APT 简介

赵曈

一、APT 是什么

APT，即 Annotation Processing Tool 注解处理器。是由 JDK 自带的（1.6 及以上）、供用户编写实现自定义 Annotation（Java 注解）处理逻辑的一系列 API 及编译工具。

Annotation 相信大家不会陌生，在我们编程中，像@Override 这类就是一种 Annotation。在 Java 世界中，Annotation 有作用域（字段、函数、类）和作用时间（源码时、编译时、运行时）区别，在此就不详细展开了。一般来说，Annotation 都伴随着 Java 反射技术一同使用，可以达到用户使用简单的 Annotation 标注一个 Java Bean，通过简单调用，相关库在程序运行时将被标注的 Java Bean 转换为格式化数据。市面上有很多流行的库，比如说 Gson、FastJson、OrmLite 等等都使用了这种技术。

而本文主角 APT 与上面所说的技术有一点最大区别，即通过使用 APT，程序员可以在程序编译前，在每个 Java 类中检测符合自定义规则的元素，并且执行自定义的逻辑，例如自动生成新的 Java 类文件。APT 动作完成之后，程序再继续正常的编译行为。

二、APT 如何使用

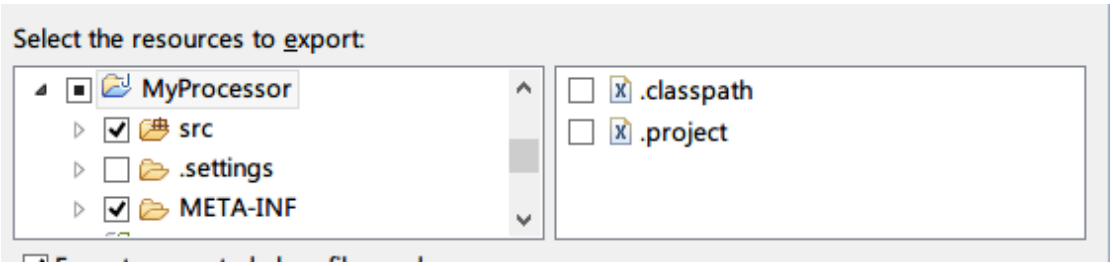
STEP 1.新建工程，新建一个 Java 类，就叫 MyProcessor 好了。MyProcessor 需要继承 javax.annotation.processing.AbstractProcessor，覆写一些方法，如图：

```
public class MyProcessor extends AbstractProcessor {

    @Override
    public boolean process(Set<? extends TypeElement> annotations,
            RoundEnvironment roundEnv) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public synchronized void init(ProcessingEnvironment processingEnv) {
        // TODO Auto-generated method stub
        super.init(processingEnv);
    }

}
```

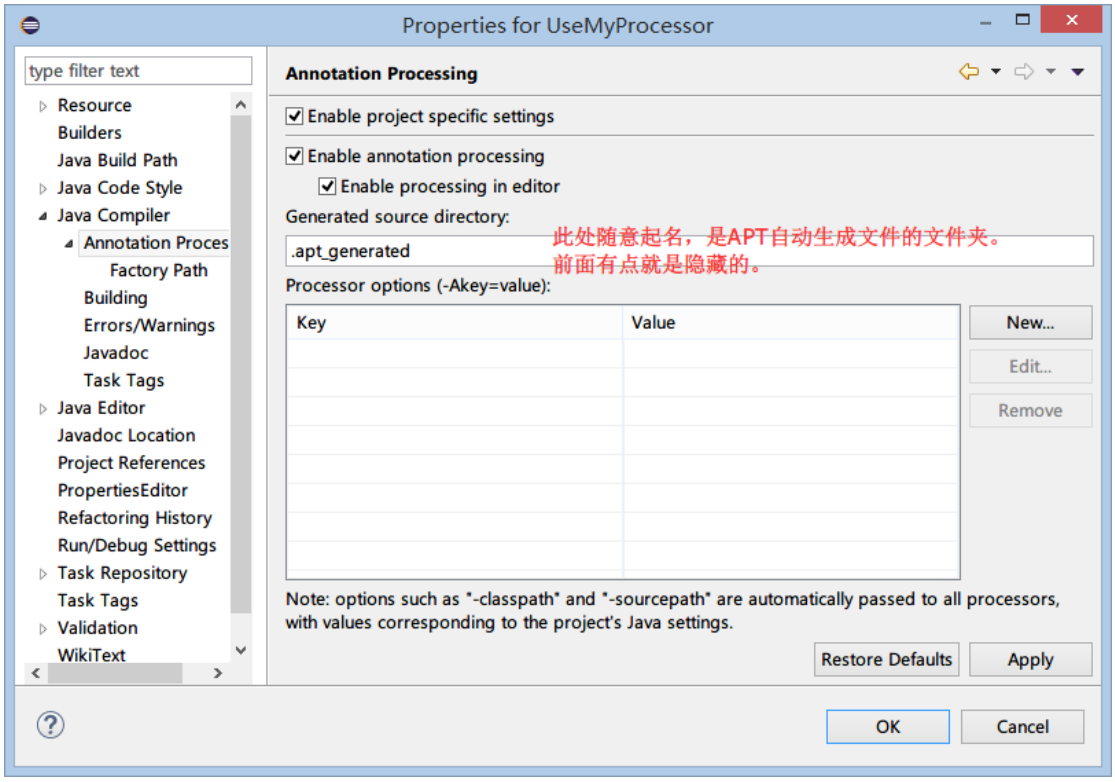
process 方法就是 APT 的执行点，我们对 Java 类的解析和处理就在此函数中进行。

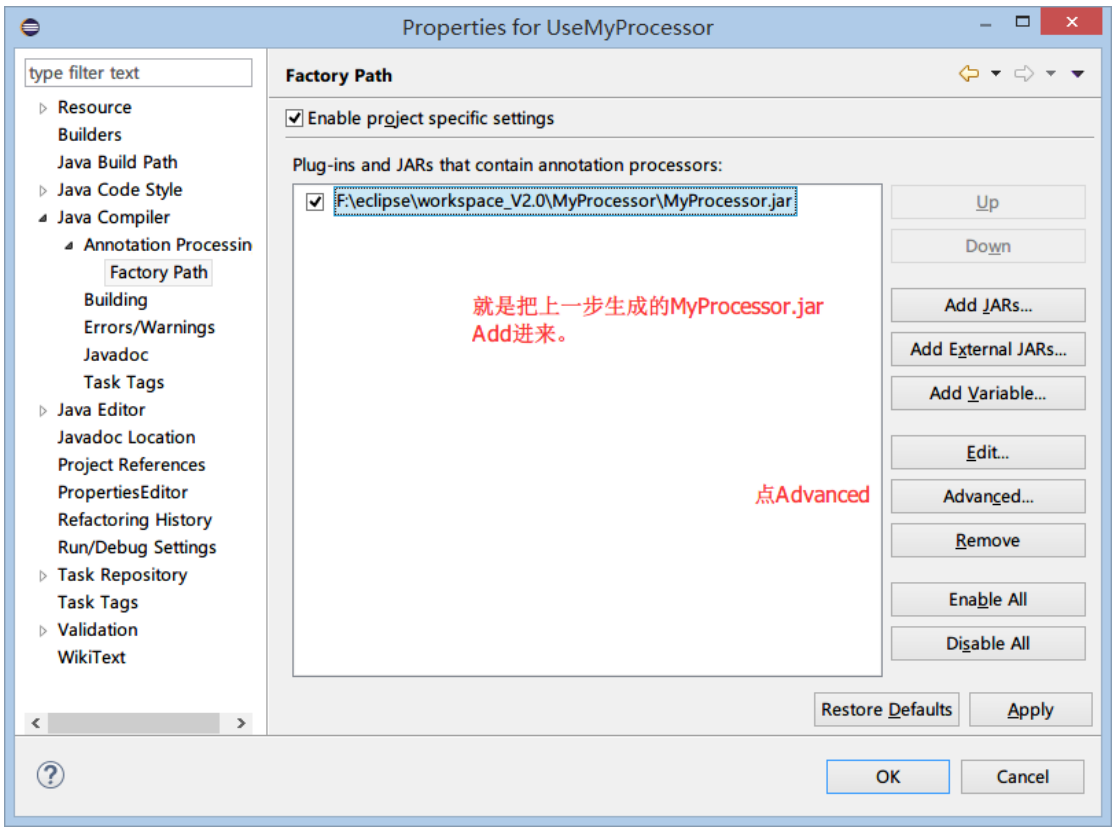
STEP 2.在工程中创建新文件夹，名为”META-INF”，二级文件夹”services”，在其中创建文件，命名为”javax.annotation.processing.Processor”，文件中写上 MyProcessor 类的完整名称，如”com.test.MyProcessor”

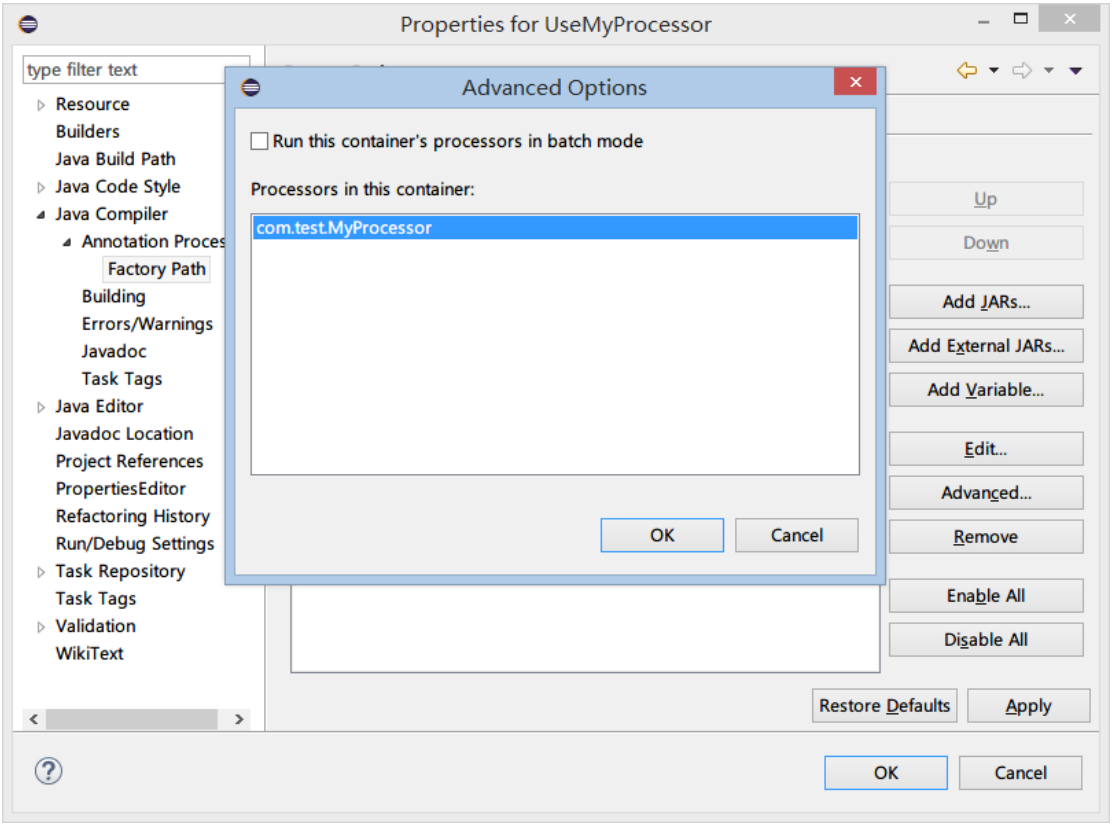
STEP 3.将此工程打包为 jar 包，勾选如图：



STEP 4.建立另一个测试工程，右键工程选择如下：







之后 OKOKYESYES 搞定。

至此我们完成了自己的注解处理器编写和使用，当然，你会发现完全没变化，那是因为我们的 process 函数里面啥都没写，下一节详细说下。

三、APT 能用来干什么

STEP 1. 让我们从简单的开始 现在 我想规定所有在我手下的程序员们给类起名字都不能用 T 开头。那么声明注解处理器要处理的文件、支持的最高 Java 代码版本。后续处理时，可以根据实际需要，在 @SupportedAnnotationTypes 的 value 中赋值你想进行处理的 Annotation 完整 className。

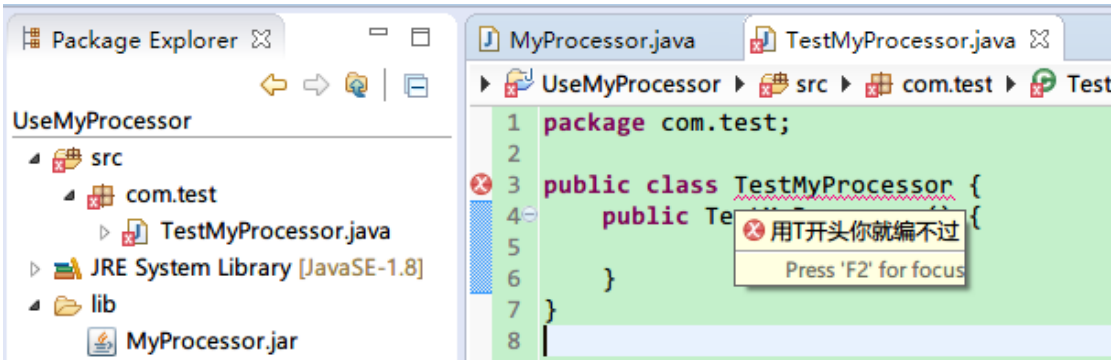
```
@SupportedAnnotationTypes(value={"*"})   支持所有文件、支持Java8下所有版本
@SupportedSourceVersion(SourceVersion.RELEASE_8)
```

STEP 2.改造 MyProcessor 类。Element 有许多实现，这些实现表示了字段、函数、类等不同类型，不同类型 Element 中可获取到不同的 Kind，可供用来进行一些判断，如是否私有函数，字段数据是什么类型等等。

```
@Override
public boolean process(Set<? extends TypeElement> annotations,
        RoundEnvironment roundEnv) {
    // 获取所有java class
    Set<? extends Element> allClass = roundEnv.getRootElements();
    for (Element e : allClass) {
        if (e.getSimpleName().toString().startsWith("T")) {
            processingEnv.getMessager().printMessage(Kind.ERROR, "用T开头你就编不过", e);
        }
    }
    return true;
}
```

STEP 3.生成 jar 包，让所有程序员按照上面的步骤导入。

STEP 4.程序员不信邪，写了一个类。看他编不过了吧。当然，我们也可以不提示 Error，还有好几种类型的提示可供选择。



STEP 5.现在换个花样，给所有用 T 开头的类，在其所属包下生成一个带 Yeah 前缀的 Java 文件。首先获取到文件操作类 Filer

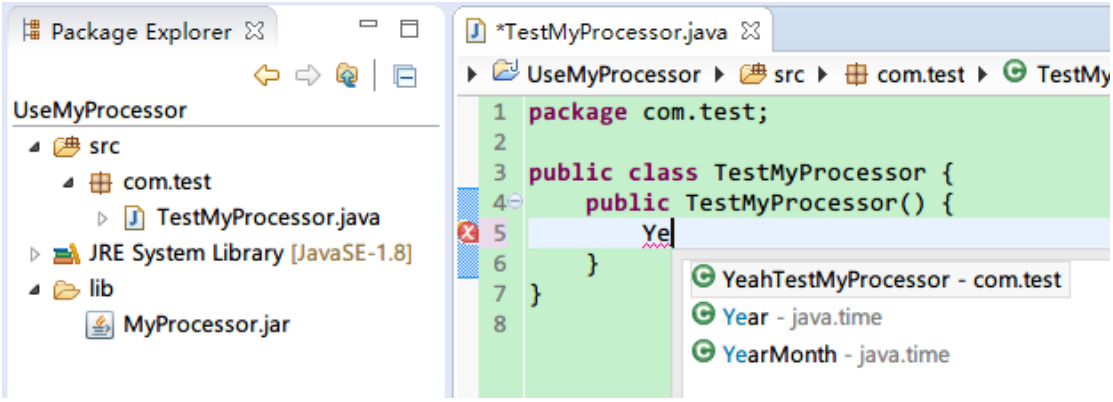
```
@Override
public synchronized void init(ProcessingEnvironment processingEnv) {
    mFiler = processingEnv.getFiler();
    super.init(processingEnv);
}
```

其次写处理的逻辑

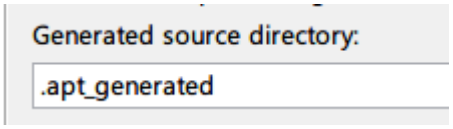
```
@Override
public boolean process(Set<? extends TypeElement> annotations,
        RoundEnvironment roundEnv) {
    // 获取所有java class
    Set<? extends Element> allClass = roundEnv.getRootElements();
    for (Element e : allClass) {
        final String name = e.getSimpleName().toString();
        if (name.startsWith("Yeah")) {
            // 这是我们生成的文件，不再重复处理
            continue;
        }
        if (name.startsWith("T")) {
            createOurJavaFile(e);
        }
    }
    return true;
}
```

```
    private void createOurJavaFile(Element element) {
        // 获取被处理的Java类包名
        final String packageName = element.toString().substring(0,
                element.toString().lastIndexOf("."));
        // 构造生成的类的类名
        final String className = "Yeah" + element.getSimpleName().toString();
        StringBuilder content = new StringBuilder();
        content.append("package ").append(packageName).append(";\n");
        content.append("public class ").append(className).append("{\n}");
        JavaFileObject file = null;
        try {
            file = mFiler.createSourceFile(packageName + "." + className,
                    element);
            file.openWriter().append(content.toString()).close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

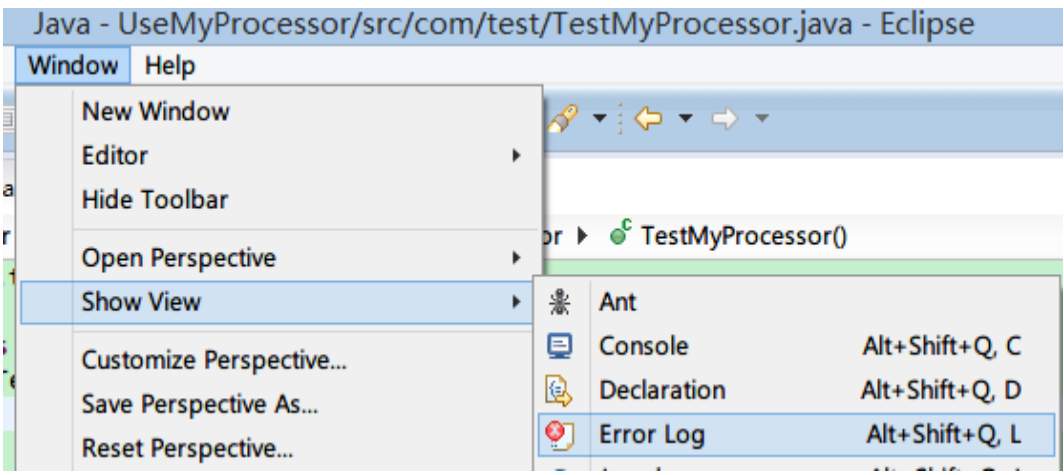
最后生成 Jar 包，更新。友情提示，更新 Jar 包操作不要在 Eclipse 里面做，会报无法覆盖。请直接在操作系统下，把生成的 Jar 包直接复制粘贴到相应工程目录下。之后刷新/Clean 让工程重新编译即可。可以看到已经有这个类了。



为什么我们在左边看不到呢？回忆下第二节中，设置中有一项指定自动生成的路径，前面默认是带点，就是隐藏的了。如果我们想看到的话改改就行。



STEP 6.注解处理器中如何 Debug。APT 处理时在正常编译前执行的，所以我们没法像普通程序一样在 Log 里面看到错误和打印。如果需要加打印，建议使用上面介绍的 printMessage 方法，直接把要加的打印显示在相关 Elment 上，也很直观。如果想看到 APT 运行时的错误，需要打开 Window-Show View-Error Log，如图



四、Android Studio 使用 APT

很多同学使用 Android Studio，Gradle 编译工程，介绍下如何配置才能让 APT 在 Android Studio 环境下正确运行。

STEP 1. 明确 Android Studio 里面有两个 build.gradle。一个是 Module(=Eclipse 的 project)，一个是 Project(=Eclipse 的 workspace)

STEP 2. STEP 2：打开 Android Studio 里 Project 的 build.gradle，加入下面几行：

classpath 'com.neenbedankt.gradle.plugins:android-apt:1.+'

```
dependencies {
    classpath 'com.android.tools.build:gradle:1.2.3'

    // NOTE: Do not place your application dependencies here; they belong
    // in the individual module build.gradle files
    classpath 'org.robolectric:robolectric-gradle-plugin:0.+'
    classpath 'com.neenbedankt.gradle.plugins:android-apt:1.+'
}
```

STEP 3. 打开 Module 的 build.grade，加入你要使用的带 APT 注解处理器的 jar 包：

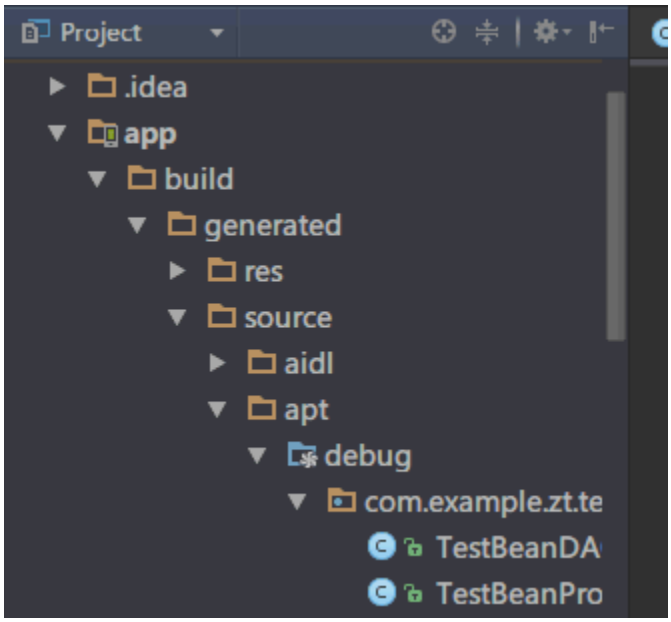
a).文件开头加上依赖 apply plugin: 'android-apt'

```
apply plugin: 'com.android.application'
apply plugin: 'android-apt'
```

b).加入 jar 包，举例 SimpleDAO.jar

```
dependencies {
    compile files('libs/SimpleDAO.jar')
    apt files('libs/SimpleDAO.jar')
}
```

STEP 4. 编译一下，应该可以看到成功，然后注解处理器自动生成的代码在 build/generated/source/apt/目录下



五、APT 工程示例

我与许绍秋同学联手制作了一款 Android SQLite ORM 框架。参考了世面上比较流行的一些 ORM 框架，比如 AFinal，xUtils，DBExecutor 等等。我们框架的优势就在于使用了 APT 技术，而其余框架使用运行时反射。众所周知反射是需要额外时间的，会导致效率变低。我们在 918 上面跑过测试，10w 行数据的增删改查，速度比上述框架快很多。当然，我们认为易用性方面也有优势，具体就不吹了。此框架目前在我们自己的应用，如全网搜索、EPGServer、呼吧 3.0 等应用中都有所使用。欢迎大家围观使用吐槽。Github 地址：https://github.com/zxfrdas/SimpleDAO

个人认为 APT 可以做的事情还有很多，比如我们可以用来生成 findViewById 的代码，或者对类、函数的命名、属性做规定。期待大家更好的点子。

Have Fun!