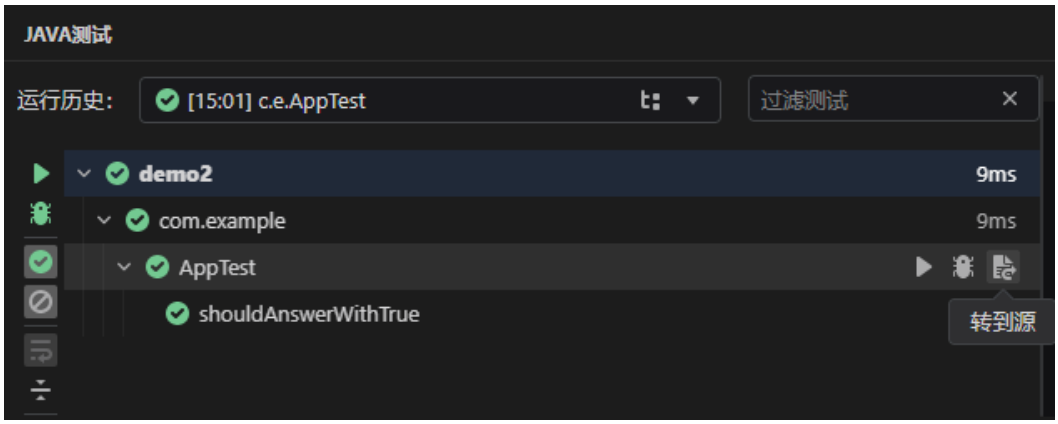
----结束

要运行或调试所有可用的测试，请单击“JAVA测试”视图工具栏上的**“运行测试”**（ ）或**“调试测试”**（ ）按钮。

导航到测试类或方法

在“JAVA测试”视图中，用户可以直接从测试用例导航到相应的测试类或方法。

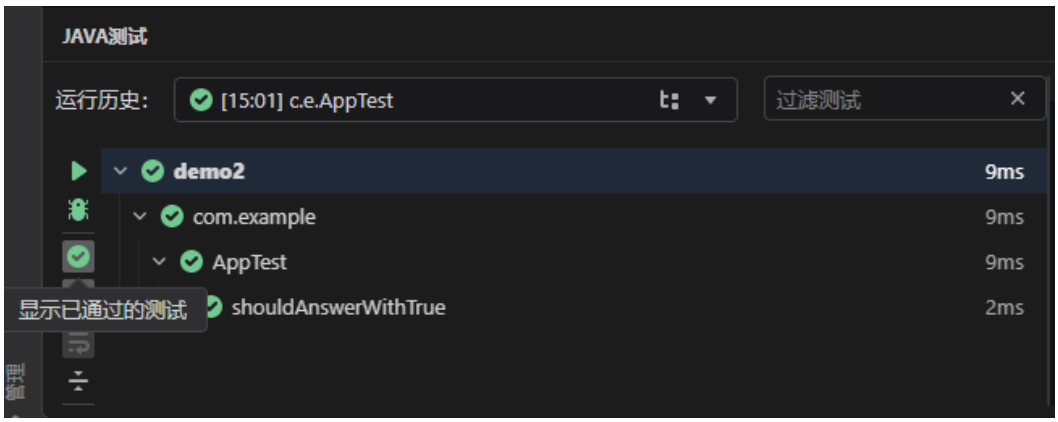
1. 在CodeArts IDE中，找到并打开“Java测试”视图。
2. 在“Java测试”视图中，你会看到一个树形结构，列出了所有的测试类和测试方法。
3. 双击与类或方法对应的树节点，或将鼠标悬停在其上并单击**“转到源”**按钮（ ）。

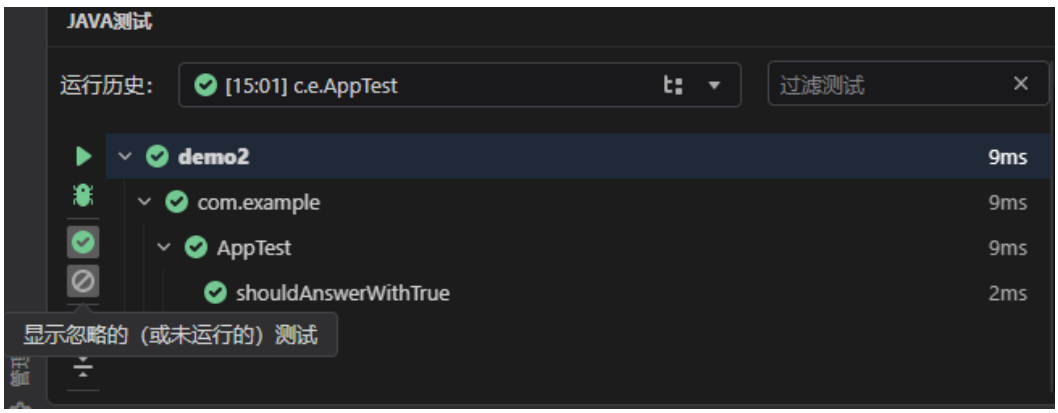


组织测试视图

“测试”视图提供了多种功能，方便地组织测试显示。

显示已通过的测试和显示忽略的（或未运行的）测试。





“测试”测试顶部的过滤字段允许用户提供文本查询以按条件筛选测试列表。

2.5.11.3.3 测试启动配置

简介

CodeArts IDE允许用户自定义运行测试用例的配置。为此，用户可以将相应的**启动配置**添加到项目的**launch.json**中。

以下配置模板可用于运行和调试测试：

- JUnit测试
- TestNG测试

JUnit 测试

启动配置属性

在启动配置中，用户可以选择相应的属性来运行单个测试方法、单个测试类、包中的所有测试或目录中的所有测试。

名称	描述
type	调试器的类型。对于运行和调试Java代码，应将其设置为**javadbg**。
name	启动配置的名称。
env	额外的环境变量。
skipBuild	跳过程序的构建过程（设置为**true**）或不跳过（设置为**false**）。
killPrevSession	终止具有相同名称的先前运行会话（设置为**true**），或中止启动（设置为**false**）。
vmOptions	JVM的额外选项。
method	完全限定的测试方法名称。
class	完全限定的测试类名称。
package	测试包名称。
directory	包含测试源代码的目录。默认情况下，此项设置为**${workspaceRoot}/src/test**。用户可以使用变量来提供路径。

启动配置示例

用户可以使用提供的示例作为工作启动配置示例。

运行**package.name**包中的所有测试：

```
{
   "type": "javadbg",
   "name": "JUnit Test (Package)",
   "request": "launch",
   "jUnit": {
      "package": "package.name"
   },
   "vmOptions": "-ea"
}
```

运行单个测试方法**qualified.method.name**：

```
{
   "type": "javadbg",
   "name": "JUnit Test (Method)",
   "request": "launch",
```

```
    "jUnit": {
        "method": "qualified.method.name"
    },
    "vmOptions": "-ea"
}
```

TestNG 测试

启动配置属性

在启动配置中，用户可以选择相应的属性来运行单个测试方法、单个测试类、包中的所有测试或目录中的所有测试。

名称	描述
type	调试器的类型。对于运行和调试Java代码，应将其设置为**javadbg**。
name	启动配置的名称。
env	额外的环境变量。
skipBuild	跳过程序的构建过程（设置为**true**）或不跳过（设置为**false**）。
killPrevSession	终止具有相同名称的先前运行会话（设置为**true**），或中止启动（设置为**false**）。
vmOptions	JVM的额外选项。
method	完全限定的测试方法名称。
class	完全限定的测试类名称。
package	测试包名称。
directory	包含测试源代码的目录。默认情况下，此项设置为**${workspaceRoot}/src/test**。

启动配置示例

用户可以使用提供的示例作为工作启动配置示例。

运行**package.name**包中的所有测试：

```
{
    "type": "javadbg",
    "name": "TestNG Test (Package)",
    "request": "launch",
    "testNG": {
        "package": "package.name"
    },
    "vmOptions": "-ea"
}
```

运行单个测试方法**qualified.method.name**：

```
{
    "type": "javadbg",
    "name": "TestNG Test (Method)",
    "request": "launch",
```



```
    "testNG": {
        "method": "qualified.method.name"
    },
    "vmOptions": "-ea"
}
```

2.5.12 启动配置

2.5.12.1 简介

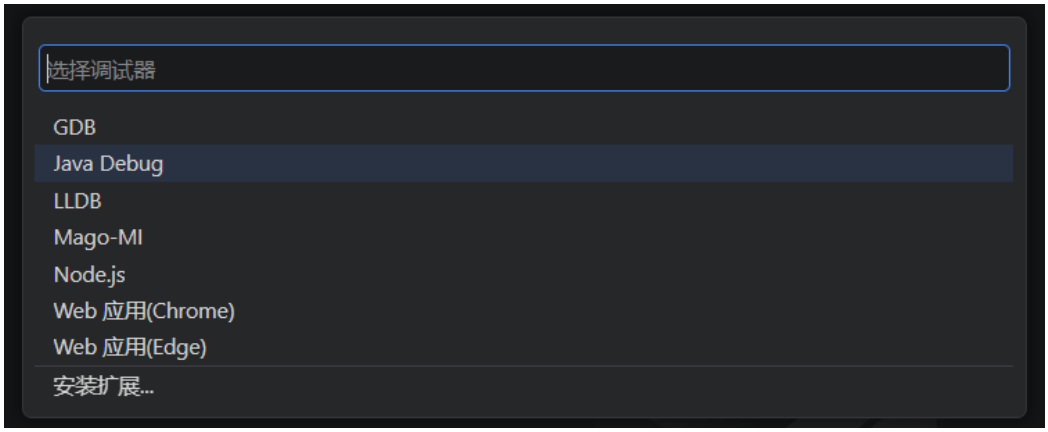
启动配置允许用户配置和保存各种场景的运行或调试设置详细信息。CodeArts IDE将配置信息保存在项目根文件夹下的“.arts”文件夹中的**launch.json**文件中。

在Java环境中，可以使用以下配置模板：

- **Java类**
- **JAR应用程序**
- **Gradle任务**
- **Maven目标**
- **JUnit测试**
- **TestNG测试**
- **远程调试**

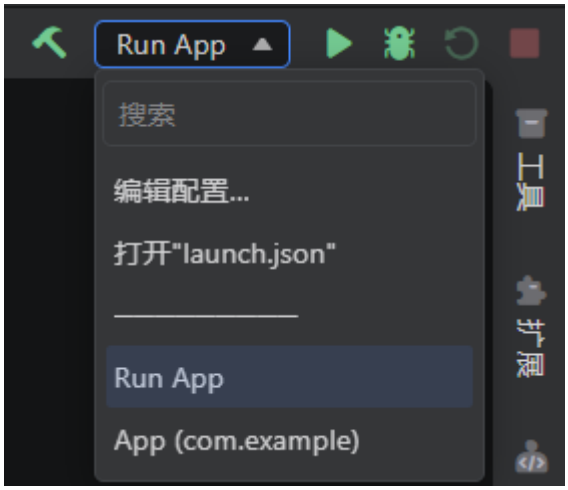
要创建**launch.json**文件，请在CodeArts IDE主工具栏上的列表中选择**“打开launch.json”**。

CodeArts IDE尝试自动检测用户的调试环境，但如果失败，用户将需要手动选择。在Java环境中，选择**“Java Debug”**。



CodeArts IDE创建一个**launch.json**文件，并根据检测到的环境填充默认内容。请检查所有自动生成的值，确保它们适用于用户的项目和调试环境。

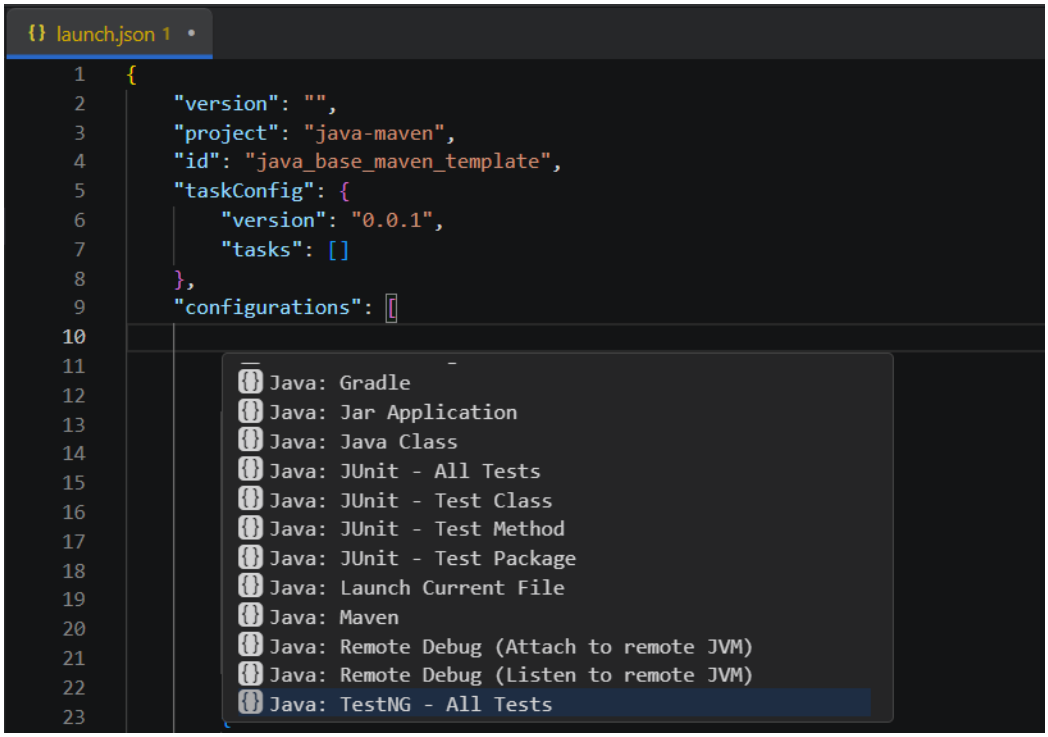
在**launch.json**中定义的所有启动配置都可以从CodeArts IDE主工具栏上的列表中选择。



向现有的 launch.json 添加新的配置

步骤1 在**launch.json**编辑器中，单击编辑器右下角的**“添加配置...”**按钮，或将光标放置在**configurations**数组内，并按**“Ctrl+I”**或**“Ctrl+Space”**或**“Ctrl+Shift+Space”**（IDEA快捷键方案）使用代码完成。

步骤2 在弹出的建议列表中，选择要使用的启动配置模板。



在**launch.json**中，按“**Ctrl+I**”或“**Ctrl+Space**”或“**Ctrl+Shift+Space**”（IDEA快捷键方案）使用代码完成查看可用属性及其值的列表。

----结束

变量替换

CodeArts IDE将常用路径和其他值作为变量提供，并支持在**launch.json**中的字符串中进行变量替换，因此用户不必在启动配置中使用绝对路径。

支持以下预定义变量：

- **${cwd}** - CodeArts IDE启动时任务运行器的当前工作目录。
- **${defaultBuildTask}** - 默认构建任务的名称。
- **${extensionInstallFolder}** - 指定扩展安装的路径。
- **${fileBasenameNoExtension}** - 当前打开文件的无扩展名的基本名称。
- **${fileBasename}** - 当前打开文件的基本名称。
- **${fileDirname}** - 当前打开文件的目录名。
- **${fileExtname}** - 当前打开文件的扩展名。
- **${file}** - 当前打开的文件。
- **${lineNumber}** - 活动文件中当前选定的行号。
- **${pathSeparator}** - 操作系统用于分隔文件路径组件的字符。
- **${relativeFileDirname}** - 相对于**workspaceFolder**的当前打开文件的目录名。
- **${relativeFile}** - 相对于**workspaceFolder**的当前打开文件。
- **${selectedText}** - 活动文件中当前选定的文本。
- **${workspaceFolderBasename}** - 在CodeArts IDE中打开的文件夹的名称，不包含任何斜杠（/）。
- **${workspaceFolder}** - 在CodeArts IDE中打开的文件夹的路径。

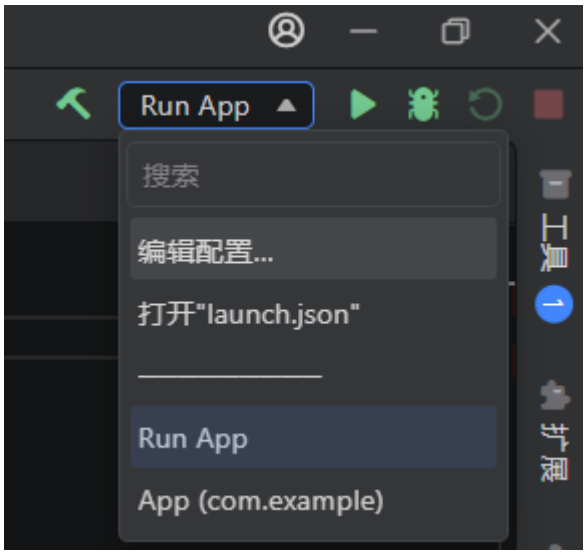
临时和永久启动配置

当用户从编辑器边栏手动运行类或方法时，CodeArts IDE会自动创建相应的启动配置，并在配置列表中显示。这些配置默认为临时配置：CodeArts IDE根据指定的限制（默认为10）保留它们的数量，并在超过此限制时自动删除最少使用的配置。

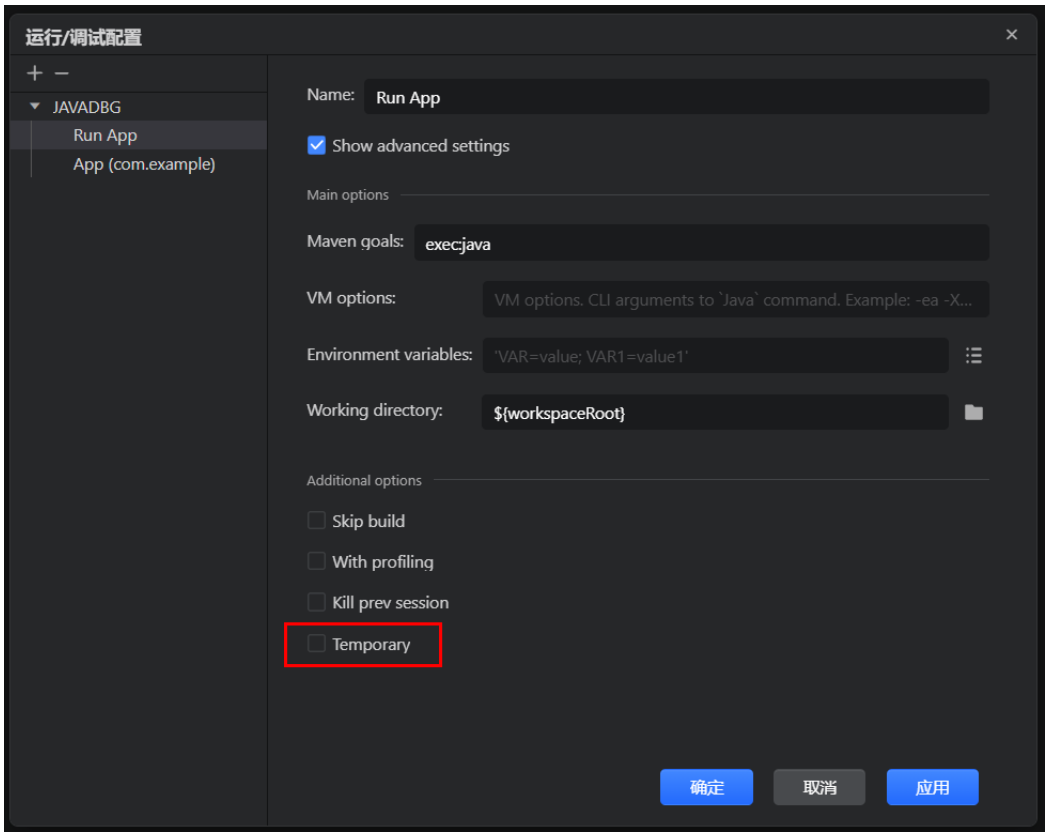
将临时启动配置保存为永久配置

用户可以将临时启动配置保存为永久配置，以防止其被删除。

步骤1 在CodeArts IDE主工具栏上的配置列表中，选择“**编辑配置...**”。



步骤2 在打开的“运行/调试配置”中，在左侧的配置列表中，选择要保存为永久配置的配置。然后，在配置参数中，将**“Temporary”**取消勾选。



----结束

或者，用户可以在**launch.json**中找到相应的启动配置记录，并为其提供**"temporary": false**属性，例如：

```
{
  "type": "javadbg",
  "name": "Java Class",
  "request": "launch",
  "mainClass": {
```

```
    "name": "com.example.Main",
    "console": "integrated"
  },
  "temporary": false
}
```

调整临时启动配置的限制

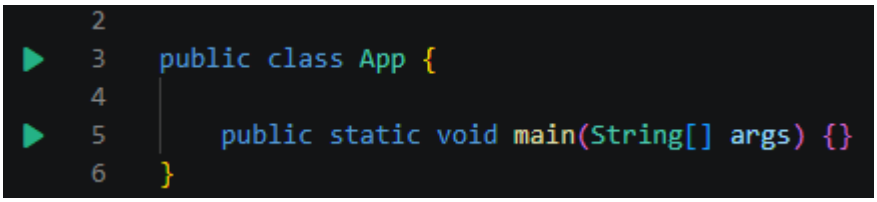
默认情况下，CodeArts IDE根据指定的限制（默认为10）保留临时启动配置的数量，并在超过限制时自动删除最少使用的配置。如果需要，用户可以通过**java.executedConfigurationsLimit**设置来调整此限制。如果设置为零，CodeArts IDE不会创建任何临时启动配置，并删除任何现有的临时启动配置。

2.5.12.2 Java 类

使用以下启动配置来运行应用程序的Main类。

说明

要快速运行一个应用程序而不必手动创建一个启动配置，只需在Main类的代码编辑器中，单击**main()**或类声明旁边的“**运行**”按钮（▶）即可。CodeArts IDE将自动创建相应的**temporary launch configuration**，并在配置列表中显示出来。



启动配置属性

名称	描述
type	描述type调试器的类型。对于运行和调试Java代码，应将其设置为**javadbg**。
name	启动配置名称。
env	额外的环境变量。
skipBuild	跳过程序的构建过程（设置为**true**）或不跳过（设置为**false**）。
temporary	指示启动配置是否为临时的（设置为**true**）还是永久的（设置为**false**）。如果临时启动配置数量超过指定限制，CodeArts IDE会自动删除最不常用的配置。有关详细信息，请参阅**启动配置**。
killPrevSession	终止具有相同名称的先前运行会话（设置为**true**），或中止启动（设置为**false**）。
vmOptions	JVM的额外选项。
name	（在**mainClass**节点下指定）限定类名。
sourcePath	（name的替代）类源文件的路径。用户可以使用**变量**来提供路径。
args	传递给**main()**方法的参数数组（**[arg1，arg2，...]**）。

名称	描述
console	在**Debug Console**（**internal**）或**集成终端**（**integrated**）中显示调试输出。

说明

已知问题是在**调试控制台**或**集成终端**中显示的中文字符显示不正确。

用户可以尝试以下解决方法来修复终端输出：

1. 在Windows控制面板中，转至时钟和区域>区域。
2. 在“管理”选项卡上，单击“更改系统区域设置”。
3. 在打开的“区域设置”对话框中，选择“测试版：使用Unicode UTF-8 获得全球语言支持”。

然后，通过将控制台属性设置为集成，调整启动配置，以使用集成终端而不是调试控制台进行输出。

启动配置示例

用户可以将提供的示例作为一个可工作的启动配置示例。

```
{
   "type": "javadbg",
   "name": "Java Class",
   "request": "launch",
   "mainClass": {
      "name": "com.example.Main",
      "console": "integrated"
   }
}
```

2.5.12.3 JAR 应用

使用此启动配置来运行打包在JAR文件中的应用程序。

启动配置属性

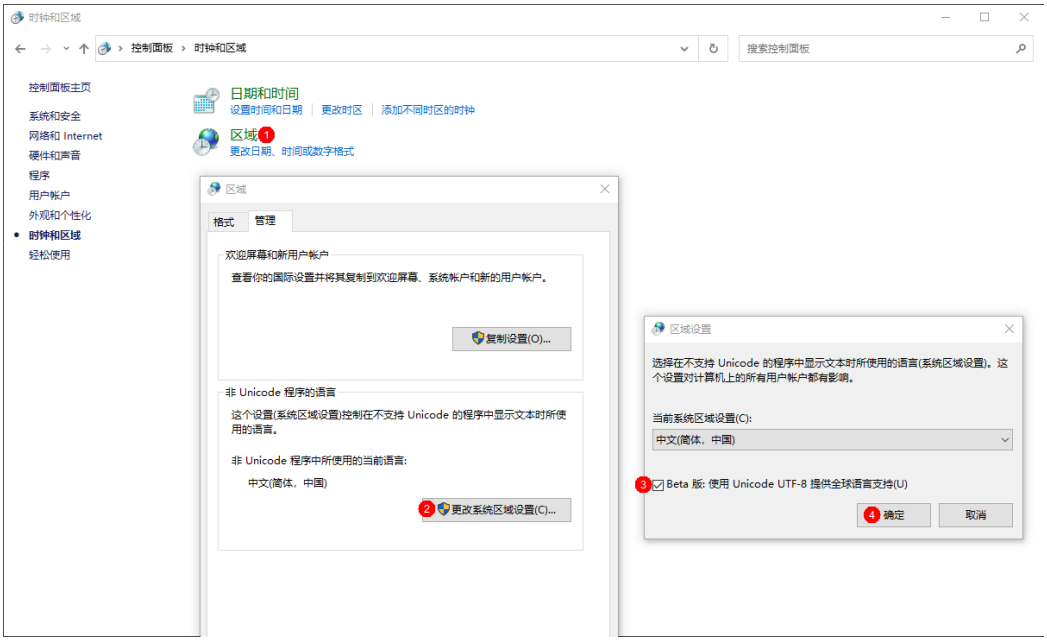
名称	描述
type	描述type调试器的类型。对于运行和调试Java代码，应将其设置为javadbg。
name	启动配置名称。
env	额外的环境变量。
skipBuild	跳过程序的构建过程（设置为**true**）或不跳过（设置为**false**）。
temporary	指示启动配置是否为临时的（设置为**true**）还是永久的（设置为**false**）。如果临时启动配置数量超过指定限制，CodeArts IDE会自动删除最不常用的配置。有关详细信息，请参阅**启动配置**。
killPrevSession	终止具有相同名称的先前运行会话（设置为**true**），或中止启动（设置为**false**）。
vmOptions	JVM的额外选项。
path	JAR文件的路径。用户可以使用变量来提供路径。
console	在**调试控制台**（**internal**）或**集成终端**（**integrated**）中显示调试输出。
args	程序传递的参数。

说明

已知问题是在**调试控制台**或**集成终端**中显示的中文字符显示不正确。

用户可以尝试以下解决方法来修复终端输出：

1. 在Windows控制面板中，转至时钟和区域>区域。
2. 在“管理”选项卡上，单击“更改系统区域设置”。
3. 在打开的“区域设置”对话框中，选择“测试版：使用Unicode UTF-8 获得全球语言支持”。



然后，通过将控制台属性设置为集成，调整启动配置，以使用集成终端而不是调试控制台进行输出。

启动配置示例

用户可以使用提供的示例作为工作的启动配置示例。

```
{
   "type": "javadbg",
   "name": "Jar Application",
   "request": "launch",
   "jar": {
      "path": "${workspaceRoot}/path/to/demo.jar",
      "console": "integrated"
   }
}
```

2.5.12.4 Gradle 任务

使用此启动配置来运行一个或多个Gradle任务。

启动配置属性

名称	描述
type	描述type调试器的类型。对于运行和调试Java代码，应将其设置为javadbg。

名称	描述
name	启动配置名称。
env	额外的环境变量。
skipBuild	跳过程序的构建过程（设置为**true**）或不跳过（设置为**false**）。
temporary	指示启动配置是否为临时的（设置为**true**）还是永久的（设置为**false**）。如果临时启动配置数量超过指定限制，CodeArts IDE会自动删除最不常用的配置。有关详细信息，请参阅**启动配置**。
killPrevSession	终止具有相同名称的先前运行会话（设置为**true**），或中止启动（设置为**false**）。
vmOptions	JVM的额外选项。
scriptArgs	传递给Gradle的参数的数组。
tasks	要运行的Gradle任务。提供一个对象数组，每个对象都有一个**名称**（任务名称）和一个可选的**args**数组，其中包含任务的参数。

启动配置示例

用户可以将提供的示例作为一个可行的启动配置示例。

```
{
   "type": "javadbg",
   "name": "Gradle",
   "request": "launch",
   "skipBuild": true,
   "gradle": {
      "scriptArgs": [
         "--info"
      ],
      "tasks": [
         "build"
      ]
   }
}
```

2.5.12.5 Maven 任务

使用此启动配置来运行一个或多个Maven任务。

启动配置属性

名称	描述
type	描述type调试器的类型。对于运行和调试Java代码，应将其设置为javadbg。
name	启动配置名称。
env	额外的环境变量。
skipBuild	跳过程序的构建过程（设置为**true**）或不跳过（设置为**false**）。

名称	描述
temporary	指示启动配置是否为临时的（设置为**true**）还是永久的（设置为**false**）。如果临时启动配置数量超过指定限制，CodeArts IDE会自动删除最不常用的配置。有关详细信息，请参阅**启动配置**。
killPrevSession	终止具有相同名称的先前运行会话（设置为**true**），或中止启动（设置为**false**）。
vmOptions	JVM的额外选项。
goals	Maven的目标是运行。可以将目标名称作为单个字符串或字符串数组提供。

启动配置示例

用户可以使用提供的示例作为工作的启动配置示例。

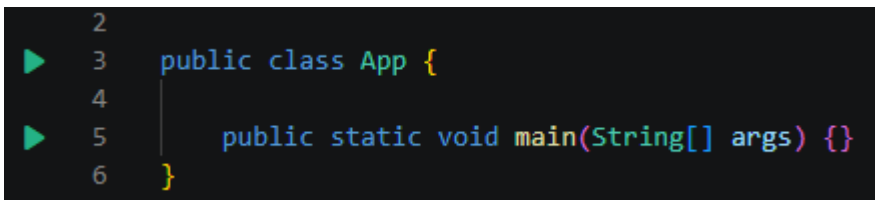
```
{
   "type": "javadbg",
   "name": "Maven",
   "request": "launch",
   "skipBuild": true,
   "maven": {
      "goals": [
         "install"
      ]
   }
}
```

2.5.12.6 JUnit 测试

使用此启动配置来运行JUnit测试。

要快速运行测试而无需手动创建启动配置，请在测试类的代码编辑器中，单击测试类声明旁边的运行按钮（▶）（以运行类中的所有测试），或者单击测试方法（以仅运行单个测试）。要快速运行一个应用程序而不必手动创建一个启动配置，只需在Main类的代码编辑器中，单击**main()**或类声明旁边的**Run**按钮即可。

CodeArts IDE将自动创建相应的**temporary launch configuration**，并在配置列表中显示出来。



启动配置属性

在启动配置中，用户只能指定以下属性之一：**方法（method）**、**类（class）**、**包（package）**或**目录（directory）**，以运行单个测试方法、单个测试类、包中的所有测试或目录中的所有测试。

名称	描述
type	描述type调试器的类型。对于运行和调试Java代码，应将其设置为javadbg。
name	启动配置名称。
env	额外的环境变量。
skipBuild	跳过程序的构建过程（设置为**true**）或不跳过（设置为**false**）。
temporary	指示启动配置是否为临时的（设置为**true**）还是永久的（设置为**false**）。如果临时启动配置数量超过指定限制，CodeArts IDE会自动删除最不常用的配置。有关详细信息，请参阅**启动配置**。
killPrevSession	终止具有相同名称的先前运行会话（设置为**true**），或中止启动（设置为**false**）。
vmOptions	JVM的额外选项。
method	完全限定的测试方法名称。
class	完全限定的测试类名称。
package	测试包名称。
directory	包含测试源代码的目录。默认情况下，此项设置为**${workspaceRoot}/src/test**。用户可以使用**变量**来提供路径。

启动配置示例

用户可以使用提供的示例作为工作启动配置的示例。

运行来自**package.name**包的所有测试：

```
{
    "type": "javadbg",
    "name": "JUnit Test (Package)",
    "request": "launch",
    "jUnit": {
        "package": "package.name"
    },
    "vmOptions": "-ea"
}
```

运行单个测试方法**qualified.method.name**：

```
{
    "type": "javadbg",
    "name": "JUnit Test (Method)",
    "request": "launch",
    "jUnit": {
        "method": "qualified.method.name"
    },
    "vmOptions": "-ea"
}
```

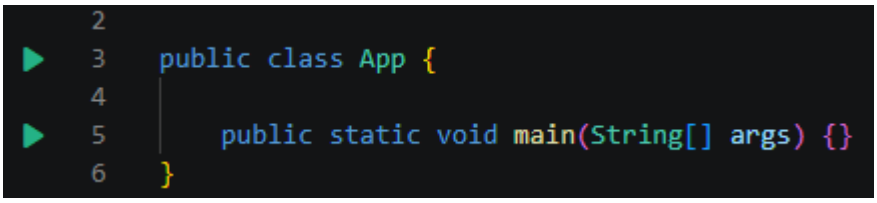
2.5.12.7 TestNG 测试

使用此启动配置来运行JUnit测试。

说明

要快速运行测试而无需手动创建启动配置，请在测试类的代码编辑器中，单击测试类声明旁边的运行按钮（▶）（以运行类中的所有测试），或者单击测试方法（以仅运行单个测试）。要快速运行一个应用程序而不必手动创建一个启动配置，只需在Main类的代码编辑器中，单击**main()**或类声明旁边的**Run**按钮即可。

CodeArts IDE将自动创建相应的**启动配置**，并在配置列表中显示出来。



启动配置属性

在启动配置中，用户只能指定以下属性之一：**方法（method）**、**类（class）**、**包（package）**或**目录（directory）**，以运行单个测试方法、单个测试类、包中的所有测试或目录中的所有测试。

名称	描述
type	描述type调试器的类型。对于运行和调试Java代码，应将其设置为javadbg。
name	启动配置名称。
env	额外的环境变量。
skipBuild	跳过程序的构建过程（设置为**true**）或不跳过（设置为**false**）。
temporary	指示启动配置是否为临时的（设置为true）还是永久的（设置为**false**）。如果临时启动配置数量超过指定限制，CodeArts IDE会自动删除最不常用的配置。有关详细信息，请参阅**启动配置**。
killPrevSession	终止具有相同名称的先前运行会话（设置为**true**），或中止启动（设置为**false**）。
vmOptions	JVM的额外选项。
method	完全限定的测试方法名称。
class	完全限定的测试类名称。
package	测试包名称。
directory	包含测试源代码的目录。默认情况下，此项设置为**${workspaceRoot}/src/test**。用户可以使用**变量**来提供路径。

启动配置示例

用户可以使用提供的示例作为工作启动配置的示例。

运行来自**package.name**包的所有测试：

```
{
    "type": "javadbg",
```

```
    "name": "TestNG Test (Package)",
    "request": "launch",
    "testNG": {
        "package": "package.name"
    },
    "vmOptions": "-ea"
}
```

运行单个测试方法**qualified.method.name**：

```
{
    "type": "javadbg",
    "name": "TestNG Test (Method)",
    "request": "launch",
    "testNG": {
        "method": "qualified.method.name"
    },
    "vmOptions": "-ea"
}
```

2.5.12.8 远程调试

使用此启动配置可以通过连接到远程JVM或使调试器监听传入连接来进行远程调试。

启动配置属性

名称	描述
type	描述type调试器的类型。对于运行和调试Java代码，应将其设置为javadbg。
name	启动配置名称。
env	额外的环境变量。
skipBuild	跳过程序的构建过程（设置为**true**）或不跳过（设置为**false**）。
temporary	指示启动配置是否为临时的（设置为**true**）还是永久的（设置为**false**）。如果临时启动配置数量超过指定限制，CodeArts IDE会自动删除最不常用的配置。有关详细信息，请参阅**启动配置**。
killPrevSession	终止具有相同名称的先前运行会话（设置为**true**），或中止启动（设置为**false**）。
debuggerMode	调试器模式，可以设置为**attach**（连接到远程JVM）或listen（监听传入连接）。默认情况下，使用attach模式。
autoRestart	仅在**debuggerMode**设置为**listen**时可用，定义调试器在远程JVM断开连接后是否自动重启。默认情况下，使用**false**。
useSocketTransport	定义是否使用套接字传输来连接进程。默认情况下，使用**true**。否则，当设置为**false**时，使用共享内存。
host	主机应用程序运行的机器的地址。默认情况下，使用**127.0.0.1**。
port	目标机器上的连接端口。默认情况下，使用**5005**。

启动配置示例

用户可以使用提供的示例作为远程调试场景的示例。

主机应用程序的启动配置是一个常规的**Java类**配置。它应该包含在**vmOptions**下提供的特殊参数，以便应用程序使用调试代理启动，并且调试器能够连接到它。

```
{
   "type": "javadbg",
   "name": "Java Class",
   "request": "launch",
   "vmOptions": "-agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=127.0.0.1:5005",
   "mainClass": {
      "name": "com.example.App",
      "console": "internal"
   }
}
```

远程调试的启动配置应该使用提供给主机应用程序启动配置的连接参数。

```
{
   "type": "javadbg",
   "name": "Remote Debug (Attach to remote JVM)",
   "request": "launch",
   "skipBuild": true,
   "remote": {
      "debuggerMode": "attach",
      "useSocketTransport": true,
      "host": "127.0.0.1",
      "port": "5005"
   }
}
```

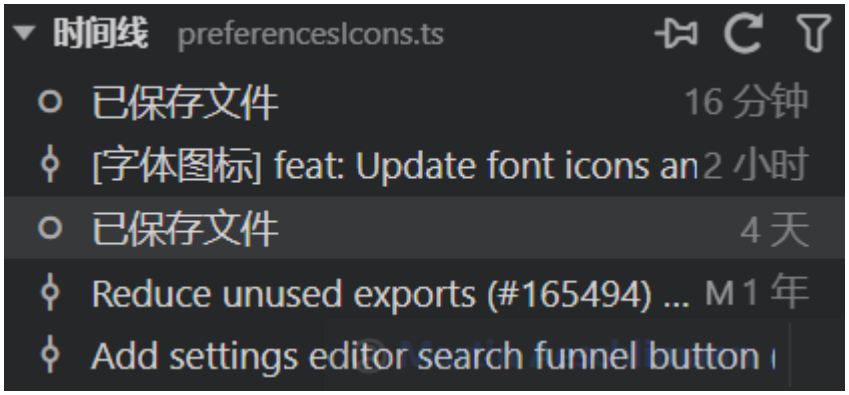
2.6 版本控制

2.6.1 简介

CodeArts IDE支持同时处理多个源代码管理（SCM）提供商，其中**Git**支持已经内置。CodeArts IDE还维护了**本地历史记录**，即使用户的项目没有关联的SCM提供商，也可以跟踪文件的更改。

时间线视图

“时间线”视图可以在“资源管理器”右侧“视图和更多操作...”（ ··· ）下，它提供了一个文件的时间序列事件的统一视图。在事件列表中，本地文件事件标记为○，与版本控制系统相关的事件标记为 。

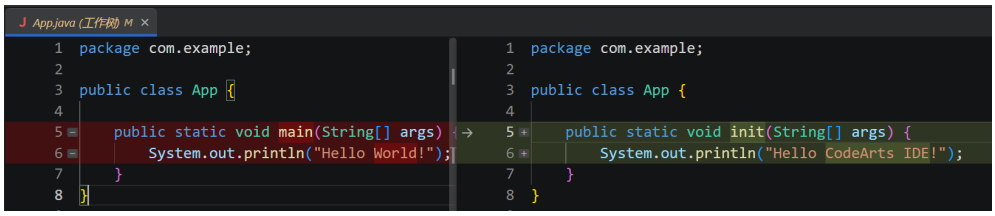


用户可以过滤“时间线”事件列表，只查看来自版本控制系统或本地历史记录的事件。在“时间线”视图工具栏中，单击**“过滤时间线”**按钮（ ）并选择所需的视图选项。

“时间线”视图显示当前在代码编辑器中打开的文件的事件，并在编辑器选项卡之间切换时自动切换文件。要禁用自动切换并始终显示某个文件的事件，请切换到其他编辑器选项卡，然后单击“时间线”视图工具栏中的**“固定当前时间线”**按钮（ ）。

差异查看器

CodeArts IDE提供了一个内置的差异查看器。



说明

首先，用户可以通过在资源管理器或打开的编辑器列表中右键单击文件，然后选择“**选择文件以比较**”，然后再右键单击要与之比较的第二个文件，并选择“**与已选项目进行比较**”来比较任意两个文件。或者，按下“Ctrl+Shift+P”或双击“Ctrl”，然后选择“**文件：比较活动文件与...**”，用户将看到最近文件的列表。

差异的默认视图是并排视图。通过单击右上角的更多操作（ ）按钮，然后选择**“内联视图”**来切换到内联视图。如果需要使用内联视图，可以选择勾选或取消勾选“diffEditor.renderSideBySide”配置项。

2.6.2 本地历史

使用本地历史记录，CodeArts IDE可以记录文件上执行的事件历史。用户可以查看和恢复文件在任何事件发生时的内容。

要访问文件的本地历史记录，请在“资源管理器”右侧“视图和更多操作...”（ ）下选择“时间线”视图。每次修改后保存文件时，都会向**“时间线”**事件列表中添加一个新记录。

查看事件引入的更改

用户可以使用CodeArts IDE差异查看器来检查每个事件引入的更改。在“时间线”视图中，右键单击一个事件，然后从上下文菜单中选择所需的操作。

- **与文件进行比较**：将文件内容与文件的当前状态进行比较。
- **与上一个版本比较**：比较此事件和上一个事件之间的文件内容。
- **选择以进行比较**：选择要进行比较的事件。选择了一个事件后，右键单击要将文件内容与之比较的另一个事件，然后从上下文菜单中选择“与已选项目进行比较”。

查看和恢复文件内容

用户可以查看特定事件发生时的文件内容，并将其重新应用于文件的当前状态。在“时间线”视图中，右键单击一个事件，然后从上下文菜单中选择所需的操作。

- **显示内容**：在单独的只读编辑器选项卡中查看事件发生时的文件内容。
- **还原内容**：重新应用文件内容，从而覆盖所有后续更改。请注意，这将丢弃此文件的任何未保存更改。

2.6.3 GIT 支持

2.6.3.1 简介

虽然这个章节讲述的是Git，但大多数源代码控制界面和工作流程在其他源代码管理系统中也是通用的。

在使用源代码管理（SCM）功能之前，CodeArts IDE将利用用户机器上的Git，因此用户需要安装Git。

说明

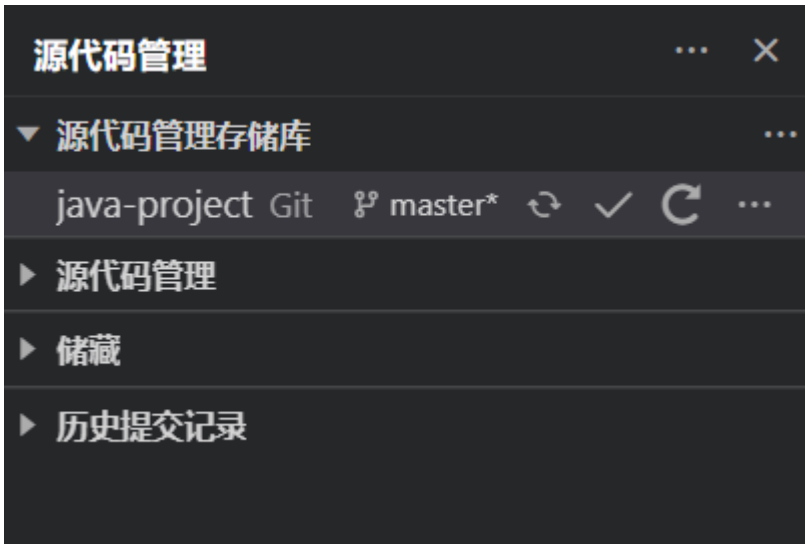
请确保安装的是2.0.0及以上的GIT版本。

2.6.3.2 访问源代码管理功能

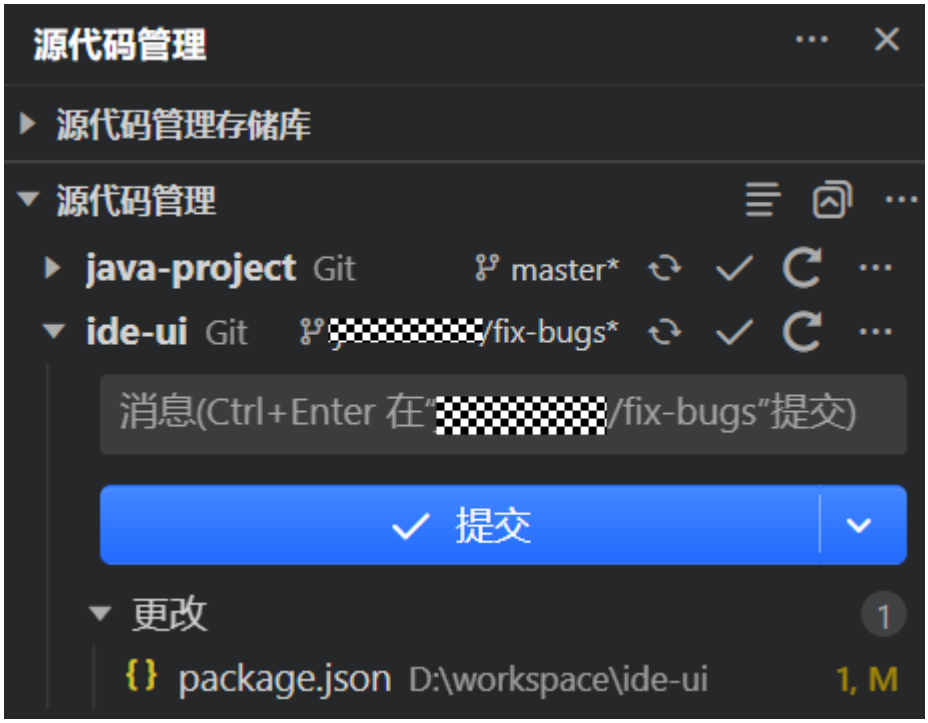
大多数与源代码管理相关的操作可以在“源代码管理”视图中执行。要打开它，请执行以下任一操作：

- 单击左侧活动栏中的**“源代码管理”**按钮（ ）。
- 在主菜单中，选择“查看 > 源代码管理”。
- 按**“Ctrl+Shift+G”**或**“Alt+9”**(IDEA快捷键方案)。

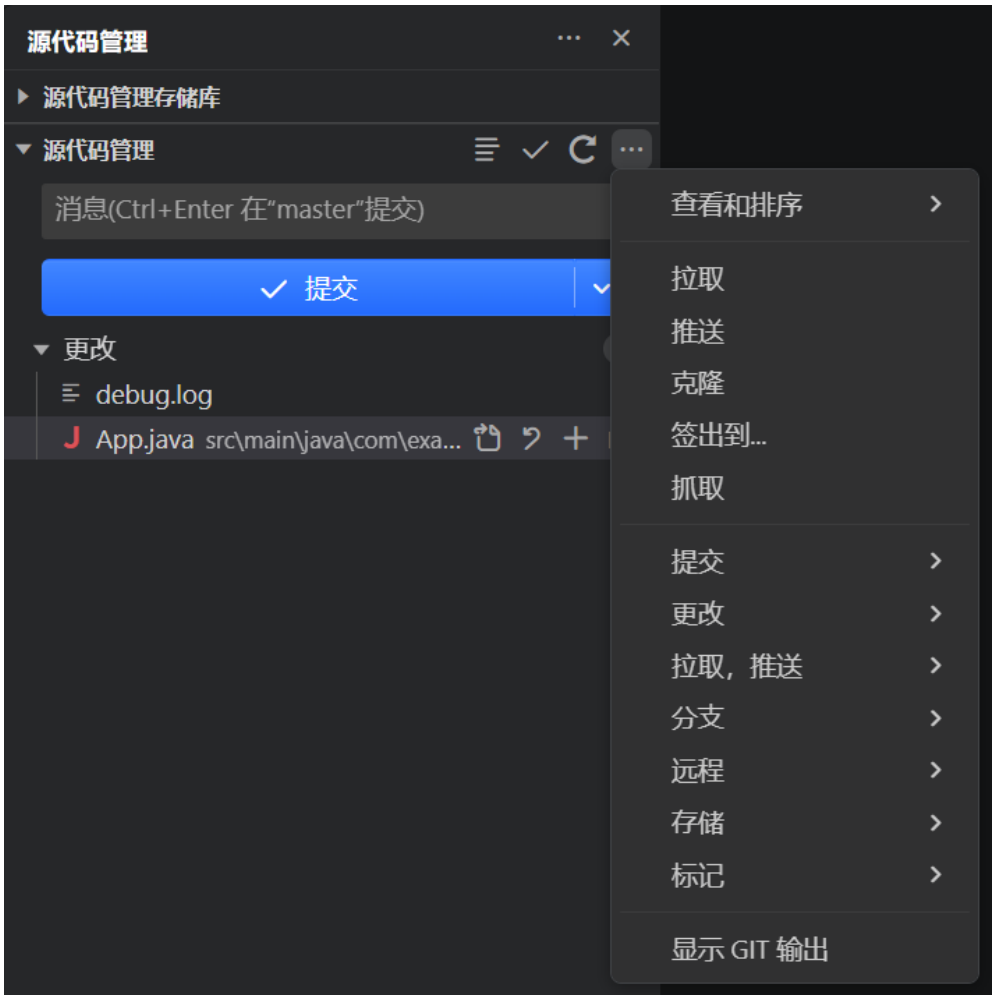
“源代码管理存储库”部分列出了当前工作区中检测到的存储库。



“源代码管理”部分显示了打开的存储库中的更改。用户可以通过选择特定的存储库来限定更改的显示。

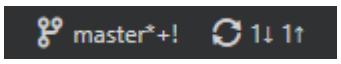


在源代码管理部分的**“更多操作”**（···）菜单中，用户可以快速访问大多数与**“源代码管理”**相关的操作。



CodeArts IDE Git 状态栏操作

CodeArts IDE状态栏允许用户运行**“同步更改”**命令，将远程更改拉取到本地仓库，然后将本地更改的代码提交推送到上游分支。用户还可以通过状态栏创建新分支并在分支之间切换。



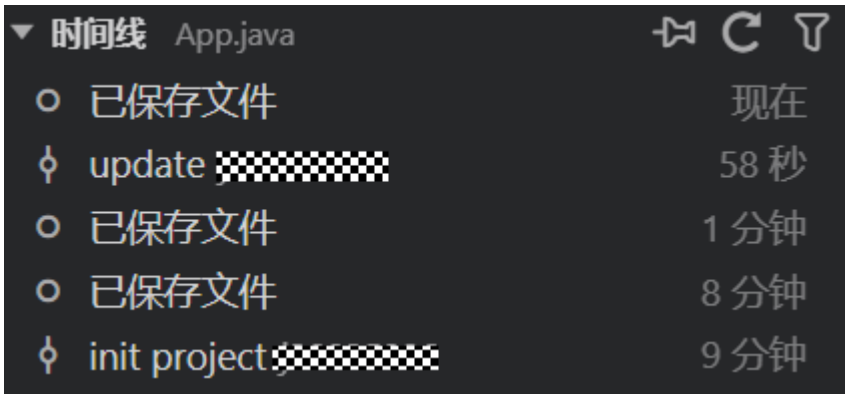
装订线指示器

当用户对一个被纳入源代码控制的文件进行更改时，CodeArts IDE会在装订线和概览标尺上添加注释。

- 红色三角形()表示已删除的行。
- 绿色条()表示新增的行。
- 蓝色条()表示修改的行。

时间线视图

“时间线”视图在“资源管理器”右侧“视图和更多操作...”（ ··· ）下，它是用于可视化当前在代码编辑器中打开的文件的时间序列事件的统一视图。

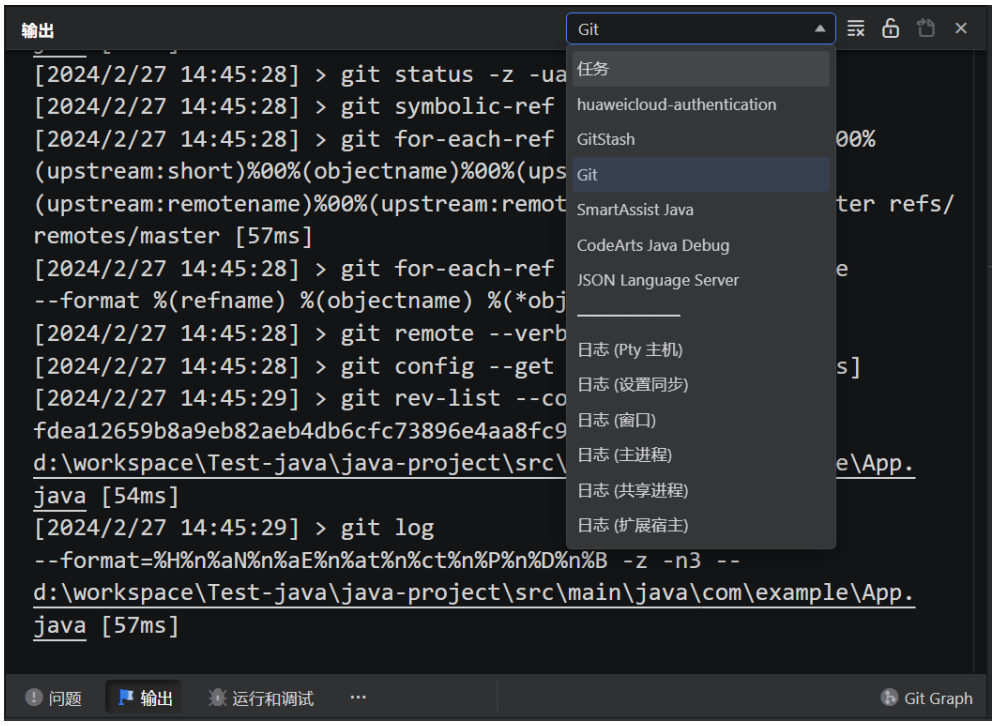


- 双击提交以打开引入该提交的更改的差异视图。
- 右键单击提交以获取复制提交ID和复制提交消息的选项。

查看 Git 输出

用户始终可以查看当前执行的Git命令的详细信息。

要打开Git输出窗口，请运行“视图 > 输出”或使用“**查看: 切换输出**”命令（“Ctrl+Shift+U”）。然后从列表中选择“**Git**”。

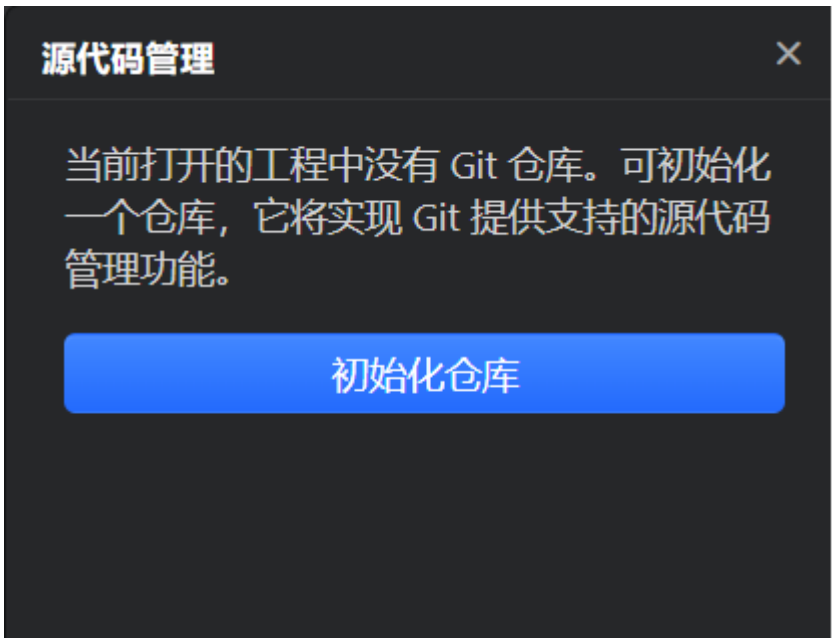


2.6.3.3 管理存储库

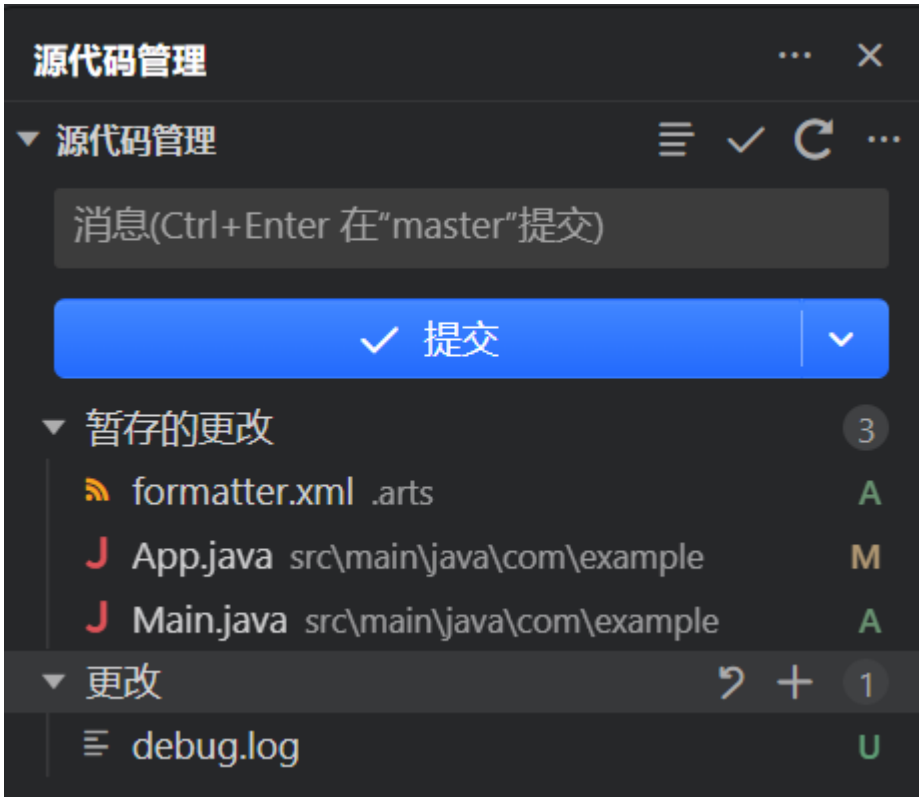
2.6.3.3.1 初始化存储库

当用户打开一个本地文件夹时，可以通过在其中初始化一个Git存储库来启用Git源代码控制。

- 打开“源代码管理”视图（按**“Ctrl+Shift+G”**或**“Alt+9”**(IDEA快捷键方案)）。
- 单击**“初始化仓库”**。

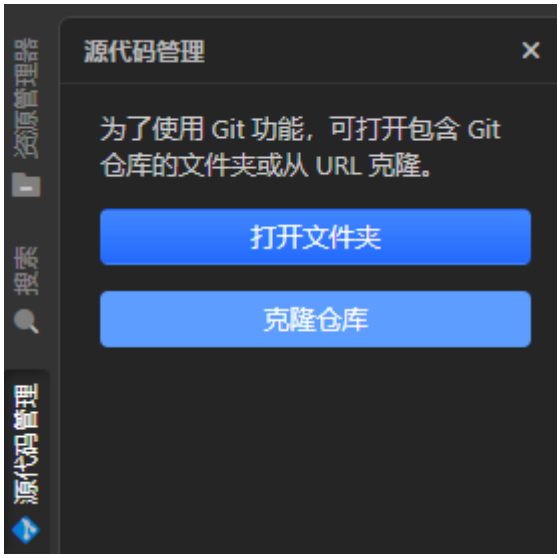


CodeArts IDE将创建必要的Git存储库元数据文件，并将工作区文件显示为未跟踪的更改，准备进行暂存。



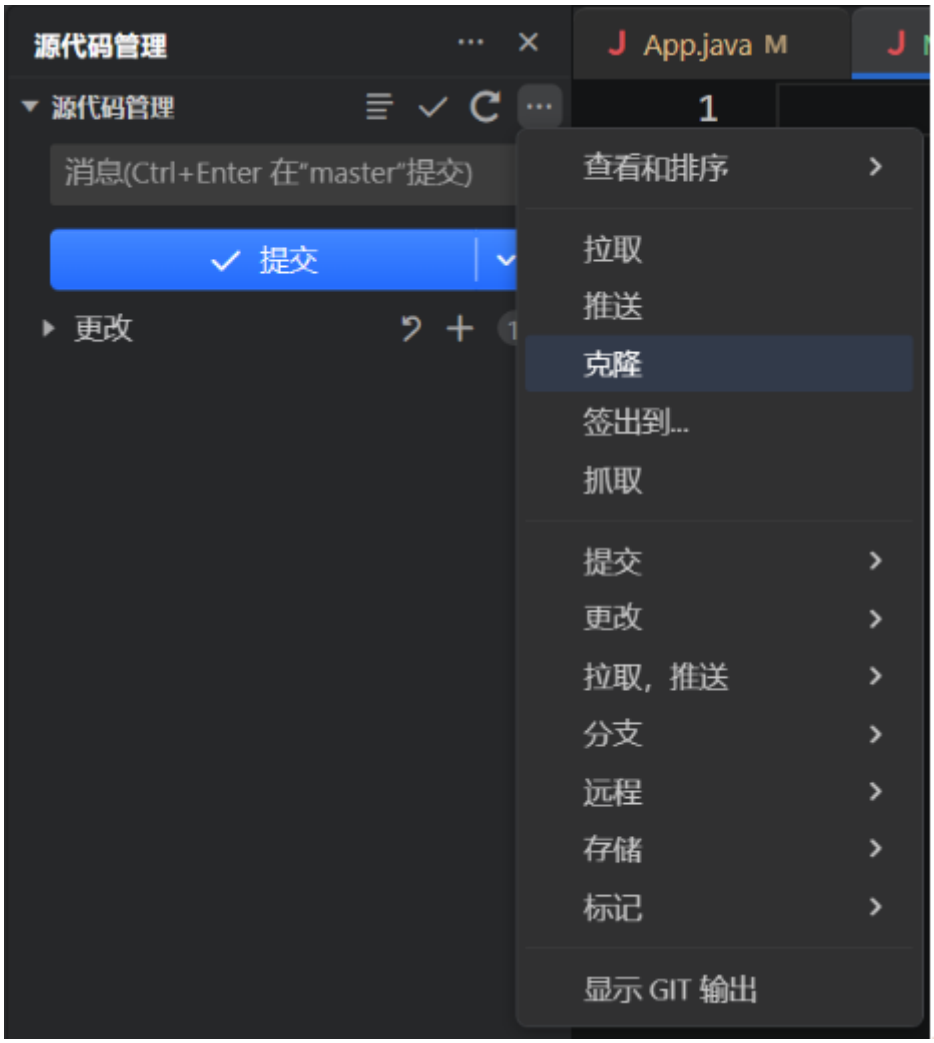
2.6.3.3.2 克隆现有存储库

如果尚未打开任何文件夹，则“源代码管理”视图可让用户打开本地文件夹或克隆存储库。

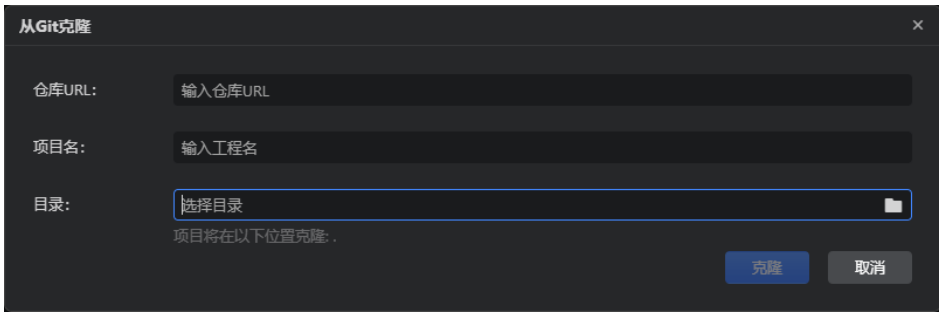


如果已经打开了某个文件夹，请按以下步骤克隆存储库。

1. 在“源代码管理”视图中，展开**“源代码管理”**部分。
2. 单击**“更多操作”**按钮（ ··· ）并选择**“克隆”**。



3. 在打开的弹出窗口中，提供远程存储库的URL和存储目录，单击“**克隆**”。



4. 存储库克隆完成后，CodeArts IDE会提示用户打开它。

2.6.3.3.3 管理远程仓库

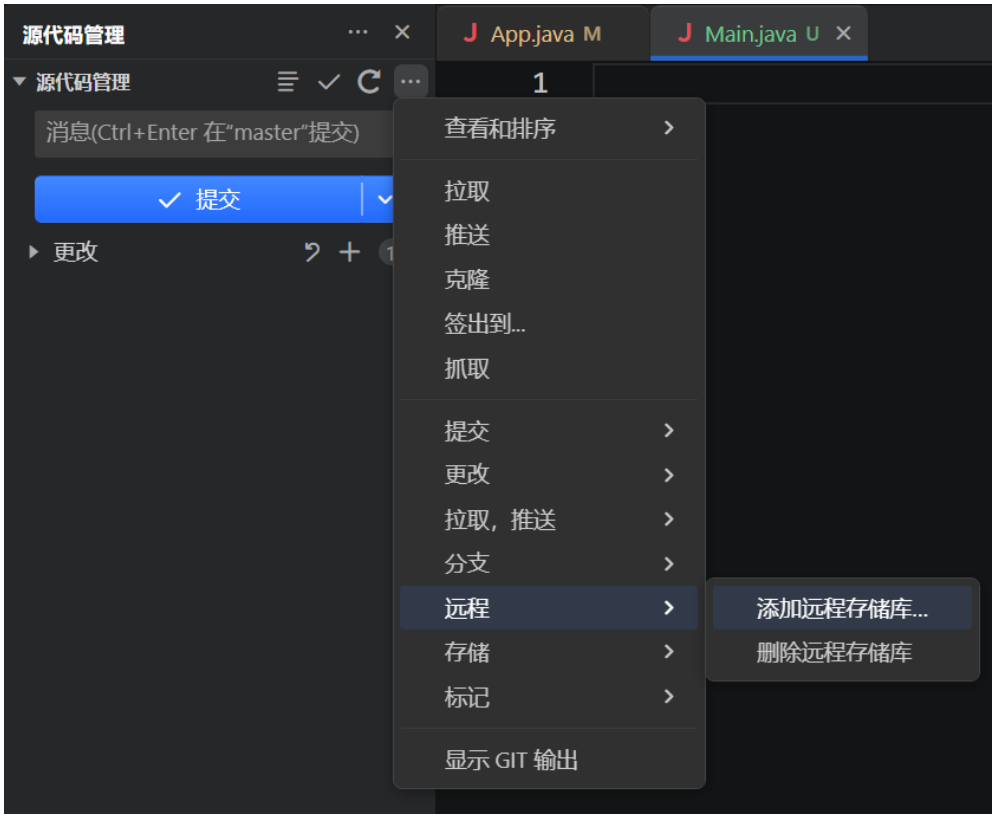
在创建本地Git仓库之后，用户可以添加一个远程仓库，以便能够在Git项目上进行协作。如果用户已经克隆了一个远程Git仓库，那么与原始克隆位置链接的远程仓库将自动配置，但用户可以添加任意多个远程仓库。

说明

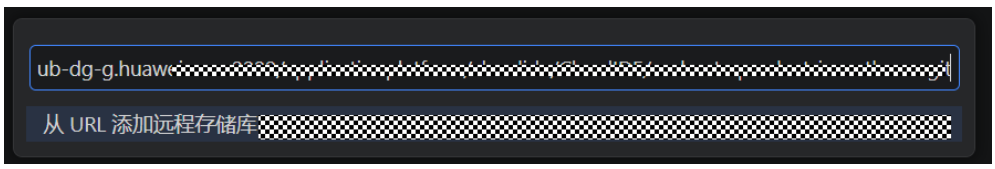
提示：用户应该设置一个凭据助手，以避免每次CodeArts IDE与Git远程仓库通信时都被要求输入凭据。否则，可以考虑通过git.autofetch设置禁用自动获取以减少身份验证提示的次数。

添加远程仓库

1. 添加远程仓库在用户选择的SCM托管平台上创建一个空仓库。
2. 在“源代码管理”视图中，展开**“源代码管理”**部分。
3. 单击要添加新的远程仓库的仓库旁边的**“更多操作”**按钮（ ），然后选择“远程 > 添加远程存储库...”。



4. 在打开的弹出窗口中，提供远程仓库的URL并按Enter键。



移除远程仓库

1. 在“源代码管理”视图中，展开**“源代码管理”**部分。

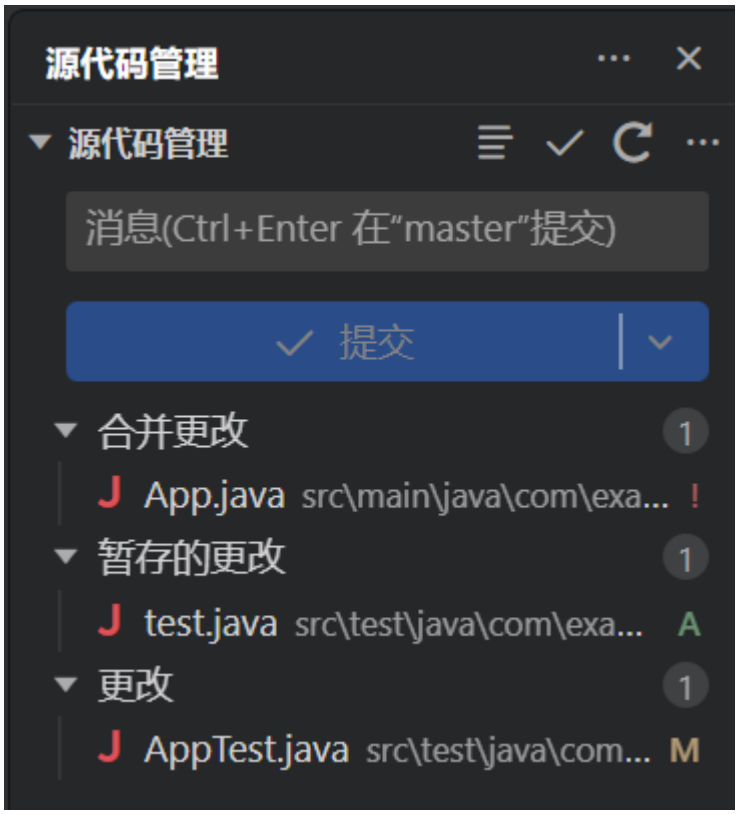
2. 单击要移除远程仓库的仓库旁边的“**更多操作**”按钮（···），然后选择“远程 > 删除远程存储库”。
3. 在打开的弹出窗口中，选择要移除的远程仓库并按Enter键。

2.6.3.4 管理版本控制下的文件

2.6.3.4.1 简介

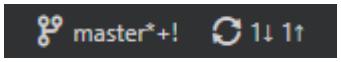
当用户的项目与源代码管理（SCM）系统关联时，CodeArts IDE会跟踪项目文件中发生的所有更改。左侧活动栏中的“**源代码管理**”按钮（）显示用户当前在存储库中拥有的更改数量。

“源代码管理”视图显示当前存储库更改的详细信息，分为“**更改**”、“**暂存的更改**”和“**合并更改**”三个组。



单击每个项目将详细显示每个文件中的文本更改。请注意，对于未暂存的更改，右侧的编辑器仍然允许用户编辑文件。

CodeArts IDE提供了存储库状态的指示：当前分支、脏状态指示以及当前分支中传入和传出提交的数量。用户可以在“**源代码管理**”部分的存储库记录旁边或CodeArts IDE状态栏中查看它们。



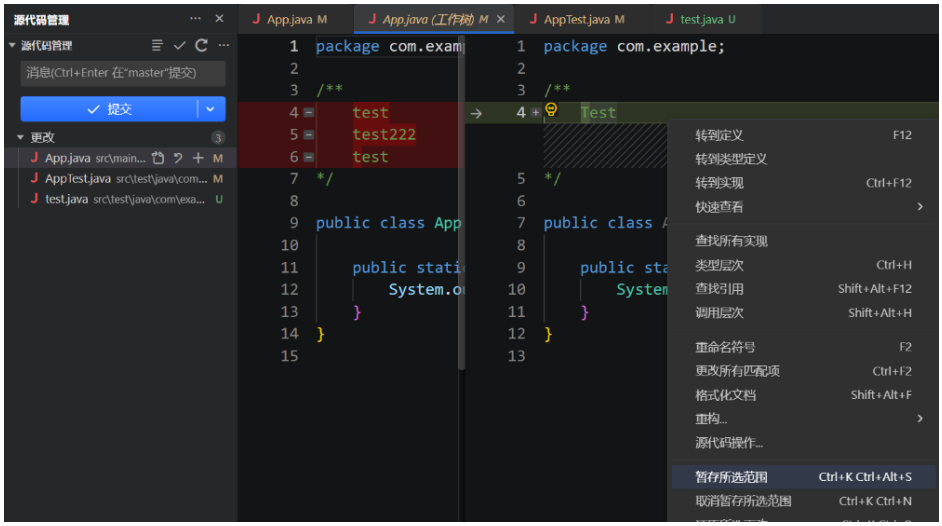
2.6.3.4.2 提交

当用户对代码进行一些更改时，用户需要将它们提交到本地项目存储库，然后将它们推送到远程存储库，以便团队成员可以使用。

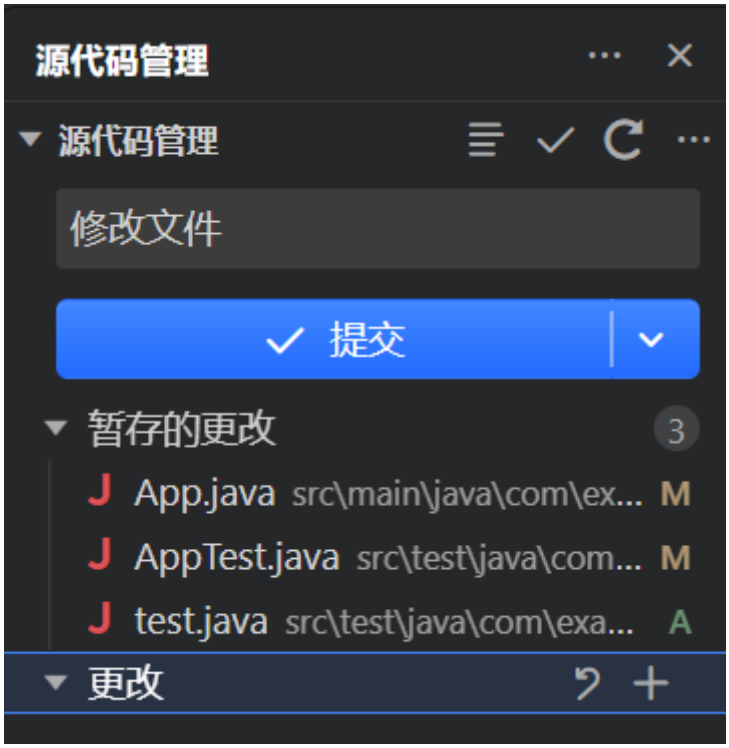
说明

在提交之前，请确保用户的Git配置中设置了用户名和/或电子邮件。否则，Git将使用本地计算机上的信息。用户可以在Git提交信息中找到详细信息。

1. 通过将更改添加到暂存区来准备提交。要执行此操作，请在源代码控制视图的更改部分中执行以下操作之一。
   - 要暂存整个文件，请单击**“暂存更改”**按钮（+），或右键单击文件并选择**“暂存更改”**。
   - 要暂存文件的一部分，请双击文件以打开差异视图，该视图提供更改的概述。选择要暂存的更改，右键单击并选择**“暂存所选范围”**，或先按下“Ctrl+K”再按下“Ctrl+Alt+S”快捷键触发。



2. 在“源代码管理”视图中，在字段中输入提交消息，然后单击**“提交”**按钮或按**“Ctrl+Enter”**。



要撤销提交，请单击“**更多操作**”按钮（···）并选择“**撤销上次提交**”。更改将重新添加到“**暂存的更改**”部分。

2.6.3.4.3 获取、拉取和推送更改

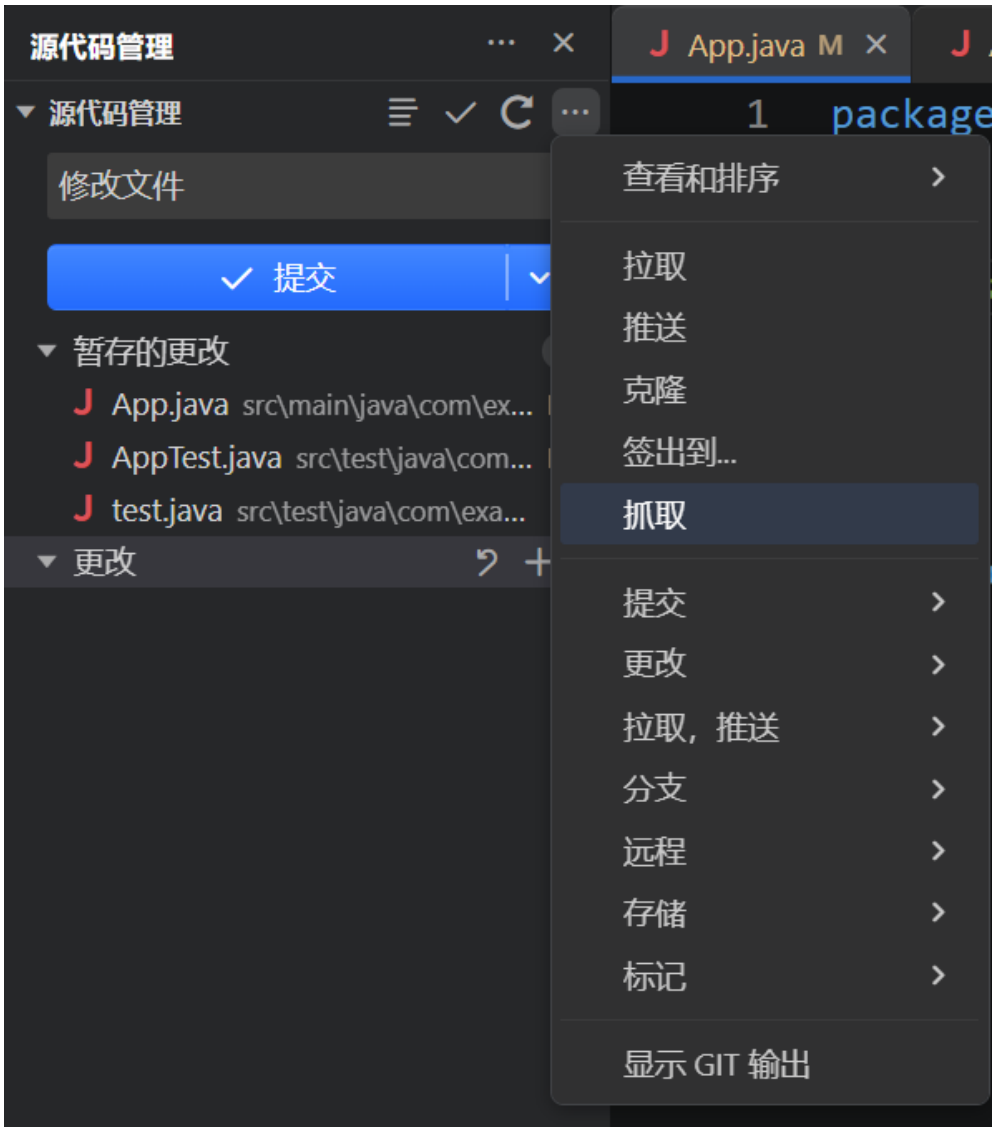
当用户的存储库连接到远程，并且用户的检出分支与远程的分支有上游链接时，CodeArts IDE允许用户推送、拉取和同步（拉取后紧接着推送）该分支。

获取

从远程获取更改可以查看本地存储库相对于远程的超前或落后情况。这些更改本身不会合并到本地工作树中。CodeArts IDE可以执行自动定期获取。此功能默认禁用，但用户可以通过git.autofetch设置启用它。

手动获取远程更改的方法如下：

- 在“源代码管理”视图中，展开“源代码管理存储库”部分。
- 单击要获取更改的存储库旁边的“**更多操作**”按钮（···），然后选择“**抓取**”。



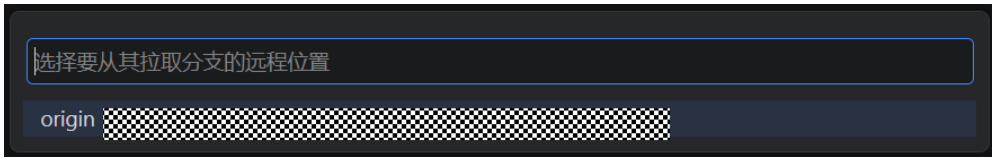
如果用户配置了多个远程，可以通过选择“拉取，推送 > 从所有远程存储库中拉取”来从所有远程获取更改。如果用户在源代码控制托管上删除了一个远程分支，它仍然

会在CodeArts IDE中可见。要清理这样的孤立分支，请选择“拉取，推送 > 获取（删除）”。

拉取

运行拉取命令时，CodeArts IDE会从远程存储库获取更改并将其集成到本地工作树中。

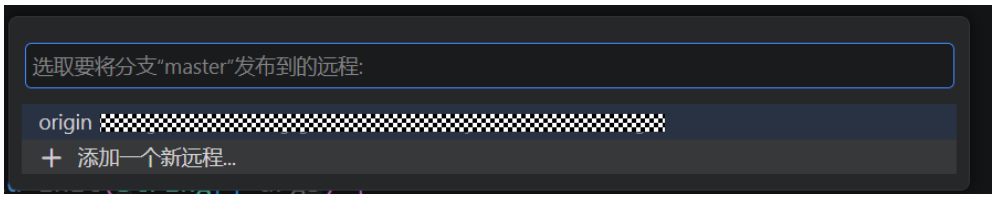
1. 在“源代码管理”视图中，展开**“源代码管理”**部分。
2. 单击要将更改拉取到的存储库旁边的**“更多操作”**按钮（···），然后执行以下操作之一：
   - 要将更改从远程跟踪分支拉取到当前本地分支，请选择**“拉取”**，或按**“Ctrl+T”**（IDEA快捷键方案）。
   - 要拉取更改并同时将本地未推送的更改rebase到已拉取的更改上，请选择“拉取，推送 > 拉取（变基）”。
   - 要从不同配置的远程存储库拉取更改，请选择“拉取，推送 > 拉取自”。然后在打开的弹出窗口中选择所需的远程存储库。



推送

在本地**提交**更改后，用户需要运行推送命令将其上传到远程存储库。

1. 在**“源代码管理”**视图中，展开**“源代码管理”**部分。
2. 单击要推送更改的存储库旁边的**“更多操作”**按钮（···），然后执行以下操作之一：
   - 要将更改从当前本地分支推送到远程跟踪分支，请选择**“推送”**。
   - 要将更改推送到不同配置的远程存储库，请选择“拉取，推送 > 拉取到”。然后在打开的弹出窗口中选择所需的远程存储库。



2.6.3.4.4 Stash 存储

使用stash存储，用户可以将当前更改移动到临时位置，而无需将其提交，从而将工作副本恢复到“干净”（即HEAD提交）状态。

要存储更改，请执行以下操作：

1. 在“源代码管理”视图中，展开**“源代码管理”**部分。
2. 单击要存储更改的存储库旁边的**“更多操作”**按钮（···），指向**“存储”**，然后执行以下操作之一：
   - 要存储已暂存的更改，请选择**“储藏”**。

- 要存储所有更改，包括未暂存和未版本化的文件，请选择**“储藏（包含未跟踪）”**。

移动更改后，用户可以随时重新应用它们到用户的工作副本中。

1. 在“源代码管理”视图中，展开**“源代码管理”**部分。
2. 单击要重新应用更改的存储库旁边的**“更多操作”**按钮（···），指向**“存储”**，然后执行以下操作之一：
   - 要应用最近的存储，请选择“应用最新储藏”。如果用户还想从存储堆栈中删除已应用的存储，请选择“删除最新储藏”。
   - 要应用任意存储，请选择“应用储藏”，然后在打开的弹出窗口中选择所需的存储。如果用户还想从存储堆栈中删除已应用的存储，请选择**“删除储藏”**。

用户可以清理存储堆栈以删除不再需要的存储。

1. 在“源代码管理”视图中，展开**“源代码管理”**部分。
2. 单击要重新应用更改的存储库旁边的**“更多操作”**按钮（···），指向**“存储”**，然后执行以下操作之一：
   - 要删除任意存储，请选择“删除储藏”，并在打开的弹出窗口中选择所需的存储。
   - 要删除所有存储，请选择“删除所有储藏”。

2.6.3.5 管理分支

2.6.3.5.1 创建/切换分支

CodeArts IDE可以方便地处理Git分支，让用户创建和切换分支，并将一个分支的更改合并到另一个分支中。

创建分支

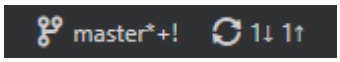
1. 在“源代码管理”视图中，展开**“源代码管理”**部分。
2. 单击要在其中创建新分支的存储库旁边的**“更多操作”**按钮（···），指向**“分支”**，然后执行以下操作之一：
   - 要从当前正在工作的分支创建新分支，请选择**“创建分支”**，并在打开的弹出窗口中提供新分支的名称，然后按“Enter”键。
   - 要从存储库中的其他分支创建新分支，请选择**“从现有来源创建新的分支”**，并在打开的弹出窗口中选择源分支。

然后在打开的弹出窗口中提供新分支的名称，然后按“Enter”键。

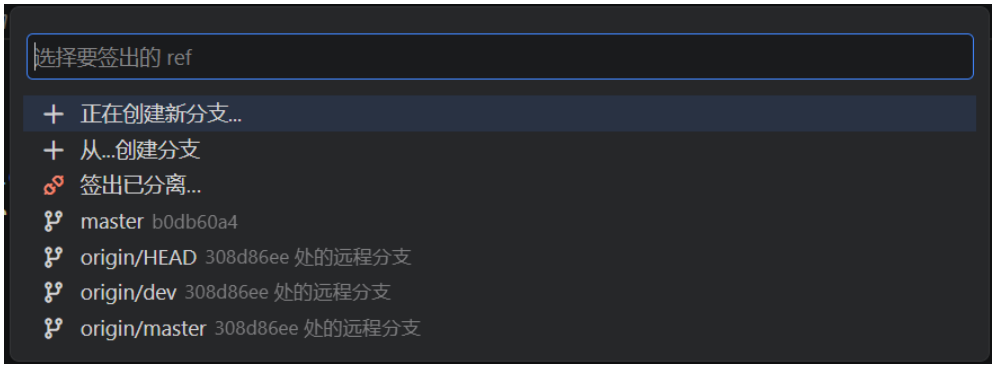
CodeArts IDE会自动创建一个新分支并切换到该分支。如果Git存储库已设置远程，可以在“**源代码管理**”部分或CodeArts IDE状态栏中单击“**发布**”按钮（）将当前分支发布到远程。

切换分支

1. 执行以下操作之一：
   - 在“源代码管理”视图中，展开“**源代码管理**”部分，单击要切换到另一个分支的存储库旁边的“**更多操作**”按钮（），然后选择“**签出到**”。
   - 在CodeArts IDE状态栏中单击分支名称。

2. 在打开的弹出窗口中，选择要切换到的分支，然后按Enter键。如果选择了一个尚不存在本地分支的远程分支，则CodeArts IDE将自动创建本地分支。

说明

用户还可以通过“**签出到**”弹出窗口创建新的本地分支。

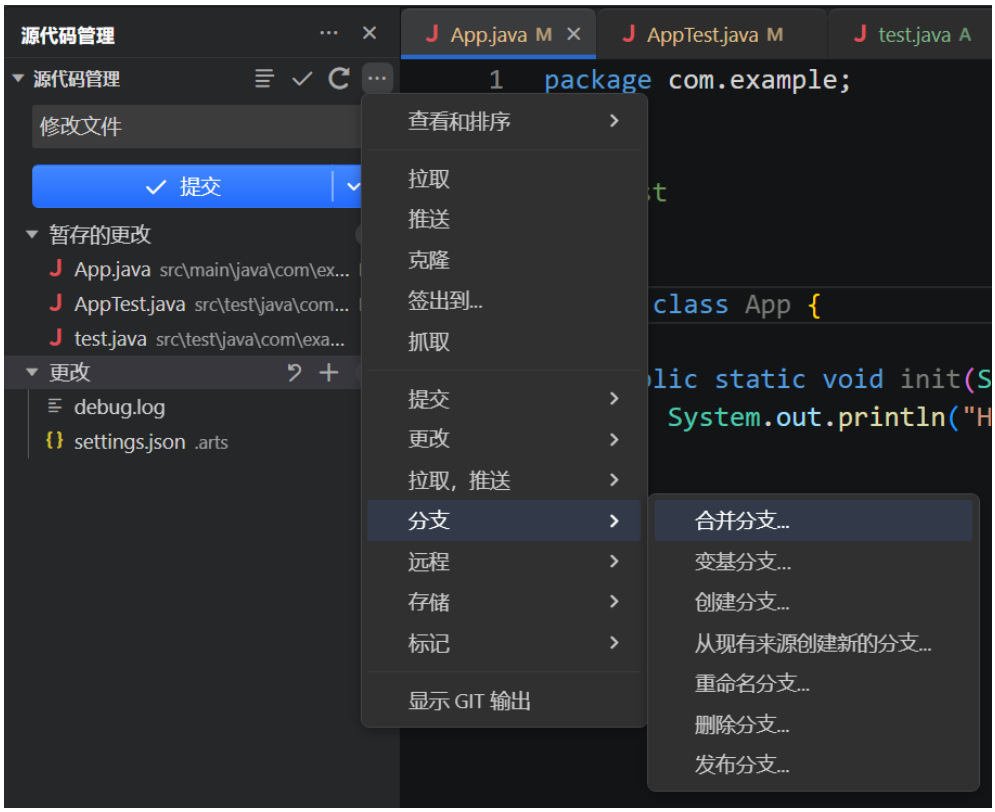
2.6.3.5.2 应用分支之间的更改

CodeArts IDE可以通过使用合并（Merge）和变基（Rebase）命令在Git分支之间应用代码更改。

合并

Merge命令允许用户将源分支的更改集成到目标分支的HEAD中。Git会创建一个新的提交（称为“合并提交”），将源分支和目标分支从两个分支分叉点开始的更改合并在一起。

1. 切换到目标分支，即想要将更改合并到的分支。有关详细信息，请参阅切换分支。
2. 在“源代码管理”视图中，展开“**源代码管理**”部分。
3. 单击要将一个分支的更改合并到另一个分支的存储库旁边的“**更多操作**”按钮（），指向“**分支**”，然后选择“**合并分支**”。

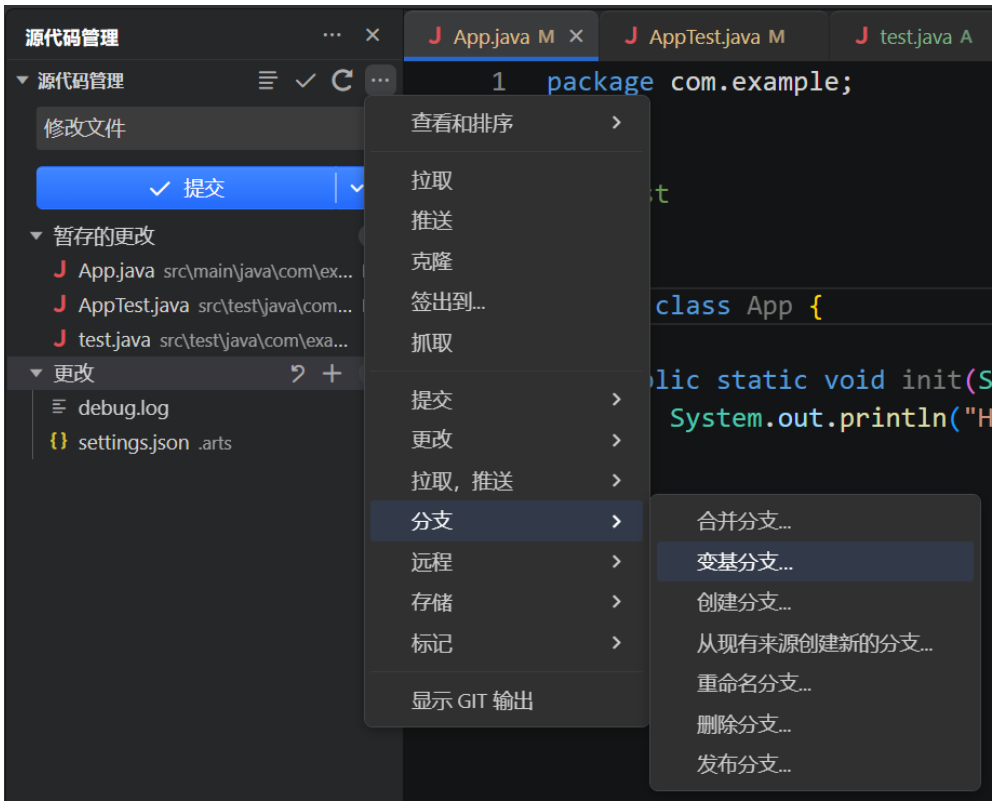


在打开的弹出窗口中，选择要从中合并更改的分支。如果发生合并冲突，请按照**解决合并冲突**中描述的方法解决它。

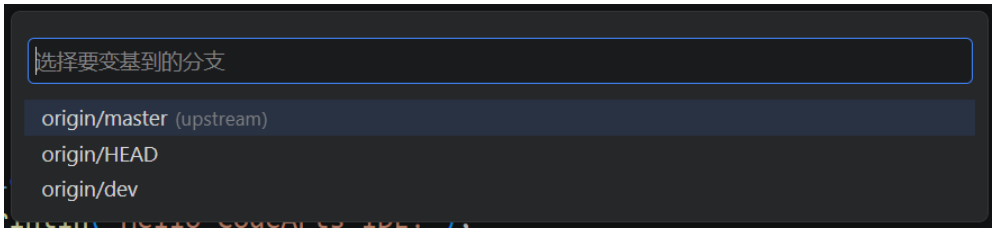
变基

Rebase命令允许用户将源分支的提交应用到目标分支的HEAD提交之上。

1. 切换到源分支，即想要将其提交应用到另一个分支上的分支。有关详细信息，请参阅**切换分支**。
2. 在“源代码管理”视图中，展开**“源代码管理”**部分。
3. 单击要将一个分支的更改合并到另一个分支中的存储库旁边的**“更多操作”**按钮（ ），指向**“分支”**，然后选择**“变基分支”**。



4. 在打开的弹出窗口中，选择用户要将更改应用到的目标分支。

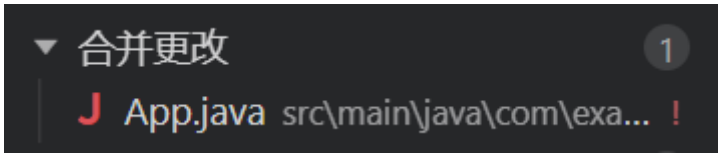


解决合并的冲突

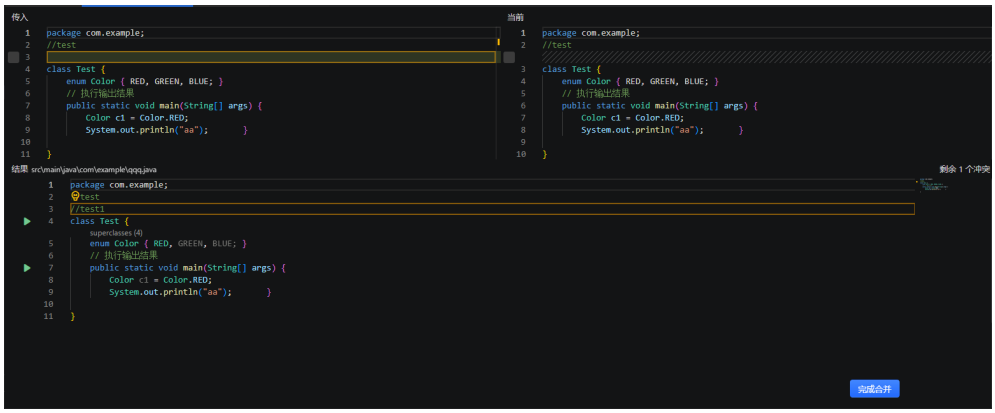
在某些情况下，用户在本地对文件所做的更改可能与其他人对同一文件所做的更改冲突。另一个常见的原因是将一个分支**合并到另一个分支**。CodeArts IDE会识别这种合并冲突并显示相应的通知。

解决合并冲突的步骤如下：

1. 在源代码控制视图的合并更改部分，找到包含冲突更改的文件。



双击该文件，在代码编辑器中打开它，进入专门的冲突视图，通过**差异查看器**详细查看更改。



一旦冲突解决完毕，用户可以将冲突的文件暂存并**提交**更改。

2.6.3.6 CodeArts IDE 作为 Git 编辑器

当用户从**命令行启动CodeArts时**，用户可以传递--wait参数，使启动命令等待直到用户关闭新的CodeArts实例。这在将CodeArts配置为Git的外部编辑器时非常有用，这样Git就会等待用户关闭启动的CodeArts实例。

1. 以Cpp客户端为例，确保用户可以从命令行运行codearts-cpp --help命令，并且能够看到帮助信息。如果用户没有看到帮助信息，请确保在安装过程中选择了“Add to PATH”。
2. 从命令行运行**git config --global core.editor "codearts-cpp --wait"**命令。

现在用户可以运行**git config --global -e**命令，并将CodeArts作为编辑器来配置Git。

CodeArts 作为 Git 差异工具

CodeArts作为Git的差异工具将以下内容添加到用户的Git配置中，以将CodeArts作为差异工具使用：

```
[diff]
    tool = default-difftool
[difftool "default-difftool"]
    cmd = codearts-cpp --wait --diff $LOCAL $REMOTE
```

这利用了用户可以传递给CodeArts的--diff选项，以便比较两个文件的差异。

以下是一些可以使用CodeArts作为编辑器的示例：

- git rebase HEAD~3 -i：使用CodeArts进行交互式变基。
- git commit：将CodeArts用作提交消息的编辑器。
- git add -p：接着输入e进行交互式添加。
- git difftool <commit>^ <commit>：将CodeArts作为差异编辑器来查看更改。

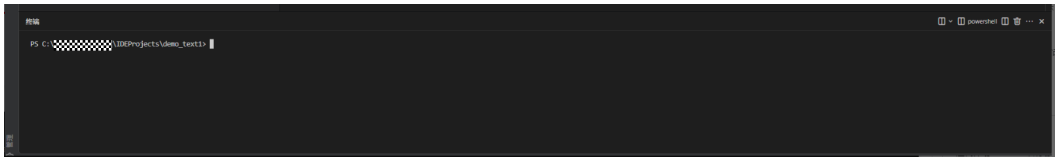
2.7 集成终端

2.7.1 简介

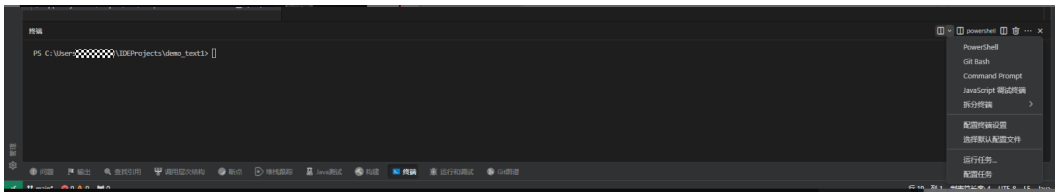
CodeArts IDE包括一个功能齐全的集成终端，其工作目录为当前工作区根目录。

要打开终端，请执行以下任一操作：

- 按“Shift+Esc”或“Alt+F12”。
- 在主菜单中，选择“查看 > 终端”。
- 从“命令”面板（“Ctrl+Shift+P”）中，运行“**查看: 切换终端**”命令。



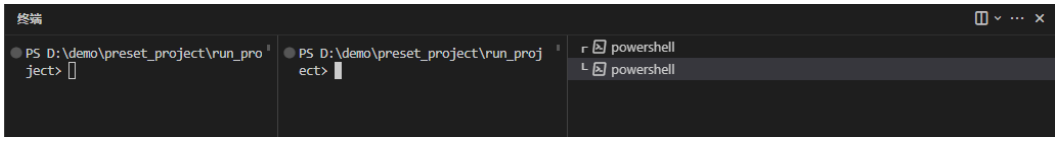
集成终端可以使用安装在计算机上的各种终端，默认为PowerShell。用户可以从“**启动配置文件...**”列表（ ）中选择其他可用的终端类型（如Command Prompt或Git Bash）。



2.7.2 终端管理

2.7.2.1 简介

终端实例以选项卡的形式显示，这些选项卡展示在“终端”视图的右侧或左侧。每个实例都有一个条目，包含其名称、图标、颜色和组合装饰（对于分组实例）。



说明

用户可以通过终端设置项terminal.integrated.tabs.location更改选项卡位置。

2.7.2.2 添加与删除终端实例

执行以下任一操作：

- 单击**终端**视图右上角的“**新建终端**”按钮（ ），或按“Ctrl+Shift+`”。
- 单击“**启动配置文件...**”按钮（ ），然后从列表中选择所需的终端类型。

在选项卡列表中执行如下操作：

悬停一个选项卡，然后单击“**终止**”按钮（ ），或选择一个选项卡并按“Delete”键。

2.7.2.3 终端实例分组

“终端”提供了多种功能，让用户可以自定义其布局。

要将当前终端实例拆分为两个，从而创建组，请执行以下任一操作：

- 在选项卡列表中，悬停选项卡，然后单击**“拆分”**按钮（ ）。
- 在选项卡列表中，选中选项卡，按“Ctrl+Shift+5”。

要将终端实例添加到组，请将对应选项卡拖入当前显示终端区域。要重新排列组中的选项卡，可以拖拽选项卡调整其在组内的顺序。

要取消拆分终端，请在选项卡列表中右键单击该终端，然后从上下文菜单中选择**“取消拆分终端”**。

要在终端组内导航，请使用以下键盘快捷键。

- 按“Ctrl+Pagedown”或“Alt+→”键聚焦下一组。
- 按“Ctrl+Pageup”或“Alt+←”键聚焦上一组。

在组中，通过使用“Alt+↑”聚焦上一个窗格，使用“Alt+↓”聚焦下一个窗格，在终端之间导航。

2.7.2.4 自定义选项卡

要更改终端的名称、图标或选项卡颜色，请右键单击其选项卡，然后从上下文菜单中选择相应的操作，或先选中要操作的选项卡，通过“命令”面板（“Ctrl+Shift+P”或双击“Ctrl”）触发命令：

命令	描述
终端: 重命名	更改终端实例的名称。
终端: 更改图标	更改终端实例的图标。
终端: 更改颜色	更改终端实例的选项卡颜色。

2.7.3 终端配置

2.7.3.1 简介

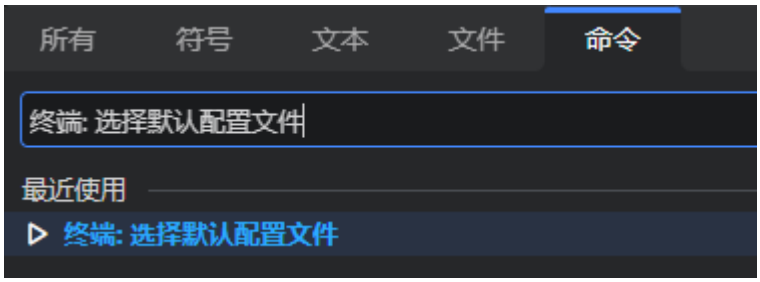
终端配置文件是特定于平台的终端配置，由可执行路径、参数和其他自定义项组成。

配置文件示例：

```
{
  "terminal.integrated.profiles.windows": {
    "My PowerShell": {
      "path": "pwsh.exe",
      "args": [
        "-noexit",
        "-file",
        "${env:APPDATA}\PowerShell\my-init-script.ps1"
      ]
    }
  },
```

```
  "terminal.integrated.defaultProfile.windows": "My PowerShell"
}
```

用户可以在终端配置文件中使用变量（例如上面示例中的APPDATA环境变量），也可以通过运行**“终端: 选择默认配置文件”**命令选择默认的集成终端。



2.7.3.2 配置模板

要创建新的配置文件，请运行**“终端: 选择默认配置文件”**命令，然后单击shell右侧的**“配置终端配置文件”**按钮（  ）以将其作为基础。这将向设置中添加一个新条目，该条目可以在配置文件**settings.json**文件中手动调整。**settings.json**所在路径为“%AppData%\codearts-java\User\settings.json”或“%AppData%\codearts-cpp\User\settings.json”。

用户可以使用路径或源以及一组可选参数来创建配置文件。源仅在Windows上可用，可用于让CodeArts IDE检测PowerShell或Git Bash的安装。或者，用户可以使用直接指向shell可执行文件的路径。以下是一些配置文件**settings.json**配置示例：

```
{
  "terminal.integrated.profiles.windows": {
    "PowerShell -NoProfile": {
      "source": "PowerShell",
      "args": ["-NoProfile"]
    }
  },
  "terminal.integrated.profiles.linux": {
    "zsh (login)": {
      "path": "zsh",
      "args": ["-l"]
    }
  }
}
```

配置文件中支持的其他参数包括：

- **overrideName**：一个布尔值，指示是否将根据运行的程序检测到的动态终端标题替换为静态配置文件名称。
- **env**：定义环境变量及其值的映射，将变量设置为**null**以将其从环境中删除。这可以使用**terminal.integrated.env.<platform>**设置为所有配置文件配置。
- **icon**：用于配置文件的图标ID。
- **color**：用于设置图标样式的主题颜色ID。

默认配置文件可以使用**terminal.integated.defaultProfile.***设置手动定义。

```
{
  "terminal.integrated.profiles.windows": {
    "my-pwsh": {
      "source": "PowerShell",
      "args": ["-NoProfile"]
    }
```

```
  },
  "terminal.integrated.defaultProfile.windows": "my-pwsh"
}
```

2.7.3.3 删除内置配置文件

要从“**启动配置文件...**”列表（ ）中删除条目，请将配置文件的名称设置为**null**。

例如，要删除**Git Bash**配置文件，请在**settings.json**文件（所在路径为“%AppData%\codearts-java\User\settings.json”或“%AppData%\codearts-cpp\User\settings.json”）进行设置。

具体设置方法请参考如下示例：

```
{
  "terminal.integrated.profiles.windows": {
    "Git Bash": null
  }
}
```

2.7.4 工作目录

默认情况下，将在资源管理器当前文件夹中打开终端。

使用**terminal.integrated.cwd**设置，用户可以指定要打开的路径。请注意，在Windows上，反斜杠符号\必须转义为\\。

```
{
  "terminal.integrated.cwd": "D:\\CodeArtsProjects"
}
```

拆分终端默认从父终端启动的目录中启动。可以使用**terminal.integrated.splitCwd**设置更改此行为，以便拆分终端在当前工作区根目录中启动。

```
{
  "terminal.integrated.splitCwd": "workspaceRoot"
}
```

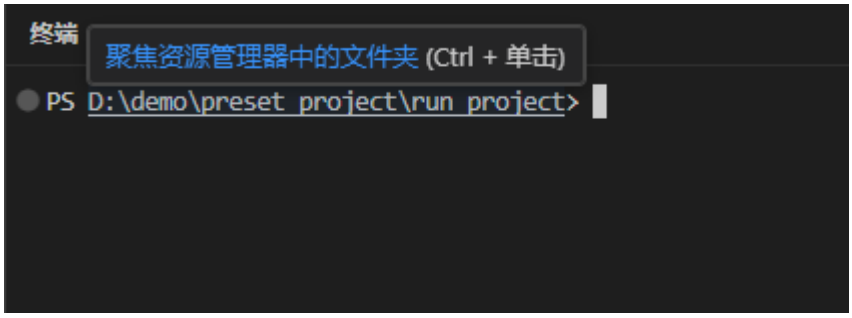
2.7.5 终端进程重连

本地和远程终端进程在窗口重新加载时恢复（例如，当扩展安装需要重新加载时），具体包括：

- 终端将重新连接。
- 终端的UI状态将恢复。
- 活动选项卡和拆分终端相对尺寸将恢复。

2.7.6 链接

终端可以检测链接，当用户悬停文件或URL时，显示下划线。用户可以按住“**Ctrl**”键并单击链接以转到该目标。



根据链接类型，单击后可以执行以下操作：

- 在编辑器中打开文件。
- 聚焦工作区中的文件夹。
- 打开一个新窗口。

2.7.7 终端外观

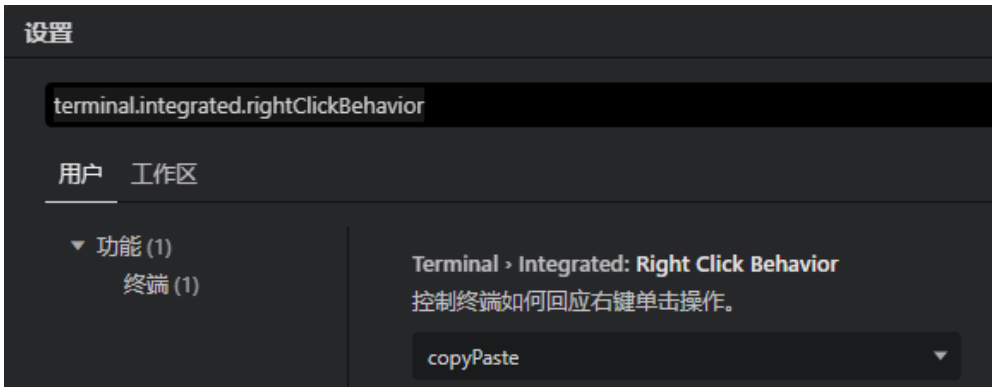
用户可以通过“Ctrl+,”打开“设置”编辑器，在搜索框输入**terminal.integrated.xxx**来设置自定义终端的外观，如：terminal.integrated.Font。

- Font：字体系列、字号和字体粗细，
- Spacing：行高和字母间距。
- Cursor：样式、宽度和闪烁。

2.7.8 鼠标右键单击行为

在终端内部单击鼠标右键时，将粘贴已选定的文本。

也可以使用快捷键**“Ctrl+,”**打开“设置”编辑器，在搜索框中输入**terminal.integrated.rightClickBehavior**，设置该行为。



2.7.9 键绑定和终端

当“终端”视图聚焦时，许多CodeArts IDE上绑定的快捷键将失效，因为此时触发的快捷键行为将传递到终端本身并由终端消耗。有一个预定义的命令列表，这些命令跳过终端处理，会发送到CodeArts IDE密钥绑定系统。用户可以使用**“terminal.integrated.commandsToSkipShell”**设置来自定义此命令列表。可以将命令添加到列表中，也可通过将命令添加到以破折号“-”开头的列表中来删除命令。请参阅设置详细信息以查看默认命令的完整列表。

```
{
  "terminal.integrated.commandsToSkipShell": [
    // Ensure the toggle sidebar visibility keybinding skips the shell
    "workbench.action.toggleSidebarVisibility",
    // Send quick open's keybinding to the shell
    "-workbench.action.quickOpen",
  ]
}
```

要覆盖**“terminal.integrated.commandsToSkipShell”**设置并将键绑定发送到终端，请启用**“terminal.integrated.sendKeybindingsToShell”**设置。

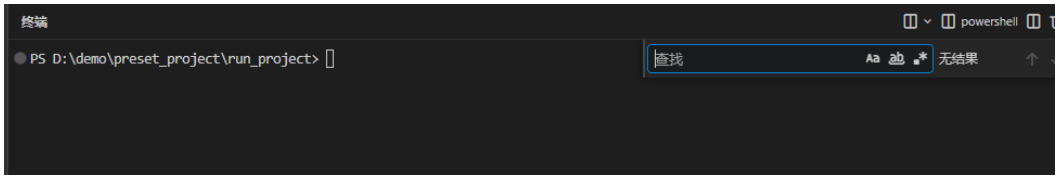
终端中的组合按键

默认情况下，当按下组合按键时，组合按键是最高优先级的，它将始终跳过终端（绕过“**terminal.integrated.commandsToSkipShell**”设置）转到CodeArts IDE。如果用户希望终端使用“Ctrl+K”（对于bash终端，这将剪切光标之后的行），可以将“**terminal.integrated.allowChords**”设置禁用：

```
{
  "terminal.integrated.allowChords": false
}
```

2.7.10 在终端中查找文本

集成终端具有查找功能，可使用“**Ctrl+F**”触发。



如果用户希望“**Ctrl+F**”转到shell而不是启动**Find**控件，请将以下内容添加到**settings.json**中，这将告诉终端不要跳过与**workbench.action.terminal.focusFind**命令匹配的键绑定。焦点查找命令匹配的键绑定的shell：

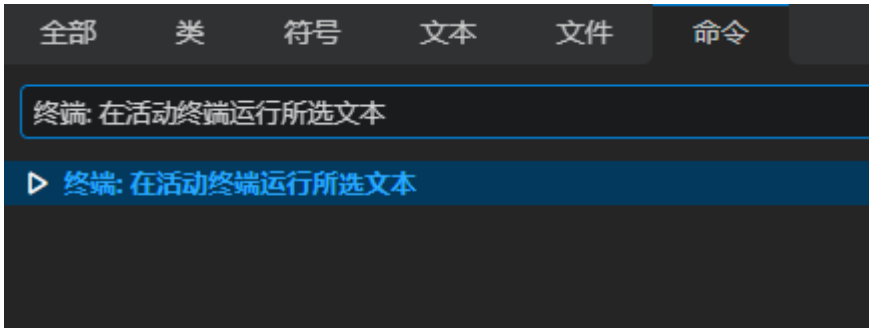
```
{
  "terminal.integrated.commandsToSkipShell": [
    "-workbench.action.terminal.focusFind"
  ],
}
```

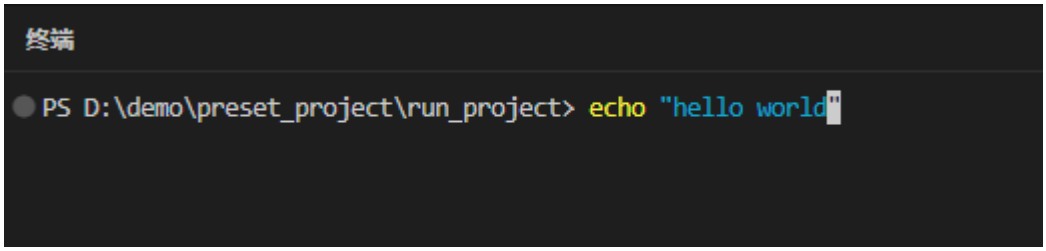
说明

有关在CodeArts IDE中搜索文本的详细信息，请参见通过**代码搜索**。

2.7.11 运行选定的文本

要通过终端执行某些文本，如脚本的一部分，请在编辑器中选中它，然后通过“**命令**”面板（按“**Ctrl+Shift+P**”或“**双击Ctrl**”）运行命令“**终端: 在活动终端运行所选文本**”：





2.8 命令行界面

2.8.1 简介

CodeArts IDE提供了一个强大的命令行界面，用户可以使用命令行选项打开文件、安装扩展、更改显示语言等。

要使CodeArts IDE命令行实用程序正常工作，必须将CodeArts IDE二进制位置（默认情况Java客户端路径是“C:\Program Files\CodeArts IDE for Java\bin”，C++客户端路径是“C:\Program Files\CodeArts IDE for Cpp\bin”）添加到系统路径中。通常，这在安装期间自动执行。用户也可以手动将位置添加到Path环境变量中。

说明

如果用户正在寻找如何在CodeArts IDE中运行命令行工具，请参见**集成终端**。

要了解CodeArts IDE命令行界面的概述，请打开终端或命令提示符，然后键入**codearts-java --help**（Java客户端）和**codearts-cpp --help**（C++客户端）。

```
PS C:\Users\[redacted]\IDEProjects\demo22> codearts-java --help
CodeArts IDE for Java 3.3.0

Usage: codearts-java.exe [options][paths...]

To read output from another program, append '-' (e.g. 'echo Hello World | codearts-java.exe

Options
  -d --diff <file> <file>                    Compare two files with each
                                             other.
  -m --merge <path1> <path2> <base> <result> Perform a three-way merge by
                                             providing paths for two modified
                                             versions of a file, the common
                                             origin of both modified versions
                                             and the output file to save merge
```

2.8.2 从命令行启动

用户可以从命令行**codearts-java .**或者**codearts-cpp .**启动CodeArts IDE以快速打开文件、文件夹或项目。

通常，用户可以在文件夹的上下文中打开CodeArts IDE。

要执行此操作，请导航到项目文件夹并运行**codearts-java**（Java客户端）和**codearts-cpp**（Cpp客户端）命令。

2.8.3 核心命令行选项

以下是通过codearts命令从命令行启动CodeArts IDE时可以使用的可选参数。

参数	描述
-h或**--help**	打印命令行参数的内置帮助。
-v或**--version**	打印CodeArts IDE版本（例如1.22.2）、提交ID和系统架构（例如x64）。
-n或**--new-window**	打开CodeArts IDE的新会话，而不是恢复上一个会话（默认值）。
-g或**--goto**	与**file:line{:character}**一起使用时，在特定行和可选字符位置打开文件。
-d或**--diff**	打开**文件差异编辑器**。需要两个文件路径作为参数。
-w或**--wait**	表示在从终端启动CodeArts IDE时，等待文件关闭后再返回到终端。通常，如果用户从终端执行**codearts-java**（Java客户端）和**codearts-cpp**（C++客户端）命令来打开一个文件或目录，如果不使用“--wait”选项，终端会立即返回，不会等待用户关闭CodeArts IDE，而使用了“--wait”选项后，终端将会等待用户关闭CodeArts IDE后才会返回。
--locale <locale>	设置CodeArts IDE会话的显示语言（例如，**en**或**zh-cn**）。

2.8.4 打开文件和文件夹

用户可以通过命令行启动的CodeArts IDE打开或创建文件。如果指定的文件不存在，CodeArts IDE将创建该文件以及任何新的中间文件夹。

对于文件和文件夹，用户可以使用绝对路径或相对路径。

如果在命令行中指定多个文件，CodeArts IDE将仅打开一个文件。

如果在命令行中指定多个文件夹，CodeArts IDE将创建一个包括每个文件夹的多根工作区。

参数	描述
file	要打开的文件的名称。如果文件不存在，则将创建文件并打开。用户可以通过用空格分隔每个文件名来指定多个文件。
file:line[:character]	与**-g**参数一起使用。要在指定行和可选字符位置打开的文件的名称。用户可以以这种方式指定多个文件，但在使用**file:line[: character]**说明符之前，必须使用**-g**参数。
folder	要打开的文件夹的名称。用户可以指定多个文件夹，并创建新的多根工作区。

2.8.5 使用扩展

用户可以从命令行安装和管理CodeArts IDE扩展。

参数	描述
--install-extension <ext>	安装扩展。提供完整的**publisher.extension**（发布商名.插件名）作为参数。使用**--force**参数来避免提示。

2.8.6 CLI 高级选项

以下几个CLI选项有助于进行更多高级设置。

参数	描述
-s、--status	打印进程使用情况和诊断信息。
--disable-gpu	禁用GPU硬件加速。
--verbose	打印启动过程中的详细输出。
--prof-startup	在启动过程中运行CPU探查器。

2.8.7 通过 URLs 打开 CodeArts IDE

用户还可以使用操作系统的URL处理机制打开项目和文件。

Java和cpp客户端使用的命令不一致，以cpp客户端为例（Java客户端请将“codearts-cpp”替换成“codearts-java”），使用以下URL格式：

- 打开项目
  ```
  codearts-cpp://file/{full path to project}/
  示例:codearts-cpp://file/c:/myProject/
  ```
- 打开文件
  ```
  codearts-cpp://file/{full path to file}
  示例:codearts-cpp://file/c:/myProject/package.json
  ```
- 在特定行和列上打开文件
  ```
  codearts-cpp://file/{full path to file}:line:column
  示例:codearts-cpp://file/c:/myProject/package.json:5:10
  ```

用户可以在浏览器或文件资源管理器等应用程序中使用URL，这些应用程序可以解析和重定向URL。例如，用户可以将**codearts-cpp://URL**直接传递给Windows资源管理器，或作为**codearts-cpp://{full path to file}**传递给命令行。

2.9 CodeArts IDE 设置

2.9.1 简介

用户可以通过CodeArts IDE的各种设置项，配置用户喜欢的CodeArts IDE。CodeArts IDE的编辑器、用户界面和函数行为等都有可以修改的设置项。

CodeArts IDE提供了几个设置范围：

- **“用户设置”**全局应用于用户打开的任何CodeArts IDE实例。
- **“工作区设置”**存储在工作区中，仅在打开工作区时应用。

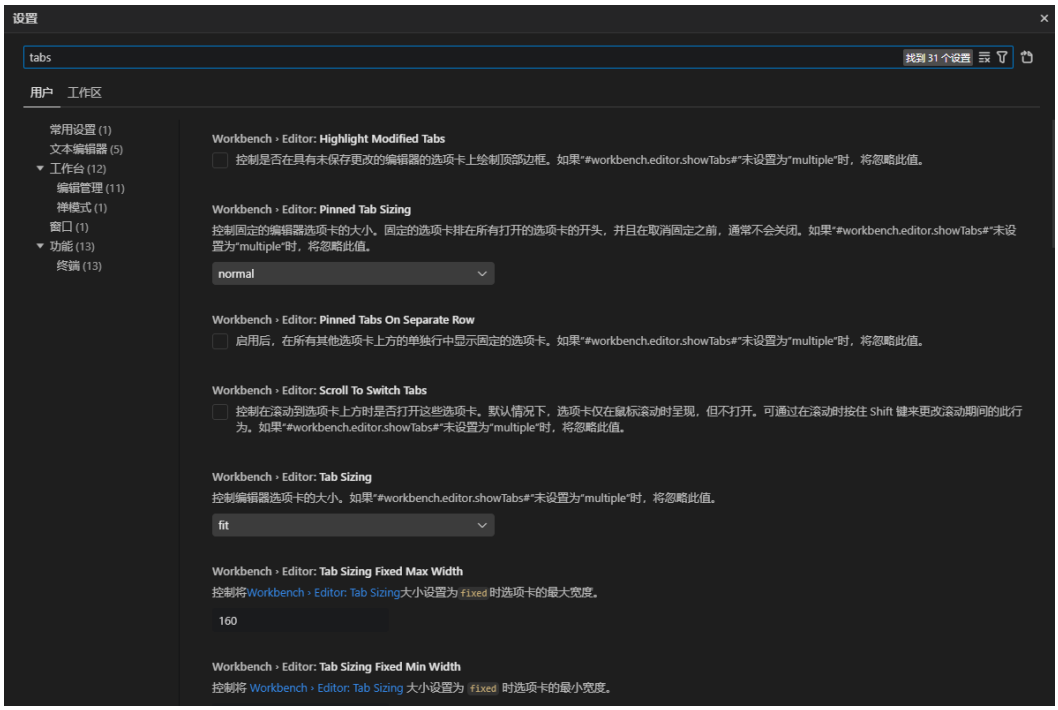
2.9.2 设置编辑器

2.9.2.1 简介

要修改用户设置，请使用**“设置”**编辑器，用户可以通过执行以下任何操作打开该编辑器。

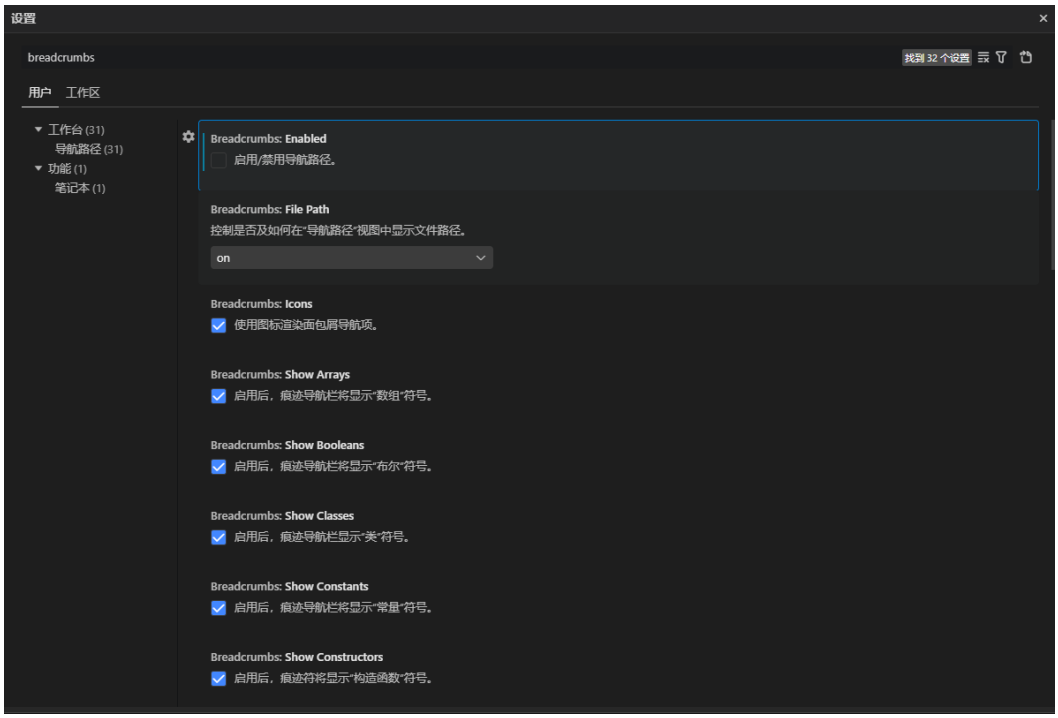
- 按**“Ctrl+,”**打开**“设置”**编辑器。
- 在左侧活动栏中选择“管理 > 设置”。
- 在“命令”面板（**“Ctrl+Shift+P”**或**“双击Ctrl”**）中，搜索并运行**首选项: 打开设置 (ui)**命令。

在“设置”编辑器中，使用搜索字段来搜索所需的设置。这将显示搜索条件匹配的设置，并过滤掉不匹配的设置。



CodeArts IDE会动态应用对设置的更改。修改后的设置用一条类似于编辑器中修改后的行的蓝线表示。

在以下示例中，**面包屑导航**已禁用。

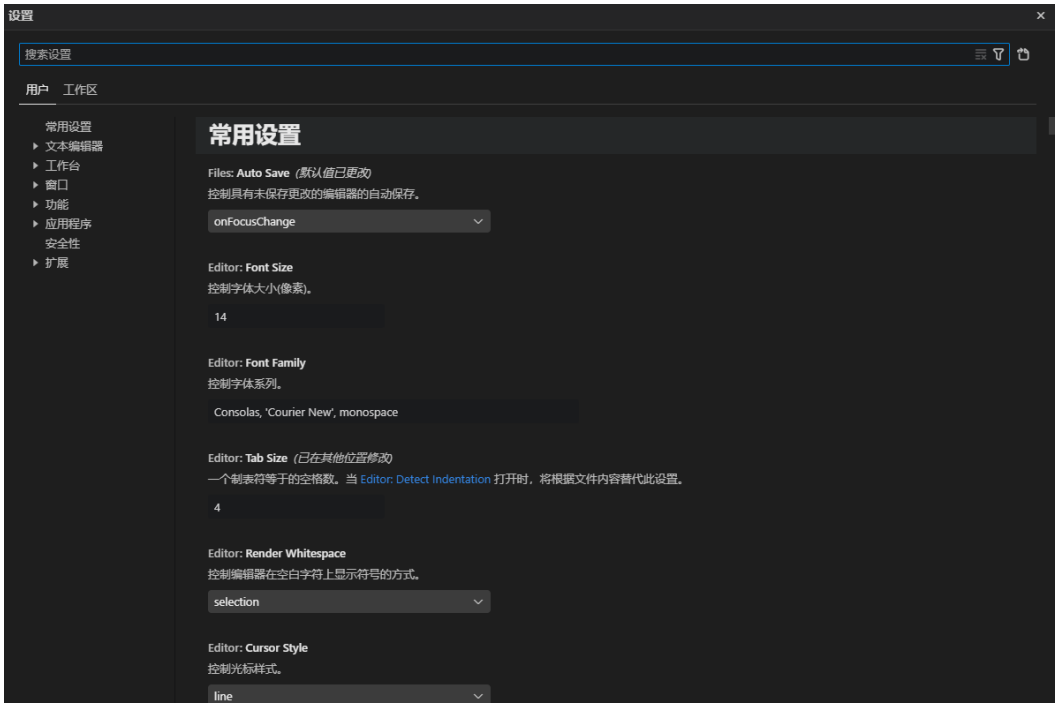


选择一个设置，然后单击**“更多操作”**（）按钮，或按“Shift+F9”打开上下文菜单，允许用户将设置重置为其默认值、复制设置ID、将设置复制为JSON文本和将设置应用于所有配置文件。

2.9.2.2 设置组

设置以组表示，以便用户可以轻松导航。顶部的**“常用设置”**组列出了常用的自定义设置项。

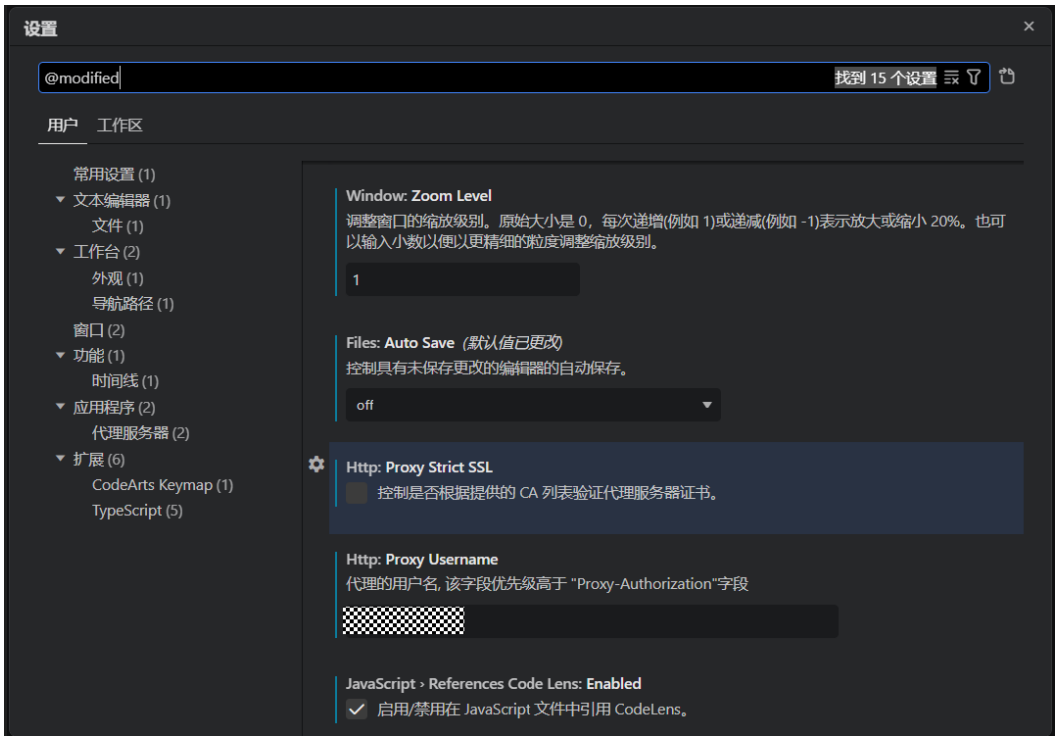
CodeArts IDE扩展还可以添加自定义设置，这些设置收集在“**扩展**”部分下。



2.9.2.3 设置编辑器筛选器

“设置”编辑器的搜索栏提供了几个筛选器，使用户更容易管理设置。使用搜索栏中的“**筛选器**”按钮（ ）轻松添加筛选器。

要检查用户配置的设置，请使用**@modified**的筛选器。如果设置的值与默认值不同，或者其值在相应的设置JSON文件中显式设置，则会显示在此筛选器下。



还有几个其他方便的过滤器可以帮助搜索设置：

- **@ext**：特定于扩展的设置。提供扩展ID，例如，**@ext:markdown-language-features**。
- **@feature**：特定于功能子组的设置。例如，**@feature:explorer**显示资源管理器的设置。
- **@id**：根据设置ID查找设置。例如，**@id:workbench.activityBar.visible**。
- **@lang**：根据语言ID应用语言筛选器。例如，**@lang:typescript**。
- **@tag**：特定于CodeArts IDE子系统的设置。

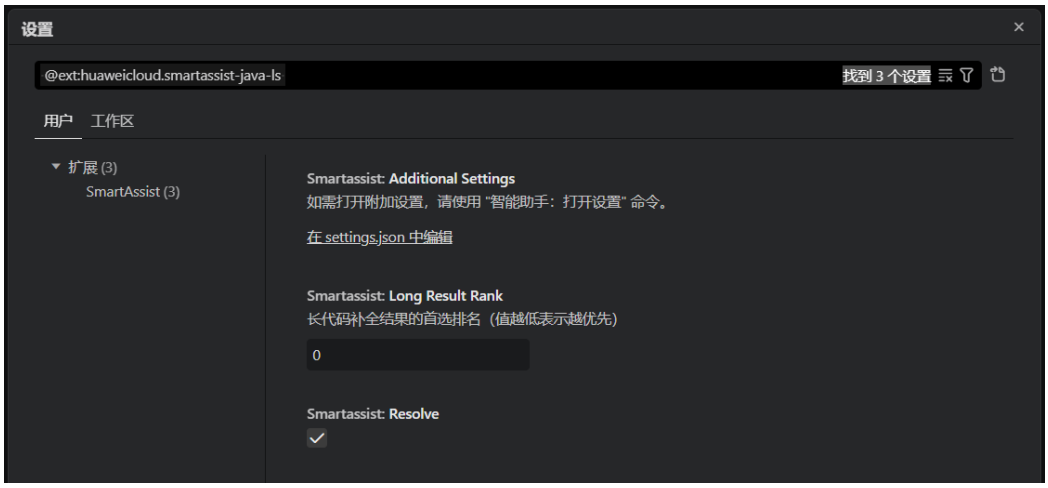
搜索栏会记住用户的设置搜索查询，并支持撤销/重做（“Ctrl+Z”/“Ctrl+Shift+Z”/“Ctrl+Y”）。用户可以使用搜索栏右侧的**“清除设置搜索输入”**按钮（ ）快速清除搜索项或筛选器。

2.9.2.4 扩展设置

安装的CodeArts IDE扩展也可以贡献自己的设置，用户可以在设置编辑器的**“扩展”**部分查看这些设置。

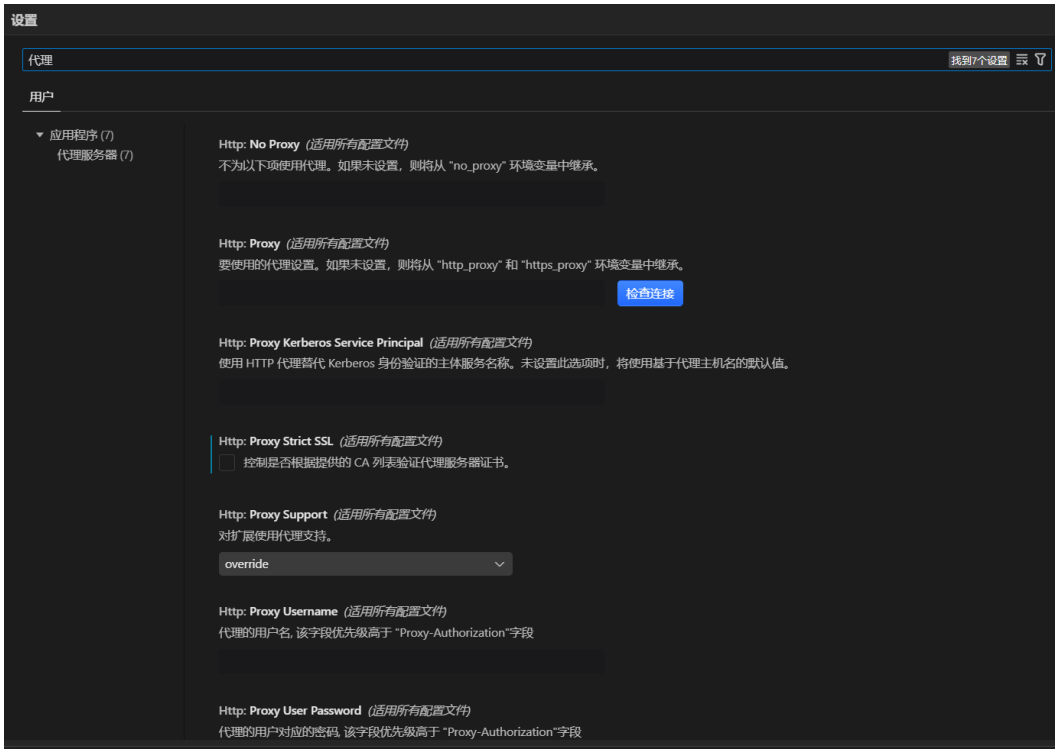
说明

要打开SmartAssist for Java设置，请在“设置”中搜索**@ext:huaweicloud.smartassist-java-ls**。



2.9.2.5 代理设置

代理设置可以允许用户在CodeArts IDE中进行网络代理的设置。在设置编辑器的搜索栏中，搜索字段**代理**，可以搜索到代理的相关配置：



CodeArts IDE提供了如下代理配置：

1. “Http: Proxy”：要使用的代理设置，包括地址和端口号。
2. “Http: No Proxy”：不使用代理的地址列表（域名或IP），多个地址用逗号分隔。
3. “Http: Proxy Strict SSL”：控制是否根据提供的CA列表验证代理服务器证书。
4. “Http: Proxy Support”：扩展使用代理的方式。
5. “Http: Proxy Username”：代理用户名。

6. “Http: Proxy User Password”：代理用户对应的密码。
7. “Http: Proxy Kerberos Service Principal”：使用HTTP代理替代Kerberos身份验证的主体服务名称。

须知

代理配置参考：

1、“Http: Proxy”：**http://proxy.******.com:8080**

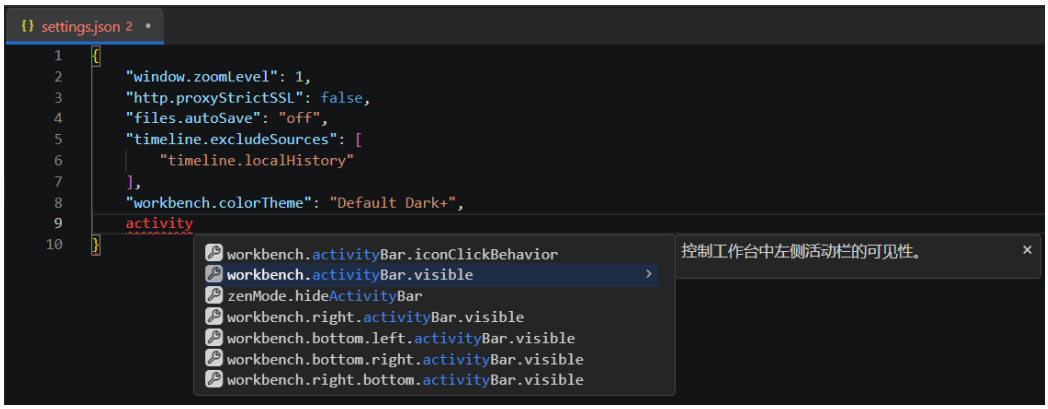
2、“Http: No Proxy”：*******.com,***.com**

在部分网络场景下，用户可能还需要配置用户的代理**用户名（Proxy Username）**和**密码（Proxy Password）**以及取消勾选**“Proxy Strict SSL”**字段。

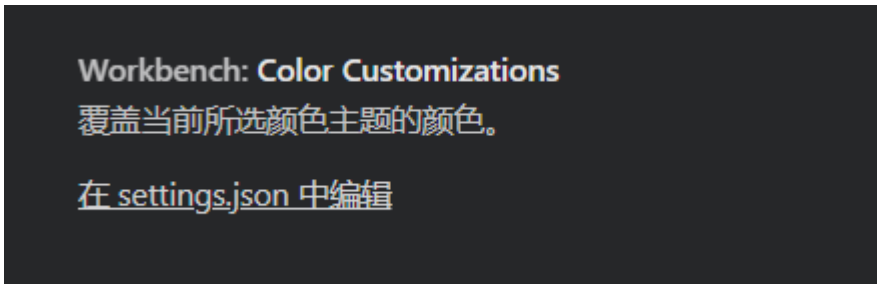
2.9.3 settings.json

设置编辑器是允许用户查看和修改存储在**settings.json**文件中的设置值的UI。用户可以通过快捷键**“Ctrl+Shift+P”**打开“命令”面板，使用**“首选项: 打开用户设置(JSON)”**命令在编辑器中打开此文件，直接查看和编辑此文件。通过指定设置ID和值，设置将写入JSON。

settings.json文件提供设置及其值的代码完成和描述悬停。由于设置名称或JSON格式不正确而导致的错误也会突出显示。



某些设置，例如**“Workbench: Color Customizations”**只能在**settings.json**中编辑。



如果用户喜欢始终直接使用**settings.json**，用户可以将“**workbench.settings.editor**”设置为“**json**”，以便将“管理 > 设置”命令和“**Ctrl+,**”键绑定，始终打开**settings.json**文件，而不是设置编辑器。

设置文件位置

不同的客户端，用户设置文件的路径不同。

- CodeArts IDE for Cpp：%APPDATA%\Roaming\codearts-cpp\User\settings.json
- CodeArts IDE for Java：%APPDATA%\Roaming\codearts-java\User\settings.json

重置所有设置

虽然用户可以通过设置编辑器中的“**重置此设置**”命令单独重置设置，但用户也可以通过打开**settings.json**并删除大括号**{}**之间的条目来重置所有更改的设置。

说明

当用户通过清除**settings.json**重置设置时，无法恢复其以前的值。

2.9.4 工作区设置

工作区设置与项目一起存储，因此可以在项目的开发人员之间共享。工作区设置覆盖用户设置。

用户可以在设置编辑器的“**工作区**”选项卡上编辑工作区设置。要快速打开此选项卡，请使用“命令面板”（“Ctrl+Shift+P”/“Ctrl+Ctrl”）中的**首选项: 打开工作区设置**。

并非所有用户设置都可用作工作区设置，例如，与更新和安全性相关的应用程序范围的设置不能被工作区设置覆盖。

工作区 settings.json 位置

与用户设置类似，工作区设置也存储在一个**settings.json**文件中，用户可以通过**首选项: 打开工作区设置 (JSON)**命令直接编辑该文件。工作区设置文件位于项目根文件夹的“.arts”文件夹下。当用户将工作区设置**settings.json**文件添加到**源代码管理**时，该项目的设置将由该项目的所有用户共享。

2.9.5 设置优先级

配置可以在多个级别上被不同的设置范围覆盖。在以下列表中，较晚的作用域覆盖较早的作用域：

- Default settings：此范围表示默认未配置的设置值。
- User settings：全局应用于所有CodeArts IDE实例。
- Workspace settings：应用于打开的文件夹或工作区。
- Language-specific default settings：这些是特定于语言的默认值，可由扩展提供。
- Language-specific user settings：特定于语言的用户设置：与用户设置相同，但特定于语言。
- Language-specific workspace settings：与工作区设置相同，但特定于语言。

设置值可以是多种类型：

- String：**"files.autoSave": "afterDelay"**
- Boolean：**"editor.minimap.enabled": true**
- Number：**"files.autoSaveDelay": 1000**
- Array：**"editor.rulers": []**
- Object：**"search.exclude": { "**/node_modules": true, "**/bower_components": true }**

具有基元类型和数组类型的值将被覆盖，这意味着使用作用域中优先于另一个作用域的配置值，而不是另一个作用域中的值。但是，具有对象类型的值将合并。

例如，**workbench.colorCustomizations**采用一个对象，该对象指定一组UI元素及其所需颜色。如果用户的用户设置将编辑器背景设置为蓝色和绿色。

```
"workbench.colorCustomizations": {
  "editor.background": "#000088",
  "editor.selectionBackground": "#008800"
}
```

打开的工作区设置将编辑器前景设置为红色：

```
"workbench.colorCustomizations": {
  "editor.foreground": "#880000",
  "editor.selectionBackground": "#00FF00"
}
```

当该工作区打开时，结果是这两种颜色自定义的组合，就像用户指定了：

```
"workbench.colorCustomizations": {
  "editor.background": "#000088",
  "editor.selectionBackground": "#00FF00",
  "editor.foreground": "#880000"
}
```

如果存在冲突的值，如上面示例中的**editor.selectionBackground**，则会发生通常的覆盖行为，工作区值优先于用户值，语言特定的值优先于非语言特定的值。

2.9.6 设置和安全性

某些设置允许用户指定CodeArts IDE将运行以执行某些操作的可执行文件。例如，用户可以选择集成终端应使用的shell。为了增强安全性，此类设置只能在用户设置中定义，而不能在工作区范围中定义。

以下是工作区设置中不支持的设置列表：

- **git.path**
- **terminal.external.windowsExec**
- **terminal.external.osxExec**
- **terminal.external.linuxExec**

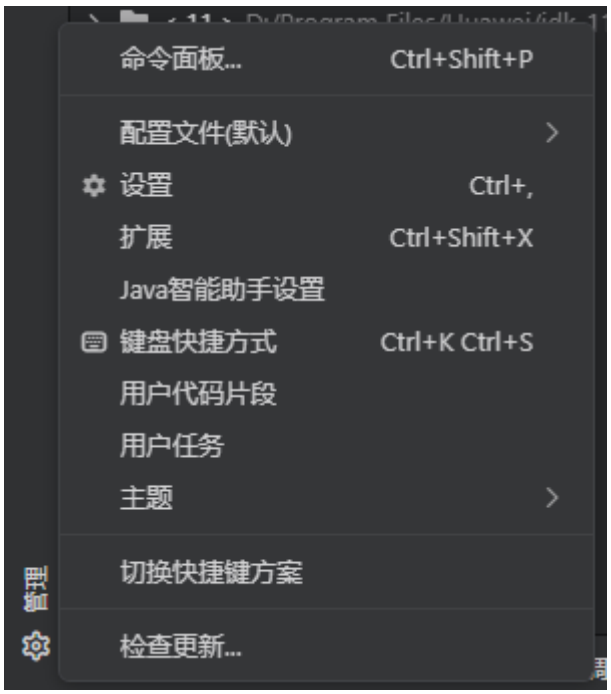
第一次打开定义这些设置中任何一个的工作区时，CodeArts IDE将警告用户，然后始终忽略之后的值。

2.10 默认键绑定

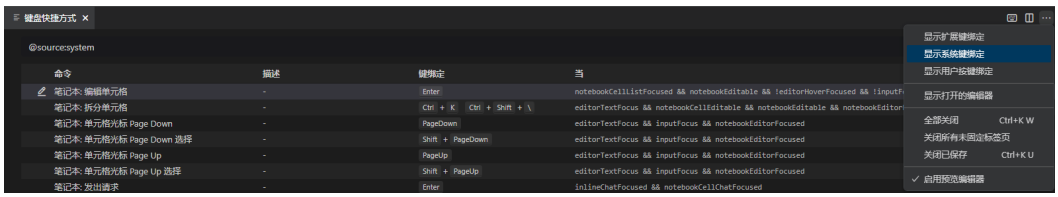
2.10.1 简介

CodeArts IDE中提供了丰富的快捷键功能，用户可以根据用户的需求随时查看当前快捷键方案或者修改快捷键方案。

如果用户需要查看当前CodeArts IDE的快捷键方案，请单击CodeArts IDE左下角的**“管理”**菜单。并选择**“键盘快捷键方式”**选项，或者先按快捷键“Ctrl+K”再按快捷键“Ctrl+S”直接唤起“键盘快捷方式”编辑器。



打开“键盘快捷键方式”编辑器后，用户可以在**“更多操作”**菜单选择**“显示系统键绑定”**，查看所有默认键盘快捷方式。这将在“键盘快捷方式”编辑器中应用@source:system过滤器（**source**为“system”）。

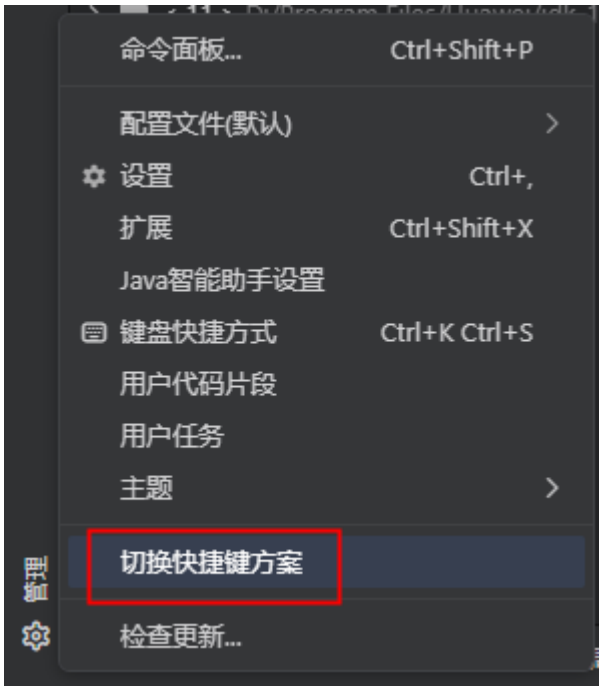


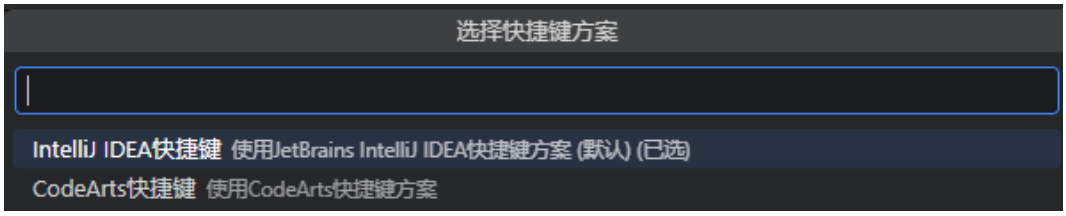
用户还可以通过使用**“首选项: 打开默认键盘快捷键 (JSON)”**命令，将默认键盘快捷方式视为JSON文件进行查看。

CodeArts IDE for Java默认提供了两个预定义的键位映射：CodeArts快捷键和IntelliJ IDEA快捷键，后者使用IntelliJ IDEA中对应的映射覆盖了一些最常用的快捷方式。如果用户习惯在IntelliJ IDEA中工作，用户可以选择IDEA键位映射，从而继续使用熟悉的快捷方式。

要在键位映射之间切换，请执行以下操作：

左下角菜单打开“管理 > 切换快捷键方案”，在列表中选择所需的键位映射。



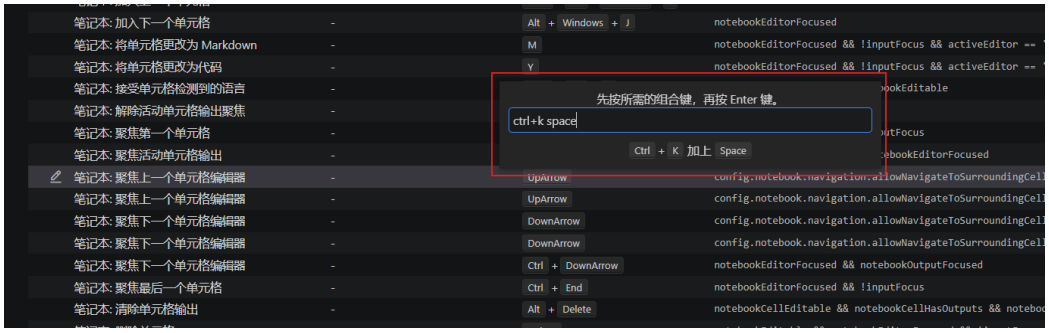


2.10.2 修改快捷键绑定

用户可以在“键盘快捷方式”编辑器中通过两种方式修改默认的快捷键。

方式1：右键单击想要修改的快捷键，选择**“更改键绑定”菜单**。

方式2：双击该快捷键方案所在的行，在唤起的弹窗中输入想要绑定的新快捷键：



2.10.3 查看和重置已修改的快捷键绑定

要查看CodeArts IDE中任何用户修改的键盘快捷键，请单击**“更多操作”**按钮（ ）并选择弹出菜单中的**“显示用户按键绑定”**。这将在“键盘快捷方式”编辑器中应用**@source:user**过滤器（**Source**为“用户”）。

执行以下操作之一：

- 要重置单个按键绑定，请右键单击列表中相应的条目，然后选择**“重置按键绑定”**。
- 要一次性重置所有按键绑定，请找到用户的**keybindings.json**文件并清空其内容。
- 以CodeArts IDE for Cpp为例，默认情况下，该文件位于“%USERPROFILE%\AppData\Roaming\codearts-cpp\User\keybindings.json”。

2.10.4 快捷键绑定参考

2.10.4.1 基本编辑

命令	快捷键（CodeArts快捷键方案）	快捷键（IDEA快捷键方案）	命令ID
剪切	Shift+Delete Ctrl+X	Ctrl+X Shift+Delete	editor.action.clipboardCutAction
复制	Ctrl+Insert Ctrl+C	Ctrl+Insert Ctrl+C	editor.action.clipboardCopyAction
粘贴	Shift+Insert Ctrl+V	Shift+Insert Ctrl+V	editor.action.clipboardPasteAction
删除行	Ctrl+Shift+K	Ctrl+Shift+K	editor.action.deleteLines
在下面插入行	Ctrl+Enter	Shift+Enter Ctrl+Shift+Enter	editor.action.insertLineAfter
在上面插入行	Ctrl+Shift+Enter	Ctrl+Alt+Enter	editor.action.insertLineBefore
向下移动行	Alt+DownArrow	Shift+Alt+DownArrow Ctrl+Shift+DownArrow	editor.action.moveLinesDownAction
向上移动行	Alt+UpArrow	Shift+Alt+UpArrow Ctrl+Shift+UpArrow	editor.action.moveLinesUpAction
向下复制行	Shift+Alt+DownArrow	Ctrl+D	editor.action.copyLinesDownAction
向上复制行	Shift+Alt+UpArrow	Shift+Alt+UpArrow	editor.action.copyLinesUpAction
撤销	Ctrl+Z	Ctrl+Z	undo
恢复	Ctrl+Shift+Z Ctrl+Y	Ctrl+Shift+Z Ctrl+Y	redo
将下一个查找匹配项添加到选择	Ctrl+D	Alt+J	editor.action.addSelectionToNextFindMatch
将上次选择移动到下一个查找匹配项	Ctrl+K Ctrl+D	Ctrl+K Ctrl+D	editor.action.moveSelectionToNextFindMatch
光标撤销	Ctrl+U	Shift+Alt+J	cursorUndo
在行尾添加光标	Shift+Alt+I	Shift+Alt+I	editor.action.insertCursorAtEndOfEachLineSelected

命令	快捷键（CodeArts快捷键方案）	快捷键（IDEA快捷键方案）	命令ID
选择所有找到的查找配置项	Ctrl+Shift+L	Ctrl+Shift+Alt+J	editor.action.selectHighlights
更改所有配置项	Ctrl+F2 F2	Shift+F6	editor.action.changeAll
展开行选择	Ctrl+L	Ctrl+L	expandLineSelection
在下面添加光标	Ctrl+Alt+DownArrow	Ctrl+Alt+DownArrow	editor.action.insertCursorBelow
在上面添加光标	Ctrl+Alt+UpArrow	Ctrl+Alt+UpArrow	editor.action.insertCursorAbove
转到括号	Ctrl+Shift+\	Ctrl+Shift+\	editor.action.jumpToBracket
行缩进	Ctrl+]	Ctrl+]	editor.action.indentLines
行减少缩进	Ctrl+[	Ctrl+[	editor.action.outdentLines
转到行首	Home	Home	cursorHome
转到行尾	End	End	cursorEnd
转到文件末尾	Ctrl+End	Ctrl+End	cursorBottom
转到文件开头	Ctrl+Home	Ctrl+Home	cursorTop
向下滚动行	Ctrl+DownArrow	Ctrl+DownArrow	scrollLineDown
向上滚动	Ctrl+UpArrow	Ctrl+UpArrow	scrollLineUp
向下滚动页	Alt+Pagedown	Alt+Pagedown	scrollPageDown
向上滚动页	Alt+Pageup	Alt+Pageup	scrollPageUp
折叠	Ctrl+Shift+[	Ctrl+- Ctrl+Numpad_Subtract	editor.fold
展开	Ctrl+Shift+]	Ctrl+= Ctrl+NumPad_Add	editor.unfold
以递归方式折叠	Ctrl+K Ctrl+[	Ctrl+Alt+- Ctrl+Alt+Numpad_Subtract	editor.foldRecursively

命令	快捷键（CodeArts快捷键方案）	快捷键（IDEA快捷键方案）	命令ID
以递归方式展开	Ctrl+K Ctrl+]	Ctrl+Alt+= Ctrl+Alt+Numpad_Add	editor.unfoldRecursively
全部折叠	Ctrl+K Ctrl+0	Ctrl+Shift+- Ctrl+Shift+Numpad_Subtract	editor.foldAll
全部展开	Ctrl+K Ctrl+J	Ctrl+Shift+= Ctrl+Shift+Numpad_Add	editor.unfoldAll
添加行注释	Ctrl+K Ctrl+C	Ctrl+K Ctrl+C	editor.action.addCommentLine
删除行注释	Ctrl+K Ctrl+U	Ctrl+K Ctrl+U	editor.action.removeCommentLine
切换行注释	Ctrl+/	Ctrl+/ Ctrl+Numpad_Divide	editor.action.commentLine
切换块注释	Shift+Alt+A	Ctrl+Shift+/ Ctrl+Shift+Numpad_Divide	editor.action.blockComment
查找	Ctrl+F	Ctrl+F	actions.find
替换	Ctrl+H	Ctrl+R	editor.action.startFindReplaceAction
查找下一个	F3 Enter	F3	editor.action.nextMatchFindAction
查找上一个	Shift+F3 Shift+Enter	Shift+F3	editor.action.previousMatchFindAction
选择所有出现的查找匹配项	Alt+Enter	Alt+Enter	editor.action.selectAllMatches
切换查找区分大小写	Alt+C	Alt+C	toggleFindCaseSensitive
切换查找正则表达式	Alt+R	Alt+R	toggleFindRegex
切换查找整个单词	Alt+W	Alt+W	toggleFindWholeWord

命令	快捷键（CodeArts快捷键方案）	快捷键（IDEA快捷键方案）	命令ID
切换自动换行	Alt+Z	Alt+Z	editor.action.toggleWordWrap

2.10.4.2 编码辅助

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
触发建议	Ctrl+I Ctrl+Space	Ctrl+Shift+Space	editor.action.triggerSuggest
触发参数提示	Ctrl+Shift+Space	Ctrl+P	editor.action.triggerParameterHints
格式化文档	Shift+Alt+F	Ctrl+Alt+L	editor.action.formatDocument
格式化选定内容	Ctrl+K Ctrl+F	Ctrl+Alt+L	editor.action.formatSelection
显示或聚焦悬停	Ctrl+K Ctrl+I	Ctrl+Q	editor.action.showHover
打开侧边的定义	Ctrl+K F12	Ctrl+K F12	editor.action.revealDefinitionAside
快速修复	Ctrl+.	Alt+Enter	editor.action.quickFix
展开选择	Shift+Alt+RightArrow	Ctrl+W	editor.action.smartSelect.expand
收起选择	Shift+Alt+LeftArrow	Ctrl+Shift+W	editor.action.smartSelect.shrink

2.10.4.3 搜索

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
显示搜索	Ctrl+Shift+F	Ctrl+Shift+F	omnisearch.open.file
在文件中替换	Ctrl+Shift+H	Ctrl+Shift+R	omnisearch.open.file.replace

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
切换匹配大小写	Alt+C	Alt+C	toggleSearchCaseSensitive
切换全字匹配	Alt+W	Alt+W	toggleSearchWholeWord
Toggle使用正则表达式	Alt+R	Alt+R	toggleSearchRegex
切换搜索详细信息	Ctrl+Shift+J	Ctrl+Shift+J	workbench.action.search.toggleQueryDetails
聚焦下一个搜索结果	F4	F4	search.action.focusNextSearchResult
聚焦上一个搜索结果	Shift+F4	Shift+F4	search.action.focusPreviousSearchResult
显示下一个搜索词	Alt+Down DownArrow	Alt+DownArrow DownArrow	history.showNext
显示上一个搜索词	Alt+UpArrow UpArrow	Alt+UpArrow UpArrow	history.showPrevious
在编辑器中打开结果	Alt+Enter	Alt+Enter	search.action.openInEditor
聚焦搜索编辑器输入	Escape	Escape	search.action.focusQueryEditorWidget
再次搜索	Ctrl+Shift+R	Ctrl+Shift+R	rerunSearchEditorSearch
删除文件结果	Ctrl+Shift +Backspace	Ctrl+Shift +Backspace	search.searchEditor.action.deleteFileResults

2.10.4.4 导航

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
转到工作区的符号	Ctrl+T	Ctrl+N	workbench.action.showAllSymbols
转到行/列...	Ctrl+G	Ctrl+G	workbench.action.gotoLine

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
转到文件...	Ctrl+E Ctrl+P	Ctrl+Shift+N	workbench.action.quickOpen
转到编辑器中的符号...	Ctrl+Shift+O	Ctrl+Shift+Alt+N Ctrl+F12	workbench.action.gotoSymbol
转到定义	F12	F12	editor.action.revealDefinition
前往声明	--	Alt+F11 F4 Ctrl+Enter Ctrl+B	editor.action.goToDeclaration
转到实现	Ctrl+F12	Ctrl+Alt+B	editor.action.goToImplementation
转到类型	Shift+Alt+T Ctrl+Shift+O Ctrl+T	Ctrl+N Ctrl+Shift+Alt+N	workbench.action.smartSearchTypes
引用:查找所有引用	Shift+Alt+F12	Alt+F7	references-view.findReferences
查看:切换问题	Ctrl+Shift+M	Shift+Escape Alt+0	workbench.actions.view.problems
转到下一个问题(错误、告警、信息)	Alt+F8	F2	editor.action.marker.next
转到上一个问题(错误、告警、信息)	Shift+F8	Shift+F2	editor.action.marker.prev
显示所有命令	Ctrl+Shift+P 双击Ctrl	Ctrl+Shift+P 双击Ctrl	omnisearch.open.command
返回	Alt+LeftArrow	Ctrl+Alt+LeftArrow	workbench.action.navigateBack
前进	Alt+RightArrow	Ctrl+Alt+RightArrow	workbench.action.navigateForward

2.10.4.5 重构

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
查看可用的重构	Ctrl+Shift+R	Ctrl+Shift+Alt+T	editor.action.refactor
复制类	Alt+F6	F5	refactor.copy.class
安全删除	Alt+Delete	Alt+Delete	refactor.safe.delete
重命名符号	F2	Shift+F6	editor.action.rename
移动	--	F6	refactor.move
移动类	F6	F6	refactor.move.classes
引入变量	Ctrl+Alt+V	Ctrl+Alt+V	refactor.extract.variable
提取方法	Ctrl+Shift+Alt+M	Ctrl+Shift+Alt+M	refactor.extract.method
介绍领域	Ctrl+Shift+Alt+F	Ctrl+Shift+Alt+F	refactor.extract.field
引入常数	Ctrl+Alt+C	Ctrl+Alt+C	refactor.extract.constant
介绍参数	Ctrl+Shift+Alt+P	Ctrl+Shift+Alt+P	refactor.introduce.parameter
内联变量	Ctrl+Alt+N	Ctrl+Alt+N	refactor.inline.variable
内联参数	--	Ctrl+Shift+Alt+P	refactor.inline.parameter
内联方法	Ctrl+Shift+Alt+L	Ctrl+Shift+Alt+L	refactor.inline.method
更改签名	Ctrl+F6	Ctrl+F6	refactor.change.signature

2.10.4.6 调试

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
切换断点	F9	Ctrl+F8	editor.debug.action.toggleBreakpoint
调试:开始调试	F5	Shift+F9	workbench.action.debug.start
继续	F5	F9	workbench.action.debug.continue
调试:开始执行（不调试）	Ctrl+F5	Shift+F10	workbench.action.debug.run
暂停	F6	F6	workbench.action.debug.pause
调试:单步调试	F11	F7	workbench.action.debug.stepInto

2.10.4.7 版本控制

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
拉取	--	Ctrl+T	git.pull
全部提交	--	Ctrl+Alt+K	git.commitAll
Git:暂存所选范围	Ctrl+K Ctrl+Alt+S	Ctrl+K Ctrl+Alt+S	git.stageSelectedRanges
Git:取消暂存选定范围	Ctrl+K Ctrl+N	Ctrl+K Ctrl+N	git.unstageSelectedRanges
Git:还原所选更改	Ctrl+K Ctrl+R	Ctrl+Alt+Z	git.revertelectedRanges

2.10.4.8 编辑器/窗口管理

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
关闭窗口	Ctrl+Shift+W Alt+F4	Ctrl+Shift+W Alt+F4	workbench.action.closeWindow
关闭编辑器	Ctrl+W Ctrl+F4	Ctrl+F4	workbench.action.closeActiveEditor

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
工作区:关闭工作区	Ctrl+K F	Ctrl+K F	workbench.action.closeFolder
查看:拆分编辑器	Ctrl+\	Ctrl+\	workbench.action.splitEditor
查看:聚焦第一个编辑组	Ctrl+1	Ctrl+1	workbench.action.focusFirstEditorGroup
查看:聚焦第二编辑组	Ctrl+2	Ctrl+2	workbench.action.focusSecondEditorGroup
查看:聚焦第三编辑组	Ctrl+3	Ctrl+3	workbench.action.focusThirdEditorGroup
向左移动编辑器	Ctrl+Shift+Pageup	Ctrl+Shift+Pageup	workbench.action.moveEditorLeftInGroup
向右移动编辑器	Ctrl+Shift +Pagedown	Ctrl+Shift +Pagedown	workbench.action.moveEditorRightInGroup
向左移动活动编辑器组	Ctrl+K LeftArrow	Ctrl+K LeftArrow	workbench.action.moveActiveEditorGroupLeft
向右移动活动编辑器组	Ctrl+K RightArrow	Ctrl+K RightArrow	workbench.action.moveActiveEditorGroupRight
将编辑器移至下一组	Ctrl+Alt +RightArrow	Ctrl+Alt +RightArrow	workbench.action.moveEditorToNextGroup
将编辑器移动到上一个组	Ctrl+Alt +LeftArrow	Ctrl+Alt +LeftArrow	workbench.action.moveEditorToPreviousGroup

2.10.4.9 文件管理

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
新文件	Ctrl+N	Alt+Insert	workbench.action.files.newUntitledFile

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
打开文件	Ctrl+O	Ctrl+O	workbench.action.files.openFile
保存	Ctrl+S	Ctrl+S	workbench.action.files.save
全部保存	Ctrl+K S	Ctrl+K S	saveAll
另存为	Ctrl+Shift+S	Ctrl+Shift+S	workbench.action.files.saveAs
关闭编辑器	Ctrl+W Ctrl+F4	Ctrl+F4	workbench.action.closeActiveEditor
关闭组中所有编辑器	Ctrl+K W	Ctrl+K W	workbench.action.closeEditorsInGroup
关闭所有编辑器	Ctrl+K Ctrl+W	Ctrl+K Ctrl+W	workbench.action.closeAllEditors
重新打开已关闭的编辑器	Ctrl+Shift+T	Ctrl+Shift+T	workbench.action.reopenClosedEditor
保留编辑器	Ctrl+K Enter	Ctrl+K Enter	workbench.action.keepEditor
复制活动文件的路径	Shift+Alt+C	Ctrl+Shift+C	copyFilePath
在Windows中显示活动文件	Ctrl+K R	Ctrl+K R	workbench.action.files.revealActiveFileInWindows
在新窗口中打开活动文件	Ctrl+K O	Ctrl+K O	workbench.action.files.showOpenedFileInNewWindow
比较活动文件与	--	Ctrl+D	workbench.files.action.compareFileWith

2.10.4.10 显示

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
切换全屏	F11	Ctrl+Alt+F	workbench.action.toggleFullScreen

命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
查看:切换禅模式	Ctrl+K Z	Ctrl+K Z	workbench.action.toggleZenMode
离开禅宗模式	Escape Escape	Escape Escape	workbench.action.exitZenMode
放大	Ctrl+Numpad_Add Ctrl+Shift+= Ctrl+=	Ctrl+Numpad_Add Ctrl+Shift+= Ctrl+=	workbench.action.zoomIn
缩小	Ctrl +Numpad_Subtract Ctrl+Shift+- Ctrl+-	Ctrl +Numpad_Subtract Ctrl+Shift+- Ctrl+-	workbench.action.zoomOut
重置缩放	Ctrl+Numpad0	Ctrl+Numpad0	workbench.action.zoomReset
显示资源管理器	Ctrl+Shift+E	Alt+1	workbench.view.explorer
显示搜索	Ctrl+Shift+F	Ctrl+Shift+F	omnisearch.open.file
查看:显示源代码管理	Ctrl+Shift+G	Alt+9	workbench.view.scm
查看:切换运行和调试	Ctrl+Shift+D	Shift+Alt+F9 Alt+5 Ctrl+Shift+F8	workbench.view.debug
查看:切换扩展	Ctrl+Shift+X	Ctrl+Shift+X	workbench.view.extensions
查看:切换输出	Ctrl+Shift+U	Ctrl+Shift+U	workbench.action.output.toggleOutput
查看:切换终端	Ctrl+`	Shift+Escape Alt+F12	workbench.action.terminal.toggleTerminal

2.10.4.11 首选项

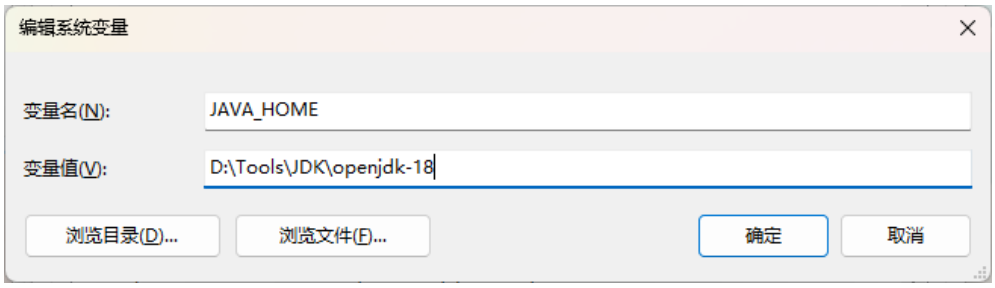
命令	键（CodeArts快捷键方案）	键（IDEA快捷键方案）	命令ID
打开设置	Ctrl+,	Ctrl+,	workbench.action.openSettings
打开键盘快捷键方式	Ctrl+K Ctrl+S	Ctrl+K Ctrl+S	workbench.action.openGlobalKeybindings
选择颜色主题	Ctrl+K Ctrl+T	Ctrl+`	workbench.action.selectTheme

2.11 附录

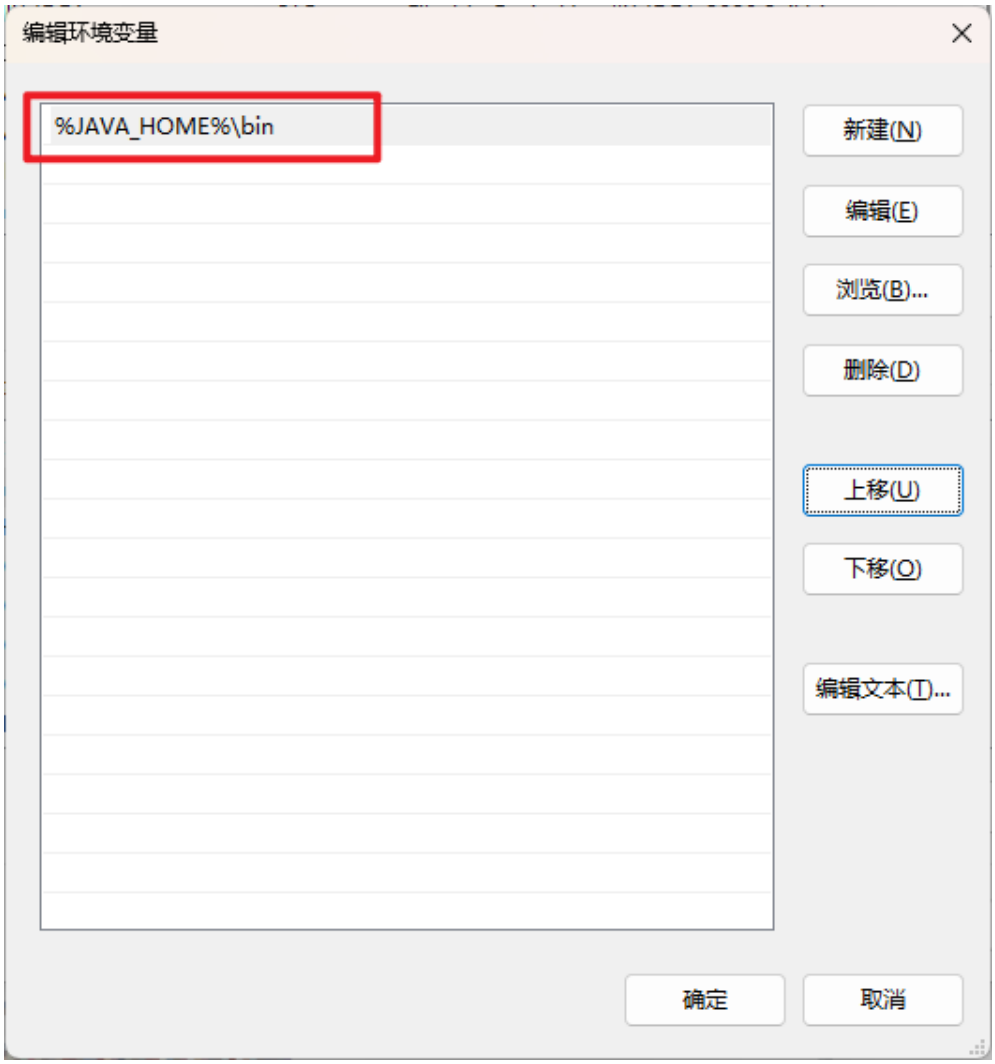
2.11.1 JDK 的安装及环境变量配置

1. 下载JDK，用户可以自行选择可访问的JDK下载链接并选择适合用户操作系统的JDK进行下载。
2. 根据用户下载的JDK类型，进行解压或安装。解压或安装成功后进入步骤3。
3. 如果用户使用的是Windows操作系统，用户可以按照如下步骤配置系统环境变量：

   （1）添加JAVA HOME变量，将步骤2中的JDK的安装/解压目录添加到该变量中，如：D:\Tools\JDK\openjdk-18。

   （2）编辑Path环境变量，添加JDK的bin目录地址：%JAVA_HOME%\bin。



4. 如果用户使用的是Linux操作系统，用户可以通过如下方式设置环境变量：

（1）在终端中输入如下命令打开Profile文件并编辑：

```
sudo vi /etc/profile
```

（2）在profile文件中添加如下内容：

```
export JAVA_HOME=JDK安装/解压路径
export PATH=$JAVA_HOME/bin:$PATH
```

（3）修改完成后，先按“**ESC**”键，然后执行**:wq!**命令，保存该文件并退出vi编辑器。

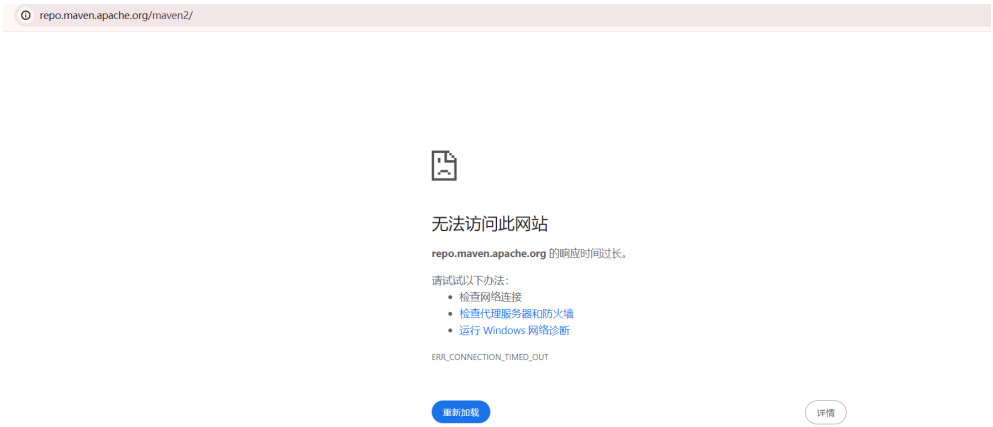
（4）在终端中输入如下命令让添加的环境变量生效：

```
source /etc/profile
```

2.11.2 Maven 镜像源检查及配置

1. CodeArts IDE for Java默认使用Maven官方镜像源，首先检查用户的网络是否可以访问Maven官方镜像源。在浏览器中访问如下地址：**https://repo.maven.apache.org/maven2**。
2. 如步骤1中地址可以访问，则用户可以直接使用CodeArts IDE进行Java项目开发。

3. 如步骤1中地址无法访问，出现下图类似的报错信息，请尝试通过配置系统网络代理等方式修复网络问题。



4. 如确认无法访问Maven官方镜像源，请配置当前网络中可以访问的Maven镜像源，具体配置可参考步骤5。
5. Maven的配置文件默认位置位于操作系统用户路径下，如在Windows中路径为：“C:\Users\用户名\.m2\settings.xml”，在Linux系统中，路径为：“~/.m2/settings.xml”。如当前系统中没有该settings.xml文件，用户可以参考上述路径新建此文件，文件内容可参考如下示例：

Maven 配置文件示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">

 <localRepository/>
 <interactiveMode/>
 <usePluginRegistry/>
 <offline/>
 <pluginGroups/>
 <servers/>
 <mirrors>
  <mirror>
   <id>Maven镜像源id</id>
   <name>Maven镜像源名称</name>
   <mirrorOf>central</mirrorOf>
   <url>Maven镜像源地址</url>
  </mirror>
 </mirrors>
 <proxies/>
 <profiles/>
 <activeProfiles/>
</settings>
```

⚠ 注意

如果是需要通过配置代理上网的环境，CodeArts IDE中也需要配置相同的代理才可以正常访问Maven官方镜像源。代理相关配置可以参考：**7.2.5-代理配置**。

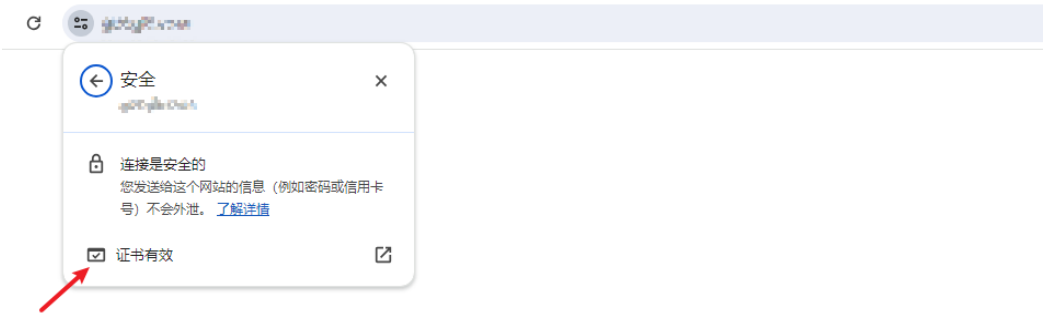
2.11.3 导入 JDK 证书

每当用户的应用程序尝试通过SSL（例如：HTTPS，IMAPS，LDAPS）连接到另一个应用程序时，它只能在可以信任该应用程序的情况下连接到该应用程序。在Java中处理信任的方式是用户拥有一个“信任库”文件（通常为$JAVA_HOME/lib/security/cacerts）。此信任存储文件包含受信任的证书，Java使用它来确定其他应用程序使用的SSL证书是否受信任。 Java只信任由证书颁发机构（CA）签名的证书，该机构的证书位于信任存储区中，或者是添加到信任存储区中的公共证书。用户可以参照如下步骤向JVM中导入证书：

步骤1 使用浏览器，访问用户所需要添加证书的网站，单击如下图标，出现对话框：

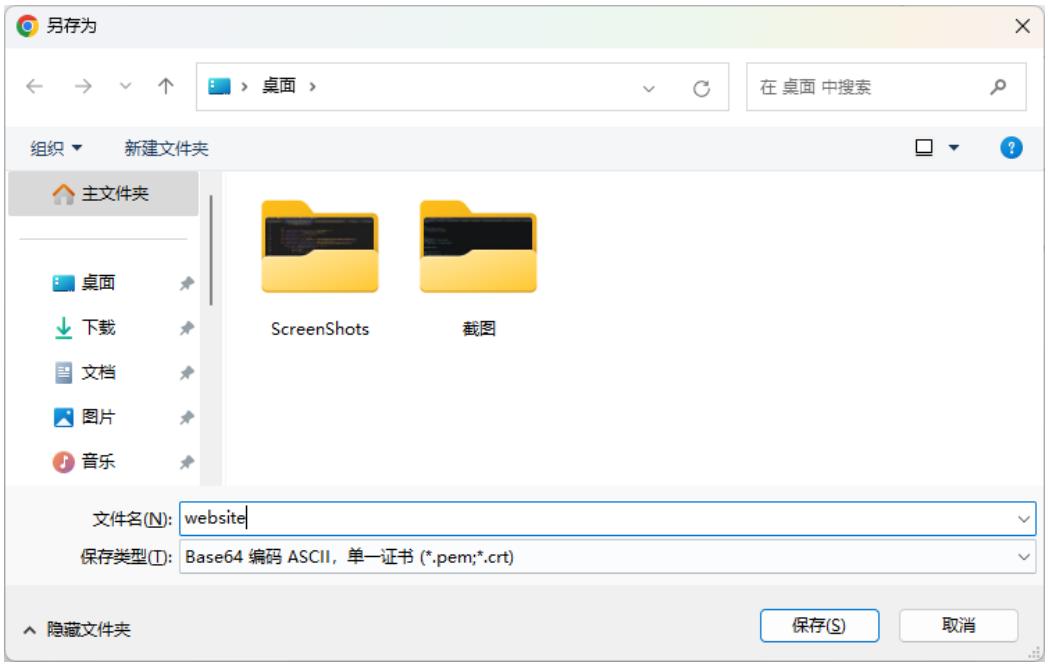
步骤2 单击如下选项：

步骤3 单击证书图标：



步骤4 在对话框中，切换到“详细信息”页签，单击导出按钮：

步骤5 修改名称，单击保存，保存类型可以参考下图：



步骤6 打开终端，输入如下命令导入网站证书：

```
<JAVA_HOME>/bin/keytool -importcert -alias <server_name> -keystore <JAVA_HOME>/lib/security/cacerts -file website.crt
```

其中<server_name>为要添加的网站的别名。

website.crt为步骤5中导出的证书地址。

步骤7 重新运行工程，查看是否可以正常从对应网站下载依赖。

----结束

注意

1. JAVA_HOME：JAVA_HOME变量的配置请参见**JDK的安装及环境变量配置**。
2. 不同浏览器在下载证书时的步骤会有一定的差异。
3. 如有更多问题，请联系CodeArts IDE工程师。

2.11.4 登录 ManageOne 运营面

步骤1 使用浏览器，通过地址“https://*ManageOne运营面的访问地址*”，登录ManageOne运营面，或通过地址“https://*ManageOne主门户的访问地址*”，登录ManageOne主门户，选择“云服务管理中心”，进入ManageOne运营面。

- 密码方式：输入账号和密码。
  - 管理面资源承载租户默认账号密码：请参见由HCC Turnkey导出的部署参数表《xxx_export_all_v2_CN.xlsx》>“基本参数”页签中的“secondlevel_vdcuser_for_deploy”、“secondlevel_vdcpassword_for_deploy”的值。
  - 管理员默认账号密码：请参见**《华为云Stack 8.6.0 账户一览表》**中“A类（Portal）”页签，产品名称为“ManageOne”，账户登录界面名称为“ManageOne运营面”获取。

- USB Key认证：插入已预置用户证书的USB Key，选择设备和用户证书，并输入PIN码。

  说明

  仅在SM系列商密算法场景下支持USB Key方式。

----结束

3 最佳实践

3.1 基于 CodeArts IDE 快速创建简单的 C++工程

3.1.1 使用 CodeArts IDE for Cpp 开发 OpenGl 示例工程

功能介绍

CodeArts IDE面向开发者提供的智能化可扩展桌面集成开发环境（IDE），结合行业和产业开发套件，实现一站式开发体验。

- 编码新体验，开发更高效：内置自研C/C++语言开发支持，提供全新的工程加载、语法着色、符号解析、编码重构和运行调试等开发体验，提升开发效率。
- 能力可扩展，生态更开放：支持基于插件的能力扩展，开放的插件标准，开源的插件框架，形成更加开放的生态系统。
- 界面可裁剪，体验更优质：支持基于组件的界面剪裁，在精简模式下形成专用工具的优质体验，又可以在需要时升级为全模式的全量IDE工具。

CodeArts IDE for Cpp：基于C/C++语言开发CMake工程，并通过CodeArts IDE完成从工程创建、代码编写、运行调试到发布测试的全过程。基于插件扩展可以将个人开发者作业流集成其中，实现从需求到提交的全部过程，更可在业务中集成提供的诸多能力，提升应用交付效率。

本实验将指导开发者通过CodeArts IDE for Cpp平台，在本地桌面快速开发一个基于Qt实现的简单项目。通过本实验用户将体验到：

- 如何在CodeArts IDE for Cpp上进行基于CMake项目的本地编译构建。
- 在CodeArts IDE上调试和运行。
- 实现一个简单OpenGL demo。

前置条件

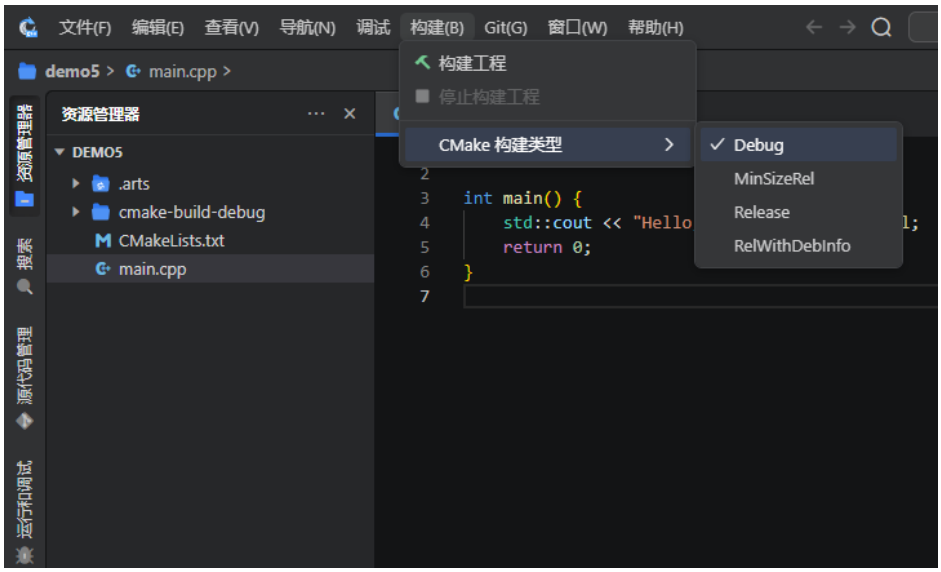
- 安装CodeArts IDE for Cpp，请参考**下载安装CodeArts IDE客户端**。
- 从CodeArts IDE下载页下载样例工程cpp-sample-v01.zip，如**下图**所示。

图 3-1 下载 Cpp 样例 demo

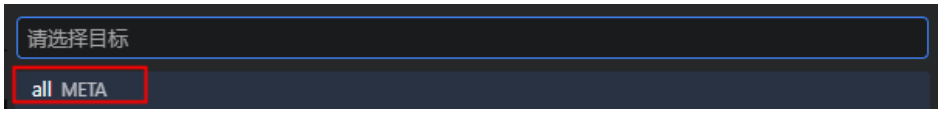
编译构建与运行调试

通过CodeArts IDE for Cpp客户端以文件夹的形式打开代码包。

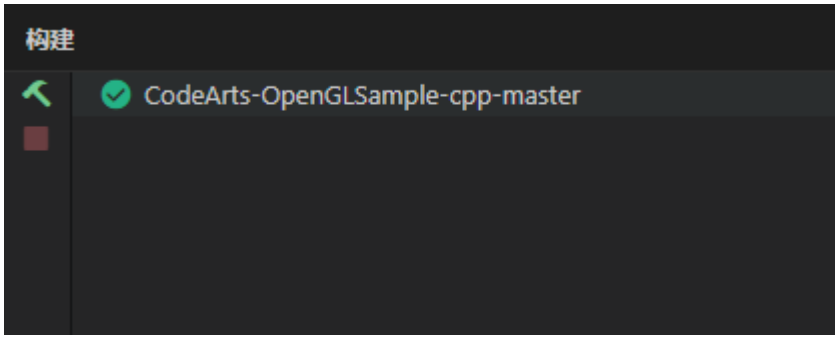
步骤1 在本地CodeArts IDE for Cpp页面，单击“构建”，选择“CMake构建类型 > Debug”。



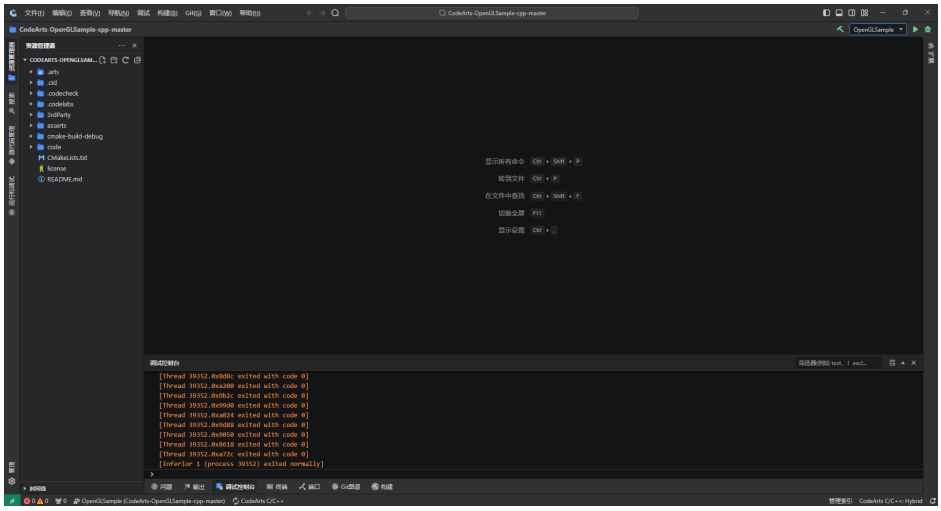
步骤2 单击右上角构建按钮，然后在弹窗中选择“all META”。



步骤3 当编译构建成功后，日志打印finished，并显示构建成功的图示。



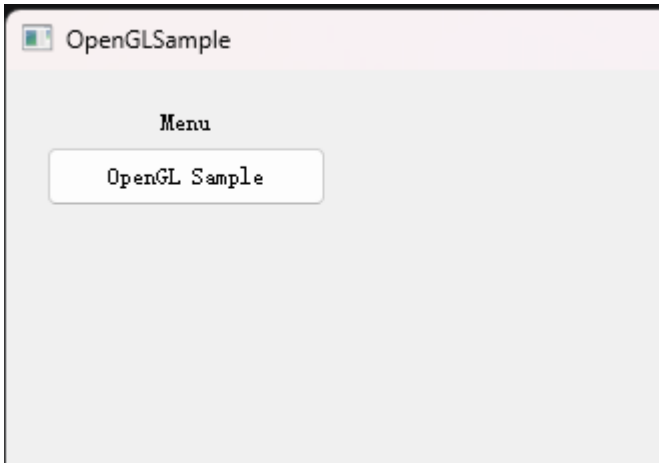
步骤4 构建成功后，在页面的右上角选择“OpenGLSample”，然后单击右侧的运行按钮 。



----结束

运行结果

运行后出现下图所示弹窗，单击弹窗内按钮“OpenGL Sample”。



单击后结果如下所示：

关键代码片段

```
// MyWidget类的构造函数，初始化一些成员变量并设置布局
MyWidget::MyWidget(QWidget *parent) : QWidget(parent)
{
    resize(800, 600);// 设置窗口大小为800*600

  // 创建一个垂直布局，用于放置菜单标题和按钮
  m_menuButtonLayout = new QVBoxLayout;
  m_menuButtonLayout->setAlignment(Qt::AlignTop);  // 设置布局的对齐方式为顶部对齐
  // 创建一个标签，设置文本为"Menu"，并设置其居中对齐
  m_menuTitle = new QLabel(this);
  m_menuTitle->setText("Menu");
  m_menuTitle->setAlignment(Qt::AlignCenter);
  // 创建一个按钮，设置文本为"OpenGL Sample"，并设置其固定大小
  m_openGLDemoButton = new QPushButton("OpenGL Sample", this);
  m_openGLDemoButton->setFixedSize(140, 30);
  // 将标签和按钮添加到垂直布局中
  m_menuButtonLayout->addWidget(m_menuTitle);
  m_menuButtonLayout->addWidget(m_openGLDemoButton);
  // 创建一个新的窗口部件，并设置其布局为上面创建的垂直布局
  m_menuWidget = new QWidget(this);
  m_menuWidget->setLayout(m_menuButtonLayout);
  // 创建一个帧，设置其形状为垂直线并设置阴影效果
  QFrame* line = new QFrame(this);
  line->setFrameShape(QFrame::VLine);
  line->setFrameShadow(QFrame::Sunken);
  // 创建一个水平布局，用于放置菜单窗口部件和帧
  QHBoxLayout *mainLayout = new QHBoxLayout;
  mainLayout->addWidget(m_menuWidget, 1);  // 将菜单窗口部件添加到布局中，并设置其伸缩因子为1
  mainLayout->addWidget(line);  // 将帧添加到布局中
  setLayout(mainLayout);  // 将主布局设置为窗口的布局
  // 连接按钮的单击信号到槽函数ShowOpenGLDemo
  connect(m_openGLDemoButton, &QPushButton::clicked, this, &MyWidget::ShowOpenGLDemo);
}
// 槽函数，用于显示OpenGL演示
void MyWidget::ShowOpenGLDemo()
{
  OpenGLDemoShow();  // 调用OpenGL演示显示函数
  return;
}
```

3.2 基于 CodeArts IDE 快速创建简单的 Java 工程

3.2.1 使用 CodeArts IDE for Java 开发简单的 Java 工程

功能介绍

CodeArts IDE for Java是一个JAVA集成开发环境，将文本编辑器和强大的开发工具（如智能代码补全、导航、重构和调试）集成在一起。

- 调试热替换：支持在调试模式下运行程序。当程序挂起时，在“运行”和“调试”视图中检查其输出。找到错误并更正它，然后重新运行程序。在Java上下文中，可通过热代码替换功能来动态修改和重新加载类，无需停止调试会话。
- 强大的编码辅助能力：CodeArts IDE for Java可以深入理解代码，提供上下文感知的机器学习辅助补全功能，可以补全单条语句或整行代码。对于代码编辑器中的任何符号，开发者可以查看其定义和相关文档。集成开发环境可高亮显示错误，并让开发者直接在代码编辑器中对错误采取行动。“问题”视图会列出当前打开的文件中检测到的所有问题。
- Java构建工具集成：CodeArts IDE for Java提供对Maven和Gradle构建系统的内置支持，包括项目和依赖关系解析，以及运行Maven目标和Gradle任务的能力。

Spring Boot和Thymeleaf是目前非常流行的Java开发框架和模板引擎。Spring Boot提供了快速构建Web应用程序的能力，而Thymeleaf则是一种功能强大且易于使用的模板引擎，可以帮助在服务器端生成动态的HTML页面。在本文中，将使用Spring Boot和Thymeleaf来实现一个完整的用户管理功能，包括增加、删除、修改和查询用户信息。 以及使用CodeArts IDE for Java构建，运行Java工程，并且学到Java语言的一些基础知识，例如Java的一些基本类型。通过Java语言创建，读取，并修改本地文件的能力。

前置条件

- 下载并安装CodeArts IDE for Java，请参考**下载安装CodeArts IDE客户端**。
- 从CodeArts IDE下载页下载样例工程java-sample-v01.zip，如**下图**所示。

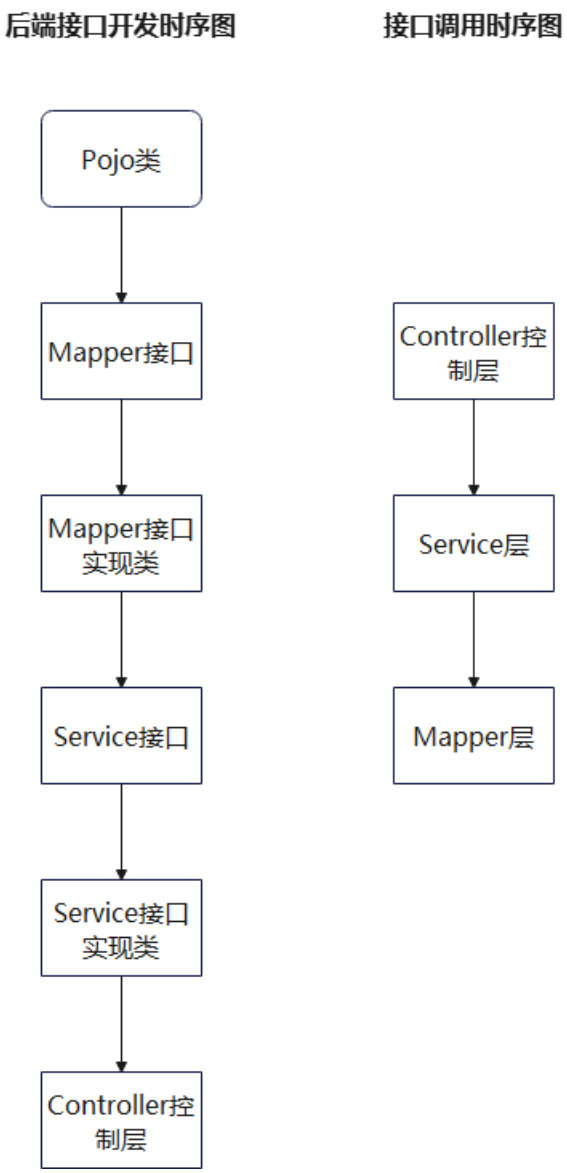
图 3-2 下载 Java 样例 Demo

- Java语言基础。
- 安装Java JDK 1.8及其以上版本。
- 引入spring-boot-starter-web、spring-boot-starter-thymeleaf和lombok（可选）相关依赖。

开发时序图

样例工程可以按照时序图节点顺序进行开发，如**图3-3**所示。

图 3-3 样例工程开发时序图



关键代码片段

- 后端实现过程

pojo类：（@Data是Lombok注解，不需要写get、set方法了。）

```
@Data
public class User {
    private String uuid;
    private Integer id;
    private String name;
    private Integer age;
    private String department;
    public User(){}
    public User(Integer id, String name, Integer age, String department) {
        this.uuid = UUID.randomUUID().toString();
        this.id = id;
        this.name = name;
```

```
        this.age = age;
        this.department = department;
    }
}
```

mapper接口：

```
public interface IUserMapper {
    public List<User> getUserList();
    public boolean addUser(User user);
    public boolean updateUser(User user);
    public boolean deleteUser(String uuid);
}
```

mapperImpl实现类：

```
@Component
public class UserMapperImpl implements IUserMapper {
    @Override
    public List<User> getUserList() {
        return UserData.userList;
    }
    @Override
    public boolean addUser(User user) {
        user.setUuid(UUID.randomUUID().toString());
        return UserData.userList.add(user);
    }
    @Override
    public boolean updateUser(User user) {
        Stream<User> userStream = UserData.userList.stream().filter(userItem -> {
            if (user.getId().equals(userItem.getId())) {
                if (user.getId() != null) {
                    userItem.setId(user.getId());
                }
                if ( !StringUtils.isEmpty(user.getName())) {
                    userItem.setName(user.getName());
                }
                if (!StringUtils.isEmpty(user.getDepartment())) {
                    userItem.setDepartment(user.getDepartment());
                }
                if (!StringUtils.isEmpty(user.getAge())) {
                    userItem.setAge(user.getAge());
                }
                return true;
            }
            return false;
        });
        return userStream.count() >= 1;
    }
    @Override
    public boolean deleteUser(String uuid) {
        int sizeBeforeDelete = UserData.userList.size();
        List<User> deletedUsers = UserData.userList.stream().filter(user ->
uuid.equals(user.getUuid())).collect(Collectors.toList());
        deletedUsers.forEach(UserData.userList::remove);
        return sizeBeforeDelete != UserData.userList.size();
    }
}
```

service接口：

```
public interface IUserService {
    public List<User> getUserList();
    public boolean addUser(User user);
    public boolean updateUser(User user);
    public boolean deleteUser(String id);
}
```

serviceImpl实现类：


```
@Service
public class UserServiceImpl implements IUserService {
    @Autowired
    IUserMapper userMapperImpl;
    @Override
    public List<User> getUserList() {
        return userMapperImpl.getUserList();
    }
    @Override
    public boolean addUser(User user) {
        return userMapperImpl.addUser(user);
    }
    @Override
    public boolean updateUser(User user) {
        return userMapperImpl.updateUser(user);
    }
    @Override
    public boolean deleteUser(String id) {
        return userMapperImpl.deleteUser(id);
    }
}
```

controller控制层： 用户管理接口Controller：

```
@RestController
@RequestMapping("/user")
public class UserController {
    @Autowired
    IUserService userServiceImpl;
    @GetMapping("/getUserList")
    public List<User> getUserList() {
        return userServiceImpl.getUserList();
    }
    @PostMapping("/addUser")
    public boolean addUser(User user) {
        return userServiceImpl.addUser(user);
    }
    @PostMapping("/updateUser")
    public boolean updateUser(User user) {
        return userServiceImpl.updateUser(user);
    }
    @PostMapping("/deleteUser")
    public boolean deleteUser(String uuid) {
        return userServiceImpl.deleteUser(uuid);
    }
}
```

用户欢迎界面Controller：

```
@Controller
public class WelcomePageController {
    @RequestMapping("/")
    public String user() {
        return "user";
    }
}
```

用户数据：

```
public class UserData {
    public static List<User> userList = new ArrayList<>();

    public static void initUserData() {
        userList.add(new User(1, "张三", 20, "人力资源部"));
        userList.add(new User(2, "李四", 18, "后勤部"));
        userList.add(new User(3, "王五", 21, "研发部"));
        userList.add(new User(4, "赵六", 25, "产品部"));
    }
}
```

启动类：（每次启动调试需要初始化用户数据）


```
@SpringBootApplication
public class UserManagerApplication {

    public static void main(String[] args) {
        UserData.initUserData(); // 初始化用户数据
        SpringApplication.run(UserManagerApplication.class, args);
    }

}
```

- 前端实现过程(thymeleaf模板+html+ajax)

user.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>User Manager</title>
    <script src="https://cdn.bootcdn.net/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
    <script>
        $(function () {
            queryData();
        });

        function queryData() {
            // 发送 AJAX 请求
            $.ajax({
                url: 'http://localhost:8080/user/getUserList',
                type: 'GET',
                dataType: 'json',
                success(res) {
                    var html = '';
                    for (var i = 0; i < res.length; i++) {
                        html += '<tr><td>' + res[i].id + '</td>' + '<td>' + res[i].name + '<td>' + res[i].age + '<td>' +
res[i].department + '</td><td> <button onclick="del(\'' + res[i].uuid + '\')">删除</button></td></tr>';
                    }
                    // console.log(html);
                    $('#tt').html(html);
                },
                error: function (xhr, status, error) {
                    console.error(error); // 处理错误情况
                }
            });
        }

        function add() {
            const id = document.getElementById('InputId').value;
            const name = document.getElementById('InputName').value;
            const age = document.getElementById('InputAge').value;
            const department = document.getElementById('InputDepartment').value;

            if (!validateId(id)) {
                return;
            }

            $.ajax({
                url: `http://localhost:8080/user/addUser?id=${id}&name=${name}&age=${age}&department=$
{department}`,
                type: 'post',
                dataType: 'json',
                success: function (res) {
                    queryData();
                    if (res == true) {
                        alert('添加成功');
                    } else {
                        alert('添加失败');
                    }
                },
```

```
            error: function (xhr, status, error) {
                console.error(error);
            }
        });
        document.getElementById('InputId').value = '',
            document.getElementById('InputName').value = '';
        document.getElementById('InputAge').value = '';
        document.getElementById('InputDepartment').value = '';
    }

    function update() {
        const id = document.getElementById('updateInputId').value;
        const name = document.getElementById('updateInputName').value;
        const age = document.getElementById('updateInputAge').value;
        const department = document.getElementById('updateInputDepartment').value;

        if (!validateId(id)) {
            return;
        }

        $.ajax({
            url: `http://localhost:8080/user/updateUser?id=${id}&name=${name}&age=${age}&department=$
{department}`,
            type: 'post',
            dataType: 'json',
            success: function (res) {
                queryData();
                if (res == true) {
                    alert('修改成功');
                } else {
                    alert('修改失败');
                }
            },
            error: function (xhr, status, error) {
                console.error(error);
            }
        });
        document.getElementById('updateInputId').value = '',
            document.getElementById('updateInputName').value = '';
        document.getElementById('updateInputAge').value = '';
        document.getElementById('updateInputDepartment').value = '';
    }

    function del(uuid) {
        $.ajax({
            url: `http://localhost:8080/user/deleteUser?uuid=${uuid}`,
            type: 'post',
            dataType: 'json',
            success: function (res) {
                queryData();
                if (res == true) {
                    alert('删除成功');
                } else {
                    alert('删除失败');
                }
            },
            error: function (xhr, status, error) {
                console.error(error);
            }
        });
    }

    function validateId(id) {
        if (id == '') {
            alert("Id值不能为空");//空值校验
            return false;
        }
        return true;
    }
```

```
    </script>
</head>

<body>
    <div class="header">
        <!--<button onclick="queryData()">查询</button>-->
        <div class="items">
            <input type="number" value="" placeholder="请输入id" id="InputId" />
            <input type="text" value="" placeholder="请输入名称" id="InputName">
            <input type="number" value="" placeholder="请输入年龄" id="InputAge">
            <input type="text" value="" placeholder="请输入部门" id="InputDepartment">
            <button onclick="add()">添加</button>
        </div>
        <div class="items">
            <input type="number" value="" placeholder="请输入id" id="updateInputId" />
            <input type="text" value="" placeholder="请输入名称" id="updateInputName">
            <input type="number" value="" placeholder="请输入年龄" id="updateInputAge">
            <input type="text" value="" placeholder="请输入部门" id="updateInputDepartment">
            <button onclick="update()">修改</button>
        </div>
    </div>
    <div class="table">
        <table>
            <thead>
                <tr>
                    <th>用户Id</th>
                    <th>姓名</th>
                    <th>年龄</th>
                    <th>部门</th>
                    <th>操作</th>
                </tr>
            </thead>
            <tbody id="tt">
            </tbody>
        </table>
    </div>
</body>
<style>
    table {
        margin: 20px;
        border-collapse: collapse;
    }

    th {
        background-color: #cccccc;
    }

    tr {
        text-align: center;
        border: 1px solid #ccc;
    }

    td {
        padding: 20px;
        border: 1px solid #ccc;
    }

    button {
        background-color: white;
        width: 70px;
        border-radius: 20px;
        border: 1px solid #ccc;
    }

    button :hover {
        background-color: #cccccc;
    }

    .header {
```



```
        display: flex;
        justify-content: center;
        flex-wrap: wrap;
    }

    .items {
        display: grid;
        margin-left: 10px;
    }

    .table {
        display: flex;
        justify-content: center;
    }
</style>

</html>
```

- 相关配置文件

相关的依赖信息如下，引入spring-boot-starter-web、spring-boot-starter-thymeleaf和lombok（可选）相关依赖：

```
<dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>annotationProcessor</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

application.properties：

```
spring.application.name=user-manager
server.port=8080
```

application.yml

```
thymeleaf:
  prefix:
    classpath: /templates   # 访问template下的html文件需要配置模板，映射
```

- 代码解析

在html代码中，主要利用onclick单击事件和jquery框架中的ajax交互。 jquery中ajax的语法格式：

```
$.ajax({
     url: 接口名,
     type: 'get',
     dataType: 'json',
     success: function(res) {
         queryData();
```



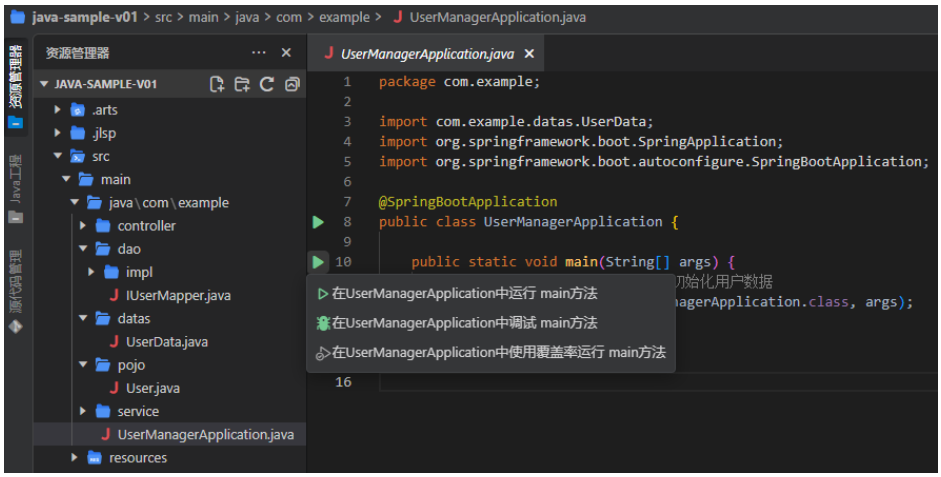
```
    },
    error: function(xhr, status, error) {
        console.error(error); // 处理错误情况
    }
});
```

运行调试

步骤1 在具有“main()”方法的“UserManagerApplication”类的代码编辑器中，单击或右键“main()”方法左侧**“运行/调试主类”**按钮（  ）。

步骤2 在弹出的下拉列表中，单击“在UserManagerApplication中运行main方法”或“在UserManagerApplication中调试main方法”项，将运行或调试项目。如下图所示：

图 3-4 代码编辑区区域“main()”方法处启动调试会话入口



步骤3 在浏览器地址栏输入“localhost:8080”。

----结束

输出结果示例

程序运行截图如下：

3.3 附录

3.3.1 下载安装 CodeArts IDE 客户端

3.3.1.1 登录 CodeArts IDE 运营面

步骤1 使用浏览器，以VDC管理员或VDC业务员**登录ManageOne运营面**。

步骤2 在页面左上角单击☰，打开服务列表。

步骤3 单击服务列表中的“软件开发生产线 > 集成开发环境 CodeArts IDE”进入CodeArts IDE运营面控制台。

如果登录用户第一次进入CodeArts IDE运营面控制台，页面将弹出提示框，单击“同意并继续”即可继续使用服务。

----结束

3.3.1.2 下载 CodeArts IDE 客户端

步骤1 在CodeArts IDE运营面首页左侧选择“CodeArts IDE > 下载客户端”。

步骤2 在下图所示的页面中，可以选择“Cpp”或者“Java”这两种开发环境的客户端。

步骤3 单击下图所示页面中间的“下载”按钮，选择Windows操作系统或者Linux操作系统的安装包进行下载。

----结束

3.3.1.3 安装 CodeArts IDE 客户端

C++客户端与JAVA客户端安装过程相同，本文以C++客户端为例，在不同操作系统安装。

- **Windows版本安装**。
- **Linux Arm版本安装**。
- **Linux X86版本安装**。

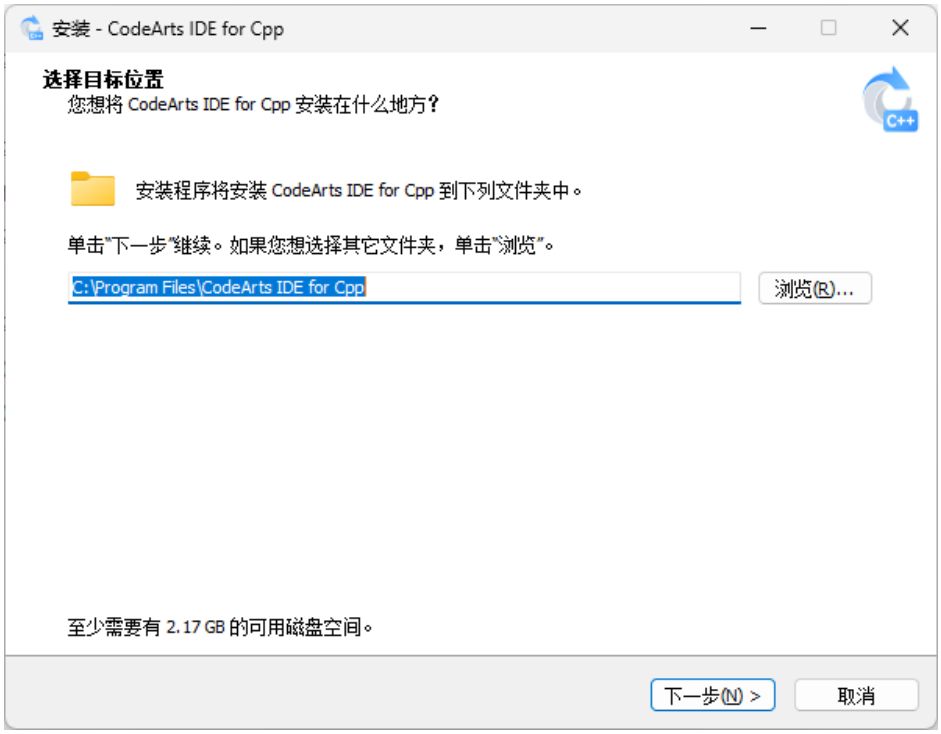
3.3.1.3.1 Windows 版本安装

约束限制

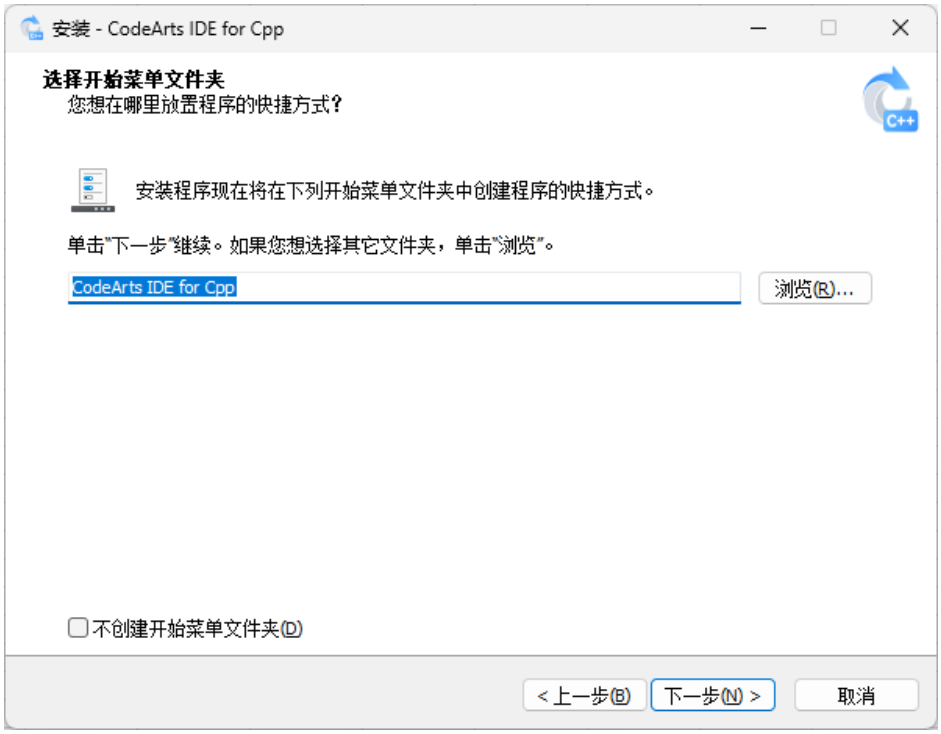
- 参考**下载CodeArts IDE客户端**，下载Windows版本CodeArts IDE安装包。
- 在安装软件包时，建议为解压和安装过程预留至少2.15GB的磁盘空间。

安装步骤

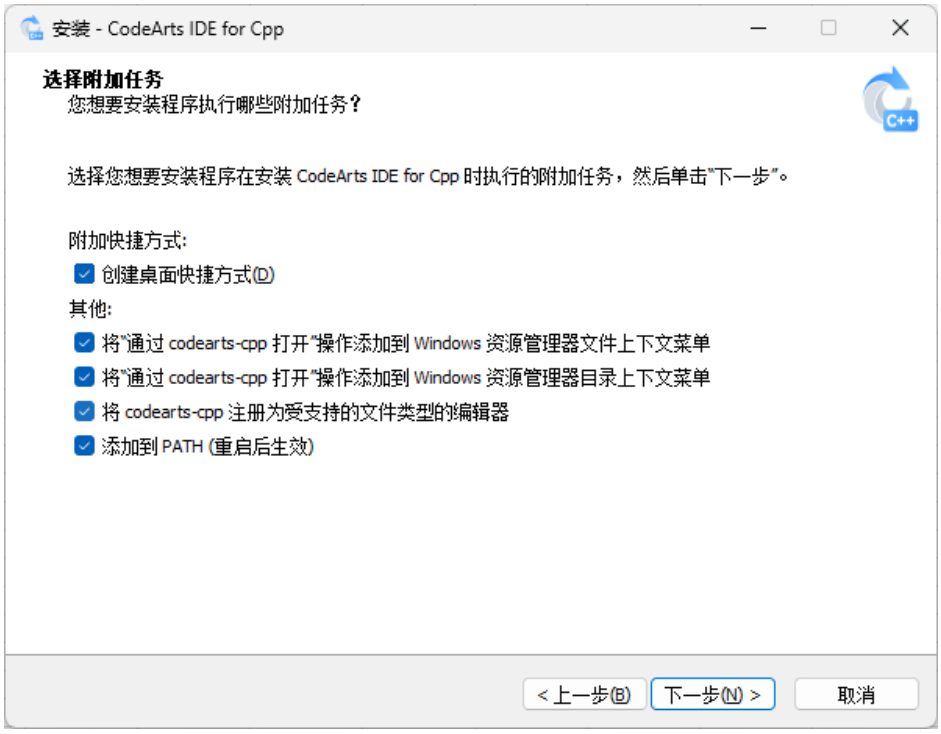
步骤1 选择目标位置，单击“下一步”。



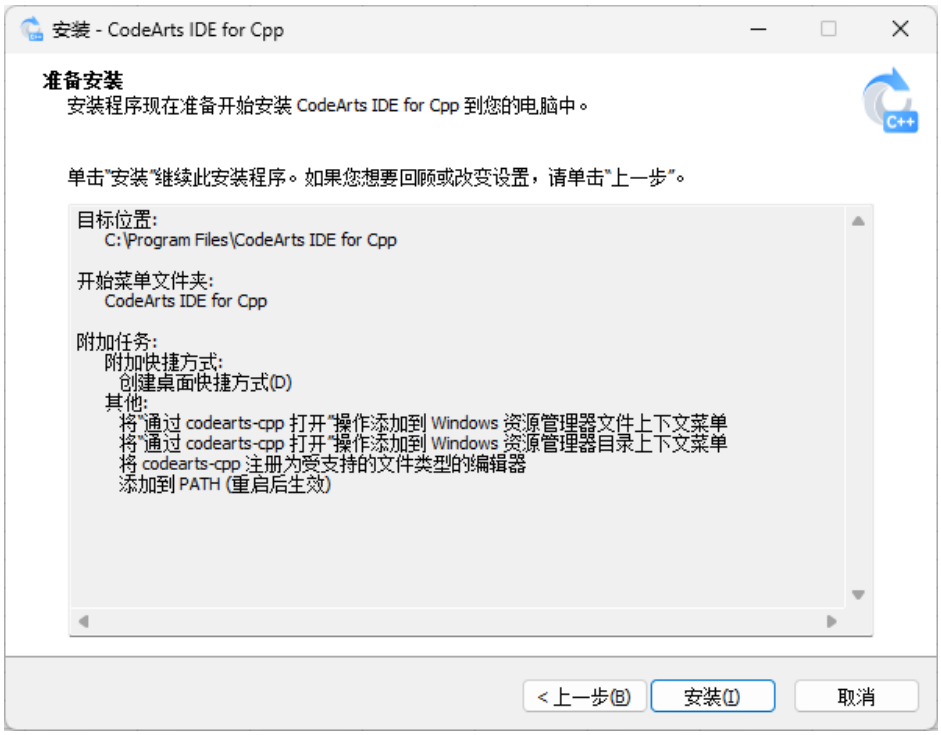
步骤2 选择开始菜单文件夹，单击“下一步”。



步骤3 按需选择附加任务，单击“下一步”。



步骤4 单击“安装”按钮，等待安装完成。



步骤5 安装完成后显示“CodeArts IDE for Cpp安装完成”，单击“完成”按钮，退出安装程序。

----结束

3.3.1.3.2 Linux Arm 版本安装

约束限制

- Linux机器需要具有图形化界面的操作系统。
- 参考**下载CodeArts IDE客户端**，下载Linux Arm版本CodeArts IDE安装包。
- 在安装软件包时，建议为解压和安装过程预留至少1GB的磁盘空间。

安装和启动步骤

步骤1 以具备sudo权限的用户（非root用户）登录Linux机器。

步骤2 执行**cd ~/Downloads**进入安装包所在目录，此命令表示安装包所在目录为“/home/用户名/Downloads”。

步骤3 执行**sudo dpkg -i codearts-cpp-arm64-linux-xxx.deb**安装客户端，此命令中的安装包名需替换为实际安装包名。

步骤4 安装完成后，在终端窗口根据安装包的版本（C/C++或JAVA）执行如下对应命令，启动CodeArts IDE客户端。其中，C/C++客户端执行**codearts-cpp**，JAVA客户端执行**codearts-java**。

----结束

3.3.1.3.3 Linux X86 版本安装

约束限制

- Linux机器需要具有图形化界面的操作系统。
- 参考**下载CodeArts IDE客户端**，下载Linux X86版本CodeArts IDE安装包。
- 在安装软件包时，建议为解压和安装过程预留至少1GB的磁盘空间。

安装步骤

步骤1 以具备sudo权限的用户（非root用户）登录Linux机器。

步骤2 执行**cd ~/Downloads**进入安装包所在目录，此命令表示安装包所在目录为“/home/用户名/Downloads”。

步骤3 执行**sudo dpkg -i codearts-cpp-x64-linux-xxx.deb**安装客户端，此命令中的安装包名需替换为实际安装包名。

步骤4 安装完成后，在终端窗口根据安装包的版本（C/C++或JAVA）执行如下对应命令，启动CodeArts IDE客户端。其中，C/C++客户端执行**codearts-cpp**，JAVA客户端执行**codearts-java**。

----结束

3.3.2 登录 ManageOne 运营面

步骤1 使用浏览器，通过地址“https://*ManageOne运营面的访问地址*”，登录ManageOne运营面，或通过地址“https://*ManageOne主门户的访问地址*”，登录ManageOne主门户，选择“云服务管理中心”，进入ManageOne运营面。

- 密码方式：输入账号和密码。

  - 管理面资源承载租户默认账号密码：请参见由HCC Turnkey导出的部署参数表《xxx_export_all_v2_CN.xlsx》>“基本参数”页签中的“secondlevel_vdcuser_for_deploy”、“secondlevel_vdcpassword_for_deploy”的值。
  - 管理员默认账号密码：请参见**《华为云Stack 8.6.0 账户一览表》**中“A类（Portal）”页签，产品名称为“ManageOne”，账户登录界面名称为“ManageOne运营面”获取。
- USB Key认证：插入已预置用户证书的USB Key，选择设备和用户证书，并输入PIN码。

  说明

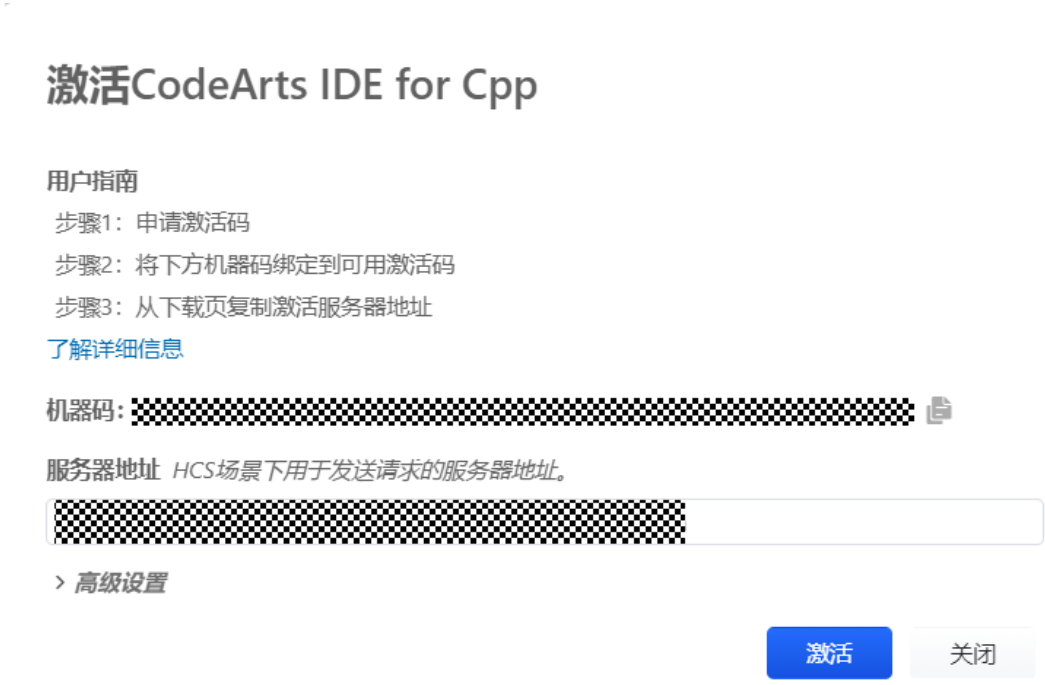
  仅在SM系列商密算法场景下支持USB Key方式。

----结束

4 常见问题

4.1 IDE 客户端激活码激活失败

打开CodeArts IDE客户端，在激活CodeArts IDE弹框中输入“服务器地址”，服务器地址可以从CodeArts IDE运营面客户端下载页面获取。输入服务器地址后，单击“激活”按钮，在正常使用的情况下激活成功，右下角弹出激活成功信息，如果遇到问题，可根据具体情况，按以下方法解决。



4.1.1 网络问题导致激活失败

问题现象

激活时出现错误提示：“由于网络错误，激活失败。请检查用户的网络连接并重试。”

解决方法

步骤1 单击“**确定**”关闭错误提示弹窗，检查“服务器地址”是否输入正确。请**登录CodeArts IDE运营面**，在CodeArts IDE运营面获取相关服务器地址。

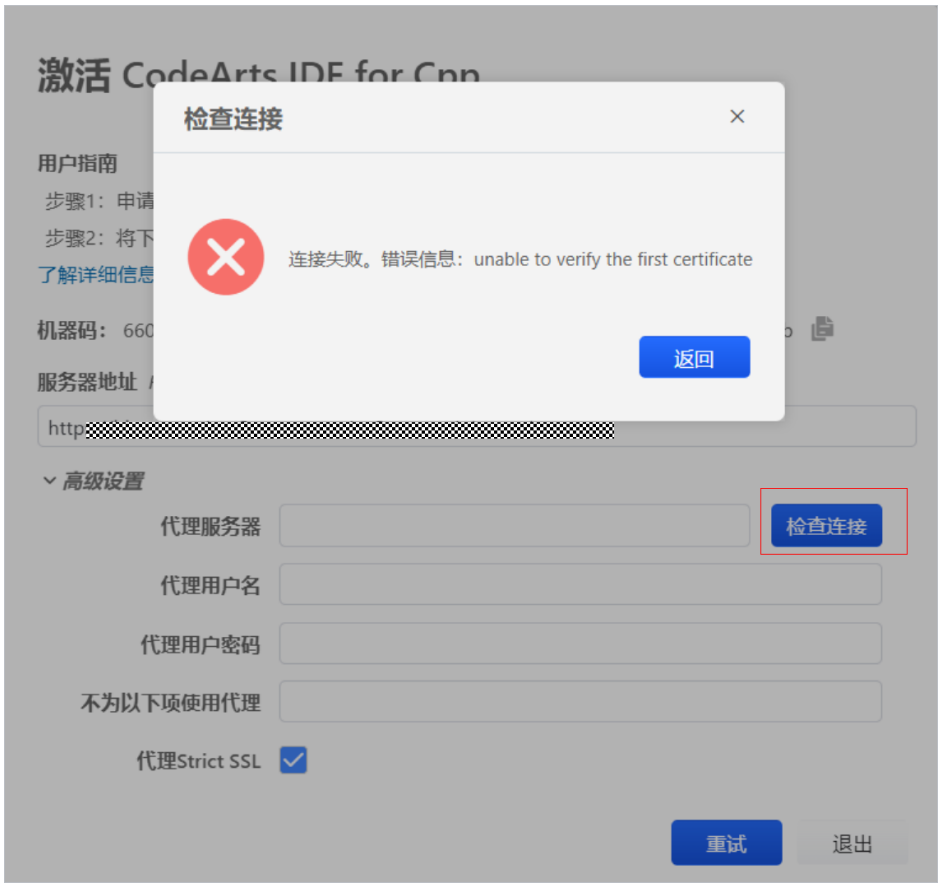
步骤2 单击“**重试**”查看检查结果。如仍失败，请根据错误提示尝试解决网络问题。

----结束

4.1.2 证书问题导致激活失败

问题现象

展开“**高级设置**”，单击“**检查连接**”后，出现错误提示：“连接失败。错误信息：unable to verify the first certificate”。



解决方法

由于未关闭SSL导致出现此现象，需要在高级设置里取消勾选“代理Strict SSL”。

4.1.3 代理问题导致激活失败

问题现象

激活时出现错误提示：“连接失败。错误信息：aborted”。

解决方法

需要用户配置正确的代理。在“高级设置”里设置相关的代理，涉及到的配置项有：

- 代理用户名
- 代理用户密码
- 不为以下项使用代理

4.1.4 未绑定激活码导致激活码激活失败

问题现象

激活时出现错误提示：“激活失败，未找到激活码。”

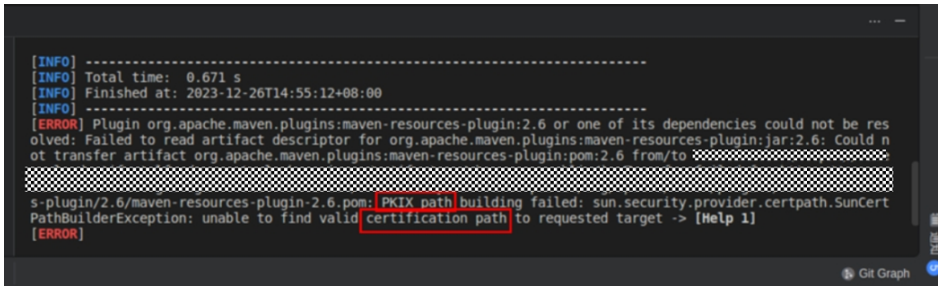
解决方法

请参考**申请激活码**与**绑定激活码并激活**，申请激活码后绑定激活码，完成激活。

4.2 Maven 工程的依赖无法下载，单击构建报 certification path 证书错误

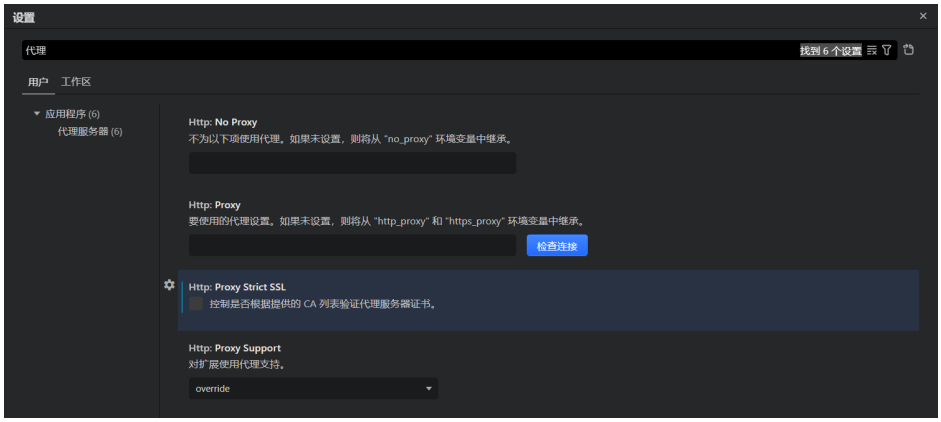
问题现象

打开Maven工程后，无法下载相关依赖，且单击底部构建视图的构建按钮，构建视图出现如下的“PKIX path”和“certification path”的错误：

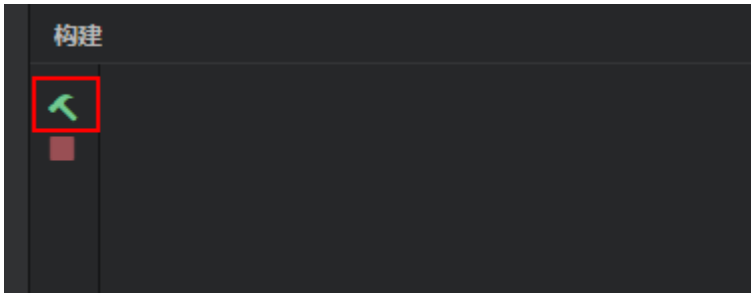


解决方法

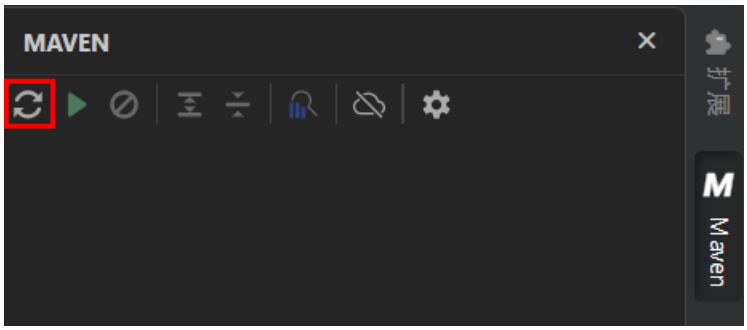
步骤1 在设置中搜索**代理**，取消勾选“**Http: Proxy Strict SSL**”。



步骤2 单击底部“构建”视图中的“**构建工程**”按钮。



步骤3 构建完成后，单击“MAVEN”视图中的“**重新加载所有Maven工程**”按钮重新加载即可。

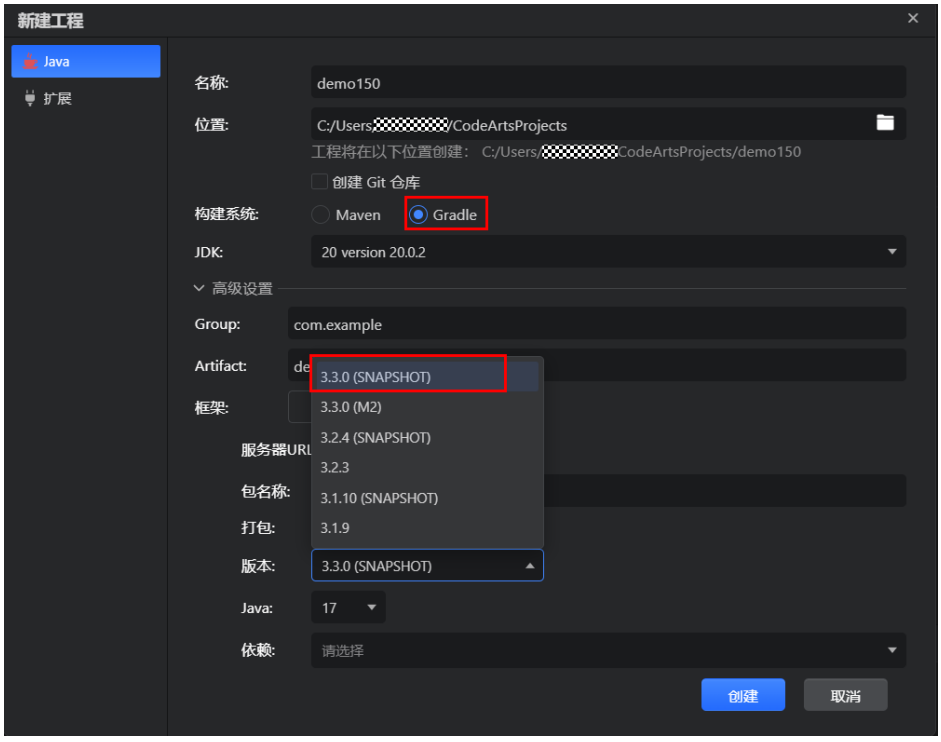


----结束

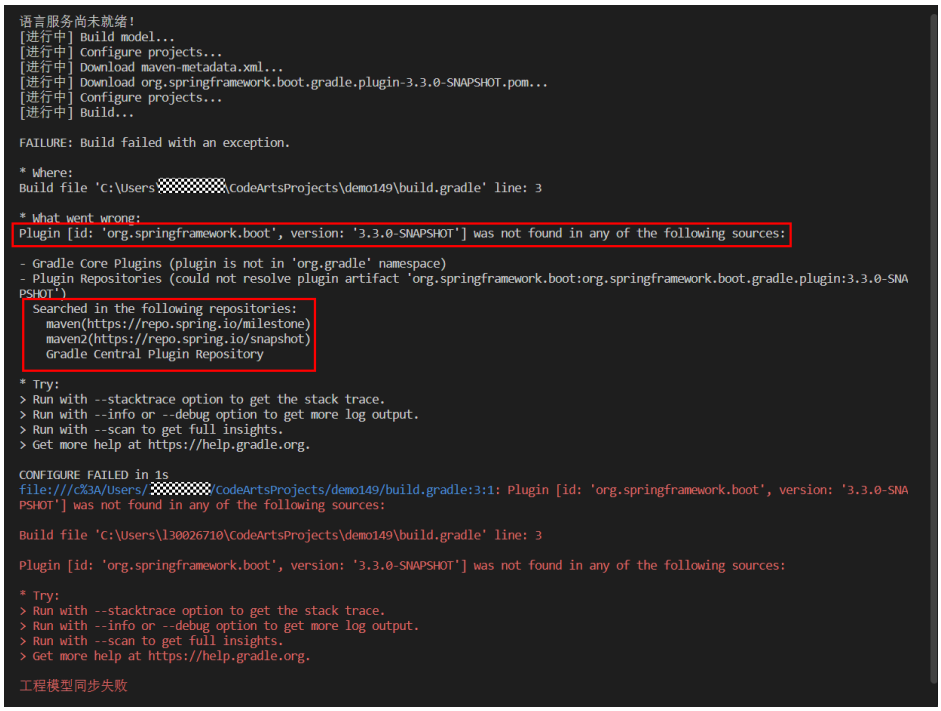
4.3 Gradle 工程的依赖无法下载

问题现象

通过“新建 > 新建工程”菜单新建SpringBoot工程时，构建系统选择“**Gradle**”，SpringBoot的版本选择“**SNAPSHOT**”版本。



打开上述创建的工程后，底部“**构建**”视图出现如下的错误信息：

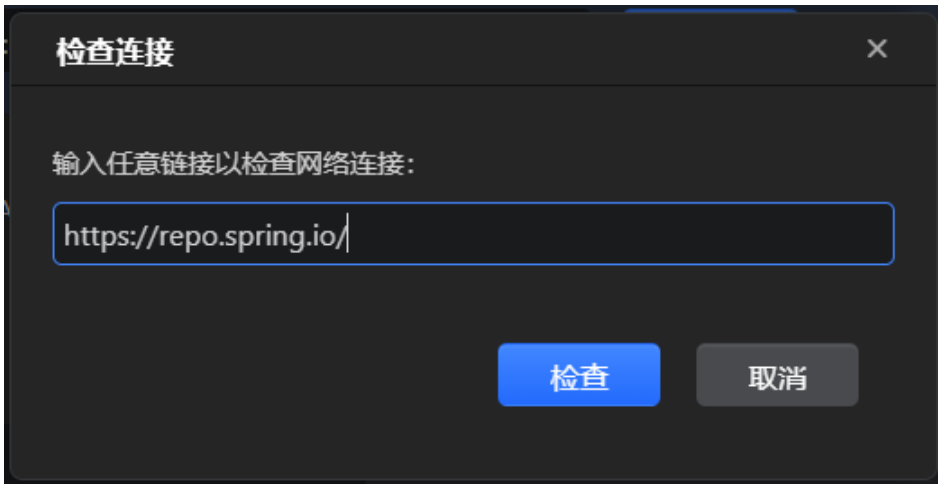


解决方法

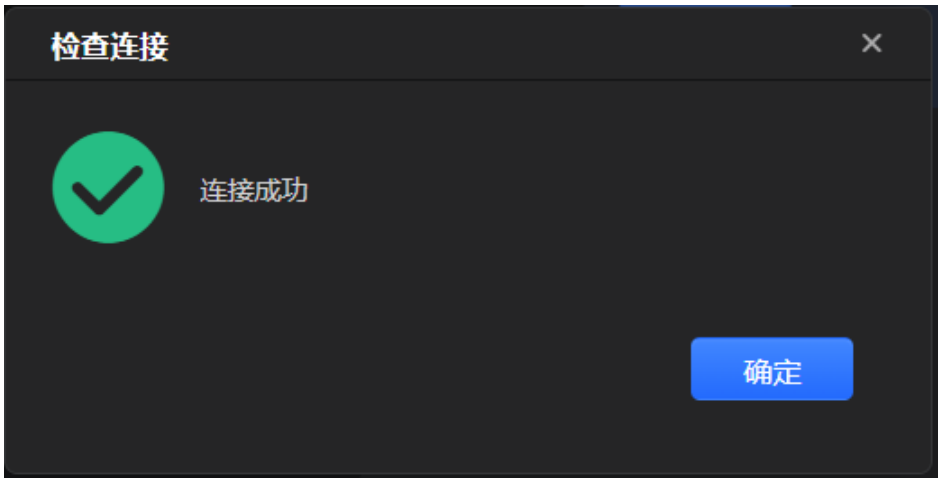
请按照下述步骤，检查当前网络环境是否可以访问**Spring镜像源**，或者出现在上述报错信息中的其他镜像源。

步骤1 单击左下角“管理 > 设置”菜单，然后在搜索框中搜索“**代理**”，单击“**Http: Proxy**”设置项中的“**检查连接**”按钮。

步骤2 在“检查连接”弹窗中，输入报错信息中的**镜像源**并单击“**检查**”按钮：

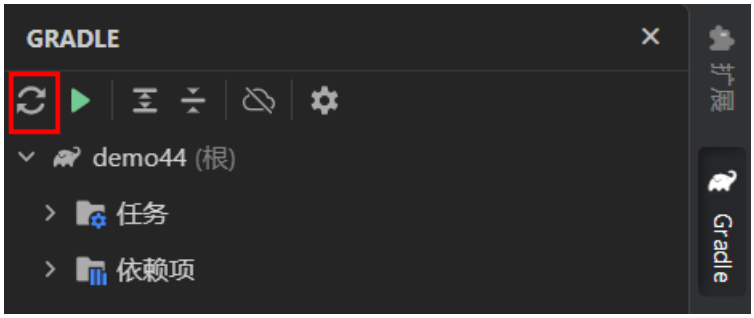


查看是否出现下述连接成功的提示：



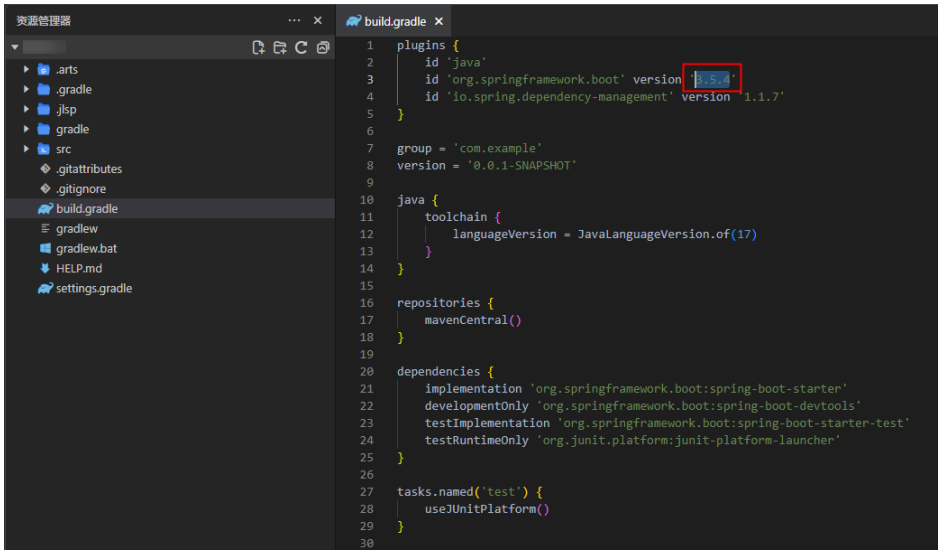
步骤3 如果出现检查连接成功的提示，则直接跳过步骤4及其后面的步骤，参考**导入JDK证书**的步骤，导入镜像源地址对应的JDK证书，

然后单击“GRADLE”视图的“**重新加载所有Gradle工程**”按钮，重新加载工程。

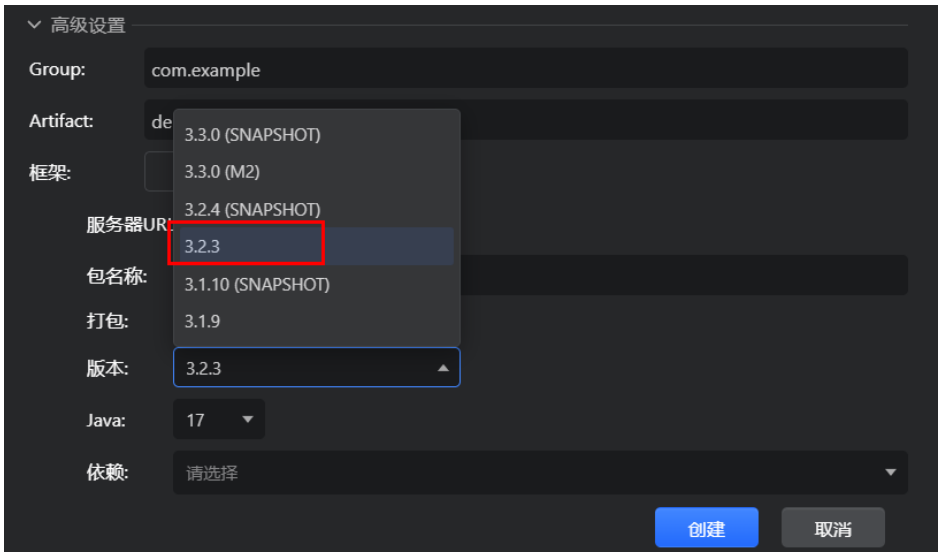


步骤4 如果未出现步骤2中检查连接成功的提示，请参考**代理设置**，设置相关代理后，再按照步骤2和步骤3检查连接，如果检查连接成功，则返回步骤3继续操作。

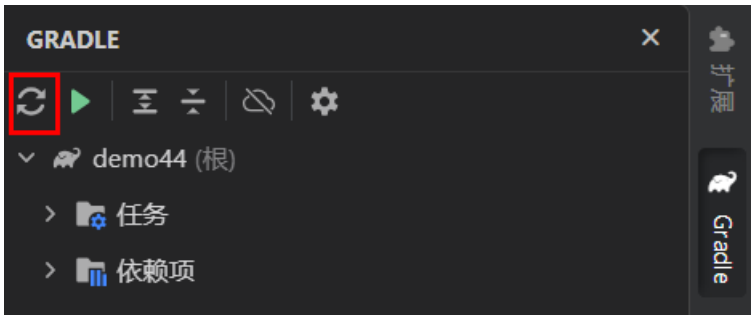
步骤5 如果配置相关代理后，检查连接依然失败，请修改工程中的build.gradle文件，将SpringBoot的版本号改为可用的发行版本。



可用的SpringBoot发行版本可以从新建SpringBoot工程界面找到，版本号不包含“**SNAPSHOT**”或“**M**”等字母：



然后单击“GRADLE”视图的**“重新加载所有Gradle工程”**按钮，重新加载工程。

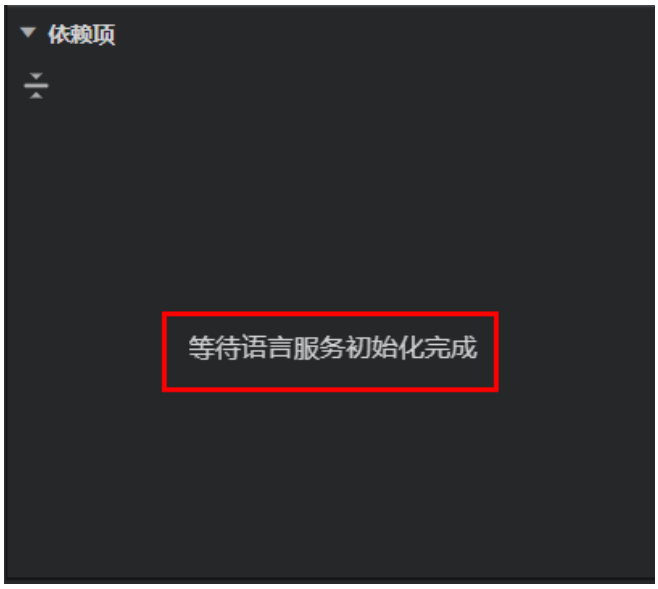


----结束

4.4 Maven/Gradle 视图一直显示“等待语言服务初始化完成”

问题现象

“依赖项”视图长时间出现“等待语言服务初始化完成”的提示信息，“MAVEN”或“GRADLE”视图也一直显示“**等待语言服务初始化完成**”。

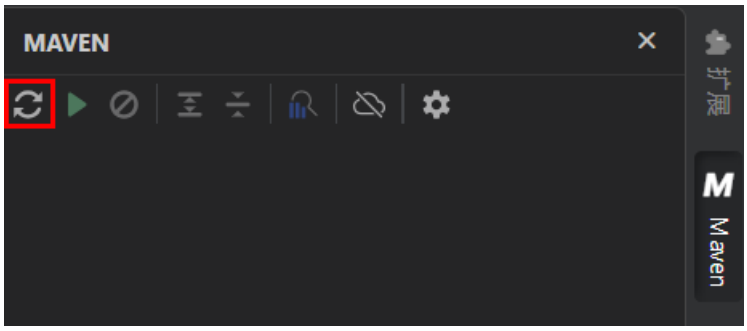


解决方法

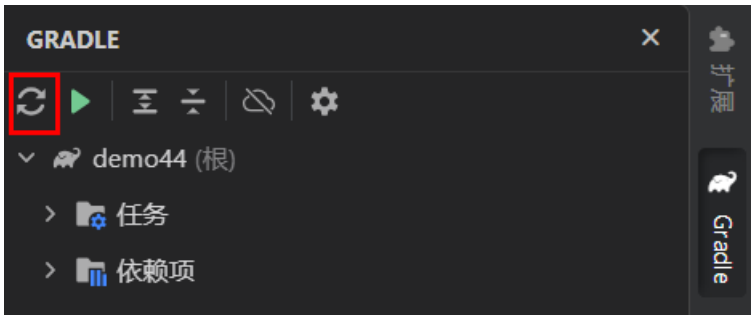
请检查系统环境是否是虚拟机环境，这种问题一般是虚拟机环境的系统缺失相关证书导致，如果是虚拟机环境，请按照如下步骤修复此问题。

步骤1 单击CodeArts IDE for Java左下角的“管理 > 设置”菜单，在设置窗口中搜索**“代理”**关键字，在搜索出的结果中，找到**“Http: Proxy Strict SSL”**设置项，**取消勾选**该设置项。

步骤2 如果是Maven工程，单击“MAVEN”视图中的**“重新加载所有Maven工程”**按钮重新加载即可。



如果是Gradle工程，单击“GRADLE”视图中的**“重新加载所有Gradle工程”**按钮重新加载即可。

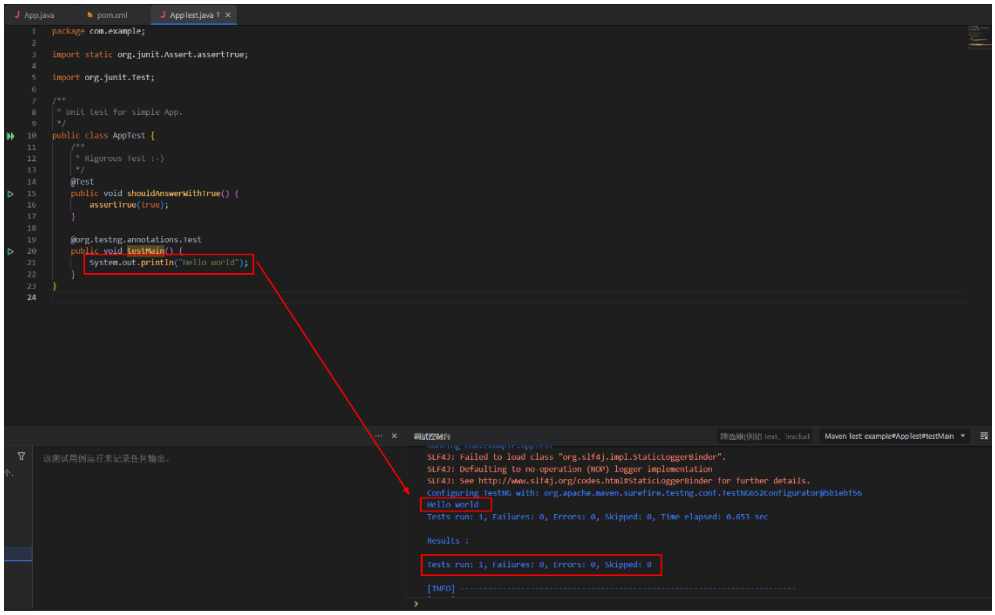


----结束

4.5 通过测试视图的运行/调试按钮，执行相关测试用例，测试视图状态无法更新

CodeArts IDE for Java版本中，部分工程在运行测试用例后，“测试”视图的状态无法更新。

如果出现上述场景，请优先检查输出结果是否在底部活动栏的“调试控制台”视图里面：



4.6 检查.ssh 密钥时出错

问题现象

CodeArts IDE在克隆存储库时，会弹出弹窗，校验密钥。当单击“是”或“否”之后，弹窗会再次出现。单击**ESC**键时，CodeArts IDE会给出错误提示：“检查主机密钥时出错”。

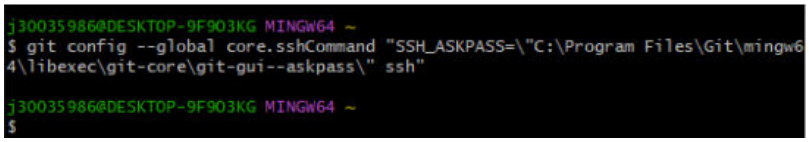
解决方法

步骤1 安装Git，安装步骤请参考**Git下载安装**。

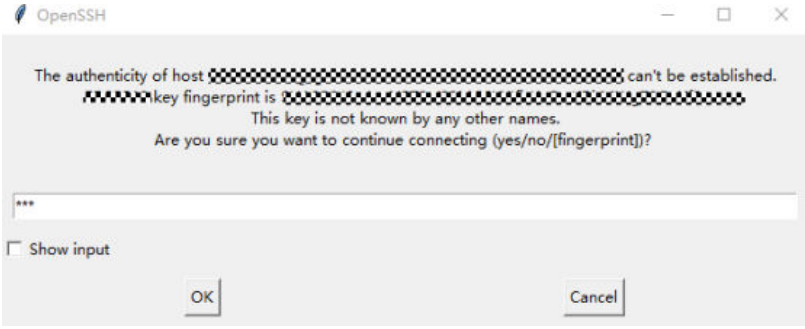
步骤2 右键单击“Open Git Bash here”，打开终端。

步骤3 输入并执行命令（按实际情况，替换Git的安装路径）。

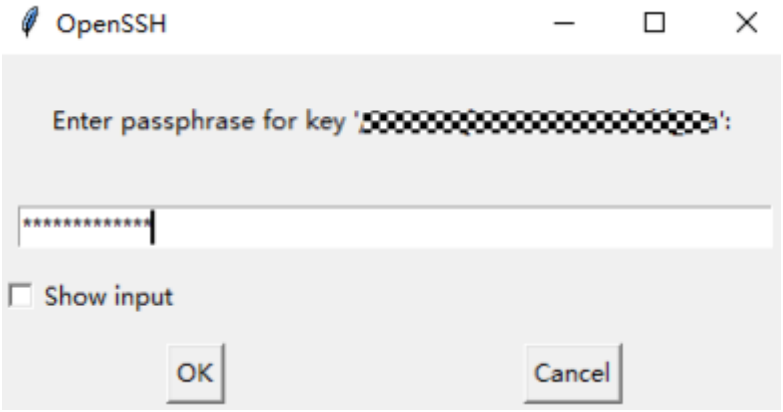
```
git config --global core.sshCommand "SSH_ASKPASS=\"C:\Program Files\Git\mingw64\libexec\git-core\git-gui--askpass\" ssh"
```



步骤4 在CodeArts IDE中单击“克隆”，输入ssh类型URL，会弹出以下弹窗。



在输入框中输入“yes”，单击“OK”，之后弹出下面弹窗，输入密钥，单击“OK”。



说明

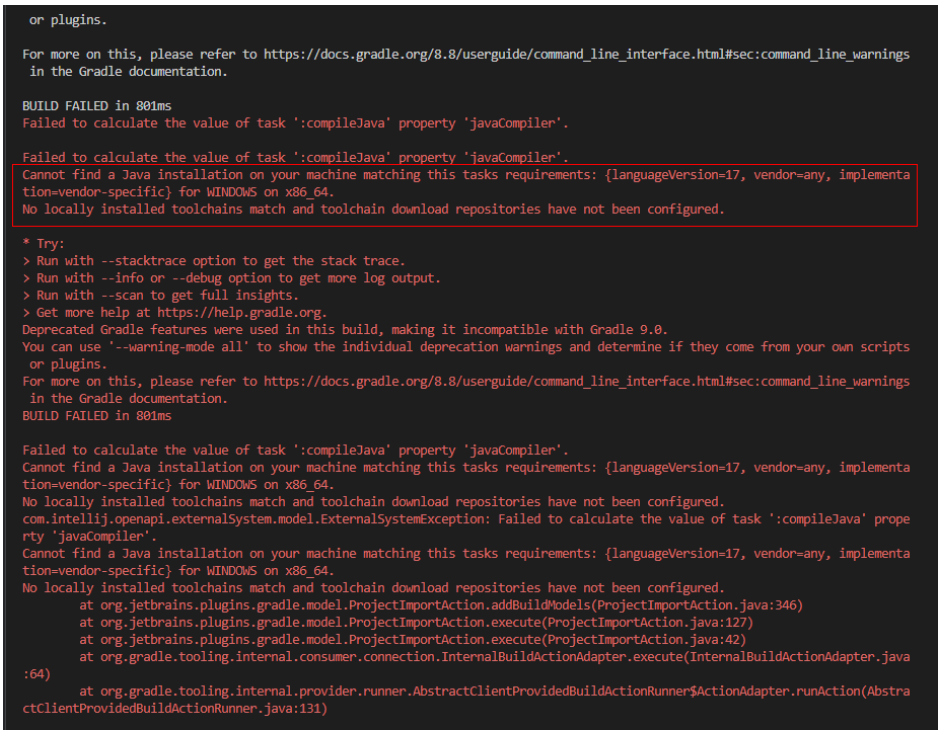
只有在第一次连接的时候，会弹出弹窗询问是否连接，输入“yes”之后，单击“OK”，会生成“C:\Users\用户名\.ssh\known_hosts”文件。删除该文件之后，会重新弹出该弹窗询问用户是否连接。

----结束

4.7 使用 JDK 20 创建 Gradle 的 SpringBoot 工程，工程解析失败或者 JDK 版本自动切换到 17 及 17 以下版本

由于Java语言服务暂不支持在Gradle的SpringBoot项目中使用JDK 20，故会出现如下两种异常场景：

1. 如果本地只安装了JDK 20，并且使用JDK 20创建Gradle的SpringBoot工程时，工程解析失败，“构建”视图出现下图“Cannot find a Java installation on your machine...”报错时，请尝试下载安装JDK 17来修复此问题。



2. 如果本地安装了JDK 20和JDK 17以及17以下版本，在使用JDK 20创建Gradle的SpringBoot项目时，JDK版本会自动切换到17或17以下的版本。

4.8 删除文件或文件夹时，无法删除到回收站

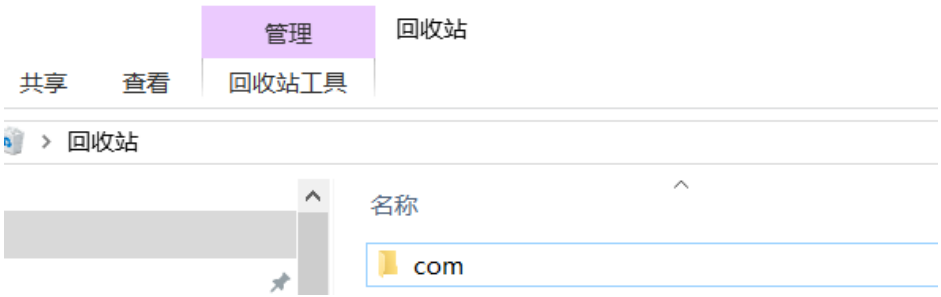
问题现象

在CodeArts IDE中删除文件或文件夹时，以删除com文件夹为例，出现如下弹窗，单击下图“移动到回收站”按钮。

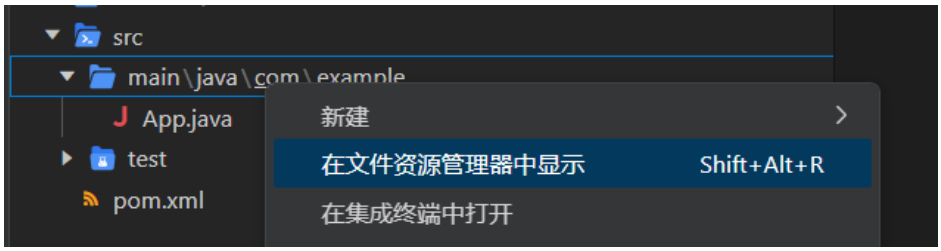
然后出现下图“无法删除到回收站。是否永久删除？”的弹窗：

解决方法

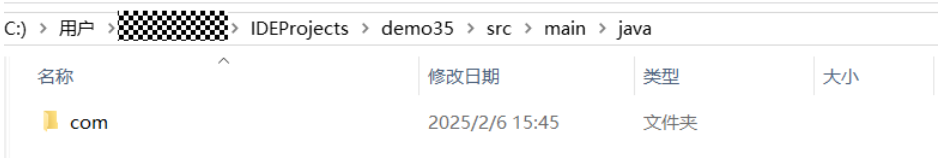
步骤1 查看电脑回收站是否存在要删除的目标文件夹com。如果存在，则单击上述“无法删除到回收站。是否永久删除？”弹窗中的“取消”按钮即可。如果回收站不存在要删除的目标文件夹com，则执行步骤2。



步骤2 鼠标左键单击目标文件夹com，鼠标右键唤出上下文菜单，单击下图“在文件资源管理器中显示”菜单。



会打开com文件夹所在的系统文件夹，如下图：



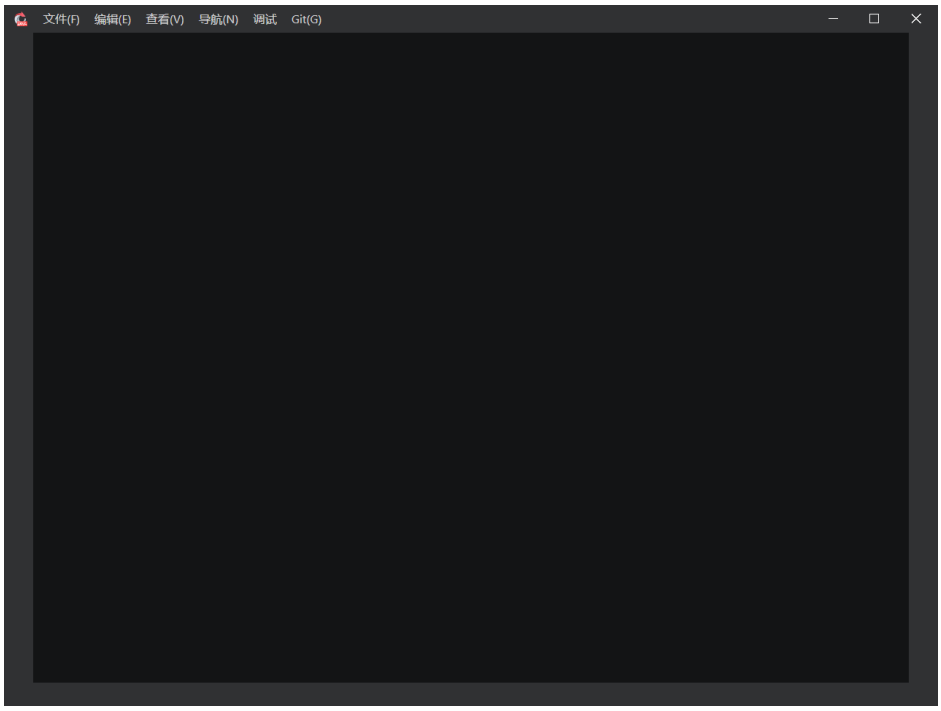
再选中上图系统文件夹com，通过鼠标右键的“删除”按钮尝试删除com文件夹，如果删除不成功，说明存在系统进程占用，在CodeArt IDE中同样无法删除此文件夹，出现“无法删除到回收站。是否永久删除？”的弹窗属于正常表现。

----结束

4.9 打开 CodeArts IDE 客户端，界面黑屏

问题现象

打开CodeArts IDE客户端后，出现界面黑屏且编辑器区域无有效内容，再次打开新的窗口仍出现同样的问题：



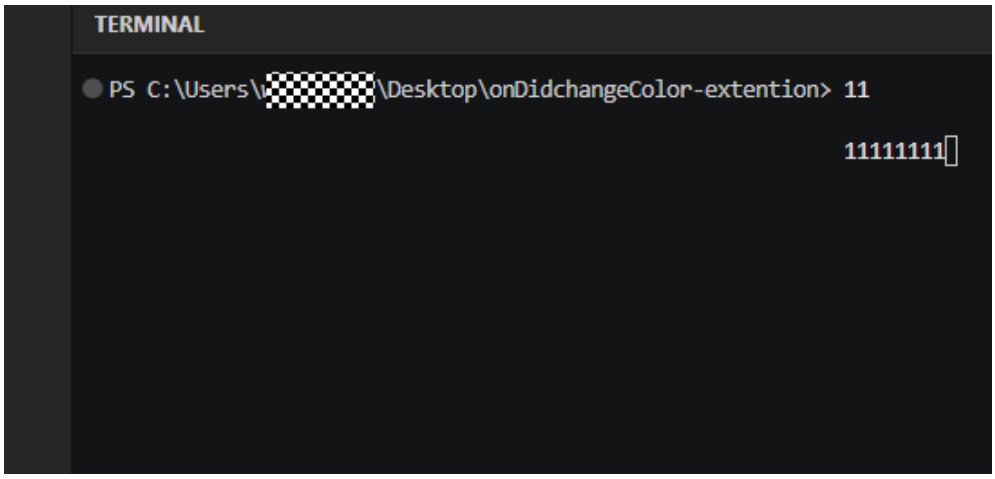
解决方法

关闭所有的CodeArts IDE窗口后重新启动CodeArts IDE，即可恢复正常。

4.10 terminal 清除终端后，终端输入内容出现换行

问题现象

默认的powershell终端，输入任意内容后回车，单击清除终端后再次输入内容时，出现跳跃换行。



解决方法

关闭当前终端新建终端或者按回车按钮进入新的命令行即可恢复。

4.11 CodeArts IDE 调试输入输出指导

问题现象

在使用默认配置调试C/C++代码时，调试控制台是不支持处理程序输入。以下面代码为例：

编译运行下面示例代码：

```
#include <stdio.h>

int main() {
    int num;
    printf("Please input: ");
    scanf("%d", &num);
    printf("your input is: %d\n", num);
    return 0;
}
```

launch.json配置为：

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "demo",
            "type": "gdb",
            "request": "launch",
            "gdbpath": "/path/to/gdb",
            "target": "/path/to/executable",
            "cwd": "${workspaceRoot}",
        }
    ]
}
```

运行结果，打印的输出结果为 13， 与输入的数字 15 不一致，如下图所示：

```
Running executable
=cmd-param-changed,param="script-extension",value="strict"
Undefined command: "export".  Try "help".
[New Thread 238544.0x3b758]
[New Thread 238544.0x3a258]
[New Thread 238544.0x3bd60]
please input:
15
your input is:13
[Thread 238544.0x3bd60 exited with code 0]
[Thread 238544.0x3b758 exited with code 0]
[Thread 238544.0x3a258 exited with code 0]
[Inferior 1 (process 238544) exited normally]
Canceled
```

解决方法

解决方法1：

在`launch.json`中添加参数`"terminal": "integrated"`，在终端调试程序。

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "demo",
      "type": "gdb",
      "request": "launch",
      "gdbpath": "/path/to/gdb",
      "target": "/path/to/executable",
      "cwd": "${workspaceRoot}",
      "terminal": "integrated"
    }
  ]
}
```

在终端可正确打印输入输出。

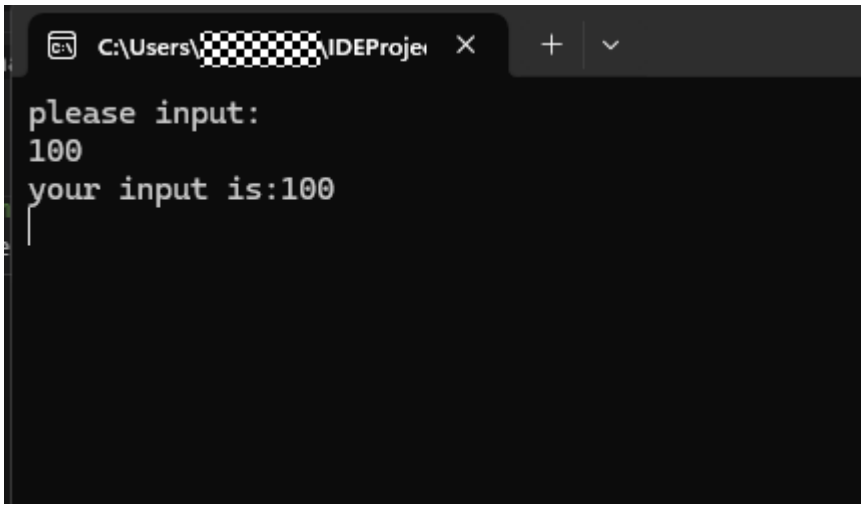
解决方法2：

在`launch.json`中添加参数`"terminal": ""`，会启动`cmd`窗口调试程序。

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "demo",
      "type": "gdb",
      "request": "launch",
      "gdbpath": "/path/to/gdb",
```

```
            "target": "/path/to/executable",
            "cwd": "${workspaceRoot}",
            "terminal": ""
        }
    ]
}
```

在`cmd`窗口可正确打印输入输出。

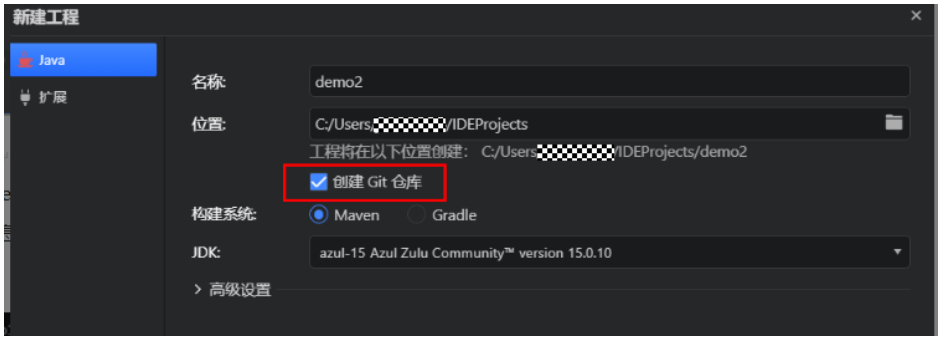


4.12 创建工程时勾选 git 仓库，添加远程仓库后推送代码时报错

问题现象

以创建一个Java工程为例，选中“创建Git仓库”。工程创建并且打开后，添加远程存储库。如下图所示：

图 4-1 创建 Java 工程



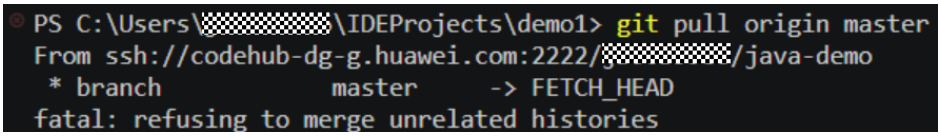
说明

默认远程仓库的名称是origin。本地分支是master，推送到远程的master的分支。

将本地修改的文件提交之后，再推送到远程。

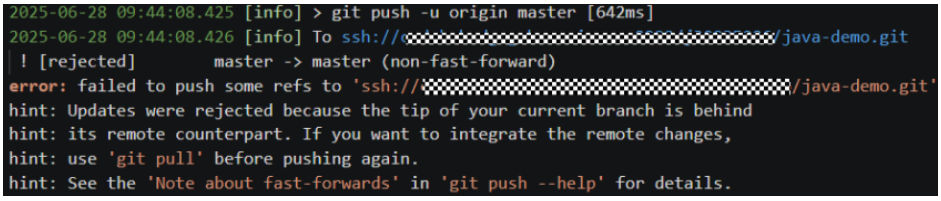
- 拉取远程仓库的代码，执行“git pull origin master”，报错信息如下图所示：

图 4-2 执行 git pull origin master 报错



- 将本地分支推送到远程，执行“git push -u origin master”，报错信息如下图所示：

图 4-3 执行 git push -u origin master 报错



解决方法

当操作过程中出现“fatal: refusing to merge unrelated histories”的错误，原因是两个仓库不同而导致的。需要在代码后面加上“--allow-unrelated-histories”进行允许合并，问题即可解决。

即执行“git pull origin master --allow-unrelated-histories”后， 同步远程分支代码，关联远程分支。

4.13 附录

4.13.1 登录 CodeArts IDE 运营面

步骤1 使用浏览器，以VDC管理员或VDC业务员**登录ManageOne运营面**。

步骤2 在页面左上角单击☰，打开服务列表。

步骤3 在服务列表页面右侧找到“软件开发生产线”，单击“软件开发生产线”下的“集成开发环境 CodeArts IDE”，进入CodeArts IDE运营面控制台。

如果登录用户第一次进入CodeArts IDE运营面控制台，页面将弹出提示框，单击“同意并继续”即可继续使用服务。

----结束

4.13.2 登录 ManageOne 运营面

步骤1 使用浏览器，通过地址“https://*ManageOne运营面的访问地址*”，登录ManageOne运营面，或通过地址“https://*ManageOne主门户的访问地址*”，登录ManageOne主门户，选择“云服务管理中心”，进入ManageOne运营面。

- 密码方式：输入账号和密码。
  - 管理面资源承载租户默认账号密码：请参见由HCC Turnkey导出的部署参数表《xxx_export_all_v2_CN.xlsx》>“基本参数”页签中的“secondlevel_vdcuser_for_deploy”、“secondlevel_vdcpassword_for_deploy”的值。

  - 管理员默认账号密码：请参见《华为云Stack 8.6.0 账户一览表》中“A类（Portal）”页签，产品名称为“ManageOne”，账户登录界面名称为“ManageOne运营面”获取。
- USB Key认证：插入已预置用户证书的USB Key，选择设备和用户证书，并输入PIN码。

  说明

  仅在SM系列商密算法场景下支持USB Key方式。

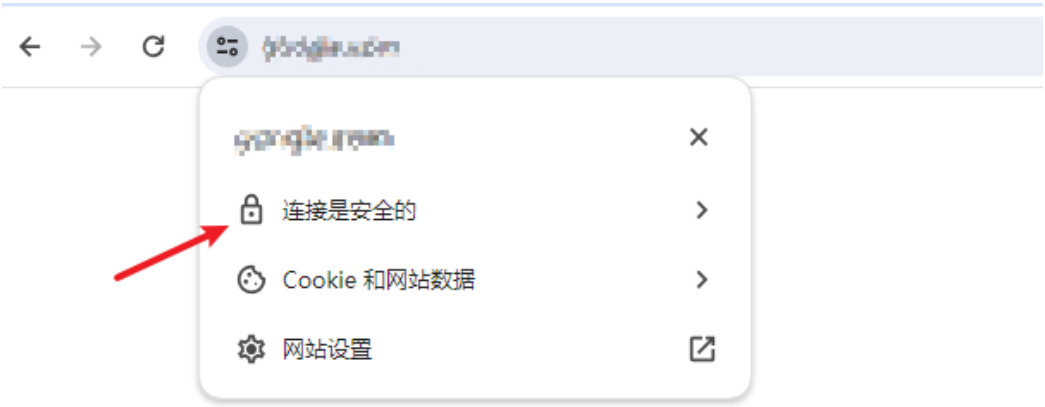
----结束

4.13.3 导入 JDK 证书

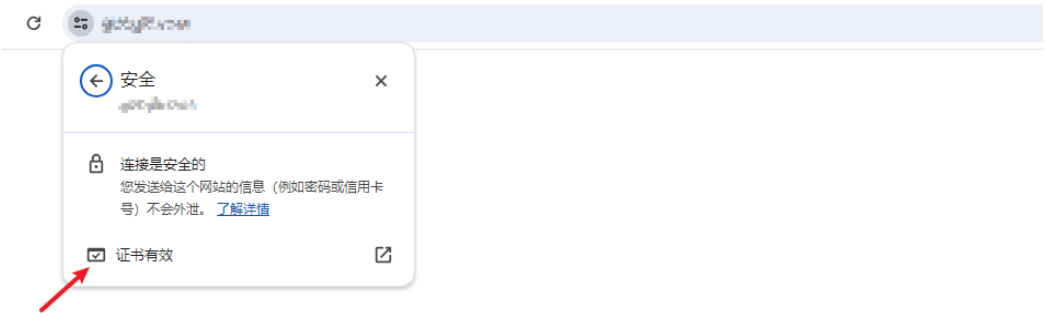
每当用户的应用程序尝试通过SSL（例如：HTTPS，IMAPS，LDAPS）连接到另一个应用程序时，它只能在可以信任该应用程序的情况下连接到该应用程序。在Java中处理信任的方式是用户拥有一个“信任库”文件（通常为$JAVA_HOME/lib/security/cacerts）。此信任存储文件包含受信任的证书，Java使用它来确定其他应用程序使用的SSL证书是否受信任。 Java只信任由证书颁发机构（CA）签名的证书，该机构的证书位于信任存储区中，或者是添加到信任存储区中的公共证书。用户可以参照如下步骤向JVM中导入证书：

步骤1 使用浏览器，访问用户所需要添加证书的网站，单击如下图标，出现对话框：

步骤2 单击如下选项：

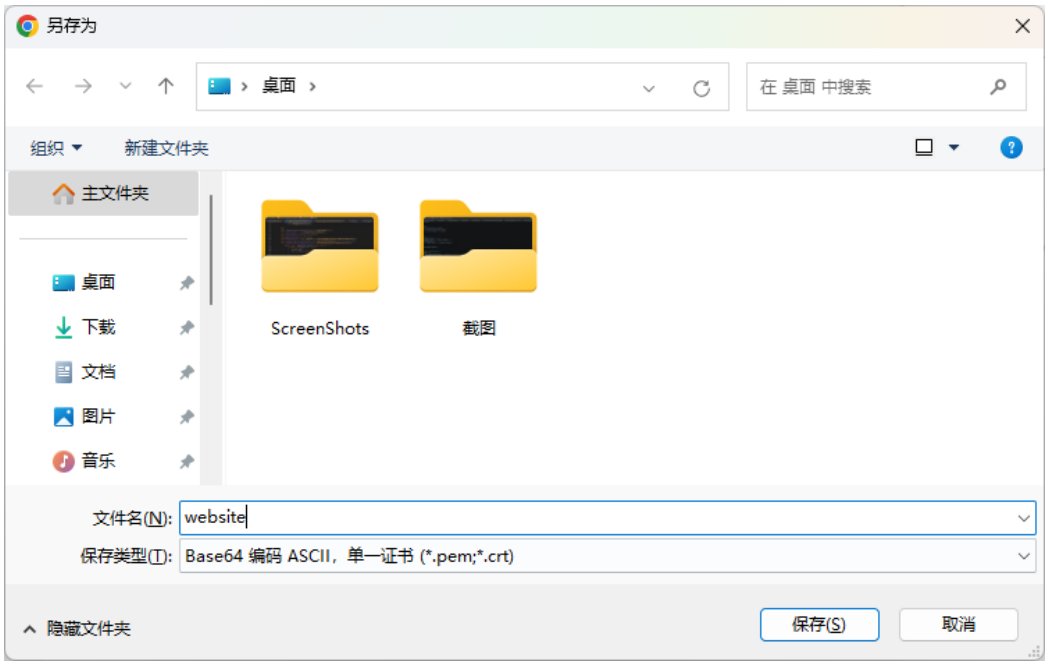


步骤3 单击证书图标：



步骤4 在对话框中，切换到“详细信息”页签，单击导出按钮：

步骤5 修改名称，单击保存，保存类型可以参考下图：



步骤6 打开终端，输入如下命令导入网站证书：


```
<JAVA_HOME>/bin/keytool -importcert -alias <server_name> -keystore <JAVA_HOME>/lib/security/cacerts -file website.crt
```

其中<server_name>为要添加的网站的别名。

website.crt为步骤5中导出的证书地址。

步骤7 重新运行工程，查看是否可以正常从对应网站下载依赖。

----结束

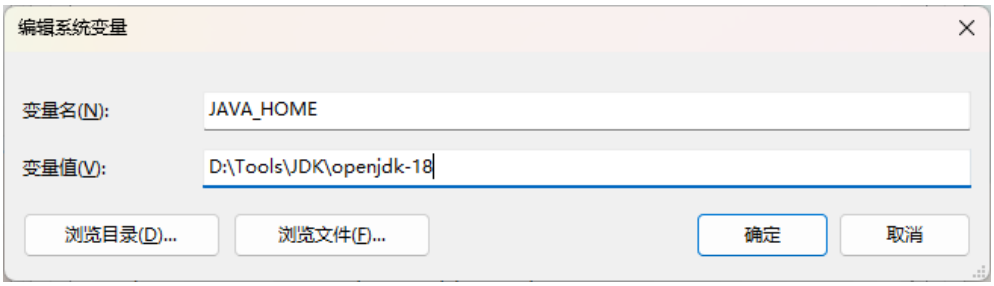
⚠ 注意

1. JAVA_HOME：JAVA_HOME变量的配置请参见**JDK的安装及环境变量配置**。
2. 不同浏览器在下载证书时的步骤会有一定的差异。
3. 如有更多问题，请联系CodeArts IDE工程师。

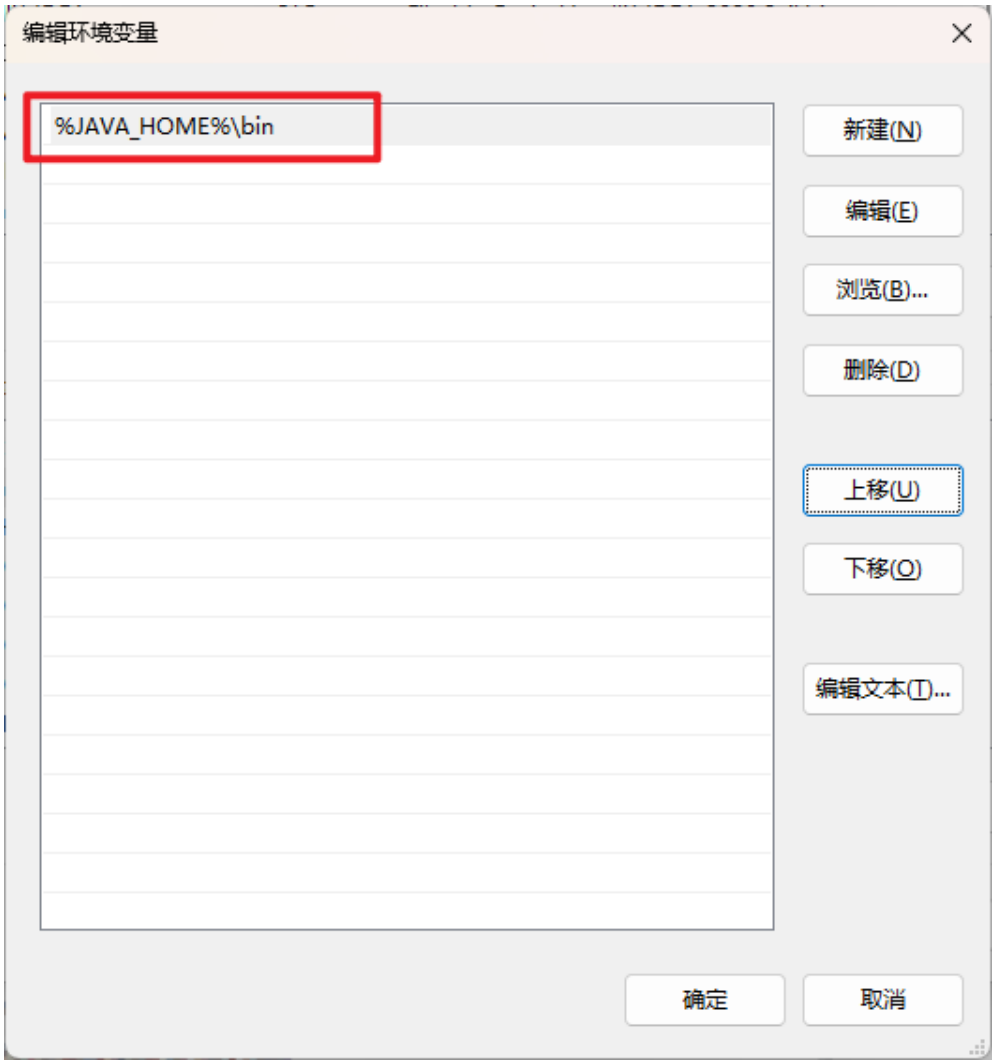
4.13.4 JDK 的安装及环境变量配置

1. 下载JDK，用户可以自行选择可访问的JDK下载链接并选择适合用户操作系统的JDK进行下载。
2. 根据用户下载的JDK类型，进行解压或安装。解压或安装成功后进入步骤3。
3. 如果用户使用的是Windows操作系统，用户可以按照如下步骤配置系统环境变量：

   （1）添加JAVA HOME变量，将步骤2中的JDK的安装/解压目录添加到该变量中，如：D:\Tools\JDK\openjdk-18。



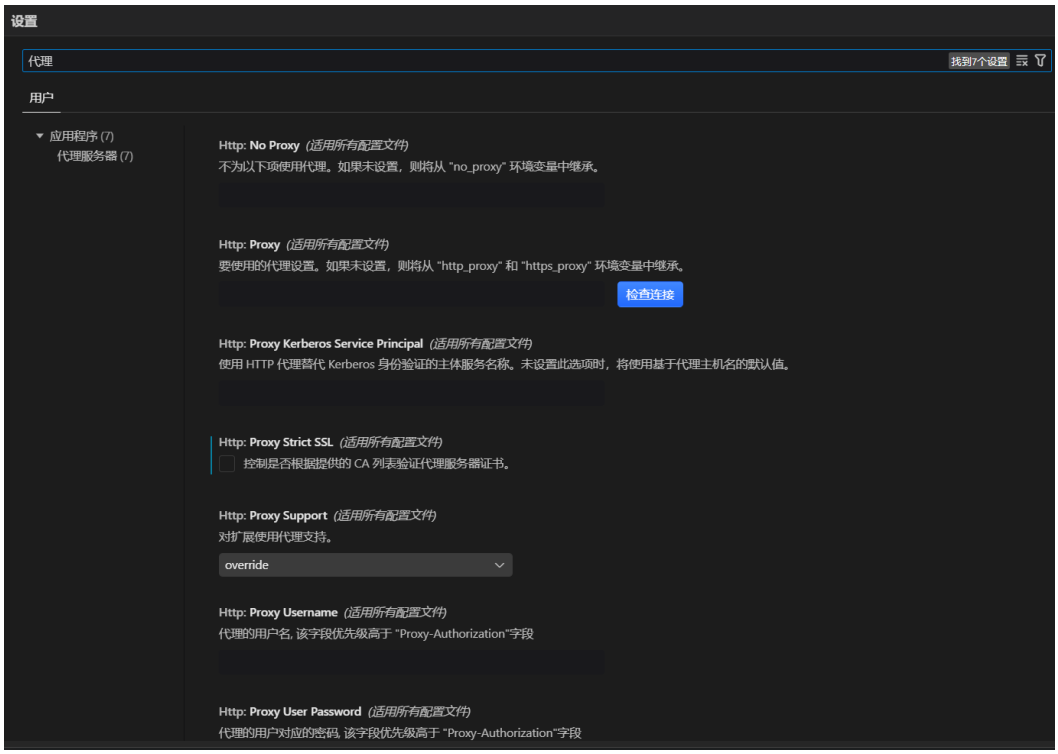
   （2）编辑Path环境变量，添加JDK的bin目录地址：%JAVA_HOME%\bin。



4. 如果用户使用的是Linux操作系统，用户可以通过如下方式设置环境变量：
   a. 在终端中输入如下命令打开Profile文件并编辑：
      ```
      sudo vi /etc/profile
      ```
   b. 在profile文件中添加如下内容：
      ```
      export JAVA_HOME=JDK安装/解压路径
      export PATH=$JAVA_HOME/bin:$PATH
      ```
   c. 修改完成后，先按“**ESC**”键，然后执行**:wq!**命令，保存该文件并退出vi编辑器。
   d. 在终端中输入如下命令让添加的环境变量生效：
      ```
      source /etc/profile
      ```

4.13.5 代理设置

代理设置可以允许用户在CodeArts IDE中进行网络代理的设置。在设置编辑器的搜索栏中，搜索字段**代理**，可以搜索到代理的相关配置：



CodeArts IDE提供了如下代理配置：

1. “Http: Proxy”：要使用的代理设置，包括地址和端口号。
2. “Http: No Proxy”：不使用代理的地址列表（域名或IP），多个地址用逗号分隔。
3. “Http: Proxy Strict SSL”：控制是否根据提供的CA列表验证代理服务器证书。
4. “Http: Proxy Support”：扩展使用代理的方式。
5. “Http: Proxy Username”：代理用户名。
6. “Http: Proxy User Password”：代理用户对应的密码。
7. “Http: Proxy Kerberos Service Principal”：使用HTTP代理替代Kerberos身份验证的主体服务名称。

须知

代理配置参考：

1、“Http: Proxy”：**http://proxy.******.com:8080**

2、“Http: No Proxy”：*******.com,***.com**

在部分网络场景下，用户可能还需要配置用户的代理**用户名（Proxy Username）**和**密码（Proxy Password）**以及取消勾选**“Proxy Strict SSL”**字段。

4.13.6 申请激活码

步骤1 参考**登录CodeArts IDE运营面**，在CodeArts IDE运营面首页左侧选择“CodeArts IDE > 激活管理”进入“激活管理”页面，再单击右上角的“申请激活码”。

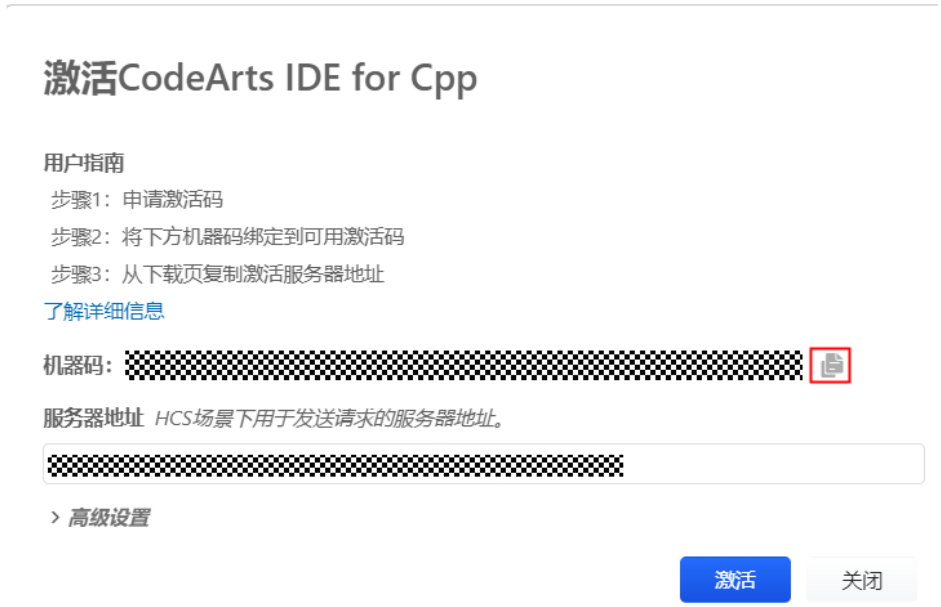
步骤2 下图所示，输入需要申请的激活码数量，再单击当前页面右下角的“立即申请”即可申请成功，并自动跳转到“激活管理”页面查看已申请的激活码。

----结束

4.13.7 绑定激活码并激活

步骤1 以激活CodeArts IDE for Cpp为例，打开已下载的CodeArts IDE客户端，会弹出提示并提供“机器码”，如图4-4。

图 4-4 复制机器码



步骤2 根据**申请激活码**，打开CodeArts IDE运营面首页的“激活管理”页面，选择上述**申请激活码**申请的激活码，如下图所示，单击右侧**“绑定”**，输入上一步获取的机器码，完成绑定。

图 4-5 绑定机器码

步骤3 如图4-6，从运营面的下载客户端页面复制“激活服务器地址”，在**步骤1**的CodeArts IDE客户端弹出的提示界面的“服务器地址”中输入，单击**“激活”**按钮，即可完成激活，如图4-8。

图 4-6 复制激活服务器地址

图 4-7 输入服务器地址并激活

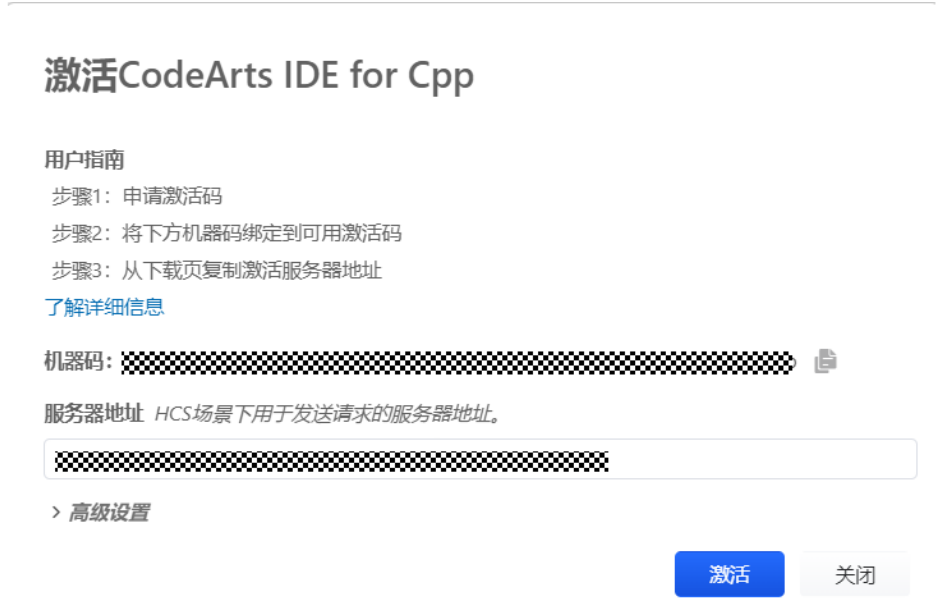
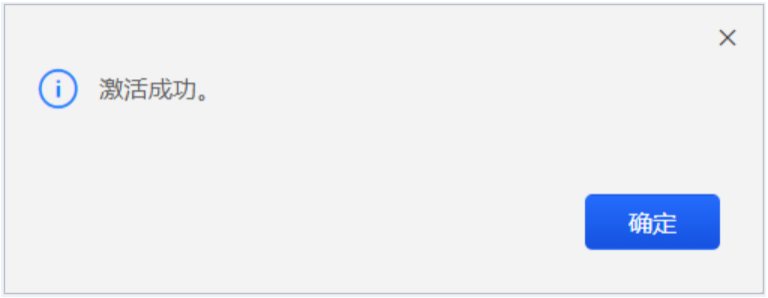


图 4-8 激活成功



----结束

须知

CodeArts IDE客户端激活成功后，会自动和许可激活服务器通信，客户端和许可服务器通信中断超过7天后，激活客户端将不可用。