**Android studio plugin**

**Step 1**

Find some simple constrains(10-15 samples).

For example,



<https://developer.android.com/guide/components/activities/activity-lifecycle>

**Step 2**

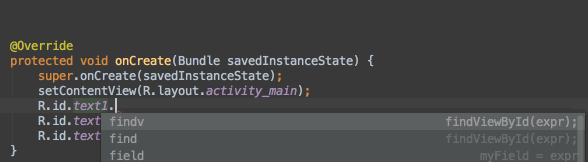
Start build plugin.

configuration environment: Gradle and DevKit

* *There are two kinds of entry location.*

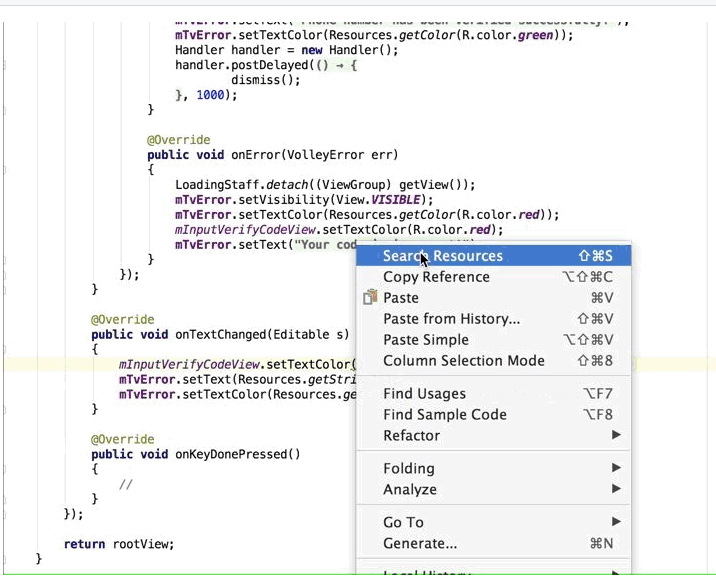
1st: Drop-down menu

<https://github.com/takahirom/android-postfix-plugin>



2nd: On right-click menu

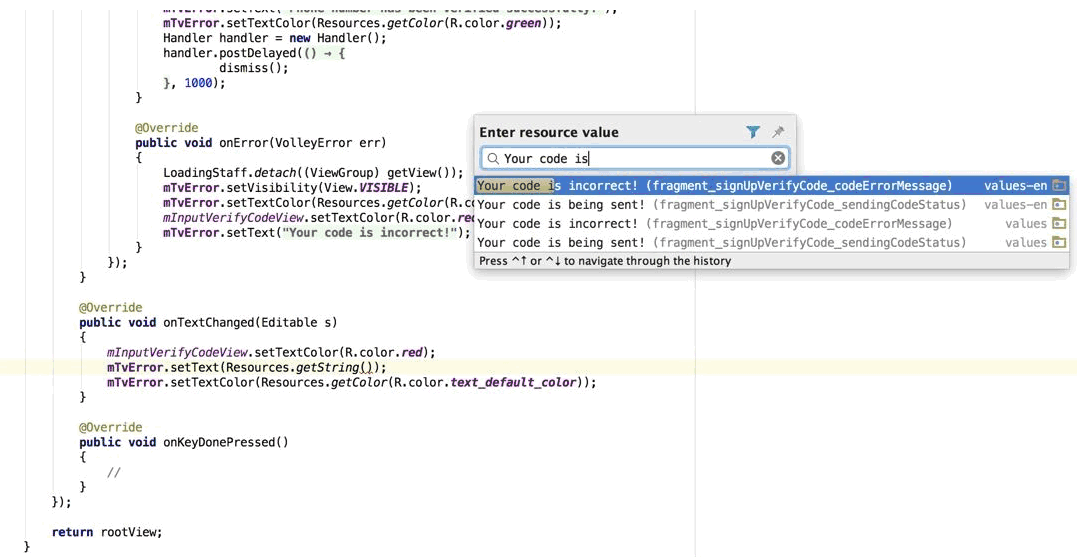
<https://github.com/hoai265/SearchResourcePlugin>



* *There are two parts to show inspection result.*

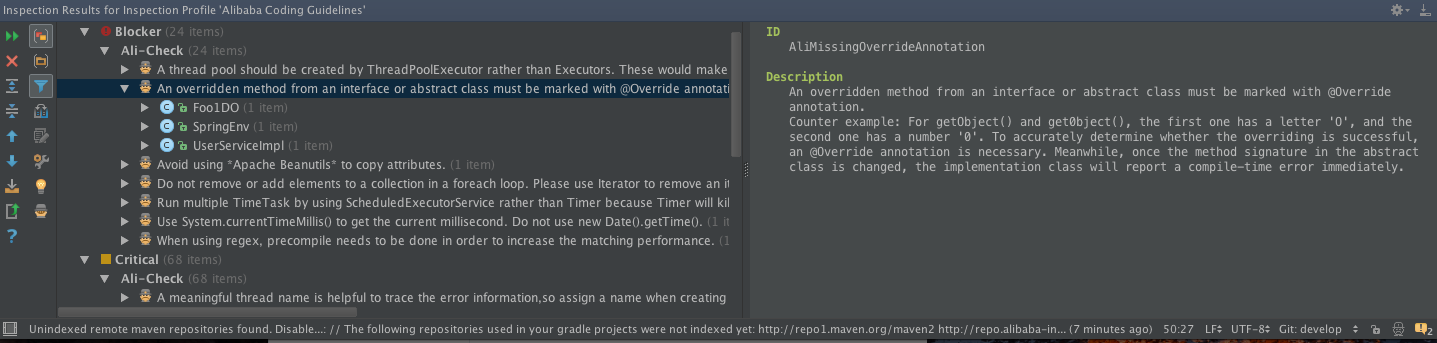
1st: Open a dialog

<https://github.com/hoai265/SearchResourcePlugin>



2nd: Shows ID and Description in dialog (ToolWindow?)

<https://github.com/XenoAmess/p3c/tree/xenoamess_maintain_fork/idea-plugin>



ToolWindow

<https://www.jetbrains.com/help/idea/tool-windows.html#general-tool-windows-layout>

<https://intellij-sdk-docs-cn.github.io/intellij/sdk/docs/user_interface_components/tool_windows.html>

<https://plugins.jetbrains.com/plugin/8558-tool-window>

* *How to search the constraint's code/function in the android studio project source code?*

Research:

Intellij -> open api -> editor -> editor.java -> getDocument(); getProject();

Intellij -> open api -> editor -> document.java ->getText();

Use PSI? <https://plugins.jetbrains.com/docs/intellij/psi-cookbook.html#how-can-i-cause-the-psi-for-a-virtual-file-to-be-rebuilt>

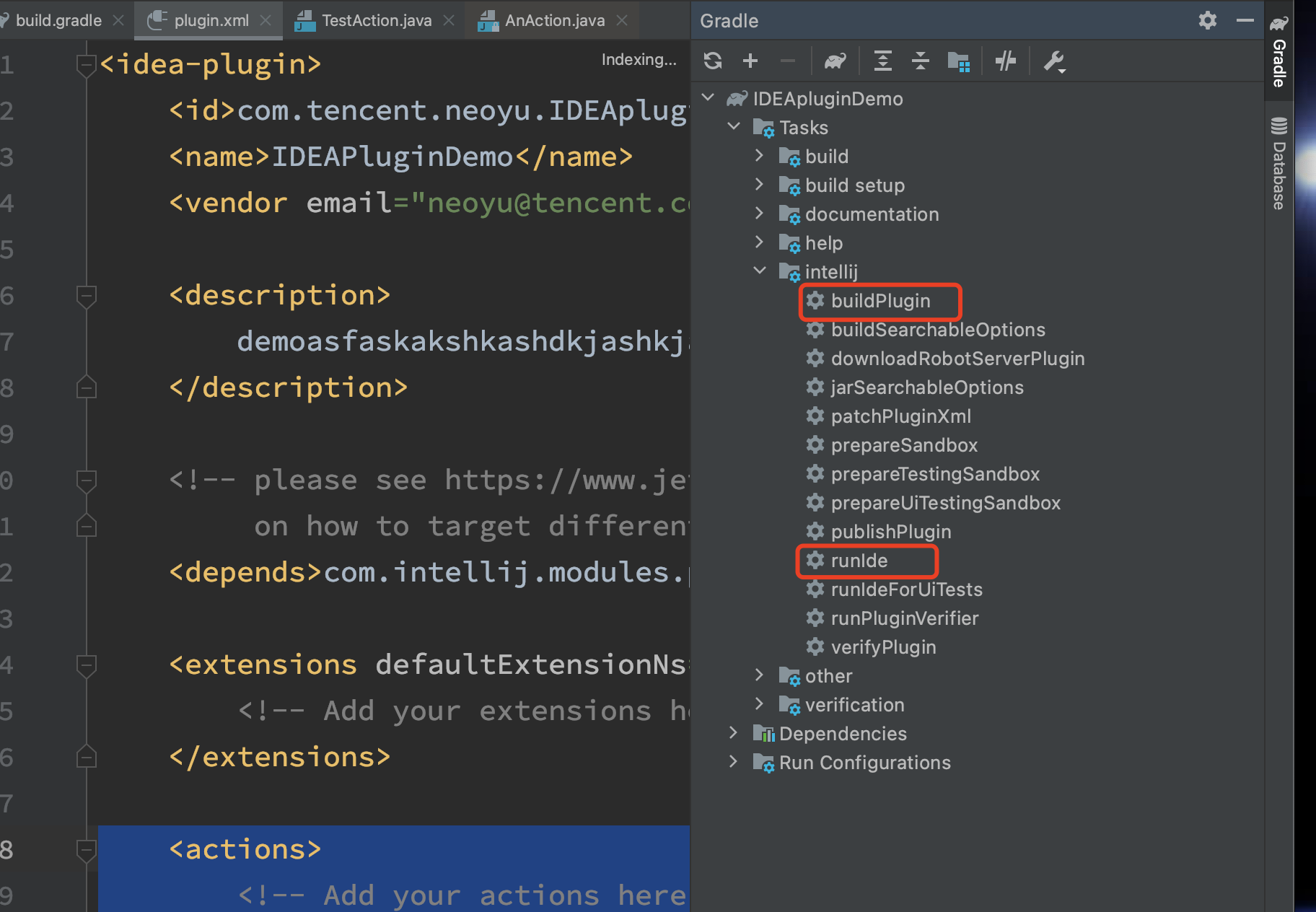
**Step 3:**

Debugging & packaging

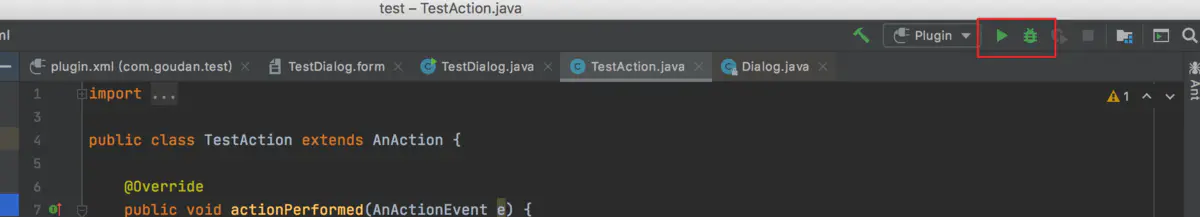
There are two ways can debugging and packaging.

1st: Gradle

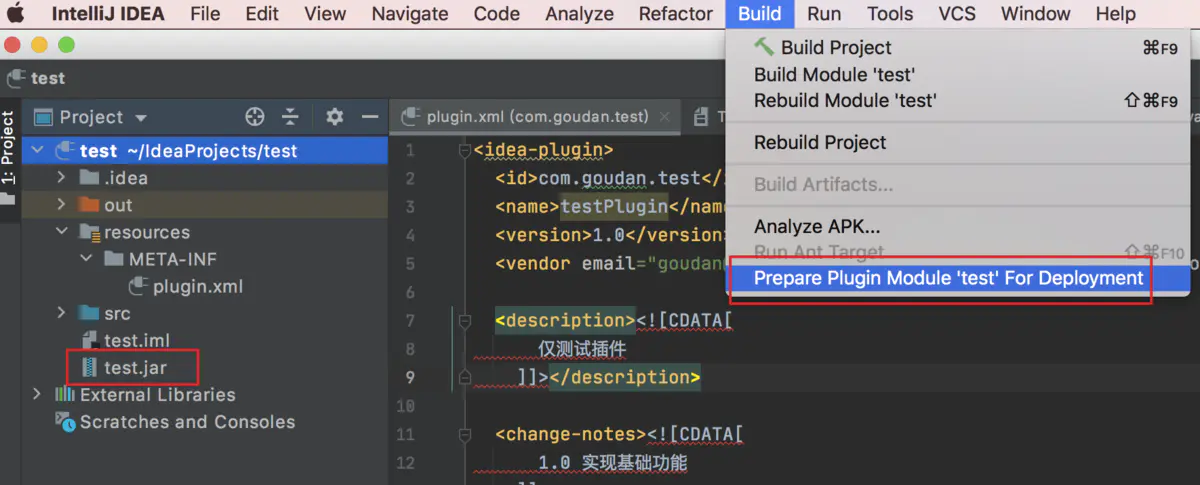
runIde is debugging, buildPlugin is packaging. Plugin build in build/distributions/ of a zip package.



2nd: Run directly or step debug



In the Build menu, after you click “Prepare Plugin Module \*\*\* For Deployment”, the jar package will be generated in the project directory on the left side. The jar package name is the project name.



**Step 4**

Testing in android studio

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**插件开发教程（中文）**

<https://www.cnblogs.com/yuxiuyan/p/14682460.html>

<https://www.jianshu.com/p/cb3e7c7d694b>

作者：狗蛋W

链接：https://www.jianshu.com/p/cb3e7c7d694b

来源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

1. **Profile(plugin.xml).**

<id> ：该项目的唯一id，当该项目被上传到应用商店后，这个id将用来区分不同插件，如果尝试上传相同id的插件，应用商店就会提示错误。

<name> ：插件名字，如果导出的插件安装到android studio上，会是已安装插件的插件名。

<version> ：插件版本号，升级使用

<vendor>：作者信息，其中作者名字会显示在已安装插件信息页面

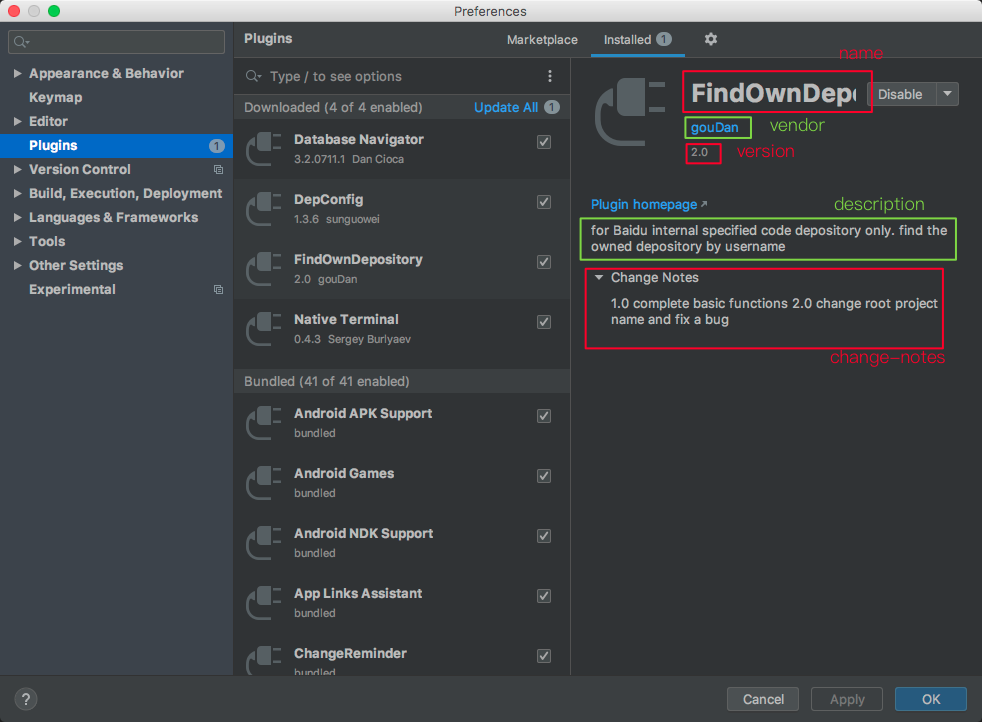
<description> ：对插件的描述，较为重要，用户对插件的初印象就是该描述的内容，描述是否清晰可能直接影响到用户对插件功能的理解，直接影响到用户是否安装茶几。IntelliJ建议至少超过40字符，过少会有文字下方红线提示

<change-notes>：这里用来写每个版本都做了哪些改动，同样会展示在插件信息页

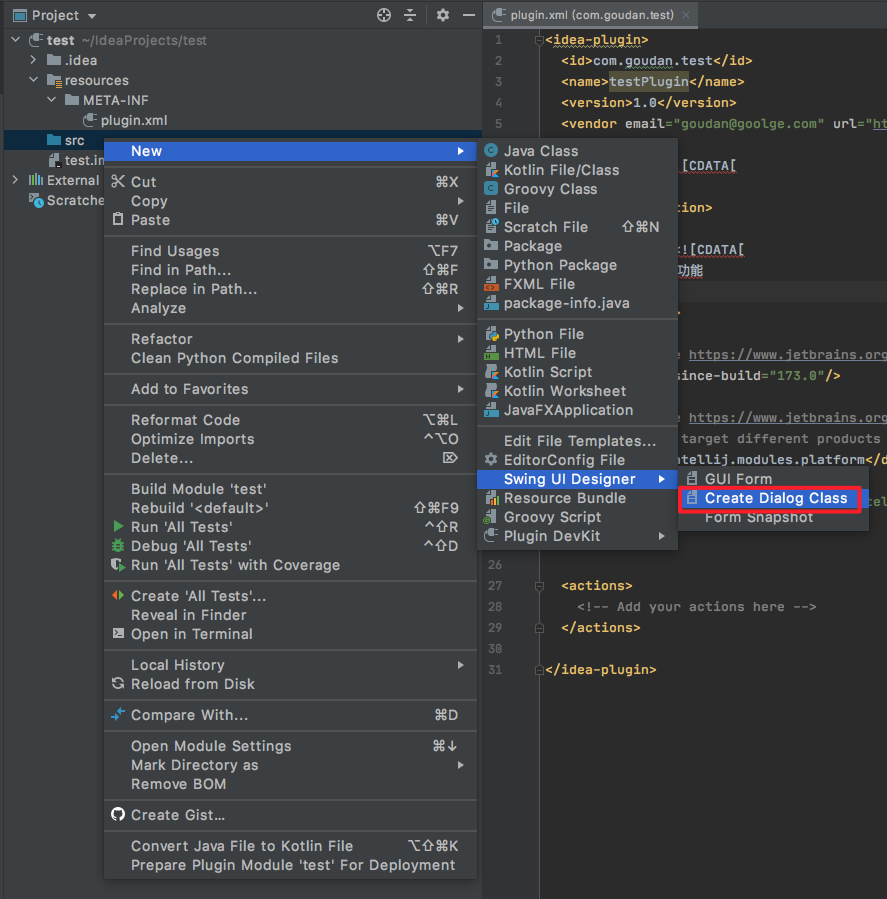
<idea-version> ：支持的idea版本，写一个较老的版本即可（比如173，代表17年3月的版本），如果这个版本过新，会导致老版本的IntelliJ IDEA无法使用该插件

<actions> ：这里存放一个或多个<action>节点，每个action节点是用户使用插件的入口，比如，可以在这里指定一个新建的AnAction类（该类作用是运行该插件功能的入口，比如弹出界面或执行后台逻辑），并指定用户可见的入口位置及入口名字，方便用户触发该插件，甚至可以配置键盘快捷键，用户可通过快捷键迅速启动插件。具体action的用法在下面进行示例讲解。

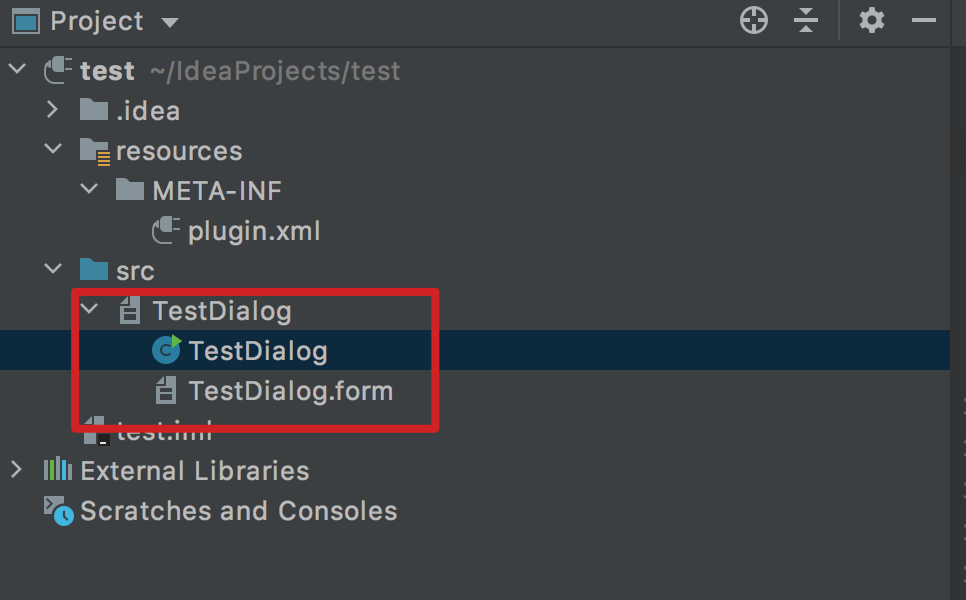
如下是android studio Preferences->Plugins页面，plugin.xml中相应的字段显示位置已经标出，便于大家理解



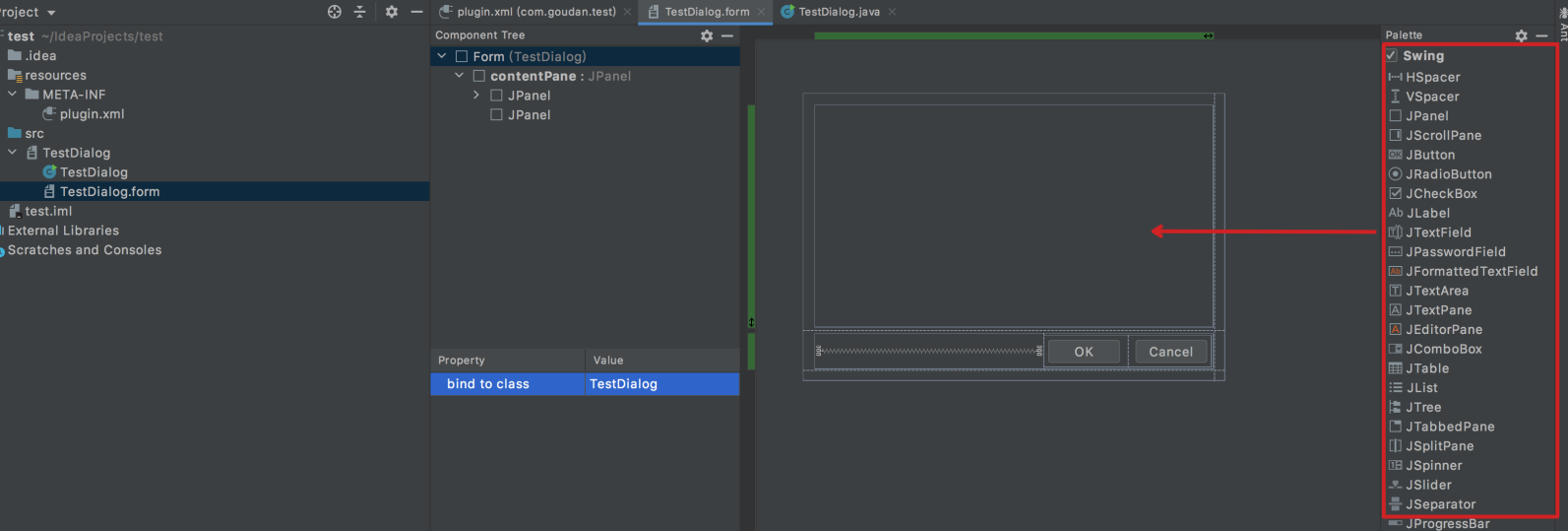
1. **Build a dialog(弹窗）**



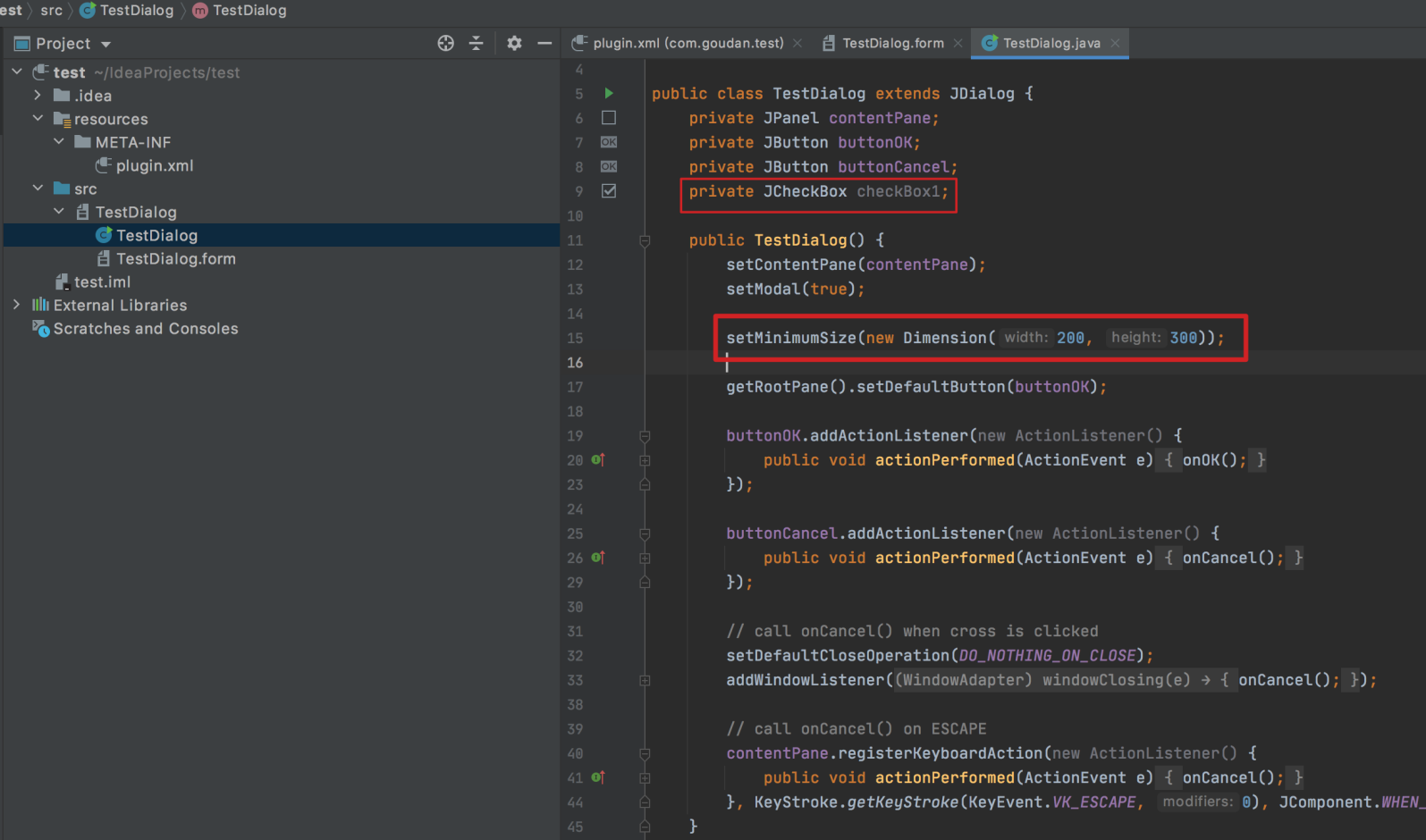
新建dialog类后，会自动的生成一个java类和一个.form文件



我们选中.form文件，可以通过拖动控件的方式修改布局，实现简单的布局效果

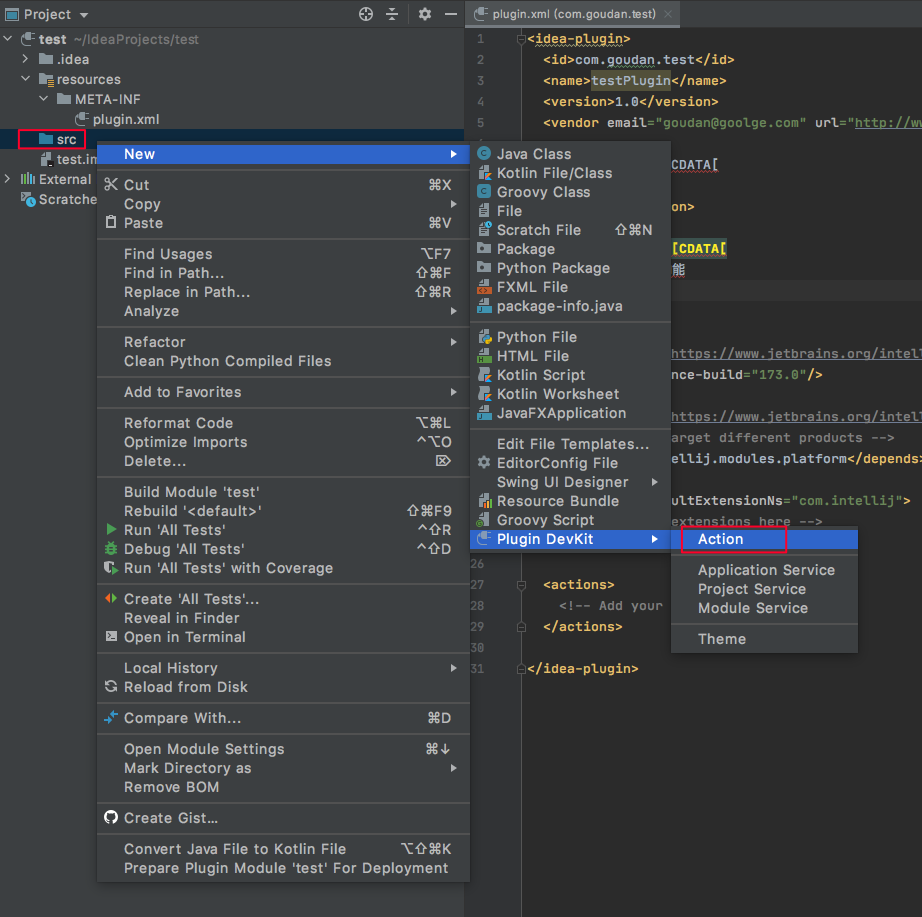


自动生成的弹窗java类文件里也贴心的帮我们设置好了点击事件、甚至我们新拖进的控件也帮我们声明好了。但是有一点还是需要注意，自动生成的类文件中没有设置dialog宽高，直接运行会发现这个dialog十分小，因此这里我们加入简单的宽高限制setMinimumSize(new Dimension(200, 300));



1. **Create an action**

相对于一个字母一个字母的手敲创建，更easy的做法是使用IntelliJ自带的Plugin DevKit生成action。注意，只有鼠标选择了项目的src或其文件夹下的文件时，才能出现Action的新建选项，如下所示



在接下来的弹出窗口中，我们可以创建一个Action。

Action ID：这个action的唯一标识

Class Name：action的类名

Name：action的名称

Description: action的描述信息

Groups：这个标签指定我们自定义的插件应该放入到哪个菜单下面。 在IntelliJ IDEA菜单栏中有很多菜单如File、Edit、View、Navigate、Code、……、Help等。他们的ID一般是菜单名+Menu的方式。比如，我们想将我们自定义的插件放到Help菜单中，作为Help菜单的子选项。那么在Groups中就指定HelpMenu。左侧的Anchor属性用于描述位置，主要有四个选项：first、last、before、after。他们的含义如下：

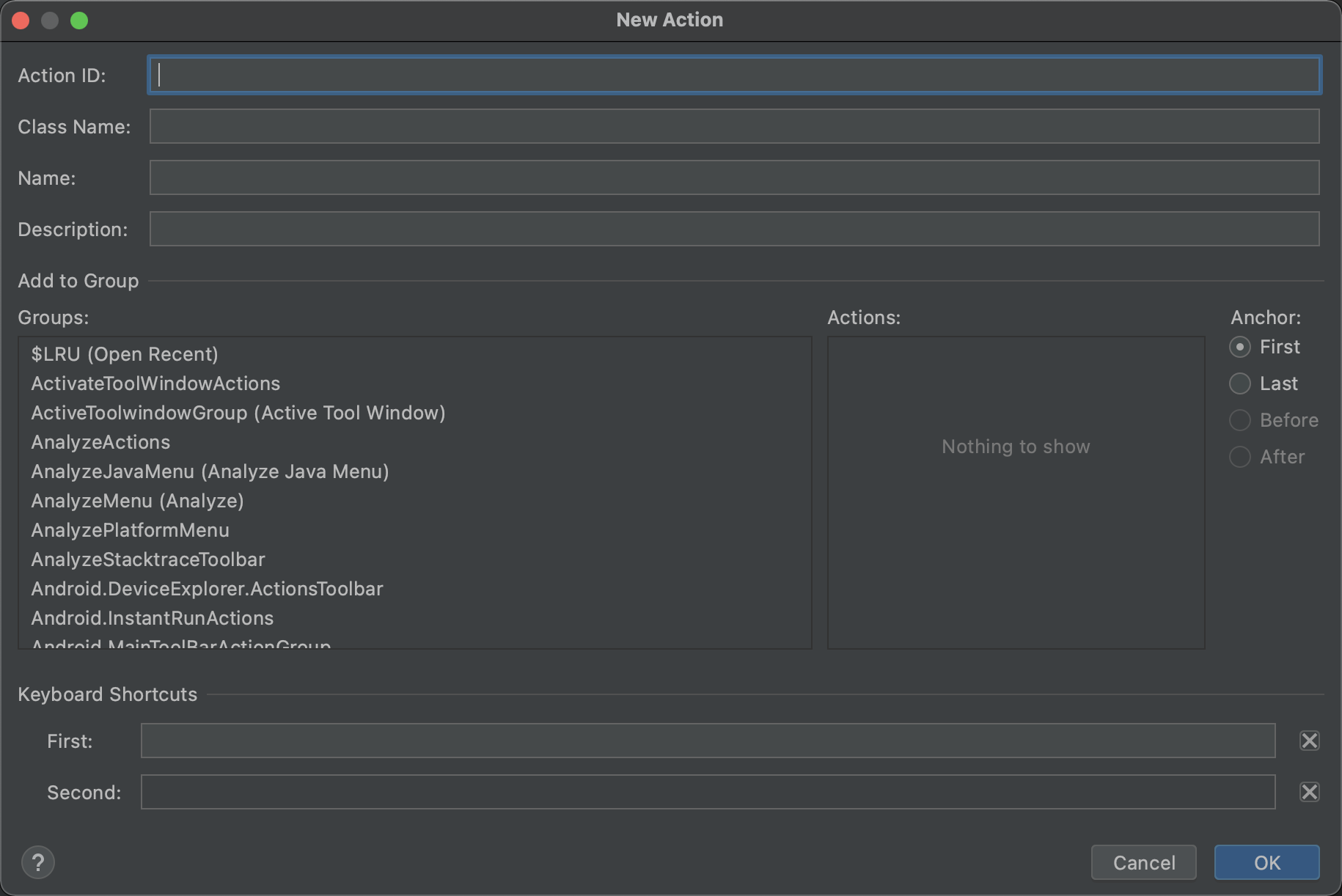
first：放在最前面

last：放在最后

before：放在relative-to-action属性指定的ID的前面

after：放在relative-to-action属性指定的ID的后面

Keyboard Shortcuts：可以为这个action指定快捷键



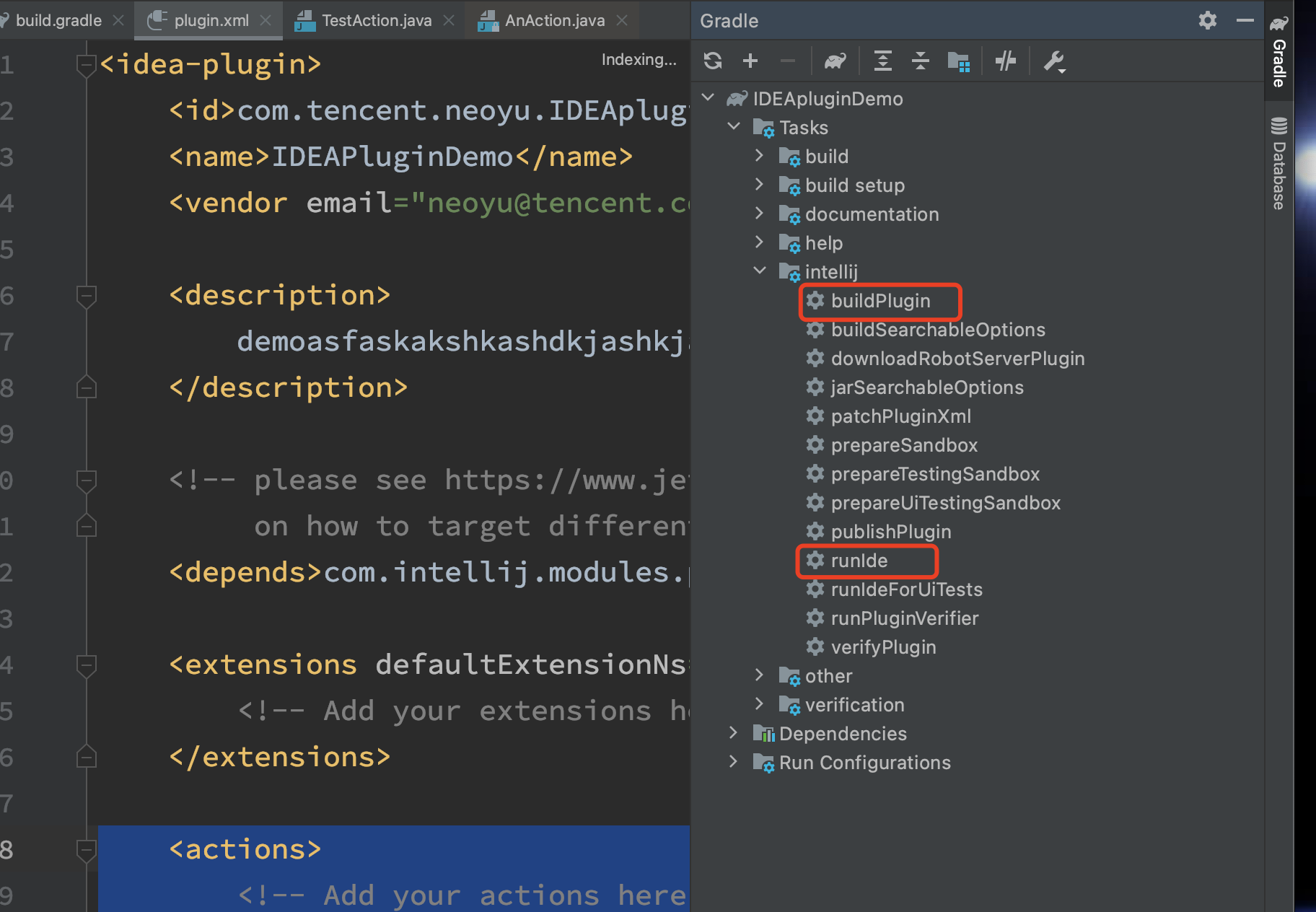
创建完之后，我们也可以在src/resources/META-INF/plugin.xml中，看到我们之前写入的action信息，如果想要修改，可以在这个配置文件中直接修改。



1. **Debugging & Packaging**

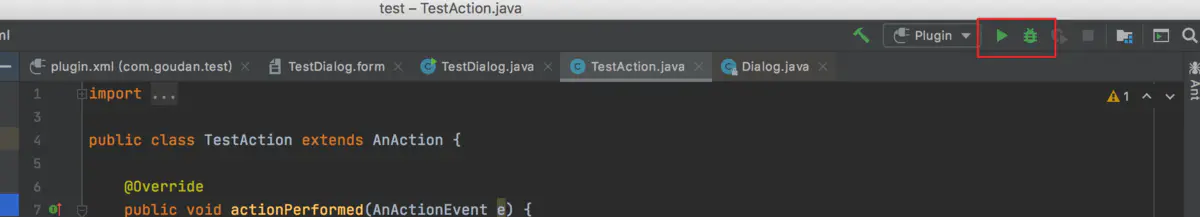
There are two ways can debugging and packaging.

1st: 等到配置完成后，在IDEA右侧的Gradle一栏中，有Intellij的集合。点击里面的runIde,可以打开一个沙盒，里面运行包含着该插件的IDEA实例。也可以右键选择debug模式运行。

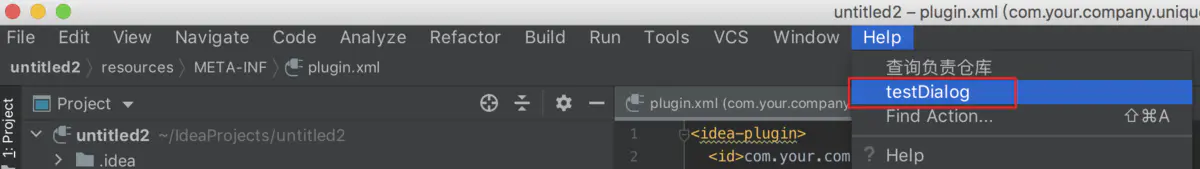


点击上图的buildPlugin，就可以在build/distributions/目录下面生成插件zip包，这个包就是我们需要的最终产物。在IDEA设置Preferences --> Plugins，点击installed旁边的设置按钮，选择Install Plugin from Disk,然后选择这个zip，就可以安装到IDEA中了。

2nd: 像android studio一样，我们可以直接运行或单步调试(如红框所示)

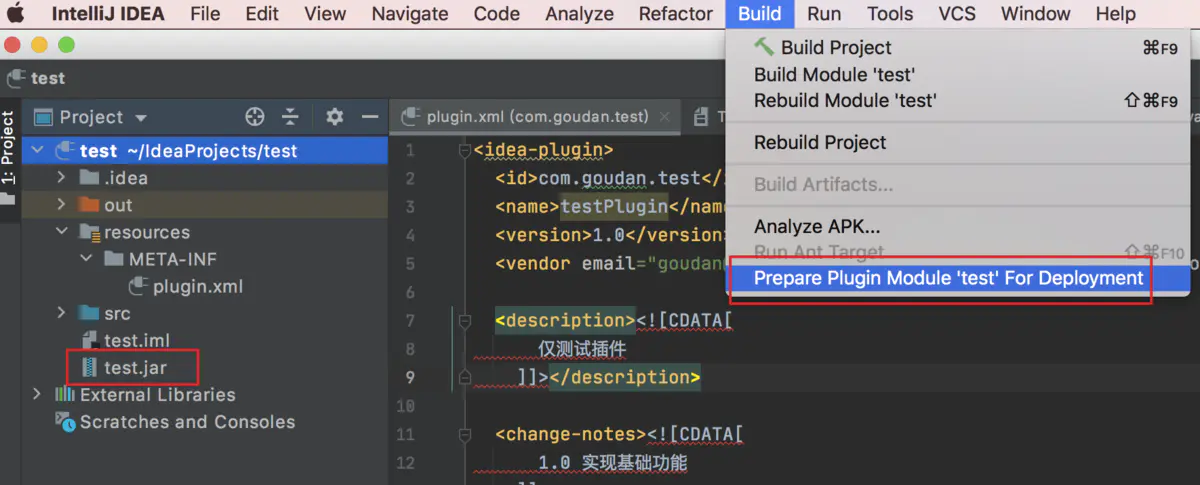


点击运行后，会弹出新的IntelliJ IDEA窗口，我们随便新建或打开一个项目，待项目index完毕后，可以在菜单栏内发现我们刚写的插件入口，点击后就会弹出自定义弹窗。



因为最终插件是运行在android studio而非IntelliJ IDEA，而且真正插件运行环境可能比自己测试的demo更复杂，会导致实际应用中会出现各种bug，因此不能仅仅满足于在intelliJ IDEA中通过debug进行测试，需要导出插件包，放到真实的环境中进行测试，但这时候的测试就无法进行单步调试了，有问题就不太容易发现，可以通过在代码一些主要逻辑节点上进行弹窗显示核心信息的方式排查问题代码位置，就像打log一样。

插件打包入口在Build菜单中，点击Prepare Plugin Module \*\*\* For Deployment后，左侧项目目录下会生成相应的jar包，jar包名字就是项目名。



android studio中可在Preference->Plugin的如下入口处选择安装本地插件，进行实际环境的测试

