Apt详解(一)

Jason骑蜗牛看世界 关注

0.296 2017.05.17 16:10:07 字数 2,042 阅读 2,789

简介

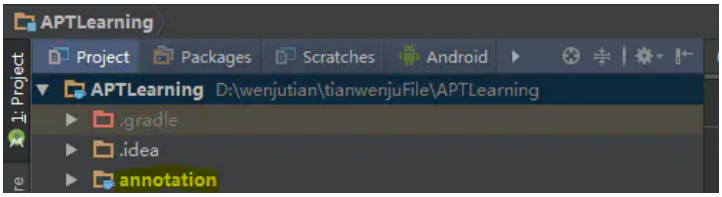
APT(Annotation Processing Tool)是一种处理注释的工具,它对源代码文件进行检测找出其中的Annotation，使用Annotation进行额外的处理。 Annotation处理器在处理Annotation时可以根据源文件中的Annotation生成额外的源文件和其它的文件(文件具体内容由Annotation处理器的编写者决定),APT还会编译生成的源文件和原来的源文件，将它们一起生成class文件。
这里面有几个关键字：处理注解，编译生成，我们总结一句话就是APT能够在编译期通过处理注解，生成我们想要的文件。

注解处理器是 javac 自带的一个工具，用来在编译时期扫描处理注解信息。你可以为某些注解注册自己的注解处理器。这里，我假设你已经了解什么是注解及如何自定义注解。如果你还未了解注解的话，可以查看官方文档。注解处理器在 Java 5 的时候就已经存在了，但直到 Java 6（发布于2006看十二月）的时候才有可用的API。过了一段时间java的使用者们才意识到注解处理器的强大。所以最近几年它才开始流行。

一个特定注解的处理器以 java 源代码（或者已编译的字节码）作为输入，然后生成一些文件（通常是.java文件）作为输出。那意味着什么呢？你可以生成 java 代码！这些 java 代码在生成的.java文件中。因此你不能改变已经存在的java类，例如添加一个方法。这些生成的 java 文件跟其他手动编写的 java 源代码一样，将会被 javac 编译

用法

工程目录结构如下：



图片.png

annotation 是我们存放注解的地方。
compiler 使我们处理注解的地方。
api 是我们提供对外调用的api。
现在有这个需求只要在我们的类上加了@Test的注解就会生成

```
package com.delta.aptlearning;
import java.lang.String;
import java.lang.System;
public class MainActivity$$helloWorld {
  public static void main(String[] args) {
    System.out.println("app");
  }
}
```

1. 创建注解libary annotation里面注解如下

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.CLASS)
public @interface Test {
}
```

我们@target是Type表明该注解只能注解类获接口

2. 创建compiler libary，因为我们的AbstractProcesso这个类是属于java不需要android一些插件所以我们可以创建javalibary。buildgradle如下：

```
apply plugin: 'java'
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    compile 'com.google.auto.service:auto-service:1.0-rc2'
    compile 'com.squareup:javapoet:1.7.0'
    compile project(':annotation')
}
sourceCompatibility = "1.7"
targetCompatibility = "1.7"
```

- 定义编译版本jdk为1.7
- AutoService主要的作用是注解processor类,并对其生成META—INF的配置信息(这里可以不用这个，也可以按照原始方式进行注解)
- javapoet主要的作用是帮助我们通过类调用的形式生成代码（也可以用string类型的方式进行拼接)
- annotaion是我们要依赖的使用的注解库

。

3. 处理注解。这时候我们要用到abstractProcessor类,我们看下api

```
@Override
    public Set<String> getSupportedAnnotationTypes() {
        return super.getSupportedAnnotationTypes();
    }

    @Override
    public SourceVersion getSupportedSourceVersion() {
        return super.getSupportedSourceVersion();
    }

    @Override
    public synchronized void init(ProcessingEnvironment processingEnvironment) {
        super.init(processingEnvironment);
    }

    @Override
    public boolean process(Set<? extends TypeElement> set, RoundEnvironment roundEnvironment)
        return false;
    }
```

- init(ProcessingEnvironment processingEnv)： 所有的注解处理器类都必须有一个无参构造函数。然而，有一个特殊的方法init()，它会被注解处理工具调用，以ProcessingEnvironment作为参数。ProcessingEnvironment 提供了一些实用的工具类Elements, Types和Filer。我们在后面将会使用到它们。

- process(Set<? extends TypeElement> annoations, RoundEnvironment env)： 这类似于每个处理器的main()方法。你可以在这个方法里面编码实现扫描，处理注解，生成 java 文件。使用RoundEnvironment 参数，你可以查询被特定注解标注的元素（原文：you can query for elements annotated with a certain annotation ）。

- getSupportedAnnotationTypes()：在这个方法里面你必须指定哪些注解应该被注解处理器注册。注意，它的返回值是一个String集合，包含了你的注解处理器想要处理的注解类型的全称。换句话说，你在这里定义你的注解处理器要处理哪些注解。注意这里也可以用注解的方式来实现

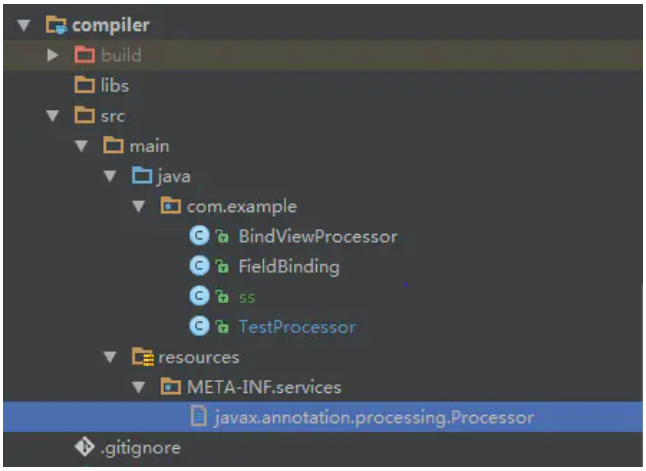
  eg: @SupportedAnnotationTypes("com.delta.annotationmodule.Test")

- etSupportedSourceVersion()： 用来指定你使用的 java 版本,注意此处也可以用注解的方式来实现
  eg:@SupportedSourceVersion(SourceVersion.RELEASE_6)

==这个内容会抽出一片文章详细介绍==

4. 注册注解
   你可能会问 “怎样注册我的注解处理器到 javac ？ ”。你必须提供一个.jar文件。就像其他 .jar 文件一样，你将你已经编译好的注解处理器打包到此文件中。并且，在你的 .jar 文件中，你必须打包一个特殊的文件javax.annotation.processing.Processor到META-INF/services目录下

   第一种方案



image

- 在main文件夹下创建resources文件夹
- 在resources资源文件夹下创建META-INF文件夹
- 然后在META-INF文件夹中创建services文件夹
- 然后在services文件夹下创建名为javax.annotation.processing.Processor的文件,在该文件中配置需要启用的注解处理器,即写上处理器的完整路径,有几个处理器就写几个,分行写么,比如我们这里是:com.example.TestProcessor

  第二种方案

- 在buildGradle文件中我们要加入 compile 'com.google.auto.service:auto-service:1.0-rc2'
- 在我们的注解处理器上加上

```
@AutoService(Processor.class)
public class TestProcessor extends AbstractProcessor
```

app用法

最终我们生成的注解需要在我们的android工程里面去应用怎么做呢?
这时候要用到Android-apt,那么问题来来了什么是android apt?
首先我们先看几个问题?

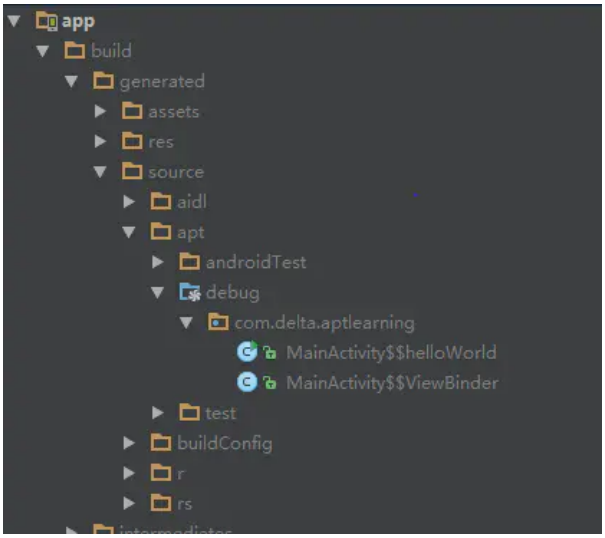
```
1. 首先注解处理器也就是我们的compile不应该打包到我们的apk中增加体积。
2. 生成的文件怎么被android studio引用
3. 怎么从buildgradle里向我们的注解处理器传递参数
```

Anroid-apt是用在Android Studio中处理注解处理的插件。他的作用如下

1. 只允许配置编译时注解处理器依赖，但在最终APK或者Library中不包含注解处理器的代码。

```
apt project(':compiler')
```

2. 这个插件可以自动的帮你为生成的代码创建目录,使注解处理器生成的代码能被Android Studio正确的引用，让生成的代码编译到APK里面去



图片.png

3. 从buildGradle里向我们的注解处理器传递参数

```
apt{
    arguments{
        module "app"
    }
}
```

处理器接收

```
Map<String, String> options = processingEnvironment.getOptions();
        Set<Map.Entry<String, String>> entries = options.entrySet();
        for (Map.Entry<String, String> entry : entries) {
            ss = entry.getValue() + ss;
            messager.printMessage(Diagnostic.Kind.NOTE, entry.getKey() + "----" + entry.getVal
        }
```

老版本的用法

整个工程的buildGradle你需要

```
 classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
```

而在你的app中需要这样

```
apply plugin: 'com.neenbedankt.android-apt'
dependencies{
    compile project(':annotation')
```

```
    apt project(':compiler')
    compile project(':api')
    }
```

到这里也许该松口气了，但是不好的消息来了android-apt插件作者近期已经发表声明表示后续不会再继续维护该插件。但也有个好消息是

Gradle从2.2版本开始支持annotationProcessor功能来代替Android-apt。另外，和android-apt只支持javac编译器相比，annotationProcessor同时支持javac和jack编译器

具体怎么用呢?
我们只需要在我们的app中这样加入

```
dependencies {
    compile project(':annotation')
    compile project(':api')
    annotationProcessor project(":compiler")
}
```

是不是很简单，如果你要传递参数你需要这样

```
defaultConfig {
        javaCompileOptions {
            annotationProcessorOptions {
                arguments = [ moduleName : project.getName() ]
            }
        }
    }
```

很方便调用但是有个前提==你的Gradle版本是2.2.X以上，就可以替换掉Android-apt。==

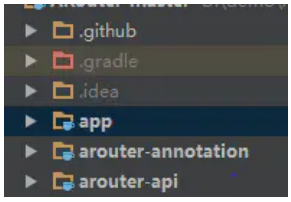
工程目录结构推荐

由于编译时注解处理器只在编译过程中使用，因此我们不希望注解处理器相关的代码在最终的APK中存在，这样能够有效的较少方法数。比如我通常在编写注解Annotation Processor的时候会引用javapoet和Guava，如果将这些代码也打进最终的APK中会造成方法数的暴增，因此建议将注解处理器相关代码单独成为一个模块。

另外为了方面注解被其他工程引用，通常我也建议将注解的定义单独划分成一个模块。

综上，我们最终的项目结构如下：

- xxx/xxx-api：主工程/提供api，Android Library类型
- xx-compiler:注解处理器模块，Java Library类型，打包apk时可以不要
- xxx-annotations：自定义注解，Java Library类型，打包apk时可以不要
  xxx/xxx-api依赖xxx-annotations,xxx-compiler依赖xxx-annotations。



图片.png