# IDEA插件开发入门

## 插件类型

### 自定义语言支持

官方文档参考：<https://plugins.jetbrains.com/docs/intellij/custom-language-support-tutorial.html>

* 文件类型识别

* 词法分析

* 语法高亮显示

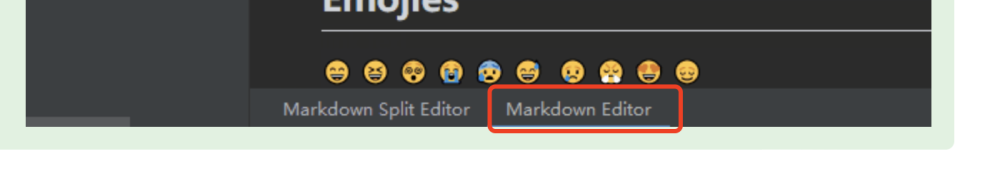
* 格式化

* 代码洞察和代码补齐

* 检查和快速修复

* 意图动作

插件还可以增强现有的（内置）自定义语言，例如通过提供额外的检查，意图及其它功能，例如：“Markdown Editor”插件。



### 框架集成

* 特定的代码洞察

* 直接访问特定于框架的功能

例如：“Struts2”插件

### 工具集成

### UI主题

主题使设计师能够自定义内置IDE UI元素的外观，自定义UI主题可以：

* 替换图标

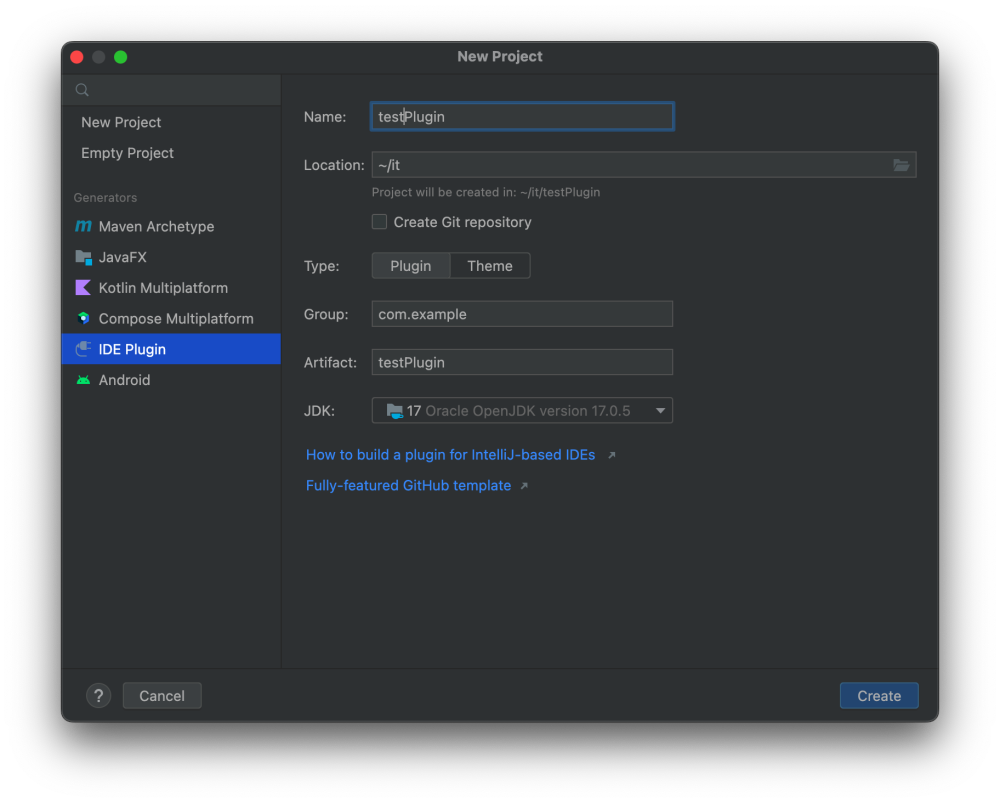
* 修改图标和UI控件的颜色

* 修改UI控件的边框和装饰物

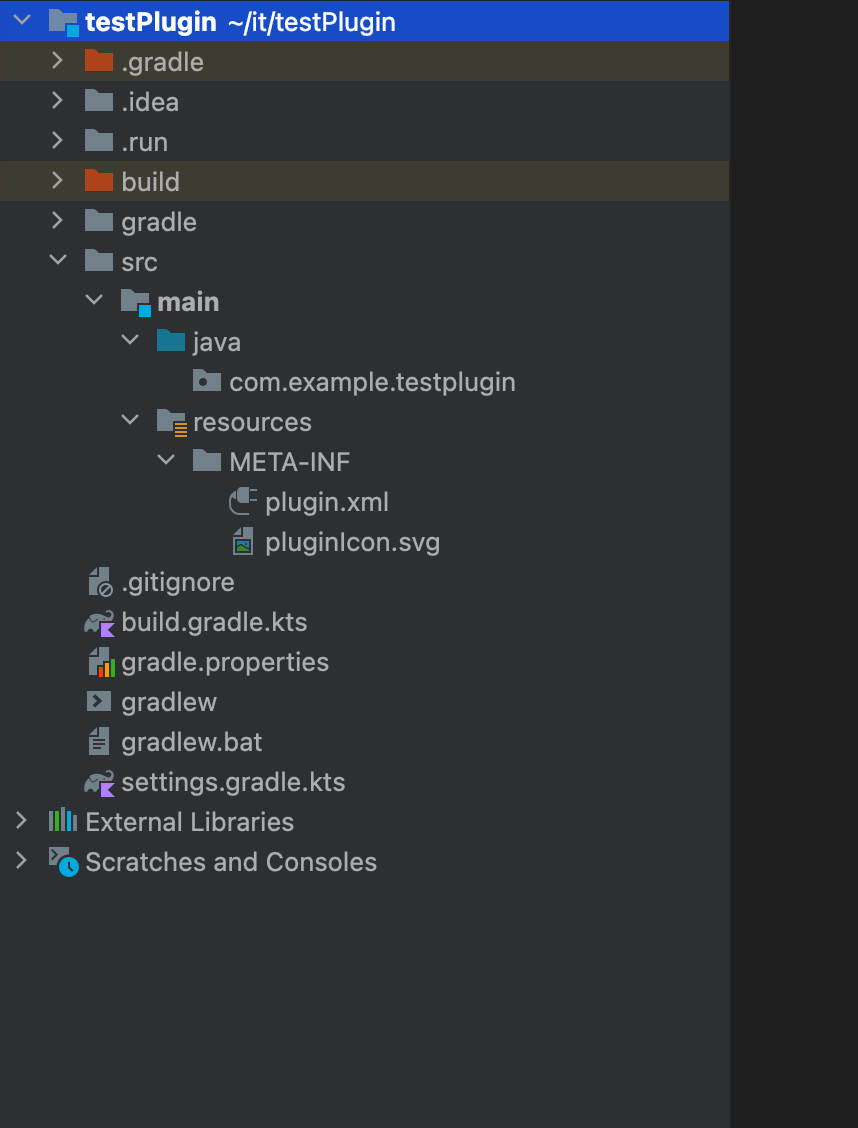
* 提供自定义编辑器主题

## 创建一个简单插件

### 1、用IDEA创建Plugin工程



#### 1.1 、Plugin项目结构



##### 1.1.1、plugin.xml

<idea-plugin>  
  
 <!-- 插件名字必须用英文且不能有标点, 可有空格和数字如 My Notes, 不能包含 plugin 字样 -->  
 <name>TestPlugin</name>  
  
 <!-- 插件唯一ID, 发不同版本时不能修改 -->  
 <id>com.example.testPlugin</id>  
  
 <!-- 插件的描述 -->  
 <description>test plugin description</description>  
  
 <!-- 插件版本变更信息，支持HTML标签；  
 将展示在 settings | Plugins 对话框和插件仓库的Web页面 -->  
 <change-notes>Initial release of the plugin.</change-notes>  
  
 <!-- 插件版本 -->  
 <version>1.0</version>  
  
 <!-- 插件页显示的开发者名称, 邮件和网址-->  
 <vendor url="http://www.jetbrains.com" email="support@jetbrains.com" />  
  
 <!-- 产品和插件依赖兼容 -->  
 <depends>com.intellij.modules.platform</depends>  
   
  
 <!-- 动作列表 -->  
 <actions>  
 </actions>  
  
 <!-- 暴露扩展列表 -->  
 <extensionPoints>  
 ...  
 </extensionPoints>  
  
 <!-- 扩展列表 -->  
 <extensions xmlns="com.intellij">  
 ...  
 </extensions>  
</idea-plugin>

##### 1.1.2、build.gradle

// 声明了3个Gradle插件  
plugins {  
 id("java")  
 id("org.jetbrains.kotlin.jvm") version "1.7.20"  
 id("org.jetbrains.intellij") version "1.13.1"  
}  
  
group = "com.example"  
version = "1.0-SNAPSHOT"  
  
repositories {  
 mavenCentral()  
}  
  
  
intellij {  
 version.set("2022.2.4") // 编译的 IDEA 版本  
 type.set("IC") // 支持的目标IDE平台，IC：社区版 IU：收费版  
  
 plugins.set(listOf(/\* Plugin Dependencies \*/)) // 依赖的插件列表  
}  
  
tasks {  
 // JVM 兼容性版本, 为 Java 11  
 withType<JavaCompile> {  
 sourceCompatibility = "11"  
 targetCompatibility = "11"  
 options.encoding = "UTF-8" // \*\* 追加项，避免汉字乱码   
 }  
   
 // 指定插件兼容的IDE构建的最小和最大版本  
 // 2022.2到2023.2  
 patchPluginXml {  
 sinceBuild.set("222")  
 untilBuild.set("232.\*")  
 }  
  
 // 插件签名（可忽略）  
 signPlugin {  
 certificateChain.set(System.getenv("CERTIFICATE\_CHAIN"))  
 privateKey.set(System.getenv("PRIVATE\_KEY"))  
 password.set(System.getenv("PRIVATE\_KEY\_PASSWORD"))  
 }  
  
 // 自动发布插件任务（可忽略）  
 publishPlugin {  
 token.set(System.getenv("PUBLISH\_TOKEN"))  
 }  
}

### 2、使用DevKit插件创建Action

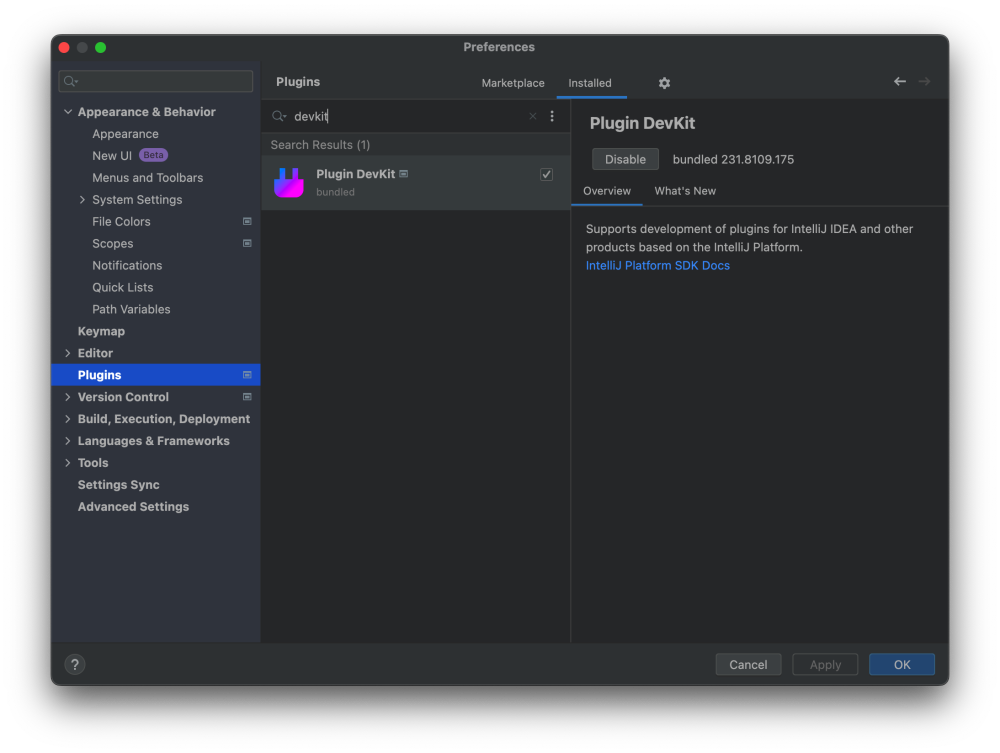
#### Action是什么

1. Action是用户调用插件功能的最常见方式，是从AnAction派生的类，当选择其菜单项或工具栏按钮时，会调用其actionPerformed()方法。

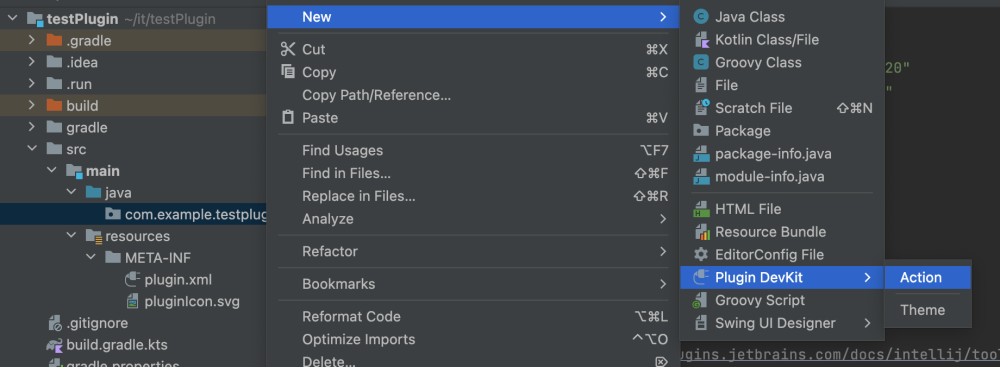
1. 可以使用键盘快捷键或Help | Find Action... 查找，从菜单或者工具栏调用。

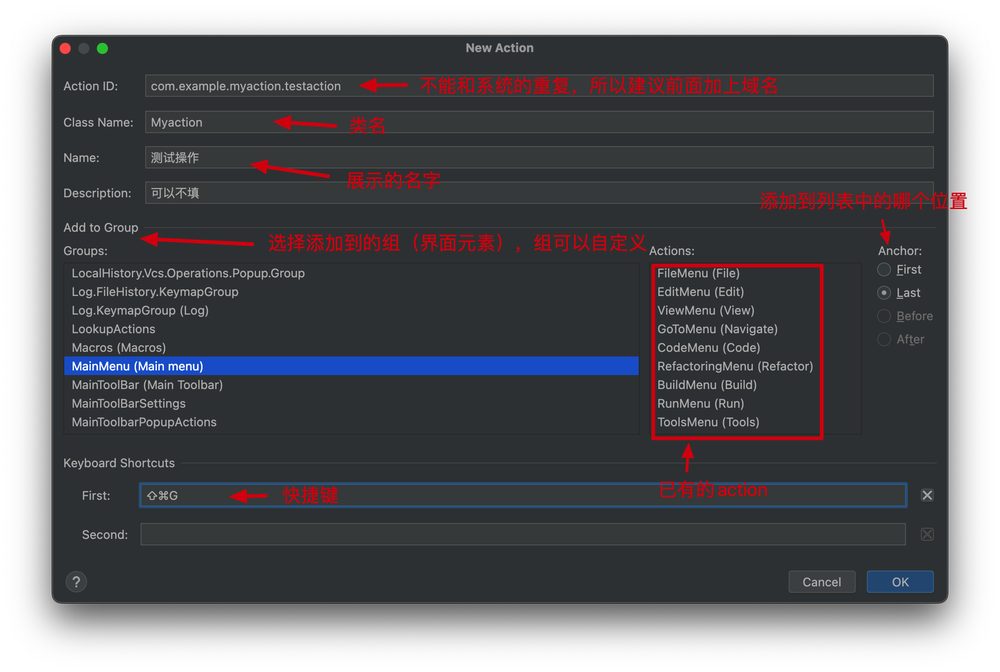
1. Action被组织成Group，而Group又可以包含其他Group。一组操作可以形成工具栏或菜单。该组的子组可以形成菜单的子菜单。

#### plugin开发插件



#### 创建Action





自动完成类的创建并注册到plugin.xml中

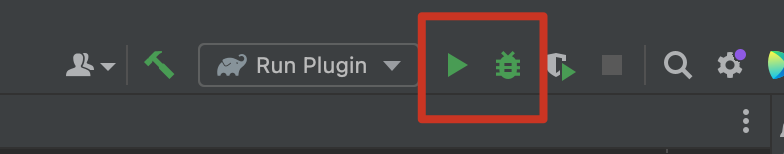


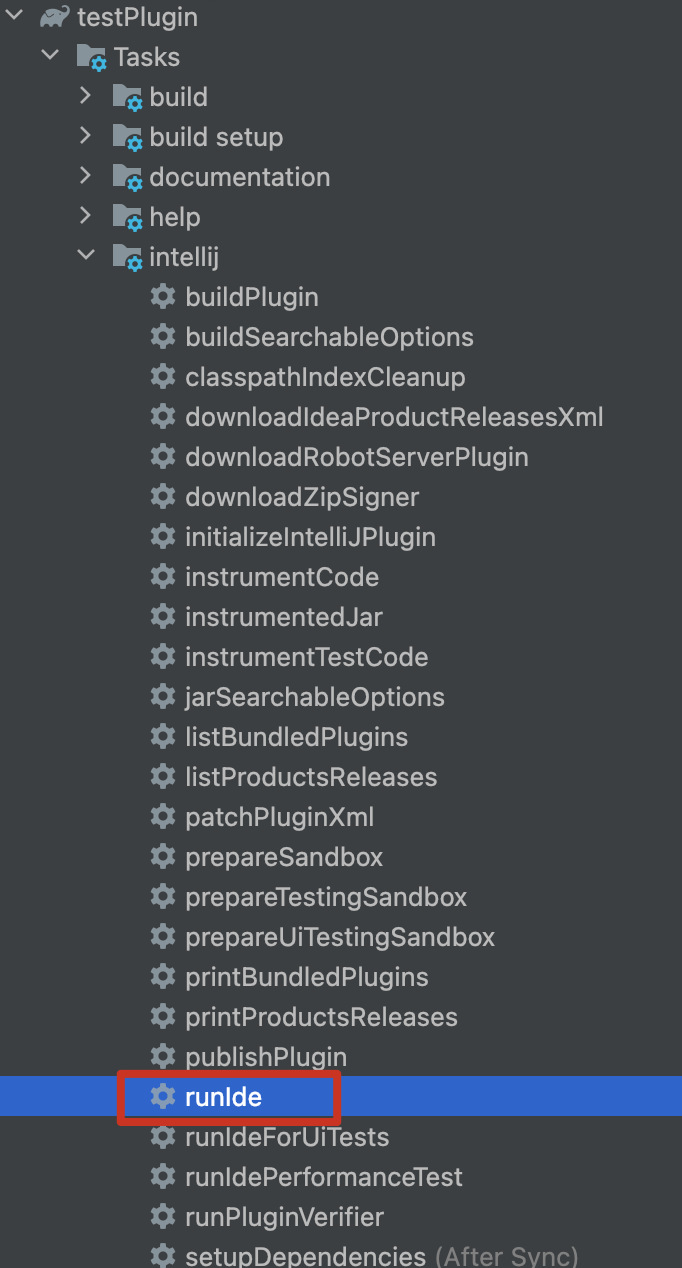
#### 编写信息提示代码

public class Myaction extends AnAction {  
 @Override  
 public void actionPerformed(AnActionEvent e) {  
 Project project = e.getData(PlatformCoreDataKeys.PROJECT);  
 Messages.showMessageDialog(project, "信息。。", "标题。。", Messages.getInformationIcon());  
 }  
}

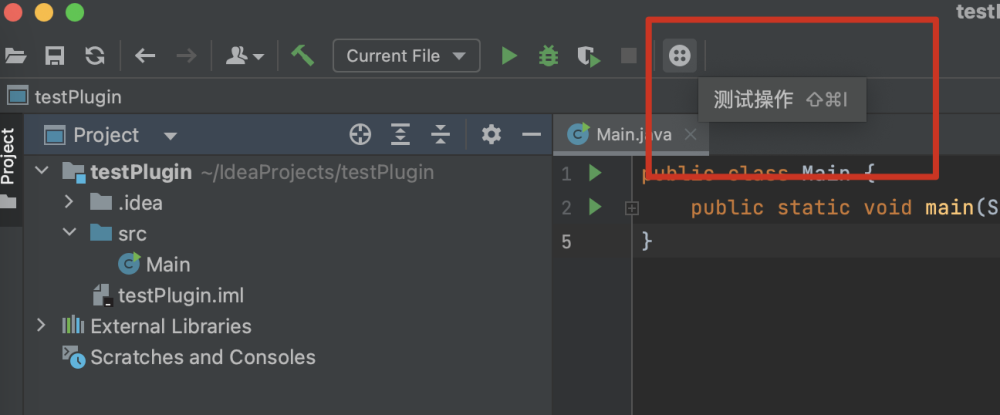
#### 执行插件

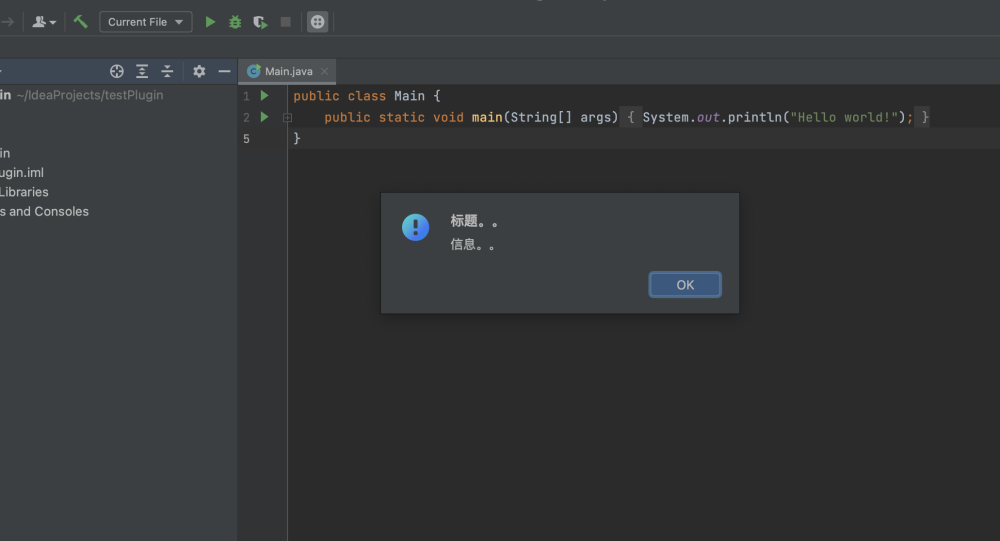
可以直接菜单栏的run和debug运行，也可以通过gradle下的runIde命令运行





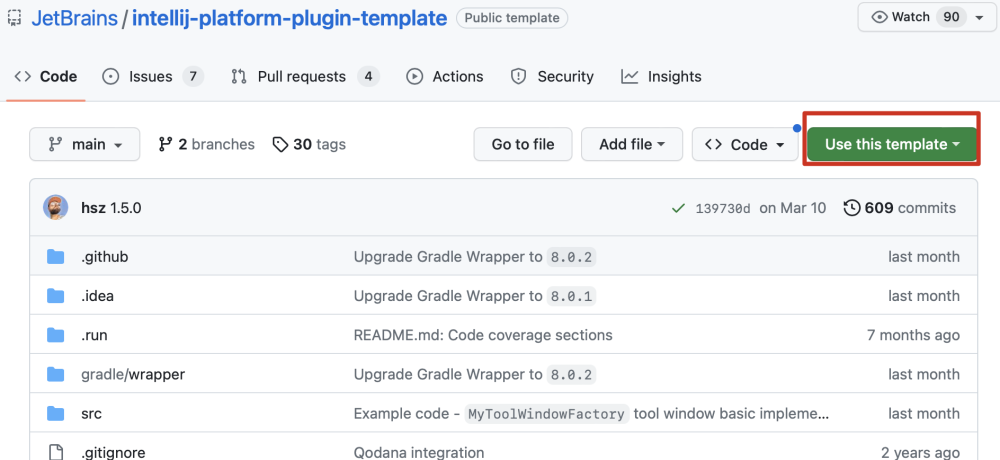
#### 执行结果



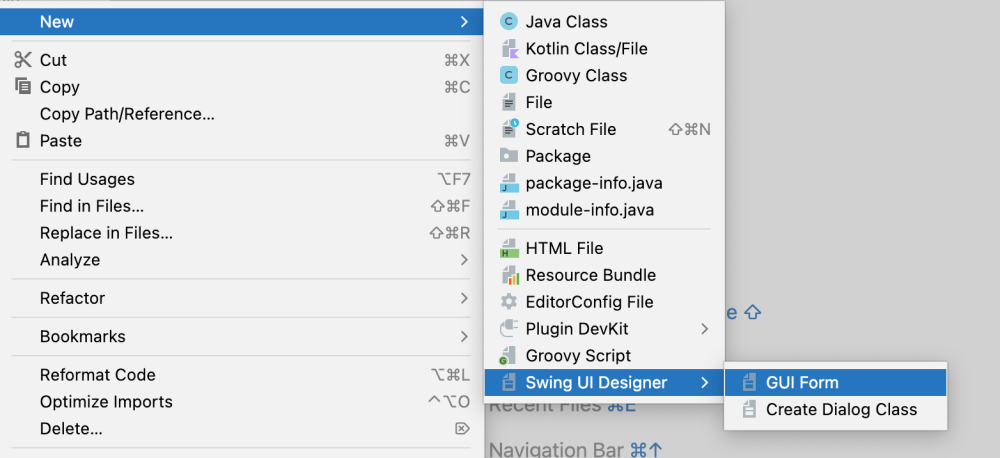


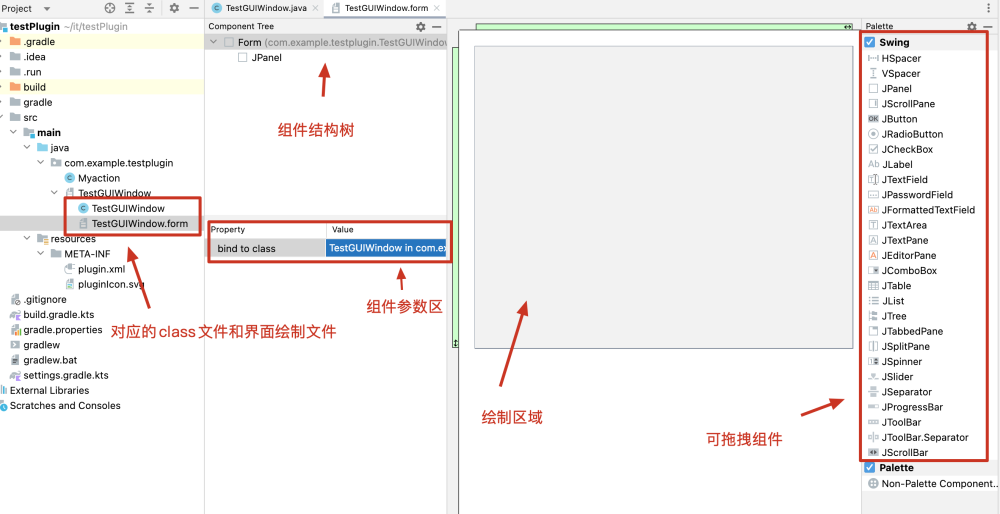
### 3、使用Github Template模版创建

<https://github.com/JetBrains/intellij-platform-plugin-template>



### 4、使用图形化界面工具拖拽画界面





#### 组件

|  |  |
| --- | --- |
| JPanel | 面板组件，所有组件可拖入面板中 |
| Hspacer、Vspacer | 控制组件间的水平间距、垂直间距 |
| JscrollPane | 滚动面板，可与JTextPane等组成可滚动的区域 |
| JScrollBar | 单向滚动条 |
| JButton、JRadioButton、JCheckBox | 按钮、单选按钮、多选按钮 |
| JLabel | 标签，显示固定文字 |
| JTextField、JPasswordField、JFormattedTextField | 单行文本编辑框 |
| JTextArea | 文本框 |
| JTextPane、JEditePane | 可以编辑和显示html，rtf和普通文本的富文本组件 |
| JComboBox | 可编辑下拉组件 |
| JTable | 表格 |
| JList | 列表 |
| JTree | 树结构，适合多级显示 |
| JTabbedPane | 选项卡面板 |
| JSplitPane | 分隔面板，用于两两分隔，多个分隔用它嵌套实现 |
| JSpinner | 单行输入框+上下选择器 |
| JSlider | 移动滑块，用来选值 |
| JSeparator | 分割线 |
| JProgressBar | 进度条 |
| JToolBar | 可以在程序的主窗口之外浮动或是托拽,里面可以添加各种组件 |

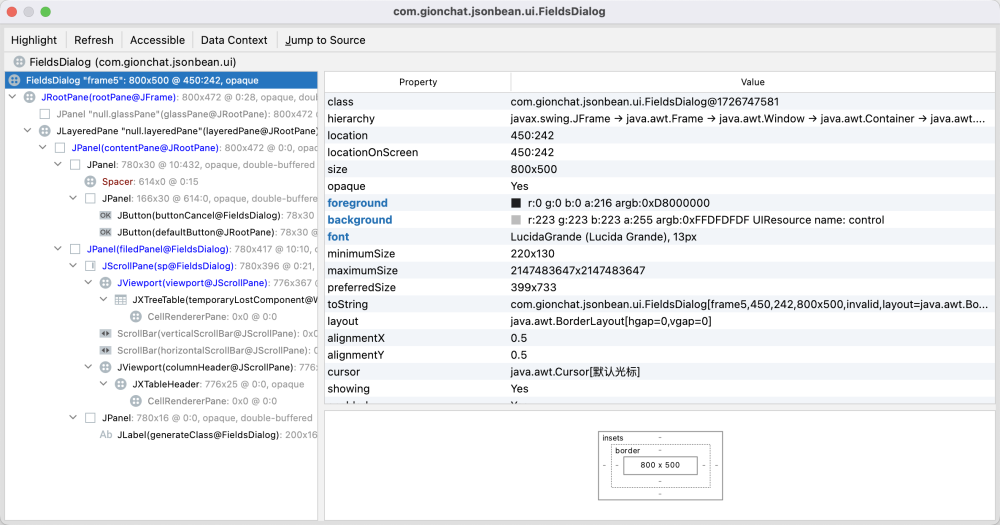
#### 监听器

|  |  |
| --- | --- |
| ActionListener | 接收动作事件 |
| ComponentListener | 接收组件事件 |
| ItemListener | 接收项目事件，在用户选择或取消选择项目时调用 |
| KeyListener | 接收按键事件 |
| MouseListener | 接收鼠标事件 |
| WindowListener | 接收窗口事件 |
| AdjustmentListener | 接收调整事件 |
| ContainerListener | 接收容器事件 |
| MouseMotionListener | 接收鼠标事件 |
| FocusListener | 接收焦点事件 |

##### UI查看器

Help -> Edit Custom Properties -> "idea.is.internal=true"

Tools -> Internal Actions -> UI -> UI Inspector



## 插件案例分析

学会了如何创建一个插件，但是还不知道如何开发一个实际可用的插件，所以接下来结合插件案例，通过案例来实践插件的开发。

官方代码案例：<https://github.com/JetBrains/intellij-sdk-code-samples>

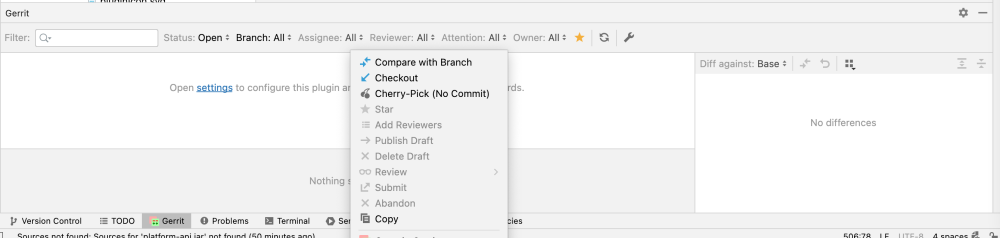
### 用户界面组件

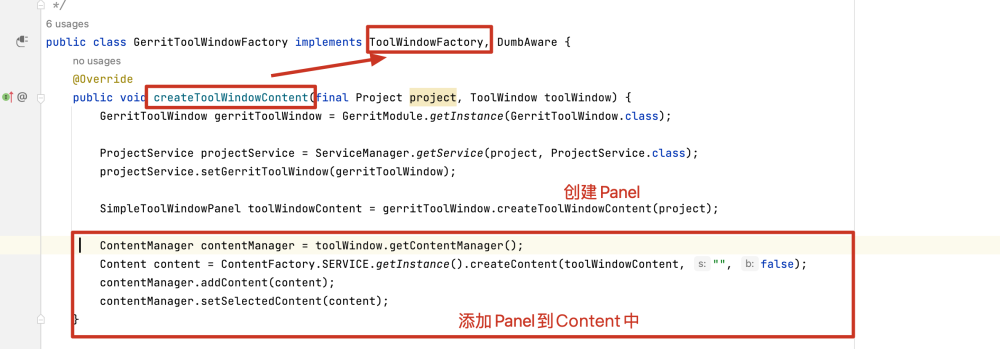
* Menus and toolbars are built using Actions

* Tool Windows

* + Gerrit插件

* + anchor="bottom"、right、left

* + 

* + 

* + 

* + 

* Dialogs

* Popups

* Notifications

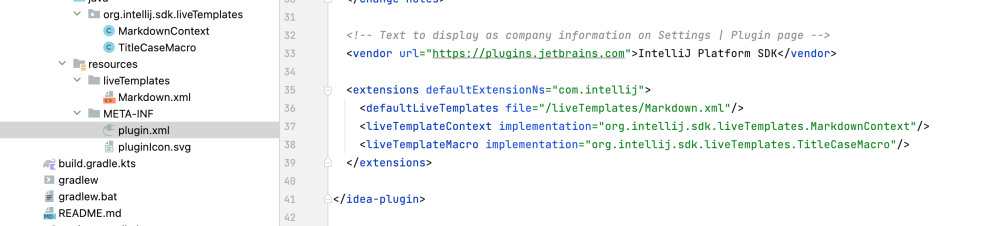
* File and Class Choosers

* Editor Components

* List and Tree Controls

* Status Bar Widgets

### 如何扩展一个插件



#### 如何找到想要的扩展点

1. 插件仓库插件扩展搜索：<https://plugins.jetbrains.com/intellij-platform-explorer/extensions>

1. 通过plugin.xml跳转到com.jetbrains的jar包，找到后缀为xxExtensions.xml的配置文件，例：

* 1. ExternalSystemExtensions.xml

* 1. LangExtensions.xml

* 1. PlatformExtensions.xml

* 1. VcsExtensions.xml

* 1. WorkspaceModelExtensions.xml

### 如何提供扩展能力

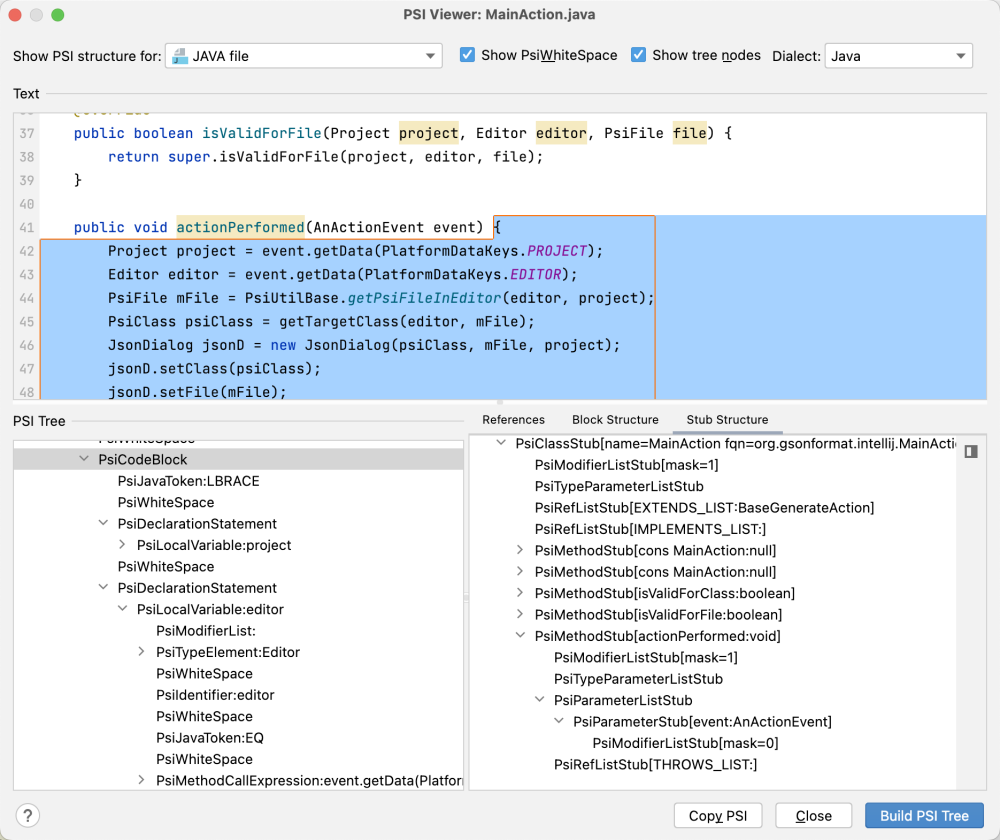
<!--  
 Declare extensions to access extension points in the IntelliJ Platform.  
 These extension points have been declared using "interface".  
 -->  
<extensions defaultExtensionNs="com.intellij">  
 <appStarter  
 implementation="com.example.MyAppStarter"/>  
 <projectTemplatesFactory  
 implementation="com.example.MyProjectTemplatesFactory"/>  
</extensions>  
  
<!--  
 Declare extensions to access extension points in a custom plugin "another.plugin".  
 The "myExtensionPoint" extension point has been declared using "beanClass"  
 and exposes custom properties "key" and "implementationClass".  
-->  
<extensions defaultExtensionNs="another.plugin">  
 <myExtensionPoint  
 key="keyValue"  
 implementationClass="com.example.MyExtensionPointImpl"/>  
</extensions>

### PSI能力

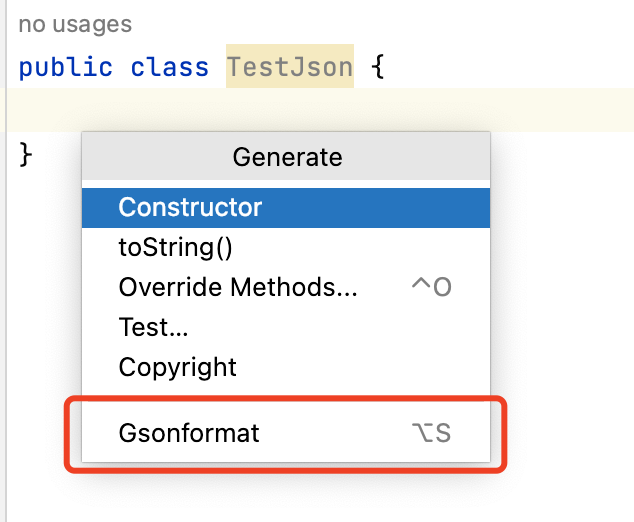
Program Structure Interface

程序结构接口，通常被称为PSI，是IntelliJ平台中负责解析文件和创建语法和语义代码模型的层，为平台的许多功能提供支持。

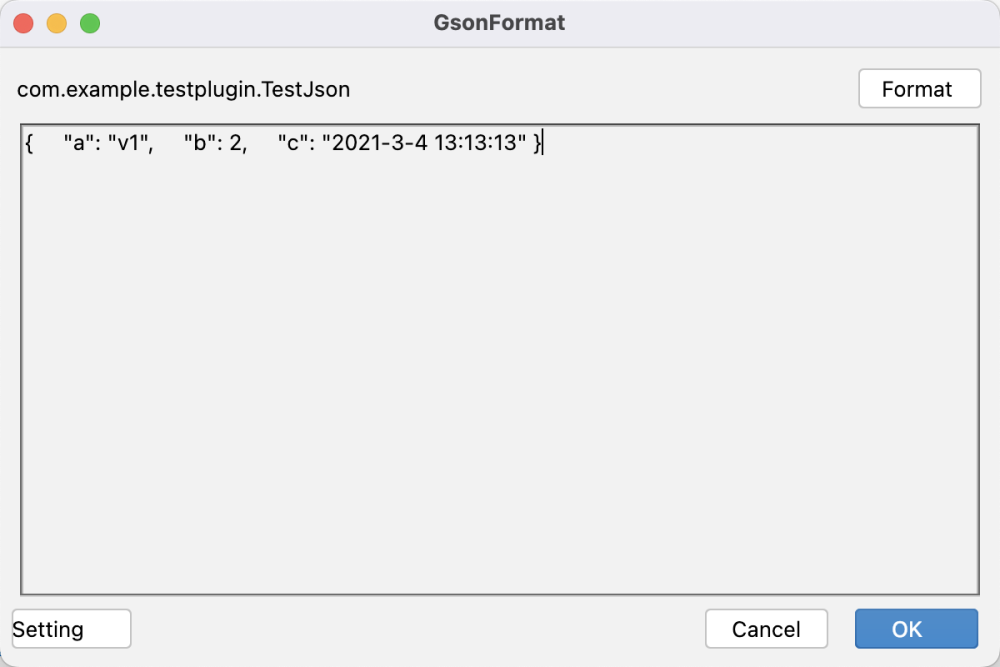
Tools -> View PSI Structure of Current File

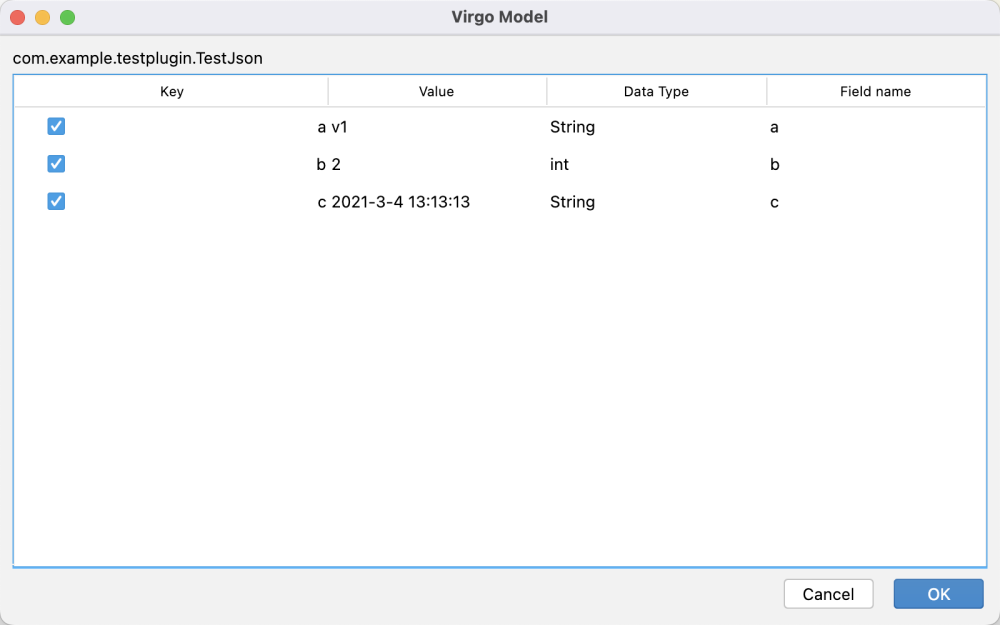


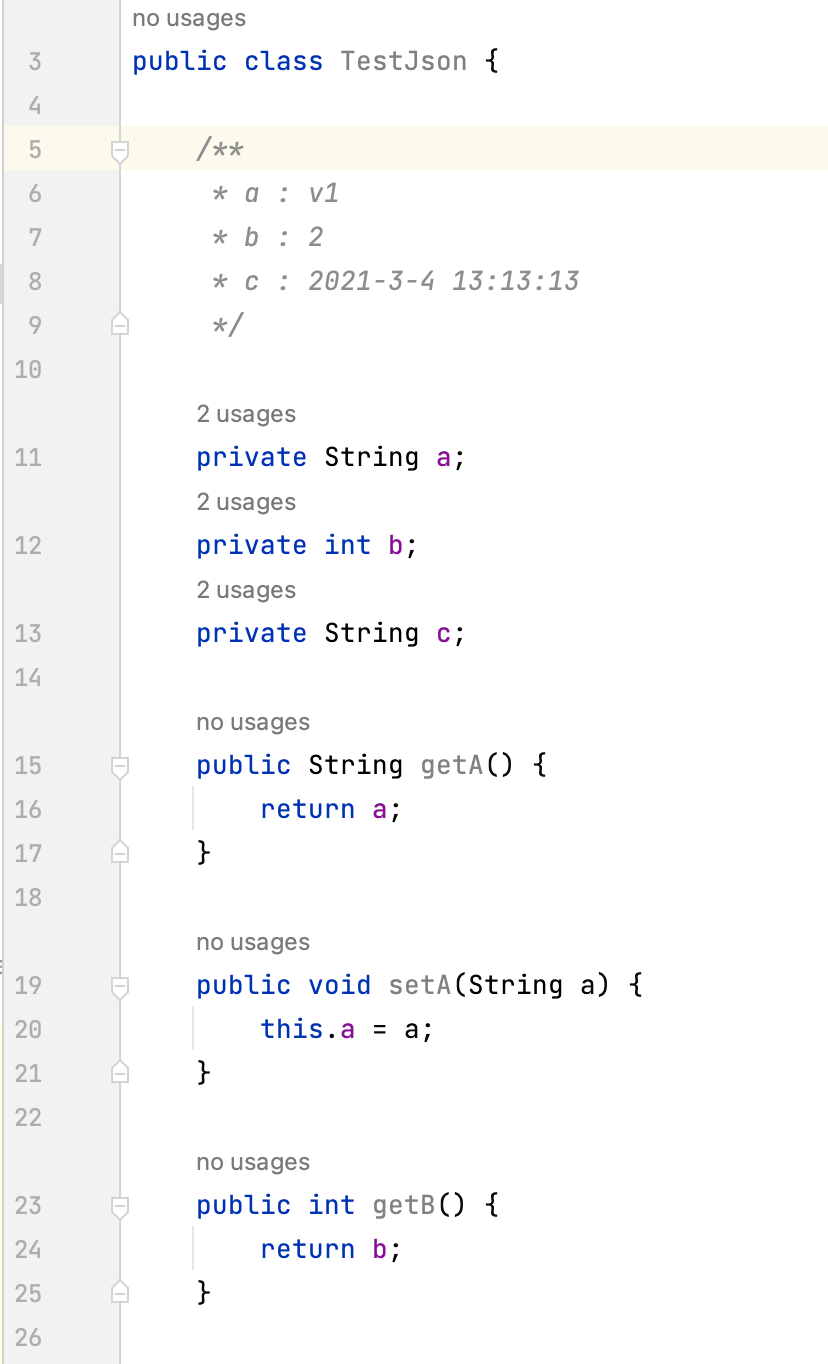
#### GsonFormat插件



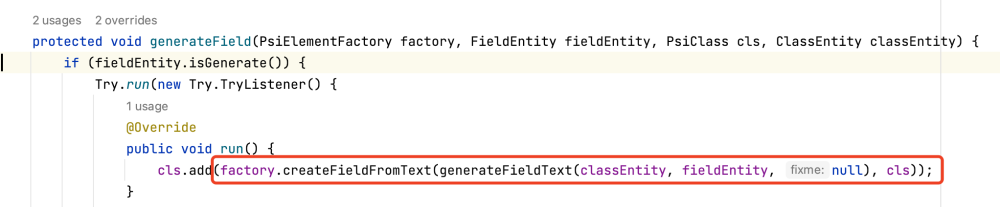


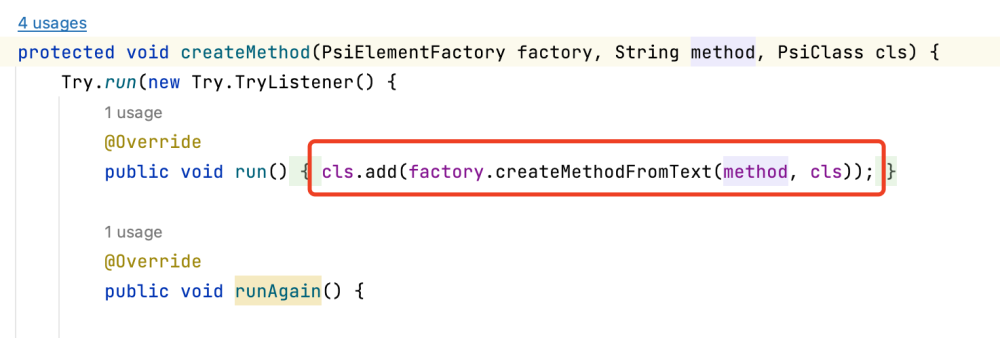


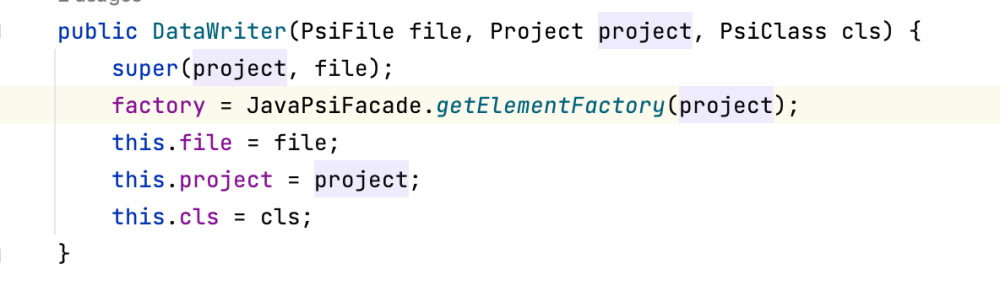




##### PSI生成代码

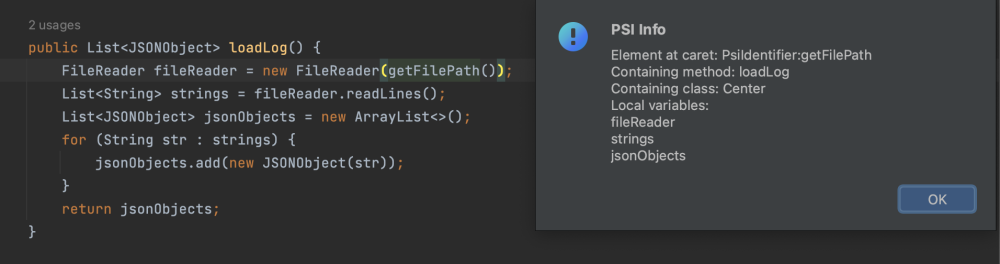


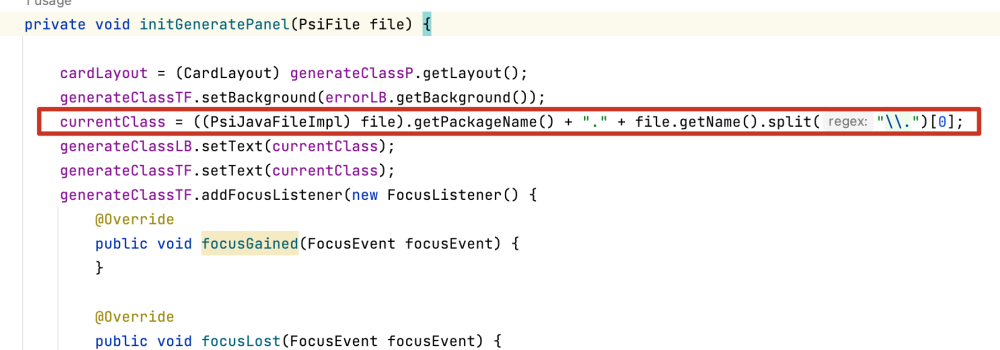




##### PSI获取文件信息







* PSI File：PSI File是将文件内容表示为特定编程语言中元素层次结构的结构的根。PsiFile类是所有PSI文件的公共基类，而特定语言中的文件通常由其子类表示。例如，PsiJavaFile类表示Java文件，XmlFile类表示XML文件。与具有应用程序范围的虚拟文件和文档不同（即使打开了多个项目，每个文件都由同一个VirtualFile实例表示），PSI具有项目范围：如果文件属于同时打开的多个项目的话，则同一文件由多个PsiFile实例表示。

* File View Providers：文件视图提供程序管理对单个文件中多个PSI树的访问。例如，一个JSPX页面有一个单独的PSI树，用于其中的Java代码（PsiJavaFile），一个单独树，用于XML代码（XmlFile），以及一个单独树形，用于整个JSP（JspFile）。每个PSI树都覆盖了文件的全部内容，并在可以找到不同语言内容的地方包含特殊的“外部语言元素”。FileViewProvider实例对应于单个VirtualFile、单个Document，并且可以检索多个PsiFile实例。

* PSI Elements：PSI文件表示PSI元素的层次结构（所谓的PSI树）。单个PSI文件（本身是PSI元素）可以以特定的编程语言公开多个PSI树。反过来，PSI元素可以具有子PSI元素。PSI元素和单个PSI元素级别的操作用于探索IntelliJ平台解释的源代码的内部结构。例如，可以使用PSI元素来执行代码分析，例如代码检查或意图操作。PsiElement类是PSI元素的公共基类。