1.1APT简介

> 1.什么是APT？ javac 附属工具

APT即为Annotation Processing Tool，它是javac的一个工具，中文意思为编译时注解处理器。

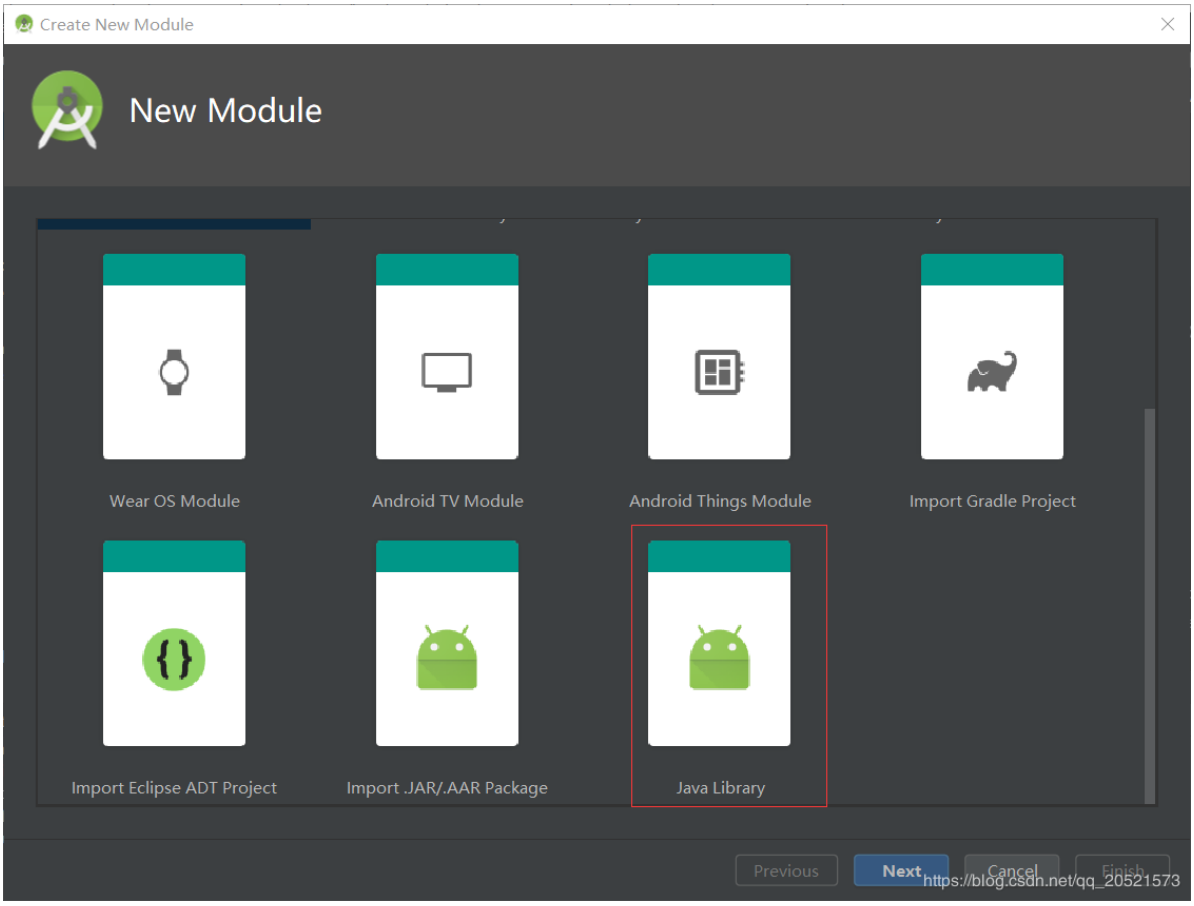
APT可以用来在编译时扫描和处理注解。通过APT可以获取到注解和被注解对象的相关信息，在拿到这些信息后我们可以根据需求来自动的生成一些代码，省去了手动编写。

注意，获取注解及生成代码都是在代码编译时候完成的，相比反射在运行时处理注解大大提高了程序性能。**APT的核心是AbstractProcessor类**

> 2.哪里用到了APT?

APT技术被广泛的运用在Java框架中，包括Android项以及Java后台项目，除了上面我们提到的ButterKnife之外，像EventBus 、Dagger2以及阿里的ARouter路由框架等都运用到APT技术，因此要想了解以、探究这些第三方框架的实现原理，APT就是我们必须要掌握的。

3.如何在Android Studio中构建一个APT项目?
APT项目需要由至少两个Java Library模块组成，不知道什么是Java Library？没关系，手把手来叫你如何创建一个Java Library。



首先，新建一个Android项目，然后File->New->New Module,打开如上图所示的面板，选择Java Library即可。刚才说到一个APT项目至少应该由两个Java Library模块。那么这两个模块分别是什么作用呢?

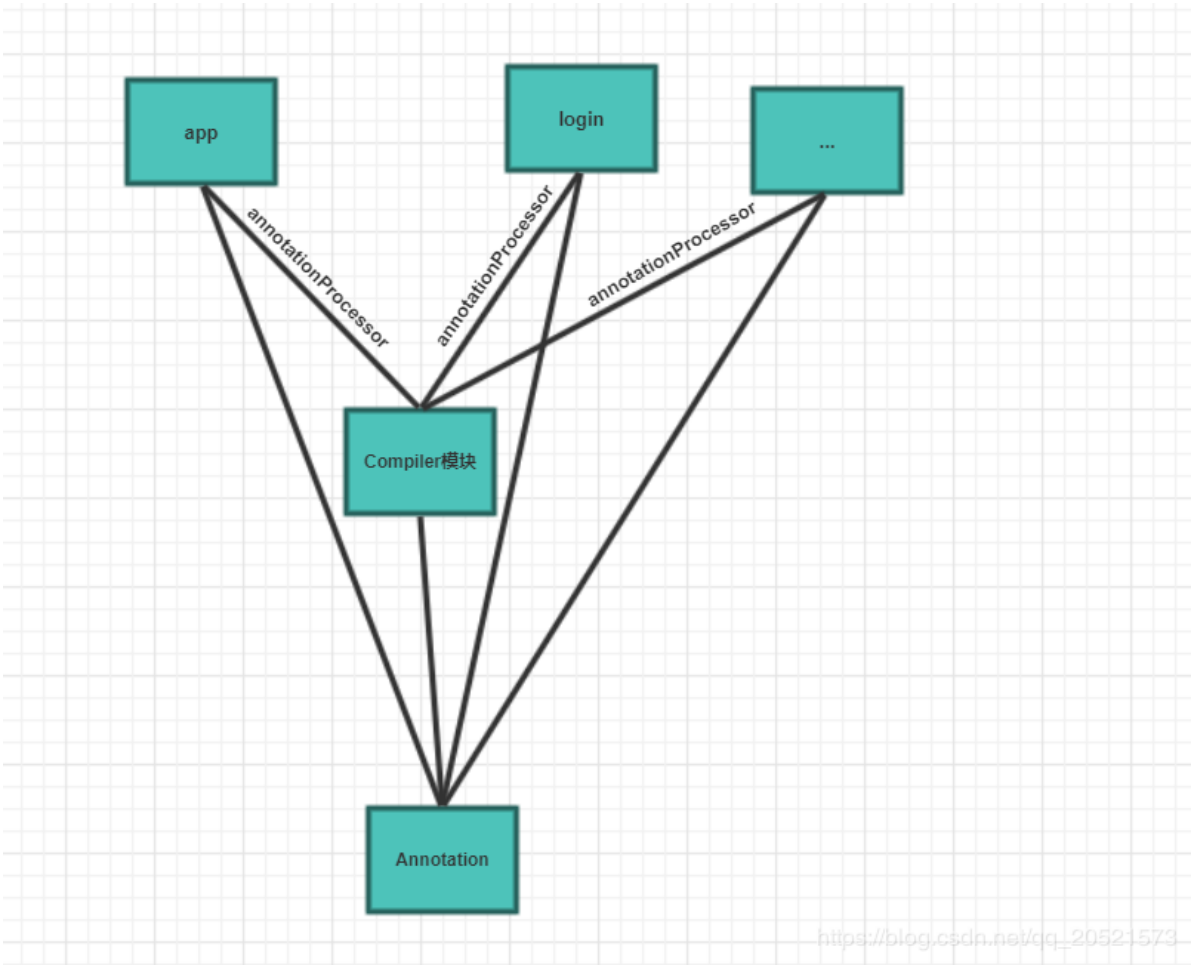
1. 先需要一个Annotation模块，这个用来存放自定义的注解。
2. 另外需要一个Compiler模块，这个模块依赖Annotation模块。

3. 项目的App模块和其它的业务模块都需要依赖Annotation模块，同时需要通过annotationProcessor依赖Compiler模块。

app模块的gradle中依赖关系如下：

```
implementation project(':annotation')
annotationProcessor project(':factory-compiler')
```

APT项目的模块的结构图如下所示：



> 为什么要强调上述两个模块一定要是Java Library？**如果创建Android Library模块你会发现不能找到AbstractProcessor这个类**，这是因为Android平台是基于OpenJDK的，而OpenJDK中不包含APT的相关代码。因此，在使用APT时，必须在Java Library中进行。

1.2APT使用方式

在处理编译器注解的第一个手段就是APT(Annotation Processor Tool),即注解处理器。在java5的时候已经存在，但是java6开始的时候才有可用的API，最近才随着butterknife这些库流行起来。本章将阐述什么是注解处理器，以及如何使用这个强大的工具。

> 什么是APT

APT是一种处理注解的工具，确切的说它是javac的一个工具，它用来在编译时扫描和处理注解，一个注解的注解处理器，以java代码(或者编译过的字节码)作为输入，生成.java文件作为输出，核心是交给自己定义的处理器去处理，

> 如何使用

每个自定义的处理器都要继承虚处理器，实现其关键的几个方法

- 继承虚处理器 AbstractProcessor

```
public class MyProcessor extends AbstractProcessor {
    @Override
    public synchronized void init(ProcessingEnvironment env){ }

    @Override
    public boolean process(Set<? extends TypeElement> annoations,
RoundEnvironment env) { }

    @Override
    public Set<String> getSupportedAnnotationTypes() { }

    @Override
    public SourceVersion getSupportedSourceVersion() { }
}
```

下面重点介绍下这几个函数：

> 1. `init(ProcessingEnvironment env)`: 每一个注解处理器类都必须有一个空的构造函数。然而，这里有一个特殊的init()方法，
>
>   它会被注解处理工具调用，并输入ProcessingEnviroment参数。ProcessingEnviroment提供很多有用的工具类Elements, Types和Filer
>
> 2. `process(Set<? extends TypeElement> annotations, RoundEnvironment env)`: 这相当于每个处理器的**主函数main()。**你在这里写你的扫描、评估和处理注解的代码，以及生成Java文件。输入参数RoundEnviroment，可以让你查询出包含特定注解的被注解元素。这是一个布尔值，表明注解是否已经被处理器处理完成，官方原文`whether or not the set of annotations are claimed by this processor`，通常在处理出现异常直接返回false、处理完成返回true。
> 3. `getSupportedAnnotationTypes()`: 必须要实现；用来表示这个注解处理器是注册给哪个注解的。返回值是一个字符串的集合，包含本处理器想要处理的注解类型的合法全称。
> 4. `getSupportedSourceVersion()`: 用来指定你使用的Java版本。通常这里返回SourceVersion.latestSupported()，你也可以使用SourceVersion_RELEASE_6、7、8

1.3 注册 处理器

由于处理器是javac的工具，因此我们必须将我们自己的处理器注册到javac中，在以前我们需要提供一个.jar文件，打包你的注解处理器到此文件中，并在在你的jar中，需要打包一个特定的文件`javax.annotation.processing.Processor到META-INF/services路径下`把MyProcessor.jar放到你的builpath中，javac会自动检查和读取javax.annotation.processing.Processor中的内容，并且注册MyProcessor作为注解处理器。

超级麻烦有木有，不过不要慌，谷歌baba给我们开发了AutoService注解，你只需要引入这个依赖，然后在你的解释器第一行加上

```
@AutoService(Processor.class)
```

然后就可以自动生成META-INF/services/javax.annotation.processing.Processor文件的。省去了打jar包这些繁琐的步骤。

> APT中的Elements和TypeMirrors

在前面的init()中我们可以获取如下引用

> - Elements：一个用来处理Element的工具类
> - Types：一个用来处理TypeMirror的工具类
> - Filer：正如这个名字所示，使用Filer你可以创建文件(通常与javapoet结合)

在注解处理过程中，我们扫面所有的Java源文件。源文件的每一个部分都是一个特定类型的Element

先来看一下Element

对于编译器来说 代码中的元素结构是基本不变的，如，组成代码的基本元素包括包、类、函数、字段、变量的等，JDK为这些元素定义了一个基类也就是`Element`类

Element有五个直接子类，分别代表一种特定类型

PackageElement	**表示一个包程序元素，可以获取到包名等**
TypeParameterElement	表示一般类、接口、方法或构造方法元素的泛型参数
TypeElement	表示一个类或接口程序元素
VariableElement	表示一个字段、enum 常量、方法或构造方法参数、局部变量或异常参数
ExecutableElement	表示某个类或接口的方法、构造方法或初始化程序（静态或实例），包括注解类型元素

开发中Element可根据实际情况强转为以上5种中的一种，它们都带有各自独有的方法，如下所示

```
package com.example;    // PackageElement

public class Test {        // TypeElement

    private int a;      // VariableElement
    private Test other;  // VariableElement

    public Test () {}    // ExecuteableElement
    public void setA (  // ExecuteableElement
                      int newA   // TypeElement
                      ) {}
}
```

再举个栗子?:

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

    String value();
}
```

这个注解因为只能作用于函数类型，因此，它对应的元素类型就是ExecutableElement当我们想通过APT处理这个注解的时候就可以获取目标对象上的Test注解，并且将所有这些元素转换为ExecutableElement元素，以便获取到他们对应的信息。

查看其代码定义

1.4 Element含义

定义如下：

```
**
 * 表示一个程序元素，比如包、类或者方法，有如下几种子接口：
 * ExecutableElement：表示某个类或接口的方法、构造方法或初始化程序（静态或实例），包括注解类型元素 ；
 * PackageElement：表示一个包程序元素；
 * TypeElement：表示一个类或接口程序元素；
 * TypeParameterElement：表示一般类、接口、方法或构造方法元素的形式类型参数；
 * VariableElement：表示一个字段、enum 常量、方法或构造方法参数、局部变量或异常参数
 */
public interface Element extends AnnotatedConstruct {
    /**
     * 返回此元素定义的类型
     * 例如，对于一般类元素 C<N extends Number>，返回参数化类型 C<N>
     */
    TypeMirror asType();

    /**
     * 返回此元素的种类：包、类、接口、方法、字段...,如下枚举值
     * PACKAGE, ENUM, CLASS, ANNOTATION_TYPE, INTERFACE, ENUM_CONSTANT, FIELD, PARAMETER, LOCAL_VARIABLE, EXCEPTION_PARAMETER,
     * METHOD, CONSTRUCTOR, STATIC_INIT, INSTANCE_INIT, TYPE_PARAMETER, OTHER, RESOURCE_VARIABLE;
     */
    ElementKind getKind();

    /**
     * 返回此元素的修饰符,如下枚举值
     * PUBLIC, PROTECTED, PRIVATE, ABSTRACT, DEFAULT, STATIC, FINAL,
     * TRANSIENT, VOLATILE, SYNCHRONIZED, NATIVE, STRICTFP;
     */
    Set<Modifier> getModifiers();

    /**
     * 返回此元素的简单名称,例如
     * 类型元素 java.util.Set<E> 的简单名称是 "Set"；
     * 如果此元素表示一个未指定的包，则返回一个空名称；
     * 如果它表示一个构造方法，则返回名称 "<init>"；
     * 如果它表示一个静态初始化程序，则返回名称 "<clinit>"；
     * 如果它表示一个匿名类或者实例初始化程序，则返回一个空名称
     */
```

```
    Name getSimpleName();

    /**
     * 返回封装此元素的最里层元素。
     * 如果此元素的声明在词法上直接封装在另一个元素的声明中，则返回那个封装元素；
     * 如果此元素是顶层类型，则返回它的包；
     * 如果此元素是一个包，则返回 null；
     * 如果此元素是一个泛型参数，则返回 null.
     */
    Element getEnclosingElement();

    /**
     * 返回此元素直接封装的子元素
     */
    List<? extends Element> getEnclosedElements();

    boolean equals(Object var1);

    int hashCode();

    /**
     * 返回直接存在于此元素上的注解
     * 要获得继承的注解，可使用 getAllAnnotationMirrors
     */
    List<? extends AnnotationMirror> getAnnotationMirrors();

    /**
     * 返回此元素针对指定类型的注解（如果存在这样的注解），否则返回 null。注解可以是继承的，也
可以是直接存在于此元素上的
     */
    <A extends Annotation> A getAnnotation(Class<A> annotationType);

    //接受访问者的访问 （？？）
     <R, P> R accept(ElementVisitor<R, P> var1, P var2);
}
```

最后一个，并没有使用到，感觉不太好理解，查了资料这个函数接受一个ElementVisitor和类型为P的参数。

```
public interface ElementVisitor<R, P> {
    //访问元素
    R visit(Element e, P p);

    R visit(Element e);

    //访问包元素
    R visitPackage(PackageElement e, P p);

    //访问类型元素
    R visitType(TypeElement e, P p);

   //访问变量元素
    R visitVariable(VariableElement e, P p);

    //访问克而执行元素
    R visitExecutable(ExecutableElement e, P p);
```

```
    //访问参数元素
    R visitTypeParameter(TypeParameterElement e, P p);

    //处理位置的元素类型，这是为了应对后续Java语言的扩折而预留的接口，例如后续元素类型添加了，
那么通过这个接口就可以处理上述没有声明的类型
    R visitUnknown(Element e, P p);
}
```

在ElementgVisitor中定义了多个visit接口，每个接口处理一种元素类型，这就是典型的访问者模式。我们制定，一个类元素和函数元素是完全不一样的，他们的结构不一样，因此，在编译器对他们的操作肯定是不一样，通过访问者模式正好可以解决数据结构与数据操作分离的问题，避免某些操作污染数据对象类。

因此，代码在APT眼中只是一个结构化的文本而已。Element代表的是源代码。TypeElement代表的是源代码中的类型元素，例如类。然而，TypeElement并不包含类本身的信息。你可以从TypeElement中获取类的名字，但是你获取不到类的信息，例如它的父类。这种信息需要通过TypeMirror获取。你可以通过调用elements.asType()获取元素的TypeMirror。

1.5 辅助接口

在自定义注解器的初始化时候，可以获取以下4个辅助接口

```
public class MyProcessor extends AbstractProcessor {

        private Types typeUtils;
        private Elements elementUtils;
        private Filer filer;
        private Messager messager;

        @Override
        public synchronized void init(ProcessingEnvironment processingEnv) {
            super.init(processingEnv);
            typeUtils = processingEnv.getTypeUtils();
            elementUtils = processingEnv.getElementUtils();
            filer = processingEnv.getFiler();
            messager = processingEnv.getMessager();
        }
    }
```

> - Filer

一般配合JavaPoet来生成需要的java文件（下一篇将详细介绍javaPoet）

> - Messager

Messager提供给注解处理器一个报告错误、警告以及提示信息的途径。它不是注解处理器开发者的日志工具，而是用来写一些信息给使用此注解器的第三方开发者的。

在官方文档中描述了消息的不同级别中非常重要的是Kind.ERROR，因为这种类型的信息用来表示我们的注解处理器处理失败了。很有可能是第三方开发者错误的使用了注解。这个概念和传统的Java应用有点不一样，在传统Java应用中我们可能就抛出一个异常Exception。如果你在process()中抛出一个异常，那么运行注解处理器的JVM将会崩溃（就像其他Java应用一样），使用我们注解处理器第三方开发者将会从javac中得到非常难懂的出错信息，

因为它包含注解处理器的堆栈跟踪（Stacktace）信息。因此，注解处理器就有一个Messager类，它能够打印非常优美的错误信息。除此之外，你还可以连接到出错的元素。在像现在的IDE（集成开发环境）中，第三方开发者可以直接点击错误信息，IDE将会直接跳转到第三方开发者项目的出错的源文件的相应的行。

> - Types

Types是一个用来处理TypeMirror的工具

> - Elements

Elements是一个用来处理Element的工具

1.6 优缺点

> 优点(结合javapoet)

- 对代码进行标记、在编译时收集信息并做处理
- 生成一套独立代码，辅助代码运行

> 缺点

- 可以自动生成代码，但在运行时需要主动调用
- 如果要生成代码需要编写模板函数