# DevEco插件开发环境搭建（window为例）

## 开发环境搭建

### 1.1安装IntelliJ IDEA

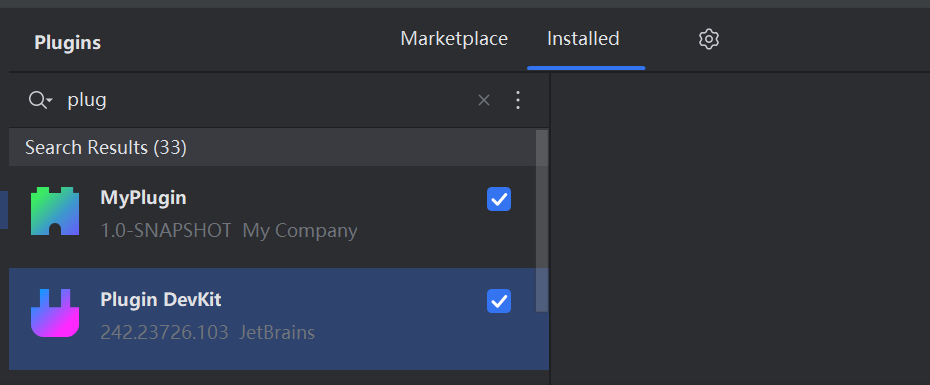
<https://www.jetbrains.com/zh-cn/idea/download>

官网下载最新IntelliJ IDEA适合自己的版本并安装

### 1.2安装plugin开发插件

File->Settings->Plugins

搜索Plugin DevKit插件并点击Install进行安装，安装后在插件列表查看确认为Enable状态



### 1.3安装Command Line Tools

如果插件功能涉及DevEco编译打包，需使用华为账号登录

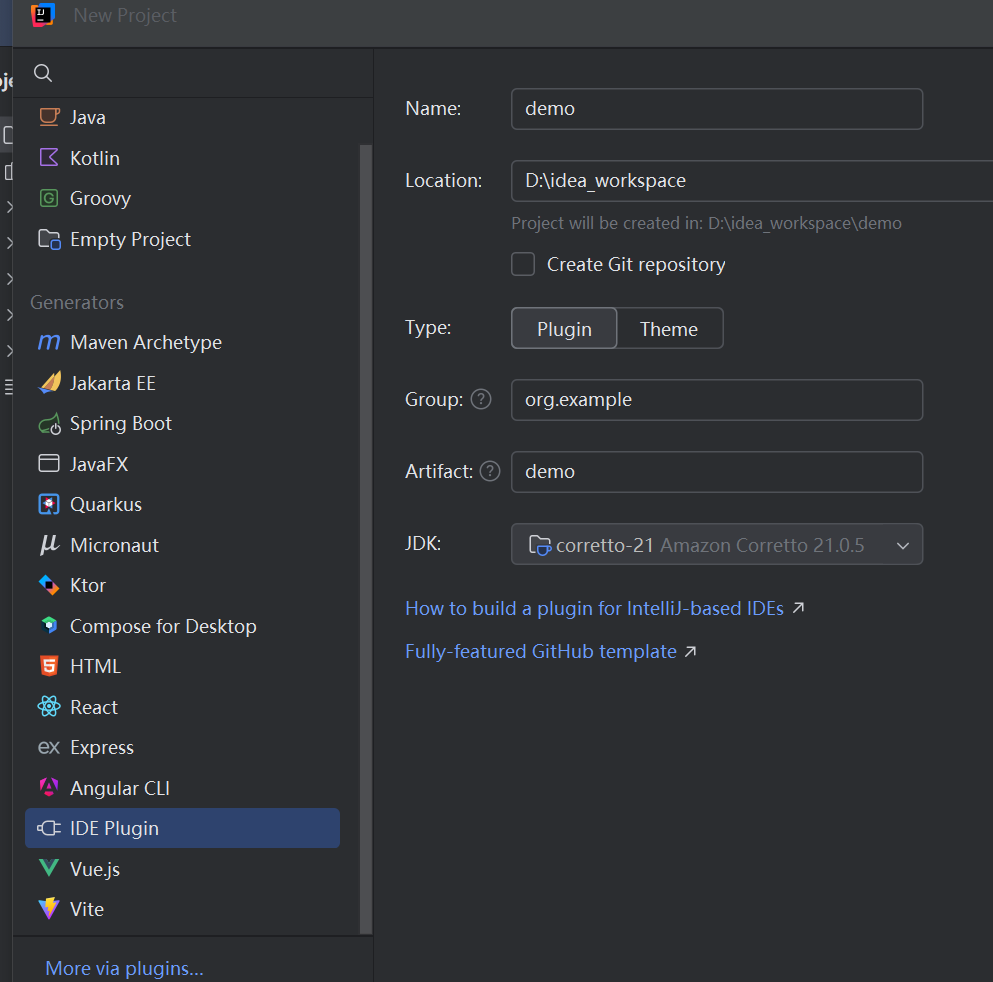
<https://developer.huawei.com/consumer/cn/download/>

官网下载Command Line Tools for HarmonyOS 5.0.0 Release并解压复制到安装路径下，例如D:\program\_toos\，注意路径中最好不要有中文及空格。复制完成后修改系统环境变量，增加path值D:\program\_toos\command-line-tools\bin

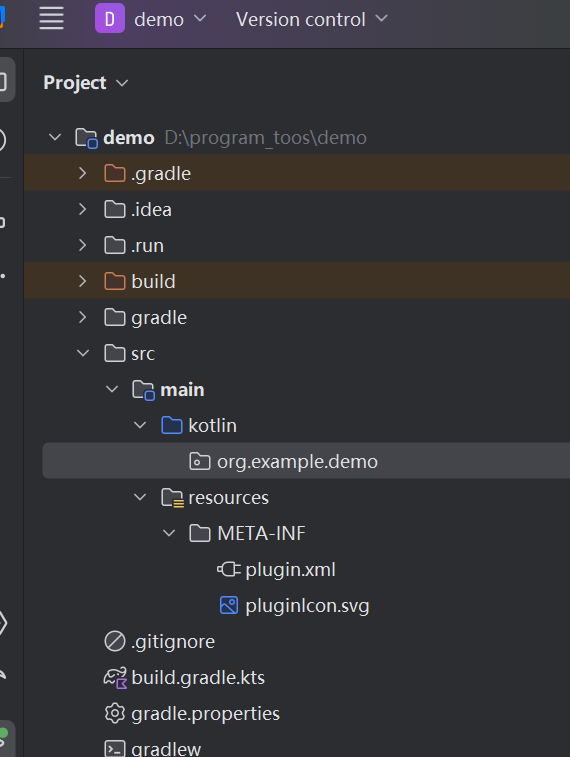
## 开发过程

### 2.1新建插件开发工程

Flie->New->Project新建工程类型选择 IDE Plugin。Type选择Plugin。选择工程存储路径，设置包名及项目名。选择JDK，如没有可在下拉选项里进行下载安装。



### 2.2项目文件介绍



默认生成的项目目录结构如上图所示。其中plugin.xml为插件配置文件，pluginIcon.svg为插件项目图标。org.example.demo为代码根目录。项目默认开发语言为kotlin。如需改为java语言开发，只需删除生成的kotlin文件夹并在main文件夹下新建java文件夹及代码包。

### 2.3HelloWorld开发

如项目使用kotlin进行开发，在HelloAction.kt中编辑如下代码

package org.example.demo  
  
import com.intellij.openapi.actionSystem.AnAction  
import com.intellij.openapi.actionSystem.AnActionEvent  
import com.intellij.openapi.ui.Messages  
  
class HelloAction : AnAction() {  
 override fun actionPerformed(e: AnActionEvent) {  
 Messages.showMessageDialog(e.*project*, "Hello world", "Alert", Messages.getInformationIcon());  
 }  
}

如项目使用java进行开发，在HelloAction.java中编辑如下代码

package org.example.demo;  
  
import com.intellij.openapi.actionSystem.AnAction;  
import com.intellij.openapi.actionSystem.AnActionEvent;  
import com.intellij.openapi.ui.Messages;  
import org.jetbrains.annotations.NotNull;  
  
public class HelloAction extends AnAction {  
 @Override  
 public void actionPerformed(@NotNull AnActionEvent e) {  
 Messages.*showMessageDialog*(e.getProject(), "Hello world",  
 "Alert",  
 Messages.*getInformationIcon*());  
 }  
}

actionPerformed为我们的工具按钮点击后的相应入口函数，Messages.showMessageDialog方法用来弹出一个写着HelloWorld的弹窗。

修改plugin.xml如下

<idea-plugin>  
 <id>com.example.plugin</id>  
 <name>MyPlugin</name>  
 <version>1.0.0</version>  
 <vendor email="developer@example.com" url="https://example.com">My Company</vendor>  
  
 <actions>  
 <action id="HelloAction"  
 class="org.example.demo.HelloAction"  
 text="Hello World Demo"  
 icon="com.intellij.icons.ExpUiIcons.Build.Build"  
 description="Test plugin">  
 <add-to-group group-id="ToolbarRunGroup" anchor="first"/>  
 </action>  
 </actions>

<depends>com.intellij.modules.platform</depends>  
</idea-plugin>

上述配置中，class="org.example.demo.HelloAction"用来指示我们的按钮响应类

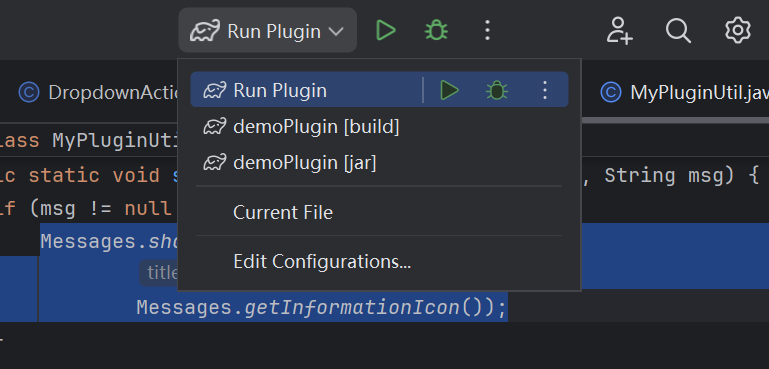
<add-to-group group-id="ToolbarRunGroup" anchor="first"/>一句配置表明我们的按钮将添加在工具栏的运行栏靠前位置。

com.intellij.icons.ExpUiIcons.Build.Build就是我们的按钮图标。

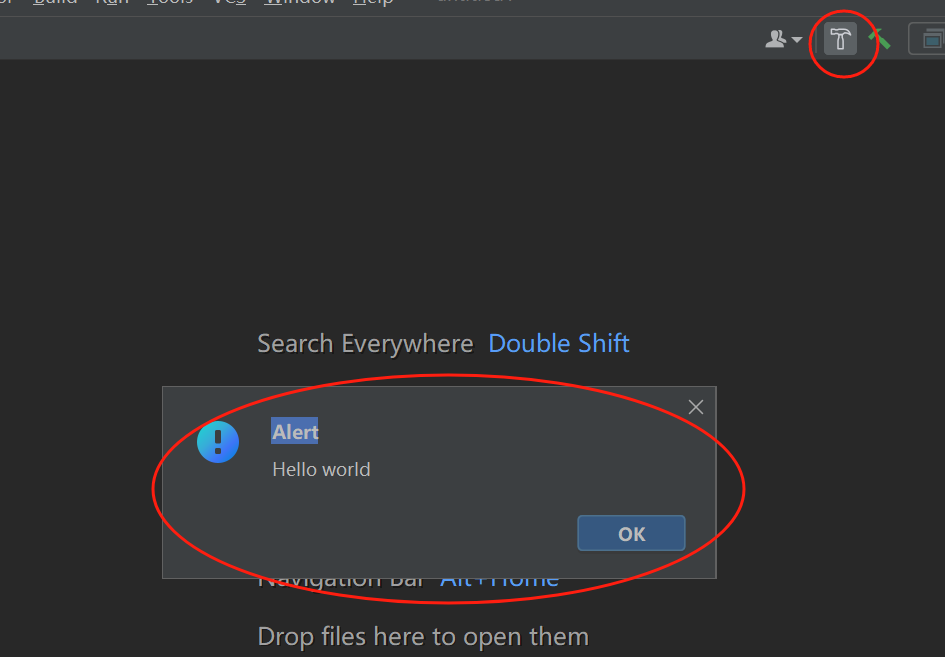
如需打包到IDEA以外的平台使用则<depends>com.intellij.modules.platform</depends>一句不能省略。

## 测试运行

### 3.1直接运行

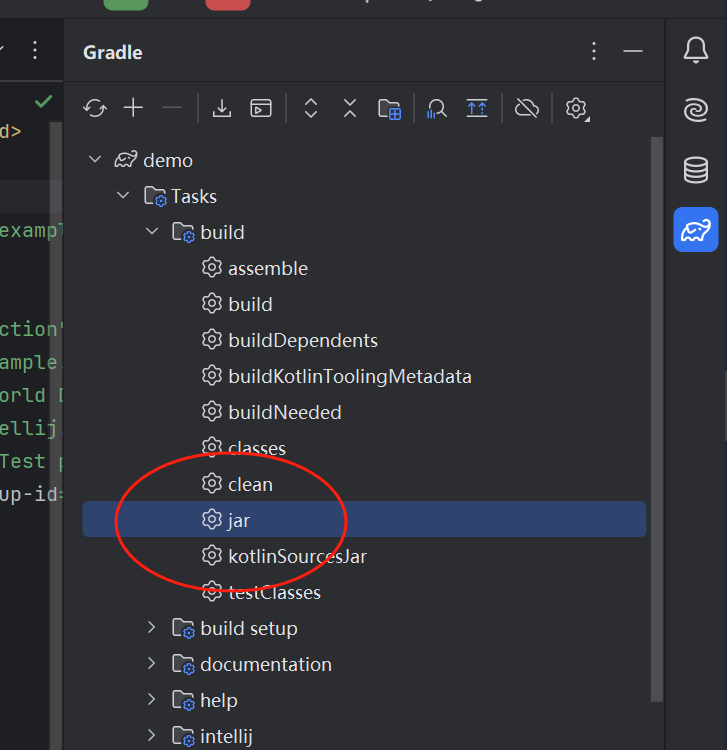


点击运行按钮，第一次运行将弹出一个新的Intellji IDEA项目窗口，选择新建个项目并打开，我们的插件将在新的项目窗口中被加载，可以看到工具栏中已显示出我们的工具图标。点击效果如下。

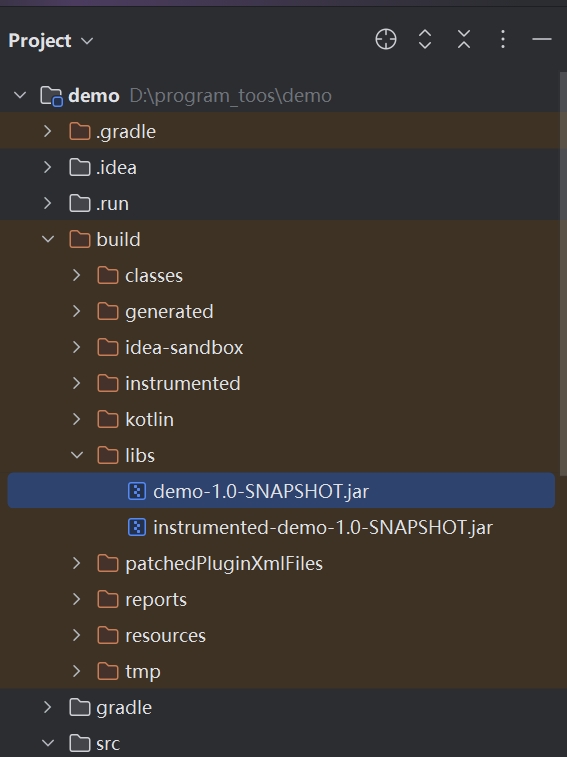


### 3.2安装运行

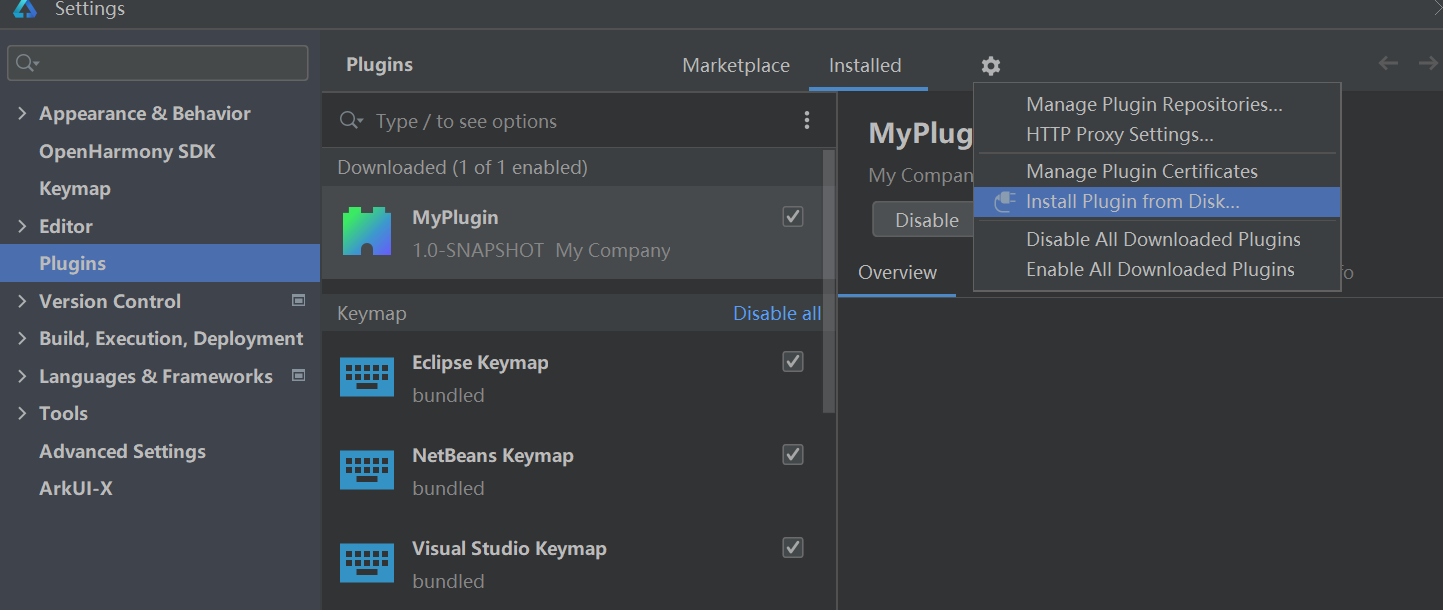
在IDEA中打jar包。



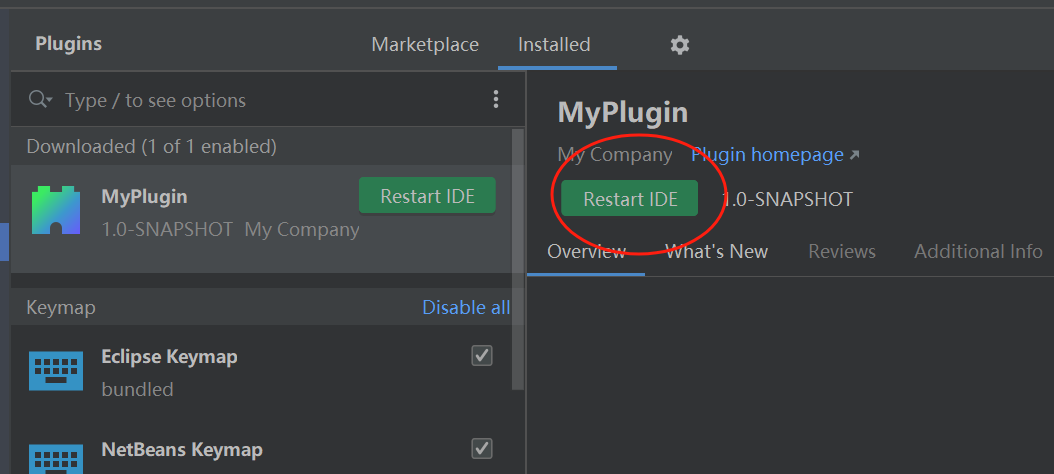
找到打好的jar包，右键open in->explorer，拷出到某路径下。



打开DevEco，File->Setting->plugin->Install Plugin from Disk，选择刚刚到处的jar包



选中插件安装完成后，点击Restart IDE

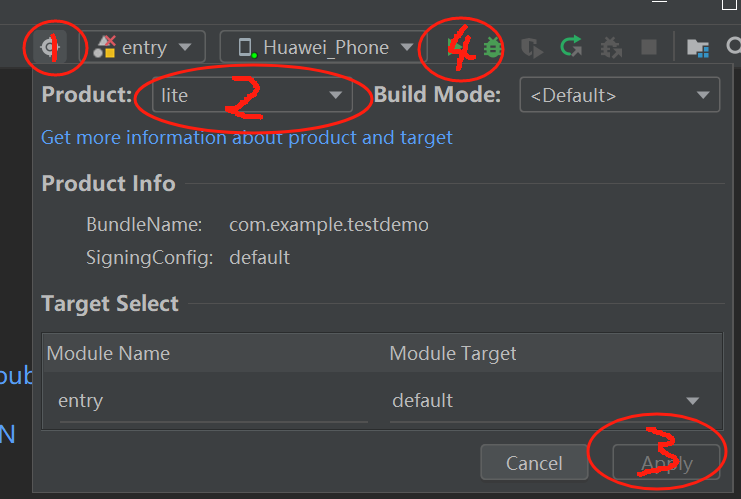


重启后即可在工具栏看到按钮图标，运行效果与IDEA中相同。

至此，已完成windows系统中开发DevEco开发工具插件的所有步骤。

# 分产品打包工具插件的开发

## 需求介绍

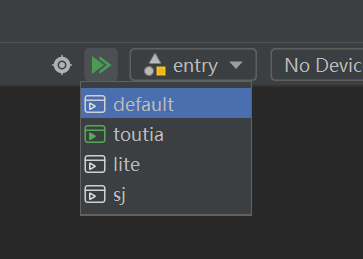
在鸿蒙开发中，可能遇到一个项目同时存在多条产品线的情况，此时对单个产品线进行打包过程中需要进行下图所示四个点击操作步骤，如果在产品线较多情况下，显得较为繁琐。另一个问题是这样操作可能同时对多个产品并行打包，打包过程存在一定安全或管理上的问题。

需求提出希望能开发一个插件，提供一个工具栏按钮的下拉列表，点击列表项即可调用hvigorw批处理命令直接进行打包，并且列表项中应能差异显示出正在打包的产品，同时只允许打包一个产品。

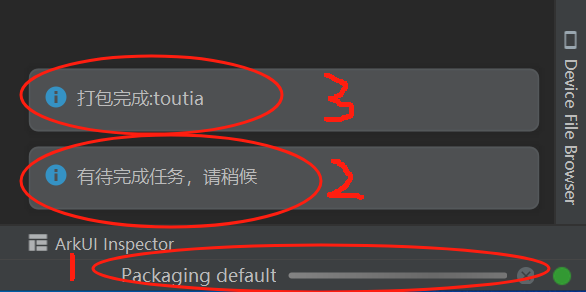
## 实现效果展示

经调研开发，最终实现如下效果：

如下图所示：点击工具栏上显示的双箭头图标，下方弹出下拉列表选项，点击某条产品名称，该产品前的图标绿色高亮并进入编译状态，下拉框自动收起。



当产品正在编译过程中，如圈1所示，下方状态栏显示Packaging进度条。此时如重新点击双箭头图标并在下拉框选择打包其他产品，如圈2所示，将弹出气泡提醒“有待完成任务，请稍候”。待打包任务完成后，如圈3所示，将弹出气泡提醒“打包完成”，此时可选择其它产品继续进行打包。



## 实现步骤

### 3.1基础xml配置文件

<idea-plugin>  
 <id>com.example.plugin</id>  
 <name>MyPlugin</name>  
 <version>1.0.0</version>  
 <vendor email="developer@example.com" url="https://example.com">

My Company</vendor>   
 <actions>  
 <action id="DropdownAction"  
 class="com.example.plugin.dropdown.DropdownAction"  
 text="选择产品进行打包"  
 icon="AllIcons.Actions.RunAll"  
 description="Show product dropdown list">  
 <add-to-group group-id="ToolbarRunGroup" anchor="first"/>  
 </action>  
 </actions>  
</idea-plugin>

定义一个id为DropdownAction的Action响应，将其绑定com.example.plugin.dropdown包下的DropdownAction类，icon选择一个适合的IDEA系统图标，类似的可用图标路径可按ctrl同时单击查看代码，并在ExternalLibrary下的com.jetbrains.idealC包中对应位置找到。

### 3.2实现下拉列表Action

点击按钮后需要通过工程路径下build-profile.json5文件解析出产品线列表后显示下拉列表。

先实现一个MyPluginUtil工具类用来放置相关的工具方法：

根据文件File读取并返回String的方法：

*/\*\*  
 \* 读取文件  
 \* @param file file文件  
 \* @return 文件内容字符换  
 \*/*public static String readStringFromFile(File file) {…}

根据传入工程，调用上面的readStringFromFile函数读取工程下的build-profile.json5文件并解析出产品列表的方法：

*/\*\*  
 \* 根据传入的Project工程读取产品列表  
 \* @param project project内容  
 \*/*public static ArrayList<ProductBean> getProductList(@Nullable Project project) {…}

用于弹出错误及异常消息弹框提示的方法：

*/\*\*  
 \* 显示弹出消息框  
 \* @param project project内容  
 \* @param msg msg内容  
 \*/*public static void showAlert(@NotNull Project project, String msg) {  
 if (msg != null && !msg.trim().isEmpty()) {  
 Messages.*showMessageDialog*(project, msg,  
 "提示：",  
 Messages.*getInformationIcon*());  
 }  
}

新建下拉列表项的实体beanDropdownItem，其中包含两个属性：String text用来进行产品名称显示，boolean isRunning用来记录该项产品是否在编译中；

新建DropdownPopup类，实现用于渲染下拉列表的DropdownItemRenderer静态内部类，该Render复写其父类Render的getListCellRendererComponent方法，通过setText、setIcon设置列表项显示，根据列表项所表示的产品是否在编译中决定显示不同路径下的icon图标。图标可使用IDEA自带图标，也可在resources下新建icons文件夹，并将图片文件拷入。需注意图片格式及大小，建议使用svg图片以适应IDEA环境配置的显示样式。

super.getListCellRendererComponent(list, value, index, isSelected, cellHasFocus);  
if (value instanceof DropdownItem item) {  
 setText(item.getText());if (item.isRunning()) {  
 setIcon(IconLoader.*getIcon*("/icons/item\_running.svg", DropdownPopup.class));  
 } else {  
 setIcon(IconLoader.*getIcon*("/icons/item.svg", DropdownPopup.class));  
 }  
}  
return this;

新建showPopup()方法并通过JBPopupFactory的createPopupChooserBuilder()创建Builder，传入下拉列表数据源List<DropdownItem>，因为后续还要获得及更改列表项编译中状态，新建一个DropDownItemDataSingleton单例来保存数据源对象。

builder.setMovable(false)设置弹框不可拖动；

.setRenderer(new DropdownItemRenderer())设置渲染器为上面实现的下拉列表渲染器；

setItemChosenCallback（）设置下拉列表项点击回调事件；

通过builder的createPopup（）创建JBPopup弹框；

调用popup.show()方法显示下拉列表弹框。

在Action中，后台任务执行文件列表加载解析，如果解析成功，调用DropdownPopup类中实现好的方法弹出下拉列表。需要注意对UI的操作需要在UI线程中，如果在后台任务代码中，应通过ApplicationManager.getApplication().invokeLater切换到UI线程执行

ProgressManager.*getInstance*().run(new Task.Backgroundable(e.getProject(), "Loading product list") {  
 @Override  
 public void run(@NotNull ProgressIndicator indicator) {  
 ArrayList<ProductBean> productList = MyPluginUtil.*getProductList*(e.getProject());  
 if (productList == null || productList.isEmpty()) {  
 ApplicationManager.*getApplication*().invokeLater(() -> {  
 MyPluginUtil.*showAlert*(Objects.*requireNonNull*(e.getProject()), "产品列表解析失败，请检查您的配置文件");  
 });  
 } else {  
 //产品配置文件读取成功切回前台显示下拉列表框  
 ApplicationManager.*getApplication*().invokeLater(() -> {  
 DropdownPopup popup = new DropdownPopup();  
 popup.showPopup(invoker, e);  
 });  
 }  
 }  
});

### 3.3实现下拉列表项的点击回调

if (selectedItem != null && DropDownItemDataSingleton.getInstance().hasNoRunningItem()) 判断是否有正在编译的产品，如果有则不进行新的编译操作，否则执行以下逻辑：

selectedItem.setRunning(true);将选中项设为运行状态；

新开一个后台任务ProgressManager.getInstance().run(new Task.Backgroundable(){…});

调用MyPluginUtil工具类中新建的executeBat()方法，传入盘符及工程路径，该方法将通过Process process = Runtime.getRuntime().exec(batFilePath)实现对工程路径根目录下.bat文件的调用，其中batFilePath参数为.bat文件所在路径及bat文件所需三个参数盘符（%1），工程路径（%2），产品线名称（%3）用空格拼接的字符串；

用process.waitFor()等待命令执行完成后，调用onBatExecutedListener.onBatExecuted回调函数，在回调方法中弹出任务完成提示并将通过selectedItem.setRunning(false)将选中项设为非运行状态。

其中.bat文件内容如下，功能为切换到工程所在路径后执行hvigorw打包命令

%1  
cd %2  
hvigorw assembleHap --mode module -p product=%3 assembleHap --analyze=normal --parallel --incremental --daemon

### 3.4实现弹出气泡

因弹出对话框需要手动关闭，影响体验，可改为通过弹出气泡进行消息提示，步骤如下：

1、在plugin.xml的<idea-plugin>节点下添加以下代码用于声明一个气泡弹出通知组。

<extensions defaultExtensionNs="com.intellij">  
 <notificationGroup id="MyPackageNotify" displayType="BALLOON"/>  
</extensions>

2、在MyPluginUtil.java中添加弹出气泡方法，注意"MyPackageNotify"参数要和plugin.xml中刚刚添加的保持一致

*/\*\*  
 \* 弹出消息气泡  
 \* @param project project内容  
 \* @param msg msg内容  
 \*/*public static void showNotification(@NotNull Project project, String msg) {  
 if (msg != null && !msg.trim().isEmpty()) {  
 Notification notification = new Notification("MyPackageNotify", msg, NotificationType.*INFORMATION*);  
 Notifications.Bus.*notify*(notification, project);  
 }

}

3、在需要弹出气泡的地方进行调用。需要注意显示气泡和弹框一样，需要执行在UI线程。

### 3.5导出Jar包到DevEco中运行

如果开发的插件需要在IDEA外的平台运行，所以需要在plugin.xml的<idea-plugin>节点下添加<depends>com.intellij.modules.platform</depends>。

在IDEA中将开发的插件打jar包，导入到DevEco中。

将test\_hvigorw.bat文件拷贝到要打包的工程根目录。

重启DevEco即可使用插件。

## 项目完整文件列表

项目中主要文件及其作用说明如下图所示：

