



河南中医药大学信息技术学院（智能医疗行业学院）智能医学工程专业《计算机程序设计》课程

第15章：使用AI辅助编程

黄子杰

河南中医药大学信息技术学院（智能医疗行业学院）与河南方和信息科技有限公司 联合建设
河南中医药大学信息技术学院智能医疗教研室

<https://aitcm.hactcm.edu.cn>

2025/2/16

本章概要

概述：

我们将使用Fitten Code辅助编程,让编程变得更简单



15.1 Fitten Code

- 作为现代编程人员，我们总是追求更快、更高效的工作方式。使用AI编程助手如同拥有一个可靠、智能的合作伙伴，它能够与你紧密合作，提供实时的建议和解决方案。无论是快速修复错误、提升代码质量，或者查找关键文档和资源，AI编程助手都能让你事半功倍。让我们携手AI编程助手，释放创造力，加速项目进程，共同迈向编程的新高度！
- Fitten Code是由非十大模型驱动的AI编程助手，它可以自动生成代码，提升开发效率，帮我们调试Bug，节省我们的时间。还可以对话聊天，解决我们编程碰到的问题。免费且支持80多种语言：Python、C++、Javascript、Typescript、Java等，目前支持以下4种编辑器与开发环境：

VS Code

JetBrains IDE 系列

Visual Studio

Vim





15.1 Fitten Code

代码自动补全

- Fitten Code 能够自动为开发者的代码补充缺失的部分，节我们宝贵的开发时间。



15.1 Fitten Code

注释生成代码

- Fitten Code 基于AI大模型对代码进行语义级翻译，支持多种编程语言互译。



15.1 Fitten Code

自动添加注释

- Fitten Code 能够根据代码自动生成相关注释，为我们的代码提供清晰易懂的解释和文档。



15.1 Fitten Code

更多功能

- 智能bug查找，解释代码，自动生成单元测试的功能，根据代码自动产生相应的测试用例等。

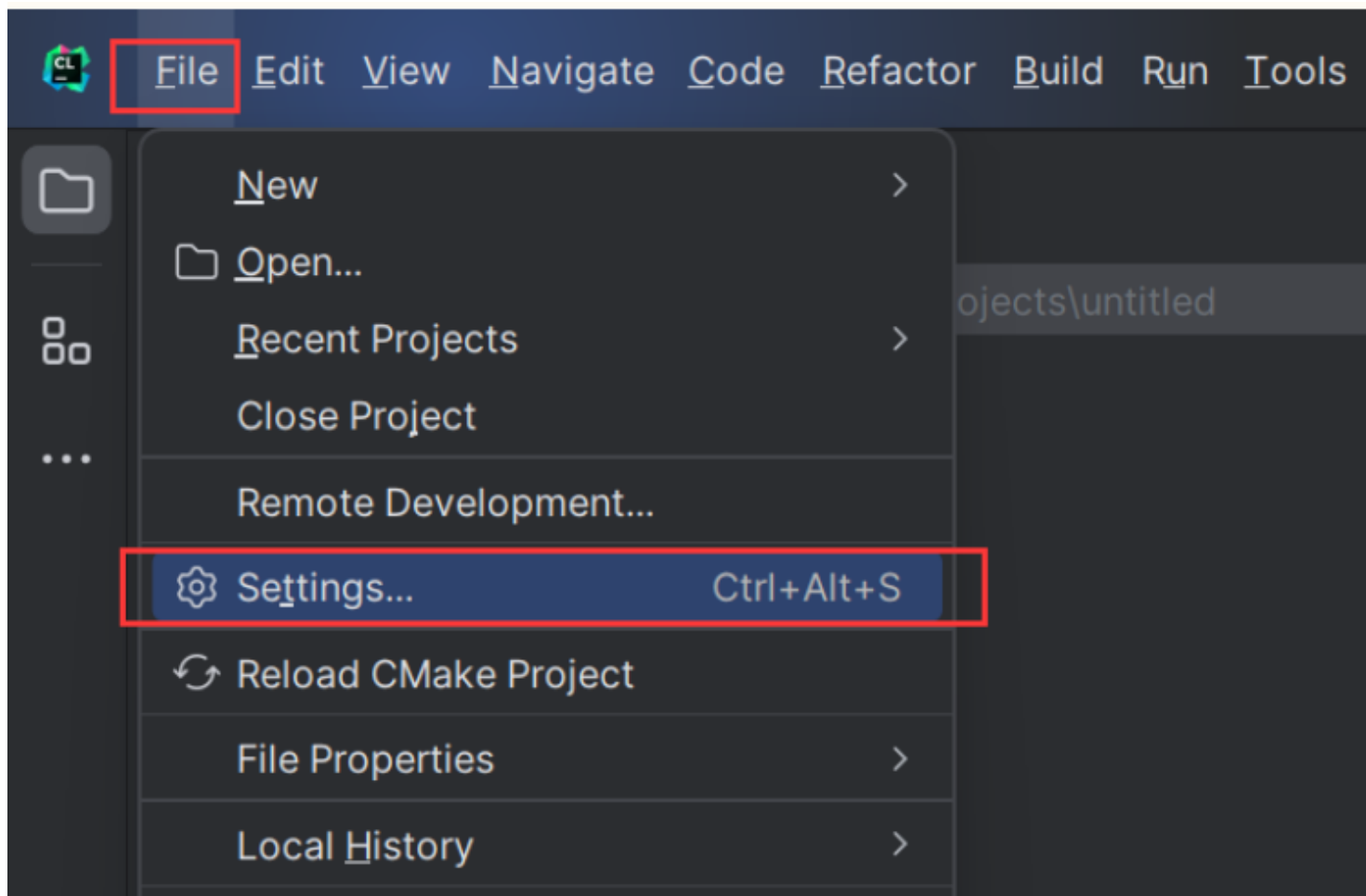
15.2 安装

- 如果我们已经安装JetBrains系列产品且版本大于等于2021.1，请直接跳过此步骤。否则请前往官网下载安装最新JetBrains系列产品。如果我们的版本低于2021.1，插件市场将无法搜索到Fitten Code插件，请升级到最新版本。



15.2 安装

- 1 打开IDE，点击左上方“File”接着点击“Settings”



15.2 安装

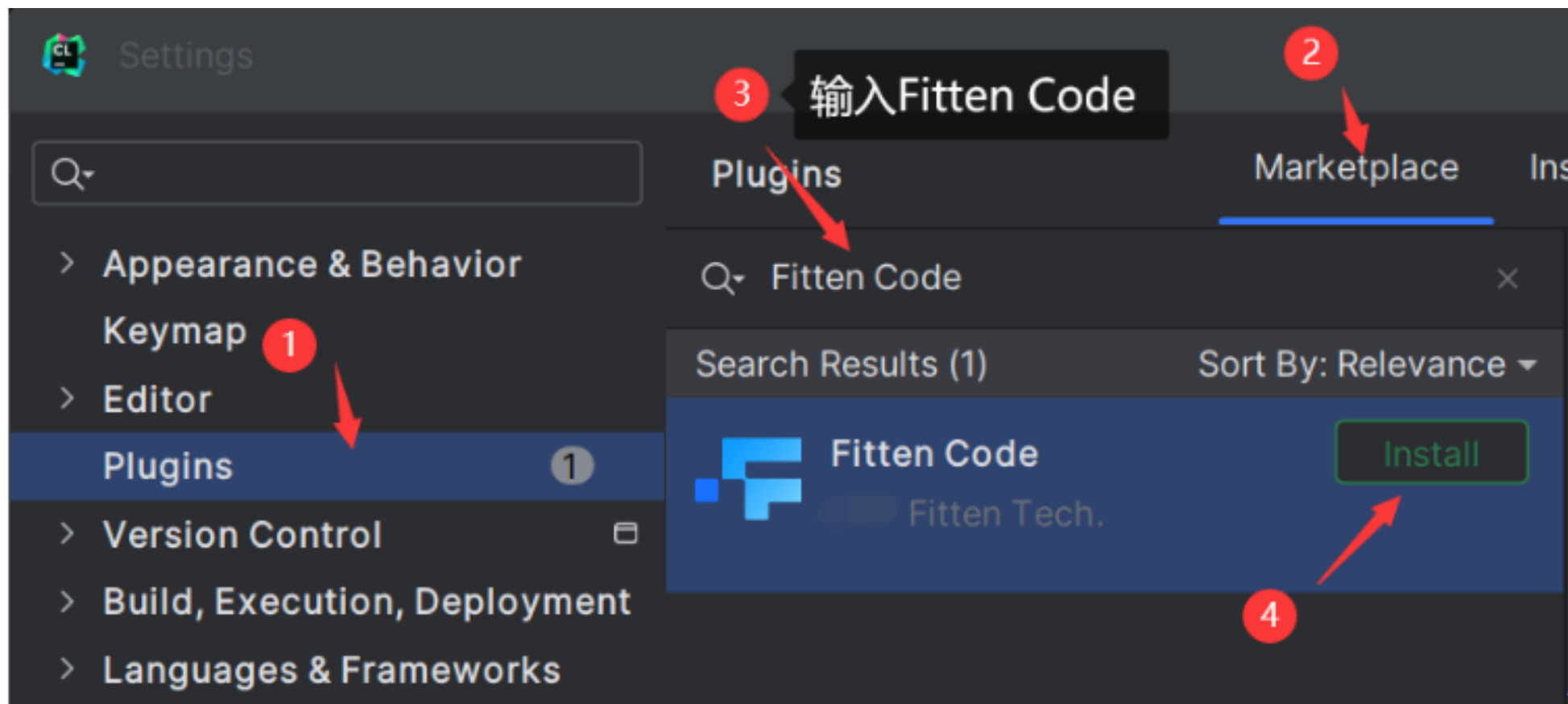
- 3 安装完毕后点击左侧Fitten Code小图标进行登录后即可使用



The image shows a login interface with a dark background. At the top, there are three tabs: '用户名登录' (Username Login) with a person icon, '手机号登录' (Mobile Number Login) with a mobile phone icon, and '邮箱登录' (Email Login) with an envelope icon. The '用户名登录' tab is selected. Below the tabs, there are two input fields: '用户名' (Username) and '密码' (Password). Below the password field, there is a link '忘记密码' (Forgot Password) on the left and a link '注册' (Register) on the right. A large blue button labeled '登录' (Login) is centered below the input fields. At the bottom, there is a horizontal line with the text '其他登录/注册方式' (Other login/register methods) in the center. Below this line is a green circular icon with a white speech bubble, representing WeChat.

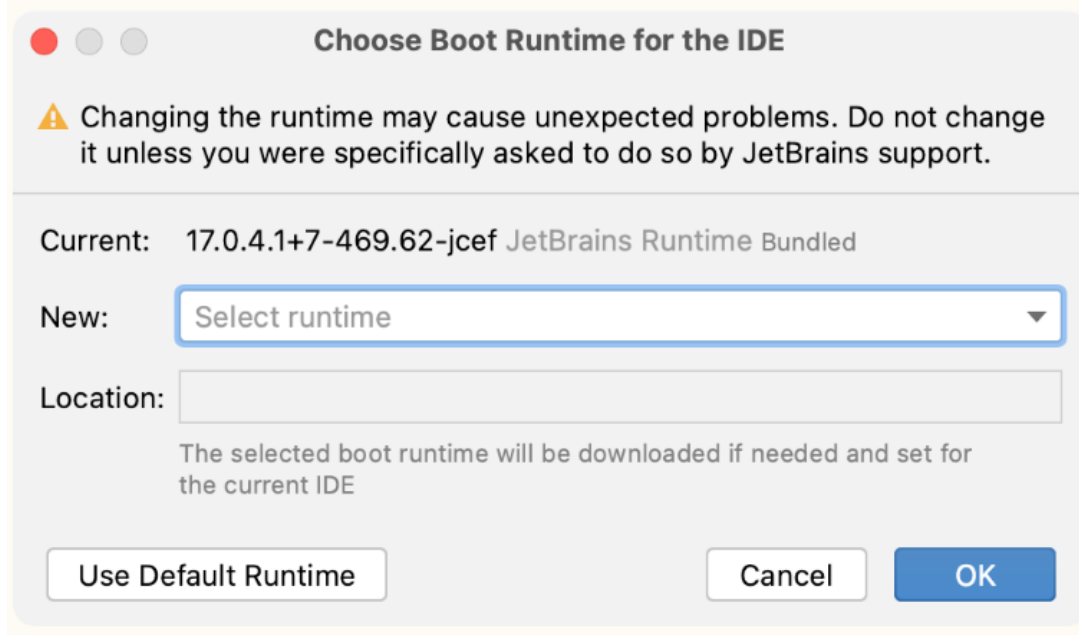
15.2 安装

- 2 点击"Plugins"然后选择"Marketplace",随后在搜索框中输入"Fitten Code", 再点击"Install"



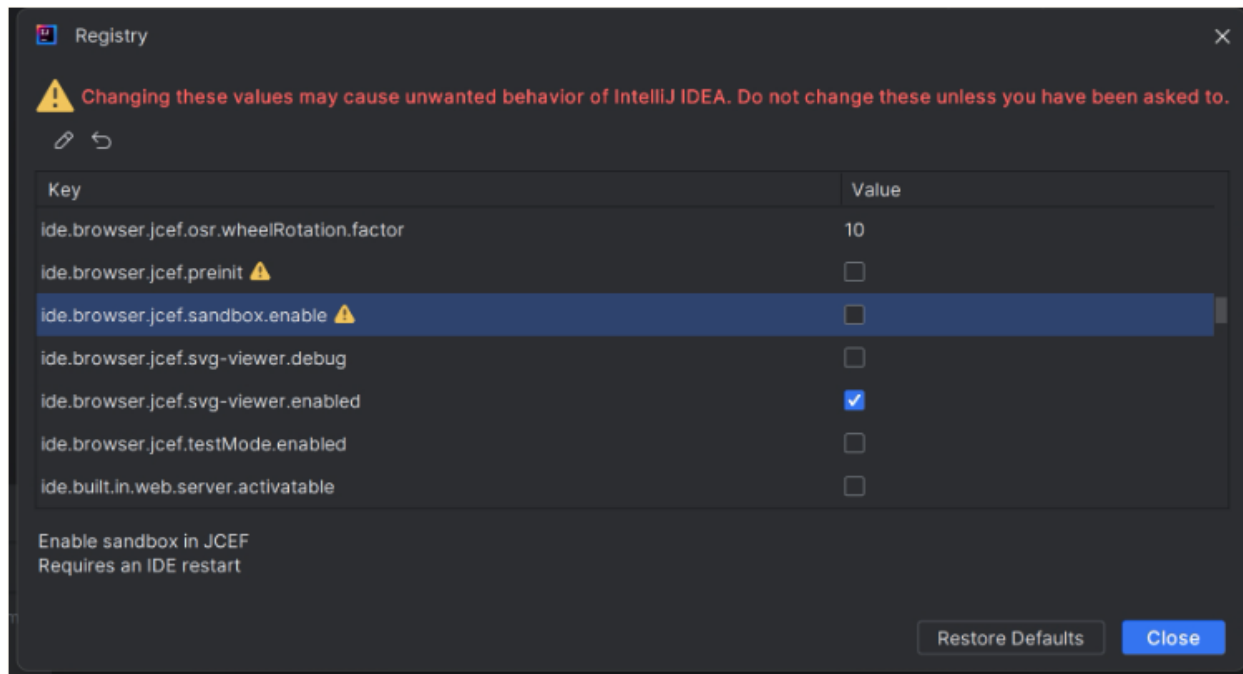
15.2 安装

- 4 如果我们出现了“请安装JCEF”的错误，将会使得Fitten Code无法正常工作，请按照以下步骤进行操作：
 - (a). 在菜单栏，点击 Help | Find Action
 - (b). 搜索并运行“Choose Boot Java Runtime for the IDE”；
 - (c). 选择带有 JCEF 的 Java 运行环境（这一步需要联网，请确保我们的网络和代理服务器畅通）；
 - (d). 等待IDEA更新并重启。



15.2 安装

- 5 如果我们出现了 “Too many restarts of GPU-process” 的错误，并且Fitten Code侧边栏无法显示，请按照以下步骤进行操作：
- (a). 在菜单栏，点击 Help | Find Action;
- (b). 搜索并运行 “Registry...” ;
- (c). 找到并关闭 “ide.browser.jcef.sandbox.enable” 的勾选;
- (d). 确定并重启IntelliJ IDEA。



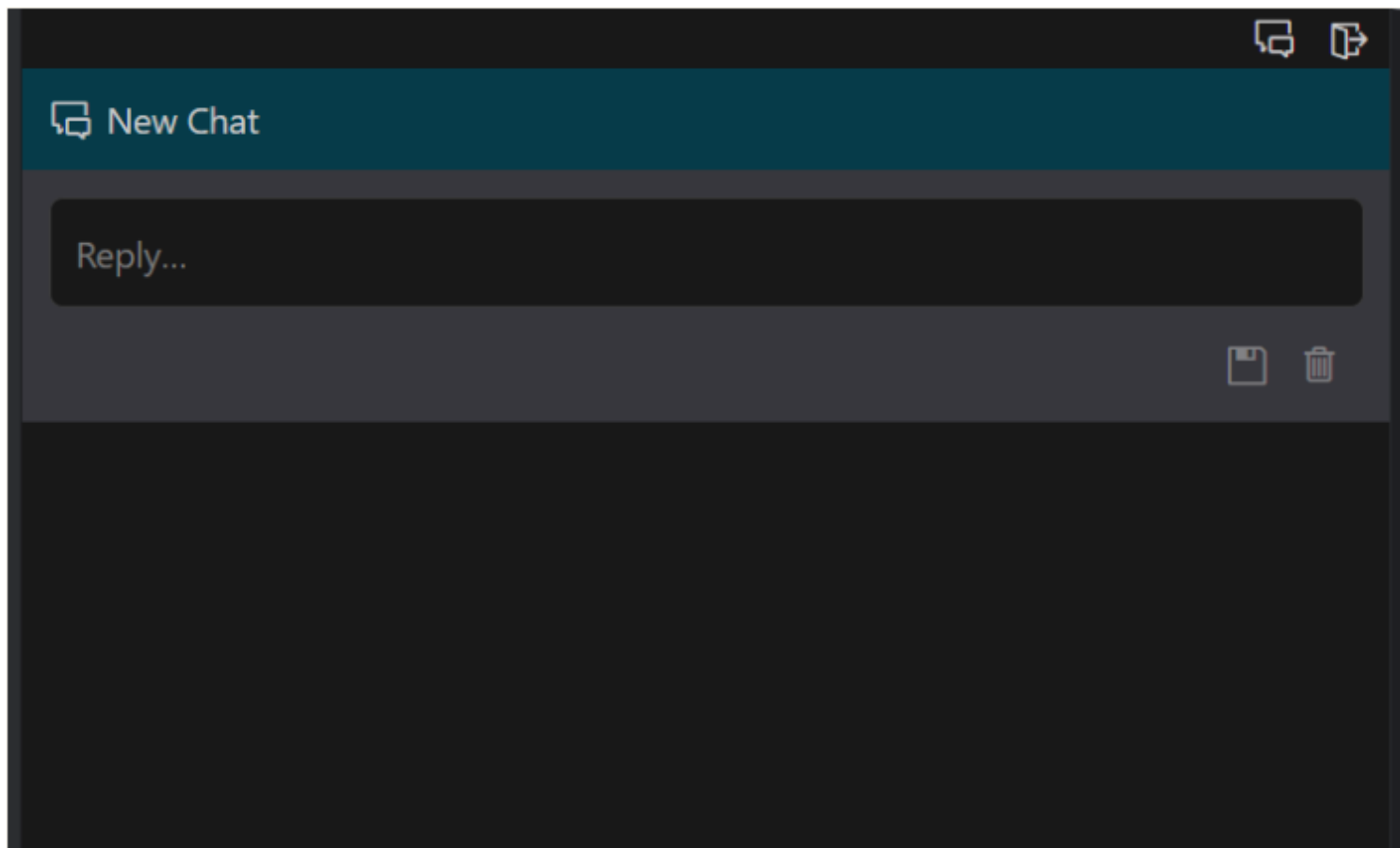
15.3 智能补全

- 打开代码文件，输入一段代码，Fitten Code就会为我们自动补全代码

```
resnet.py x
1 import torch
2
3 class ResNet(torch.nn.Module):
4     def __init__(self, num_classes=10):
5         super(ResNet, self).__init__()
6         self.conv1 = torch.nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
7         self.bn1 = torch.nn.BatchNorm2d(64)
8         self.relu = torch.nn.ReLU(inplace=True)
9         self.maxpool = torch.nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
10        self.layer1 = self._make_layer(64, 64, 3)
11        self.layer2 = self._make_layer(64, 128, 4, stride=2)
12        self.layer3 = self._make_layer(128, 256, 6, stride=2)
```

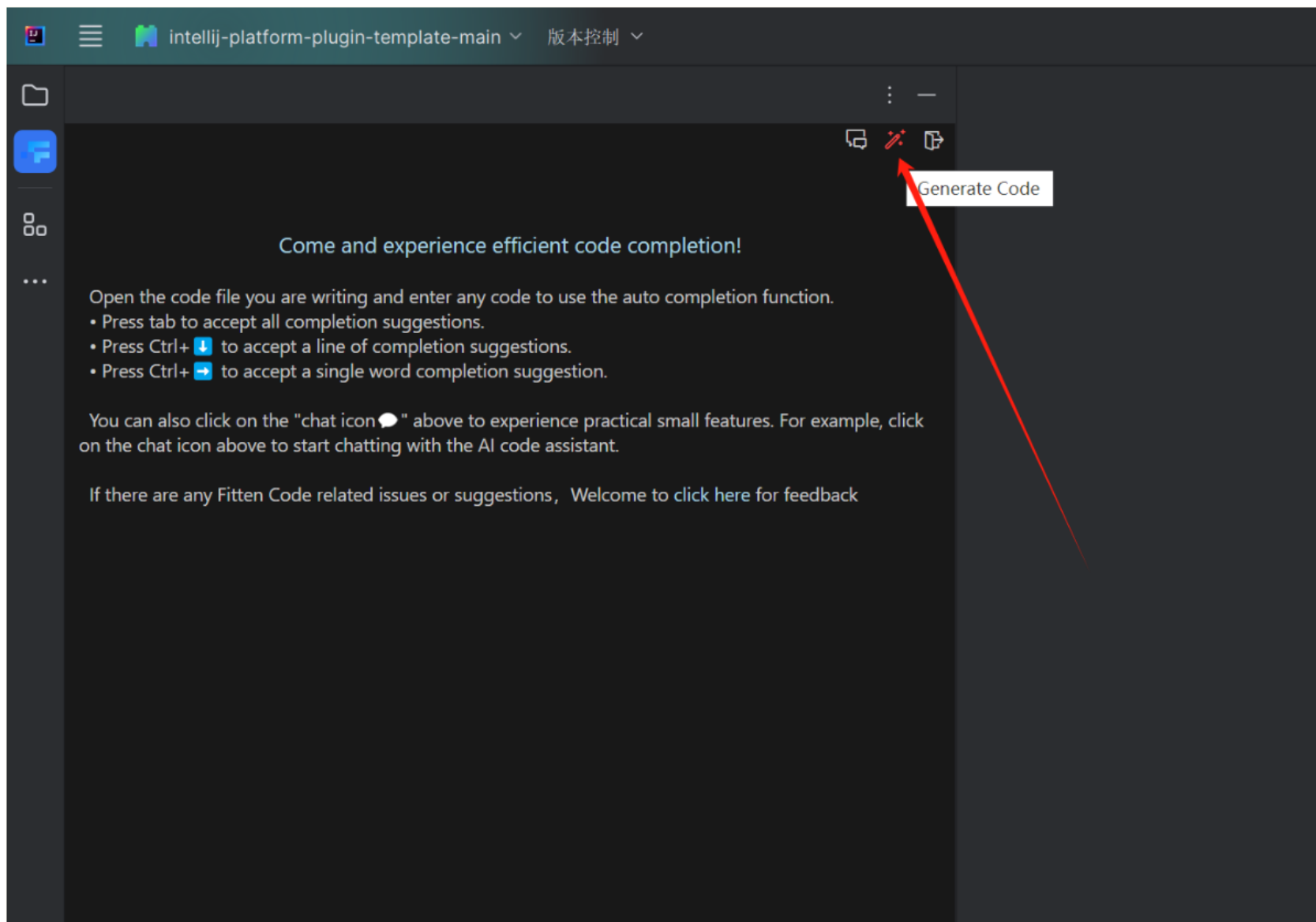
15.4 AI问答

- 用户可通过点击左上角工具栏中的“Fitten Code - 开始对话”或者使用快捷键“Ctrl+Alt+C” 打开对话窗口进行对话



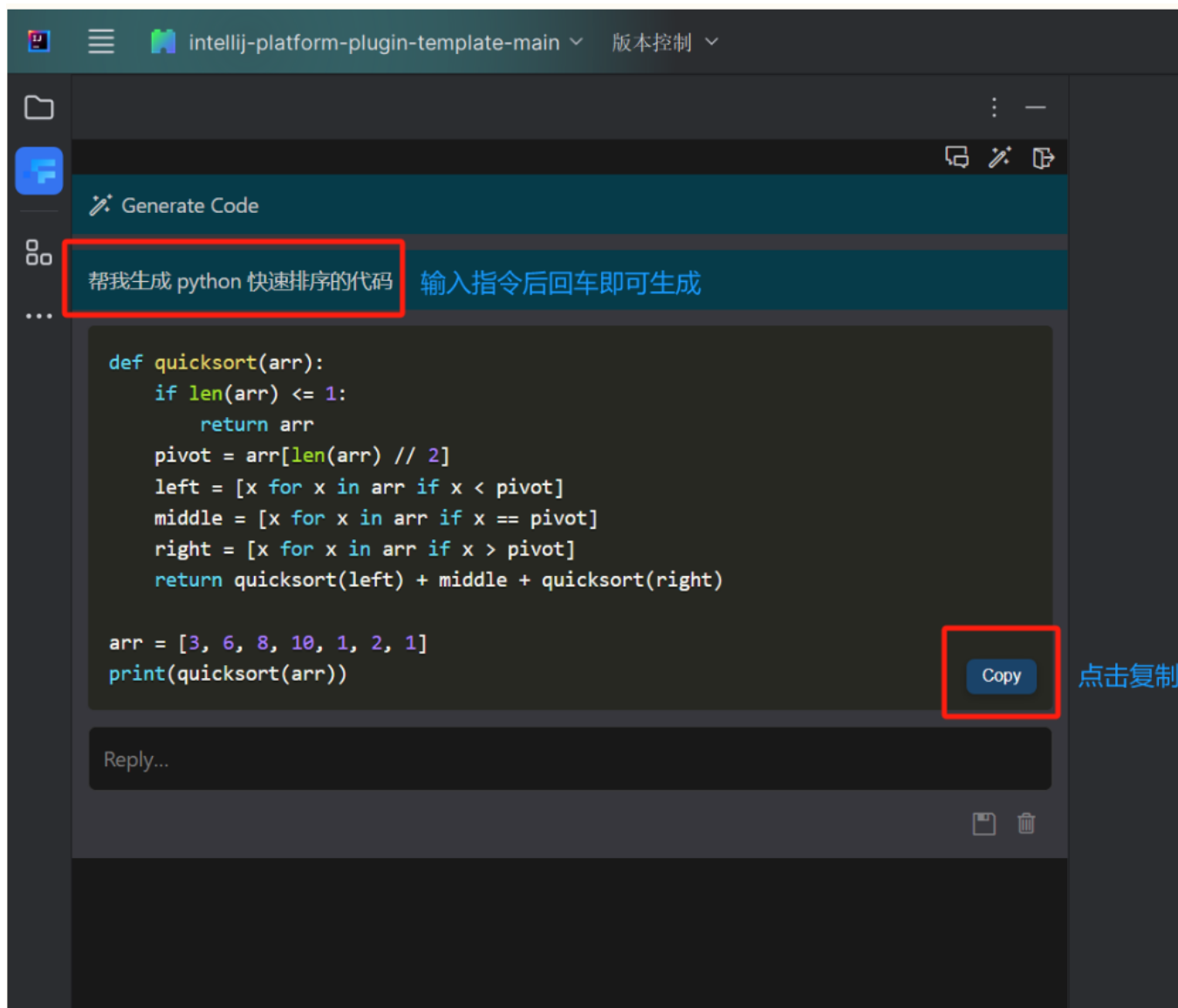
15.5 生成代码

- 生成代码



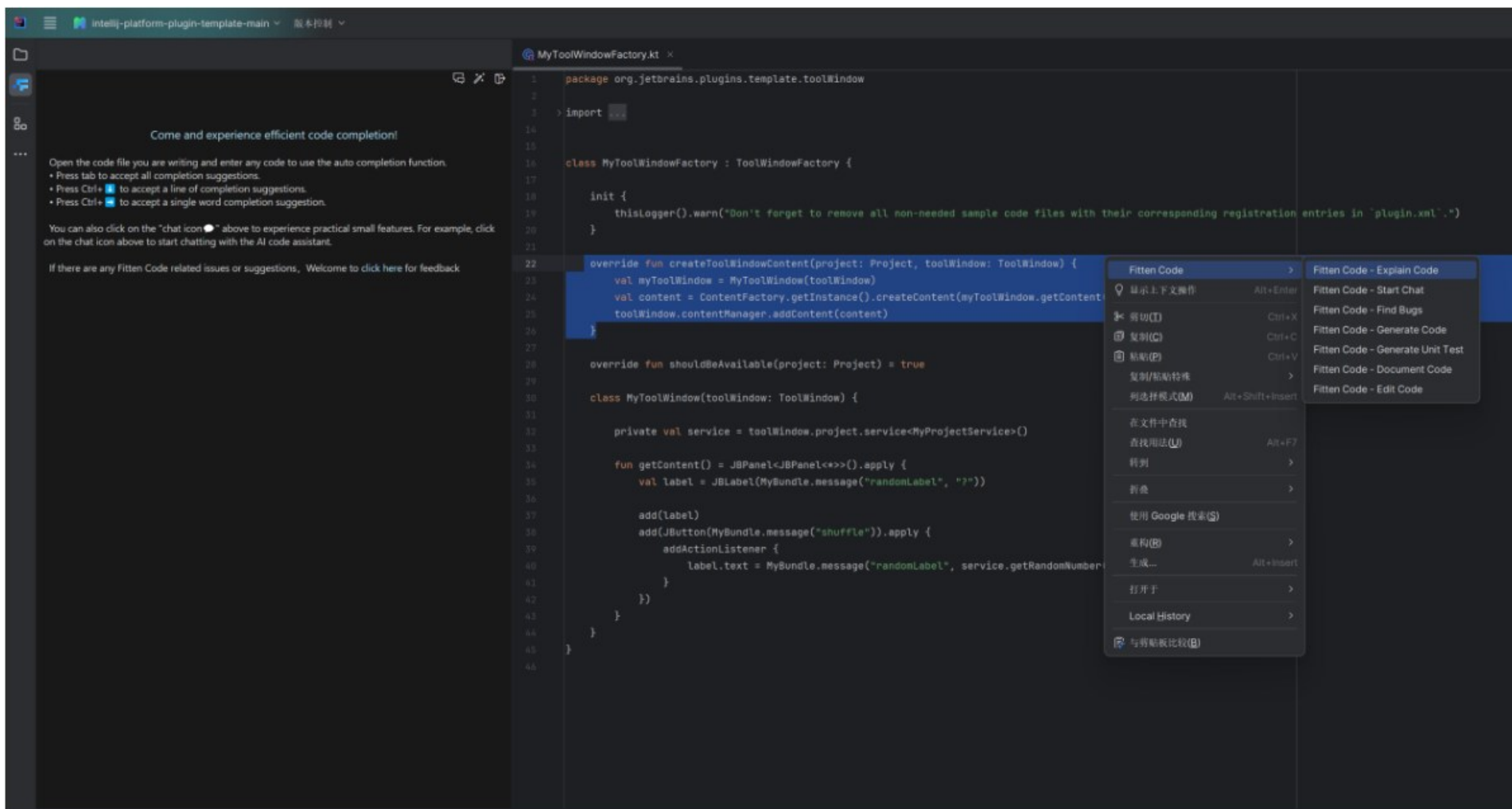
15.5 生成代码

- 生成代码



15.6 代码翻译

- 解释代码（选中代码段后右键，后续操作同理）



15.6 代码翻译

- 解释代码（选中代码段后右键，后续操作同理）

The provided code snippet is from a Kotlin file, likely within an IntelliJ IDEA plugin, and it contains an override function to create the content of a tool window within the IDE.

Let's break down the code step by step:

- 1. override fun createToolWindowContent(project: Project, toolWindow: ToolWindow) {**
This line declares a function `createToolWindowContent` which overrides a function defined in the `ToolWindowFactory` interface. When a tool window is created within the IDE, this function is called with the project and tool window as parameters.
- 2. val myToolWindow = MyToolWindow(toolWindow)**
Here, a new instance of the `MyToolWindow` class is created, passing the `toolWindow` from the function parameters to its constructor.
- 3. val content =**
`ContentFactory.getInstance().createContent(myToolWindow.getContent(), null, false)`
This line creates the content for the tool window. It uses the `ContentFactory` to create a content instance using the content of the `MyToolWindow` instance created in the previous step.
- 4. toolWindow.contentManager.addContent(content)**
Finally, the newly created content is added to the `contentManager` of the `toolWindow`.

In summary, the main function of this code snippet is to override the `createToolWindowContent` function and provide the content for the tool window by creating an instance of `MyToolWindow` and adding its content to the tool window's content manager. The `MyToolWindow` class likely defines the specific content and behavior of the tool window, including UI components and event handlers.

override的作用是什么 可继续提问

override 关键字用于在子类中重写（覆盖）父类或接口中已定义的函数或属性。

在给定的示例中，override 关键字表明 `createToolWindowContent` 函数在实现 `ToolWindowFactory` 接口时被重写。这意味着在该特定类中，`createToolWindowContent` 函数将覆盖该接口中原本的实现。

重写的作用在于提供一个新的实现，以满足特定子类的需求或逻辑。通过重写父类或接口中的函数或属性，子类可以定制化行为，使其符合特定的上下文或需求。

在 Kotlin 中，使用 `override` 关键字来重写父类或接口中已定义的函数或属性，有助于理解代码的结构和实现，并且可以提高代码的可读性和可维护性。

```
package org.jetbrains.plugins.template.toolWindow

import ...

class MyToolWindowFactory : ToolWindowFactory {

    init {
        this.logger().warn("Don't forget to remove all non-needed sample code files with their corresponding registration entries in 'plugin.xml'.")
    }

    override fun createToolWindowContent(project: Project, toolWindow: ToolWindow) {
        val myToolWindow = MyToolWindow(toolWindow)
        val content = ContentFactory.getInstance().createContent(myToolWindow.getContent(), null, false)
        toolWindow.contentManager.addContent(content)
    }

    override fun shouldBeAvailable(project: Project) = true

    class MyToolWindow(toolWindow: ToolWindow) {

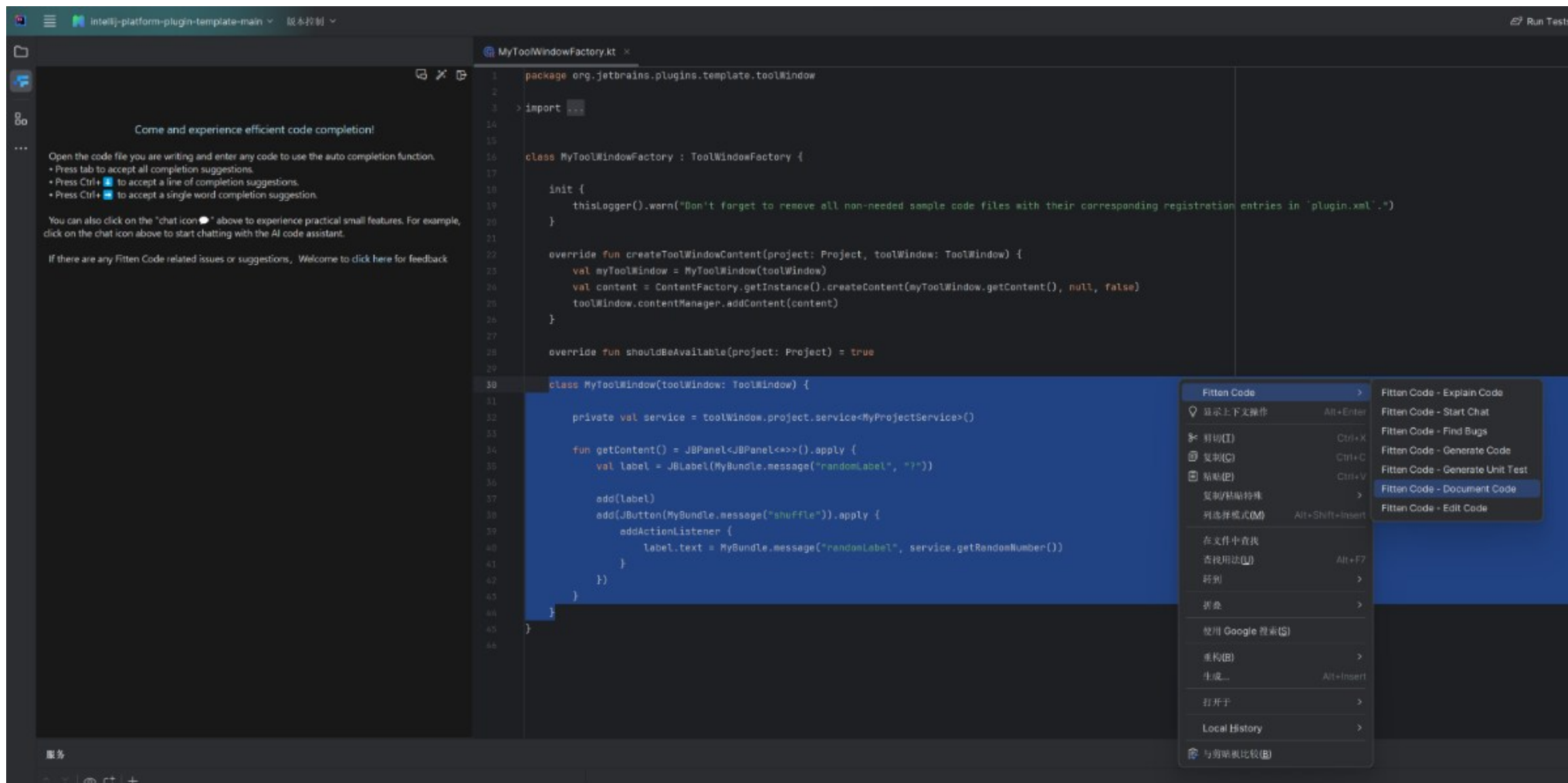
        private val service = toolWindow.project.service<MyProjectService>()

        fun getContent() = JPanel<JPanel<*>>>().apply {
            val label = JLabel(MyBundle.message("randomLabel", "?"))

            add(label)
            add(JButton(MyBundle.message("shuffle"))).apply {
                addActionListener {
                    label.text = MyBundle.message("randomLabel", service.getRandomNumber())
                }
            }
        }
    }
}
```

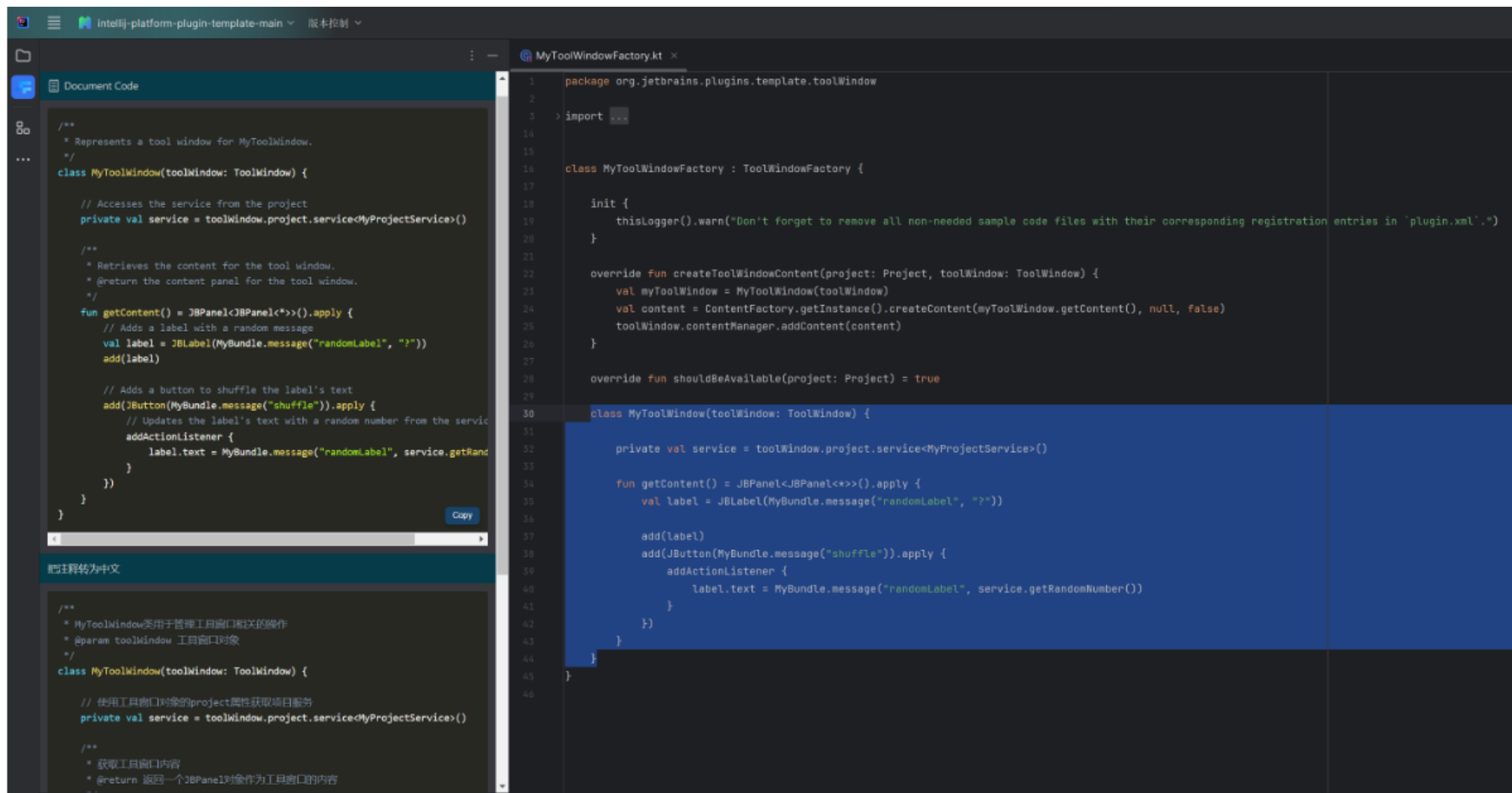
15.7 生成注释

● 生成注释



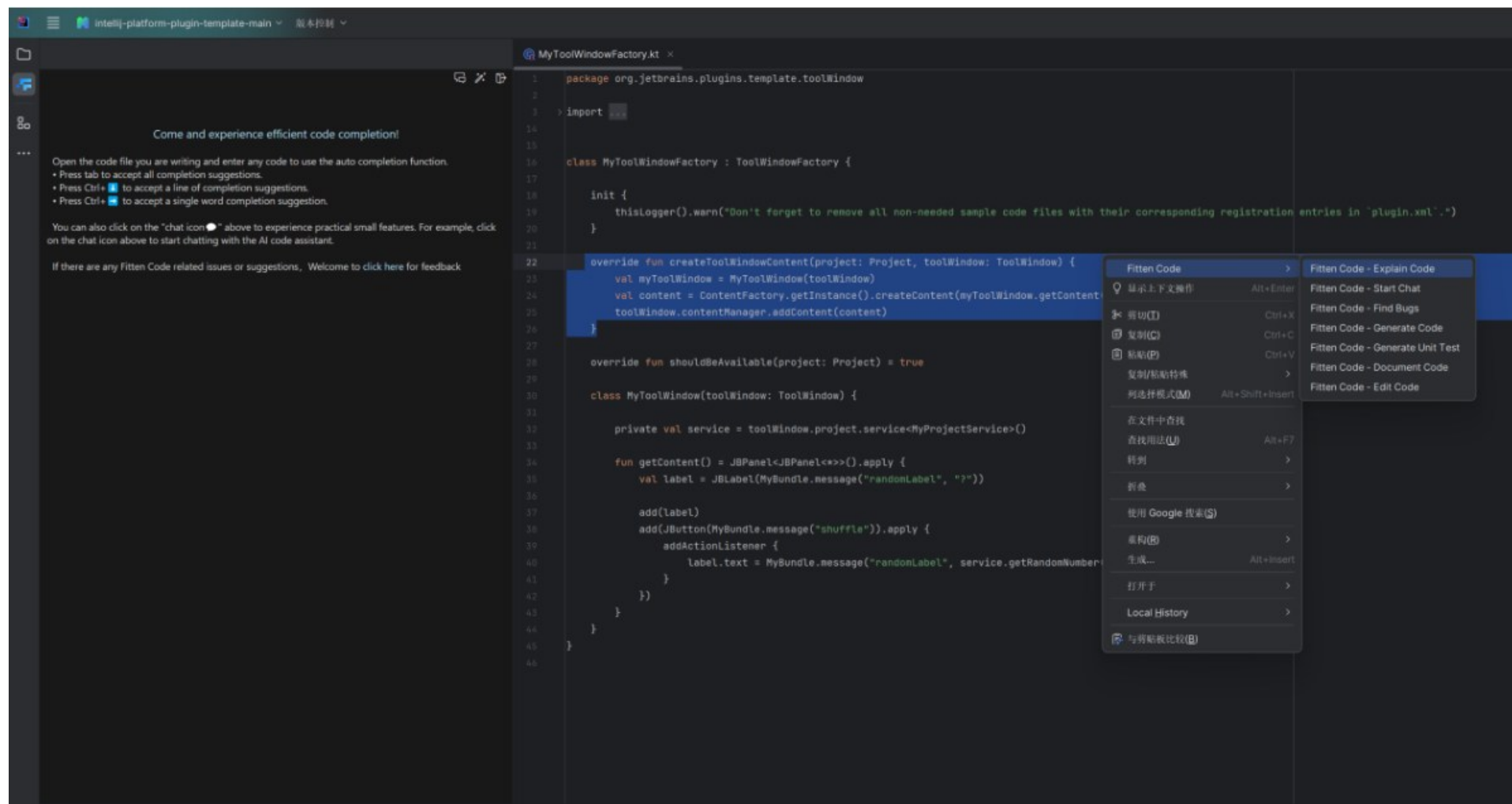
15.7 生成注释

- 生成注释



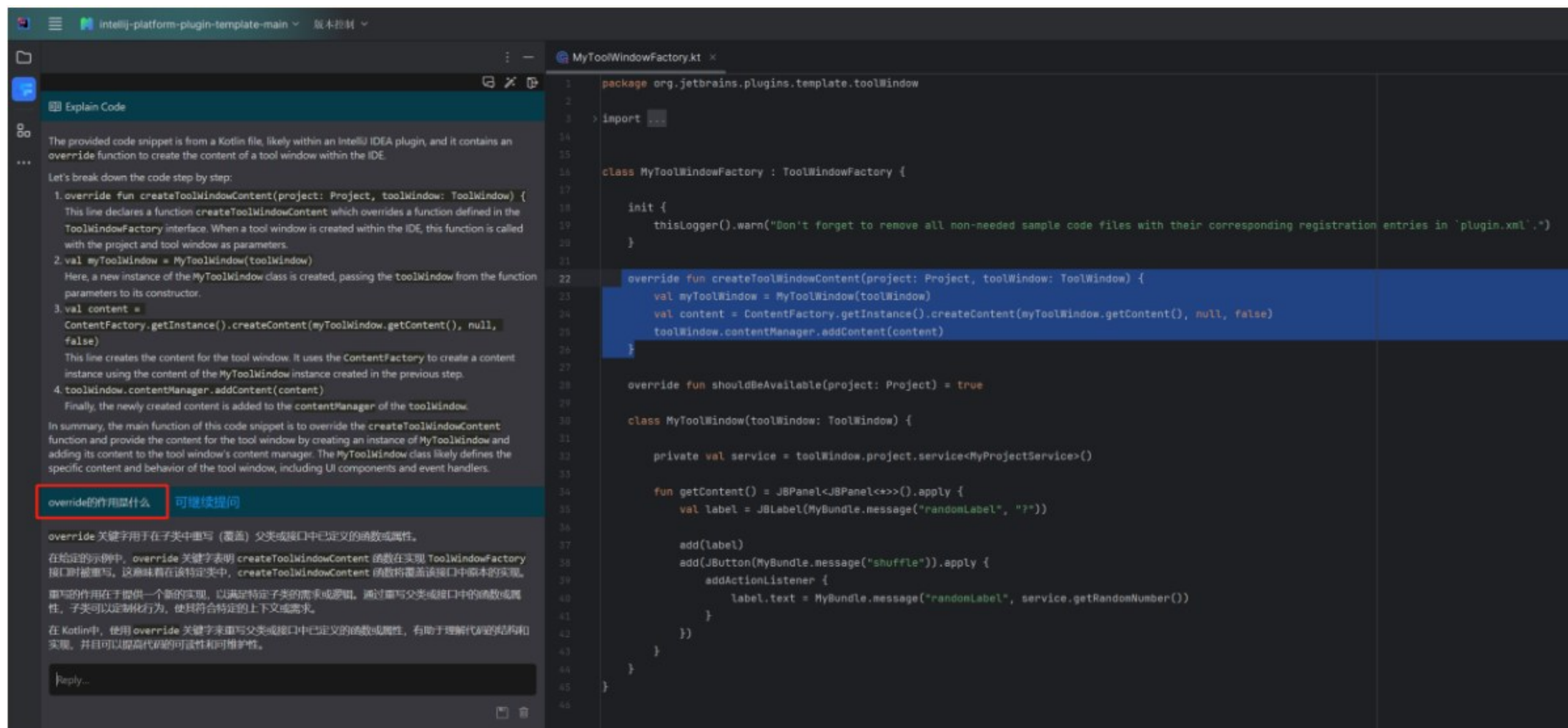
15.8 解释代码

- 解释代码（选中代码段后右键，后续操作同理）



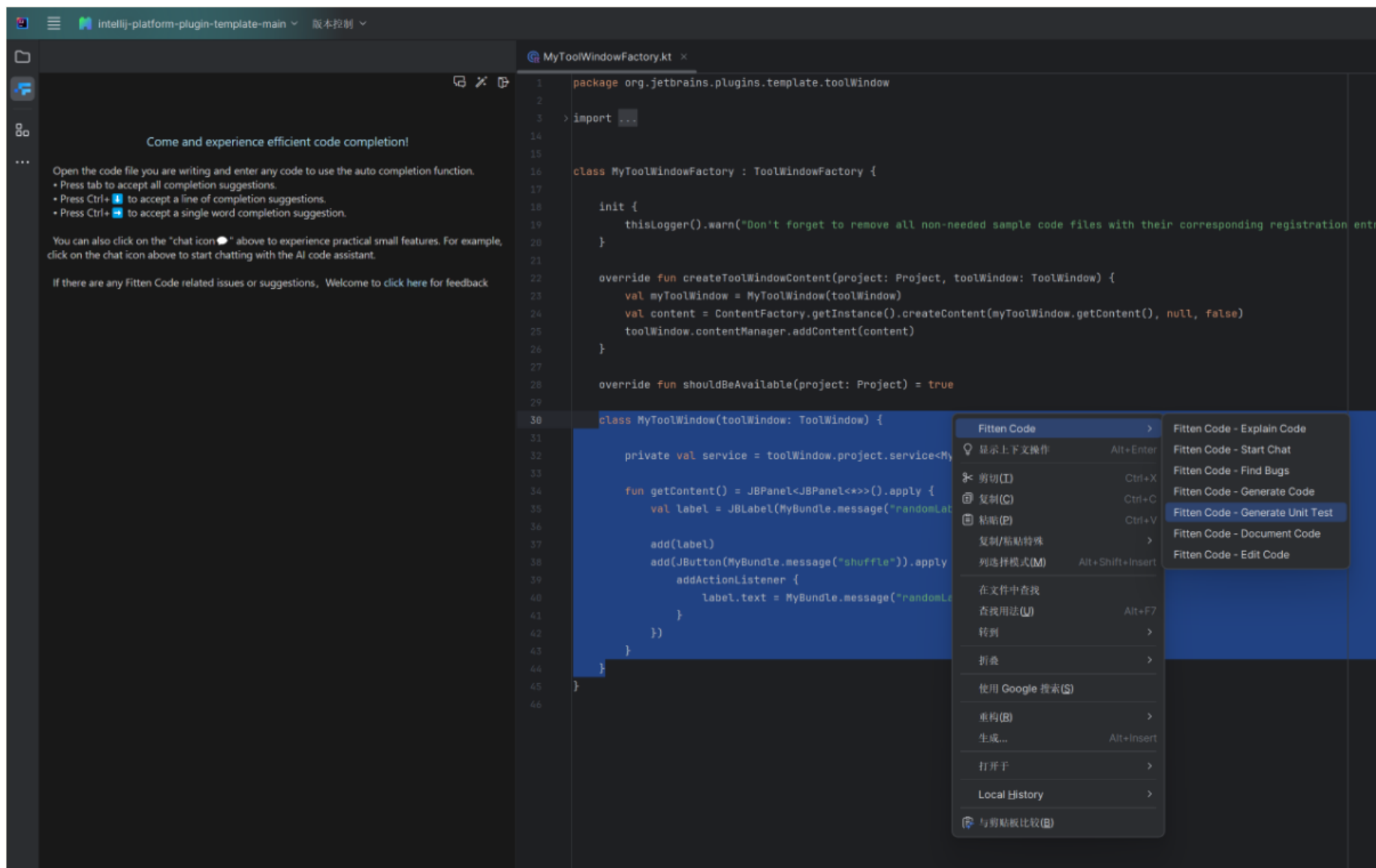
15.8 解释代码

- 解释代码（选中代码段后右键，后续操作同理）



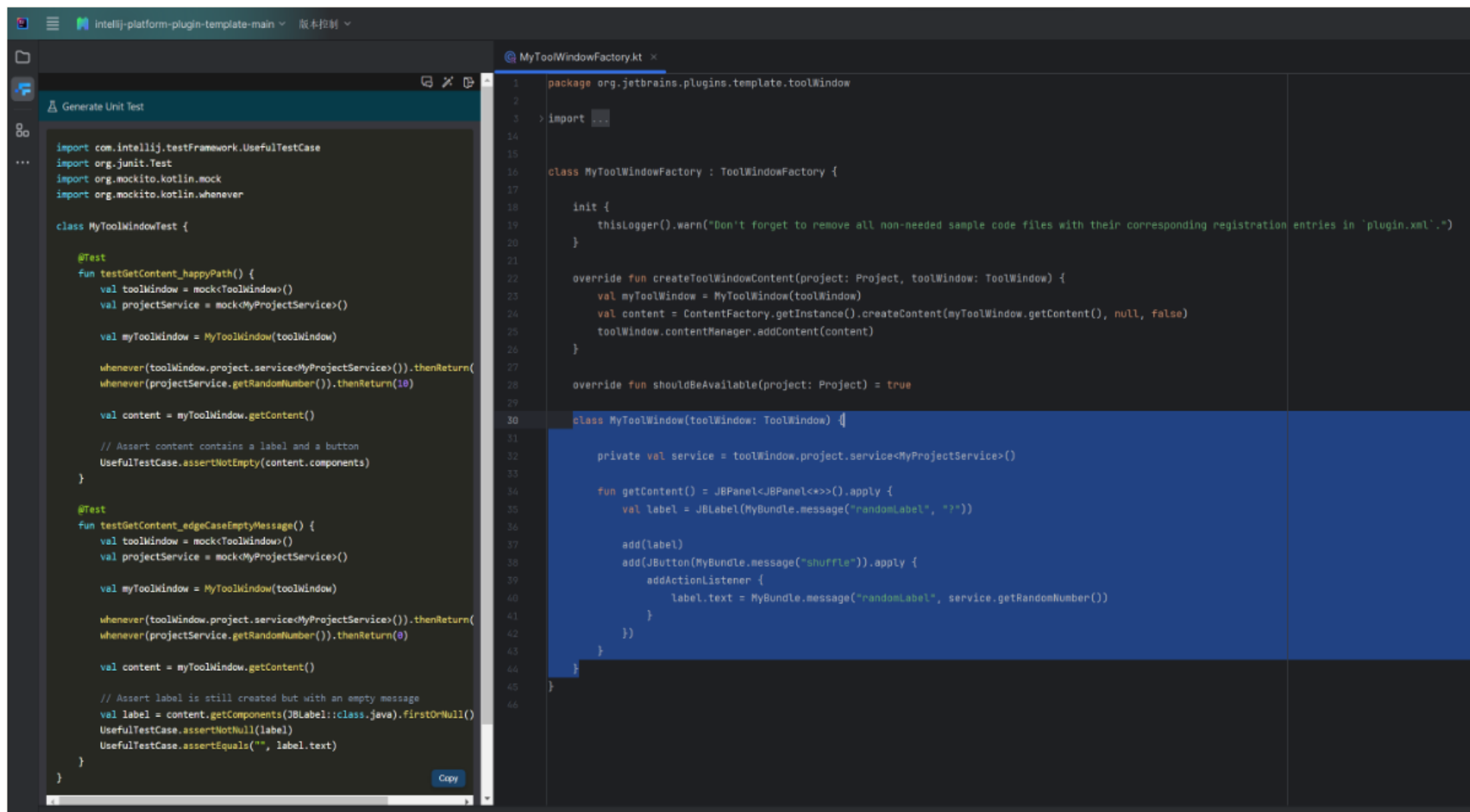
15.9 生成测试

● 生成单元测试



15.9 生成测试

- 生成单元测试



The screenshot displays the IntelliJ IDEA IDE interface. On the left, the 'Generate Unit Test' dialog is open, showing the generated unit test code for 'MyToolWindowTest'. The code includes imports for 'com.intellij.testFramework.UsefulTestCase', 'org.junit.Test', 'org.mockito.kotlin.mock', and 'org.mockito.kotlin.whenever'. It defines two test methods: 'testGetContent_happyPath()' and 'testGetContent_edgeCaseEmptyMessage()'. The right pane shows the source code for 'MyToolWindowFactory.kt' and 'MyToolWindow.kt'. The 'MyToolWindowFactory' class implements the 'ToolWindowFactory' interface, providing an 'init' block and a 'createToolWindowContent' method. The 'MyToolWindow' class is an inner class that implements the 'ToolWindow' interface, containing a 'private val service' and a 'getContent()' method that returns a 'JPanel' with a 'JLabel' and a 'JButton'.

```
import com.intellij.testFramework.UsefulTestCase
import org.junit.Test
import org.mockito.kotlin.mock
import org.mockito.kotlin.whenever

class MyToolWindowTest {

    @Test
    fun testGetContent_happyPath() {
        val toolWindow = mock<ToolWindow>()
        val projectService = mock<MyProjectService>()

        val myToolWindow = MyToolWindow(toolWindow)

        whenever(toolWindow.project.service<MyProjectService>()).thenReturn(
            whenever(projectService.getRandomNumber()).thenReturn(10)
        )

        val content = myToolWindow.getContent()

        // Assert content contains a label and a button
        UsefulTestCase.assertNotNull(content.components)
    }

    @Test
    fun testGetContent_edgeCaseEmptyMessage() {
        val toolWindow = mock<ToolWindow>()
        val projectService = mock<MyProjectService>()

        val myToolWindow = MyToolWindow(toolWindow)

        whenever(toolWindow.project.service<MyProjectService>()).thenReturn(
            whenever(projectService.getRandomNumber()).thenReturn(0)
        )

        val content = myToolWindow.getContent()

        // Assert label is still created but with an empty message
        val label = content.getComponents(JLabel::class.java).firstOrNull()
        UsefulTestCase.assertNotNull(label)
        UsefulTestCase.assertEquals("", label.text)
    }
}
```

```
package org.jetbrains.plugins.template.toolWindow

import ...

class MyToolWindowFactory : ToolWindowFactory {

    init {
        this.logger().warn("Don't forget to remove all non-needed sample code files with their corresponding registration entries in 'plugin.xml'.")
    }

    override fun createToolWindowContent(project: Project, toolWindow: ToolWindow) {
        val myToolWindow = MyToolWindow(toolWindow)
        val content = ContentFactory.getInstance().createContent(myToolWindow.getContent(), null, false)
        toolWindow.contentManager.addContent(content)
    }

    override fun shouldBeAvailable(project: Project) = true
}

class MyToolWindow(toolWindow: ToolWindow) {

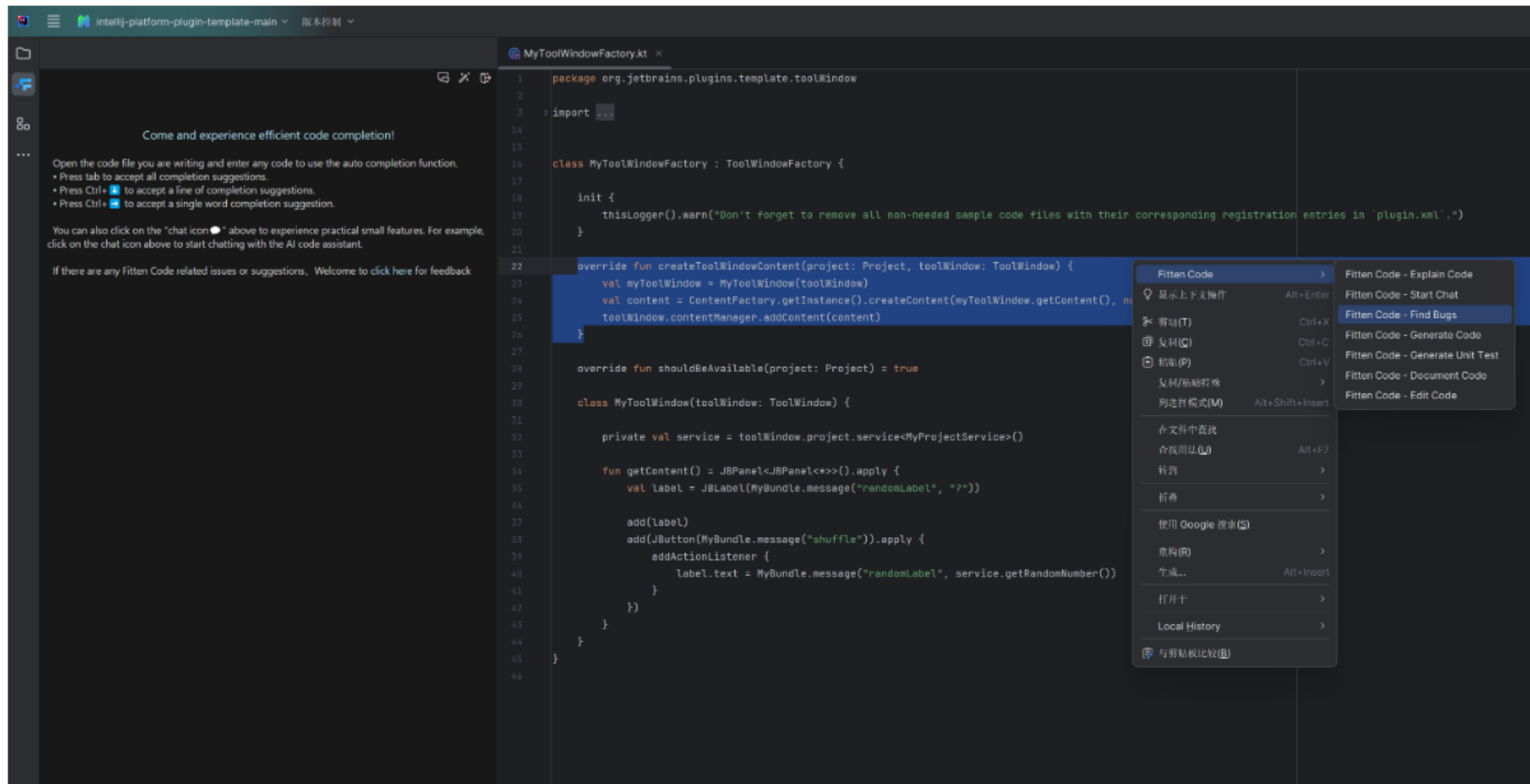
    private val service = toolWindow.project.service<MyProjectService>()

    fun getContent() = JPanel<JPanel<*>>().apply {
        val label = JLabel(MyBundle.message("randomLabel", ""))

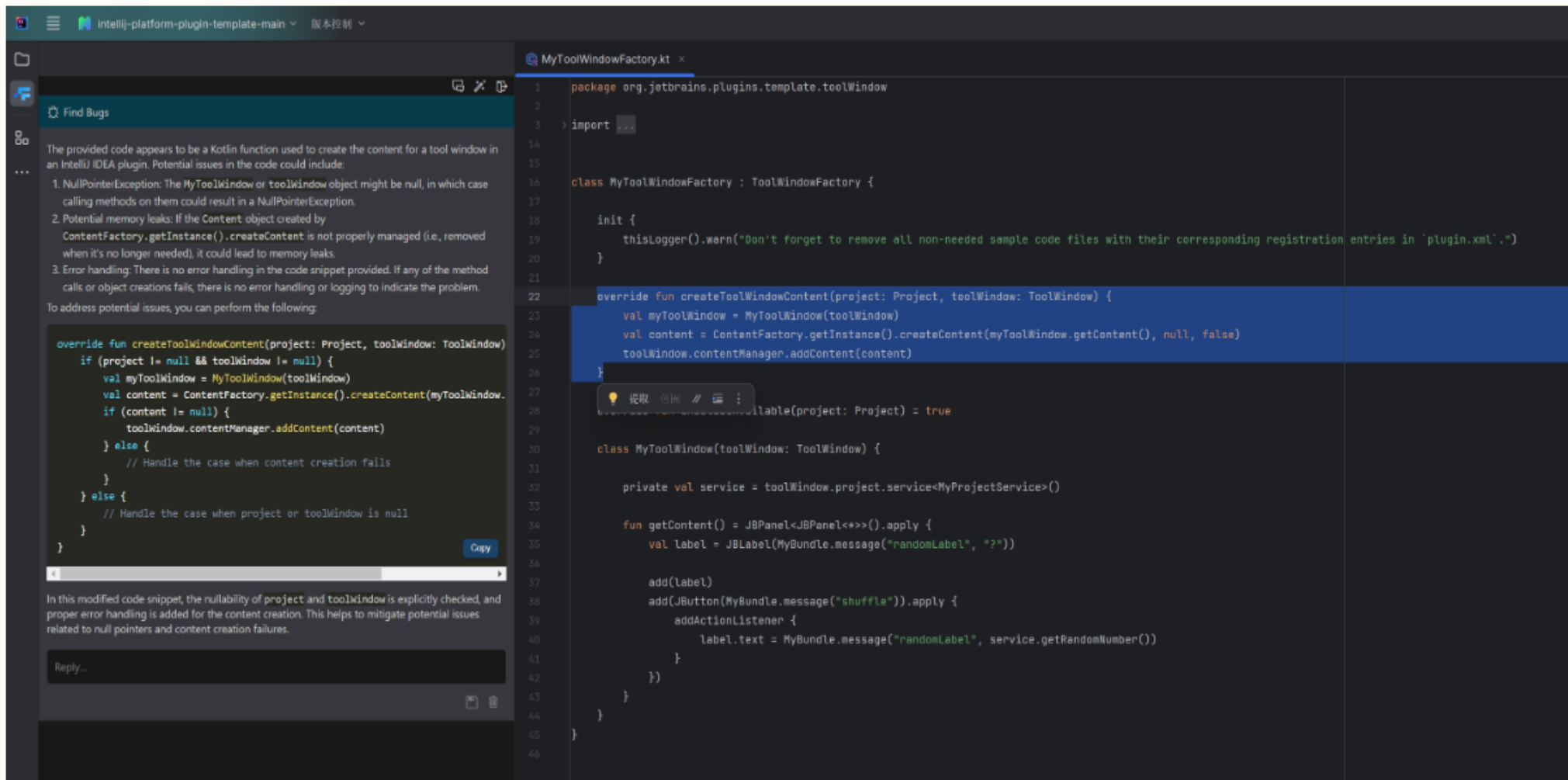
        add(label)
        add(JButton(MyBundle.message("shuffle")).apply {
            addActionListener {
                label.text = MyBundle.message("randomLabel", service.getRandomNumber())
            }
        })
    }
}
```

15.10 检查 BUG

● 查找bug

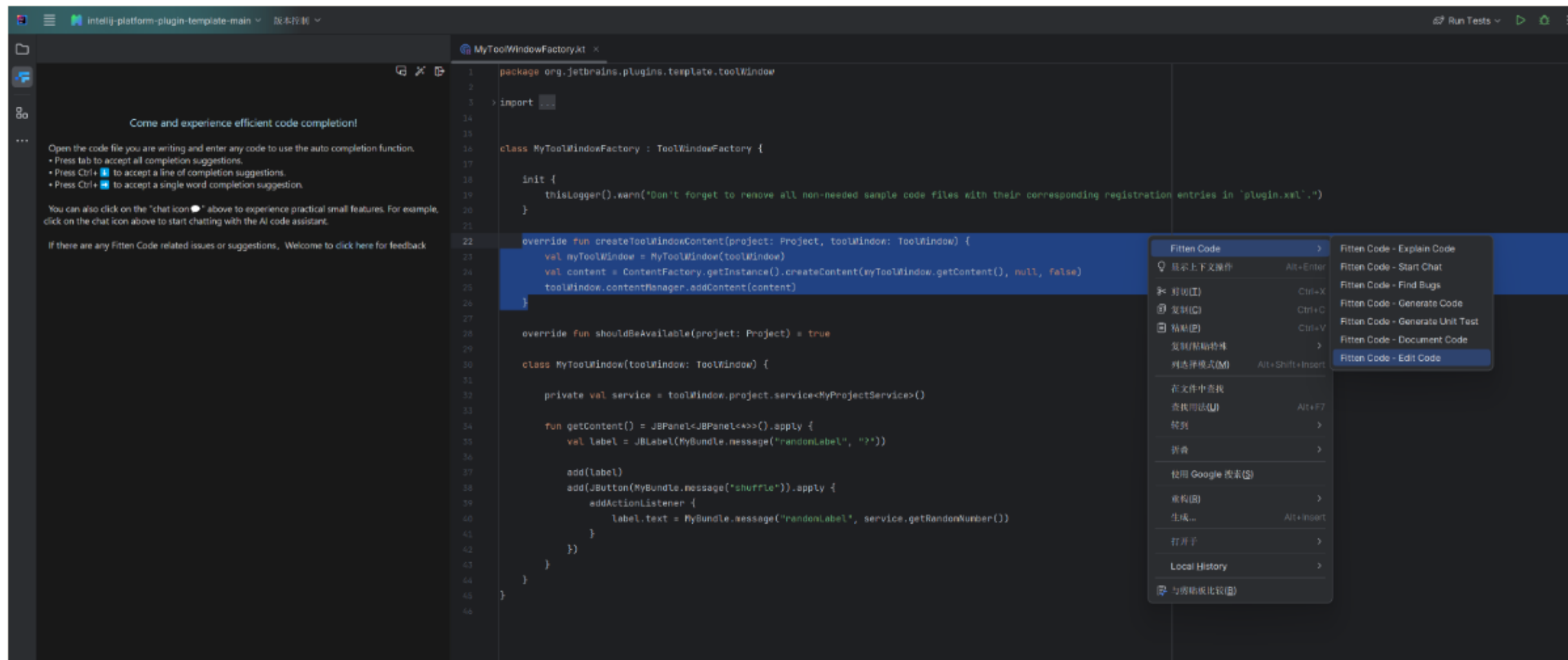


● 查找bug



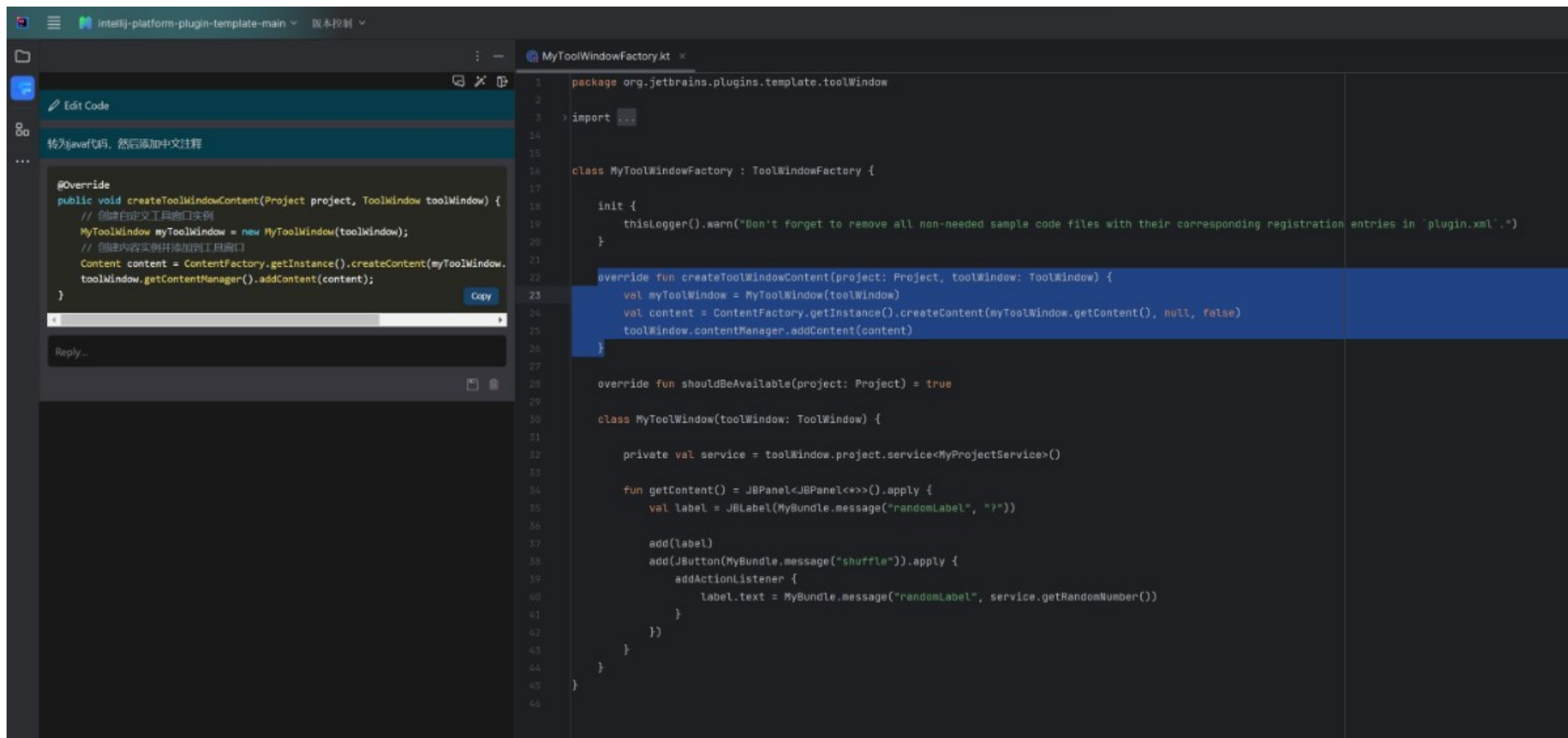
15.11 编辑代码

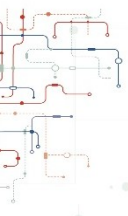
● 编辑代码



15.11 编辑代码

- 编辑代码

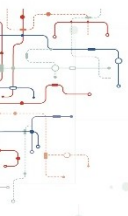




15.12 常见问题

- 点击右下角Fitten图标，然后选择“打开插件设置”，即可打开Fitten Code的设置页面。以下为可设置选项

1. 延迟自动补全的时间(单位：毫秒)
2. Fitten Code的显示语言
3. 使用"Fitten Code - 生成注释"功能时的语言首选项



15.12 常见问题

- 12.1 IDEA低版本搜不到Fitten Code

答：2024/3/21号发布了0.10.3版本，最低支持2021版本JetBrain。

- 12.2 Android Studio侧边栏没有反应

答：安装JCEF，并且关闭sandbox即可。

- 12.3 idea英文但是想让fitten是中文的方便看代码解释

答：点击右下角Fitten图标，然后选择“打开插件设置”，即可设置Fitten Code的语言为中文。

- 12.4 如何选择各功能回答时的语言？

答：点击右下角Fitten图标，然后选择“打开插件设置”，即可设置对应功能回答的语言。

本章重点

- 掌握以下内容:

Fitten Code安装

使用Fitten Code对话的方式生成代码

使用Fitten Code查找BUG



本章作业

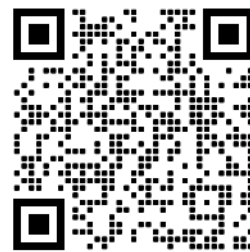
- 完成以下内容:

- 1 IntelliJ IDEA 中安装Fitten Code组件
- 2 使用Fitten Code对话的方式生成代码
- 3 使用Fitten Code查找BUG



信创智能医疗系统研发课程体系

河南中医药大学信息技术学院（智能医疗行业学院）



河南中医药大学信息技术学院（智能医疗行业学院）智能医疗教研室

河南中医药大学医疗健康信息工程技术研究所