前面有提到注解按生命周期来划分可分为 3 类：

1、RetentionPolicy.SOURCE：注解只保留在源文件，当 **Java** 文件编译成 class 文件的时候，注解被遗弃；
2、RetentionPolicy.CLASS：注解被保留到 class 文件，但 jvm 加载 class 文件时候被遗弃，这是默认的生命周期；
3、RetentionPolicy.RUNTIME：注解不仅被保存到 class 文件中，jvm 加载 class 文件之后，仍然存在；

这 3 个生命周期分别对应于：Java 源文件(.java 文件) ---> .class 文件 ---> 内存中的字节码。

那怎么来选择合适的注解生命周期呢?

首先要明确生命周期长度 SOURCE < CLASS < RUNTIME ，所以前者能作用的地方后者一定也能作用。一般如果需要在运行时去动态获取注解信息，那只能用 RUNTIME 注解；如果要在编译时进行一些预处理操作，比如生成一些辅助代码（如 ButterKnife），就用 CLASS 注解；如果只是做一些检查性的操作，比如 @Override 和 @SuppressWarnings，则可选用 SOURCE 注解。

下面来介绍下运行时注解的简单运用。

获取注解

你需要通过反射来获取运行时注解，可以从 Package、Class、Field、Method...上面获取，基本方法都一样，几个常见的方法如下：

```
/**
 * 获取指定类型的注解
 */
public <A extends Annotation> A getAnnotation(Class<A> annotationType);

/**
 * 获取所有注解，如果有的话
 */
public Annotation[] getAnnotations();

/**
 * 获取所有注解，忽略继承的注解
 */
public Annotation[] getDeclaredAnnotations();

/**
 * 指定注解是否存在该元素上，如果有则返回 true，否则 false
 */
public boolean isAnnotationPresent(Class<? extends Annotation> annotationType);

/**
 * 获取 Method 中参数的所有注解
 */
public Annotation[][] getParameterAnnotations();
```

要使用这些函数必须先通过反射获取到对应的元素：Class、Field、Method 等。

自定义注解

来看下自定义注解的简单使用方式，这里先定义 3 个运行时注解：

```java
// 适用类、接口（包括注解类型）或枚举
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface ClassInfo {
    String value();
}
// 适用 field 属性，也包括 enum 常量
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface FieldInfo {
    int[] value();
}
// 适用方法
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface MethodInfo {
    String name() default "long";
    String data();
    int age() default 27;
}
```

这 3 个注解分别适用于不同的元素，并都带有不同的属性，在使用注解是需要设置这些属性值。

再定义一个**测试**类来使用这些注解：

```java
/**
 * 测试运行时注解
 */
@ClassInfo("Test Class")
public class TestRuntimeAnnotation {

    @FieldInfo(value = {1, 2})
    public String fieldInfo = "FiledInfo";

    @FieldInfo(value = {10086})
    public int i = 100;

    @MethodInfo(name = "BlueBird", data = "Big")
    public static String getMethodInfo() {
        return TestRuntimeAnnotation.class.getSimpleName();
    }
}
```

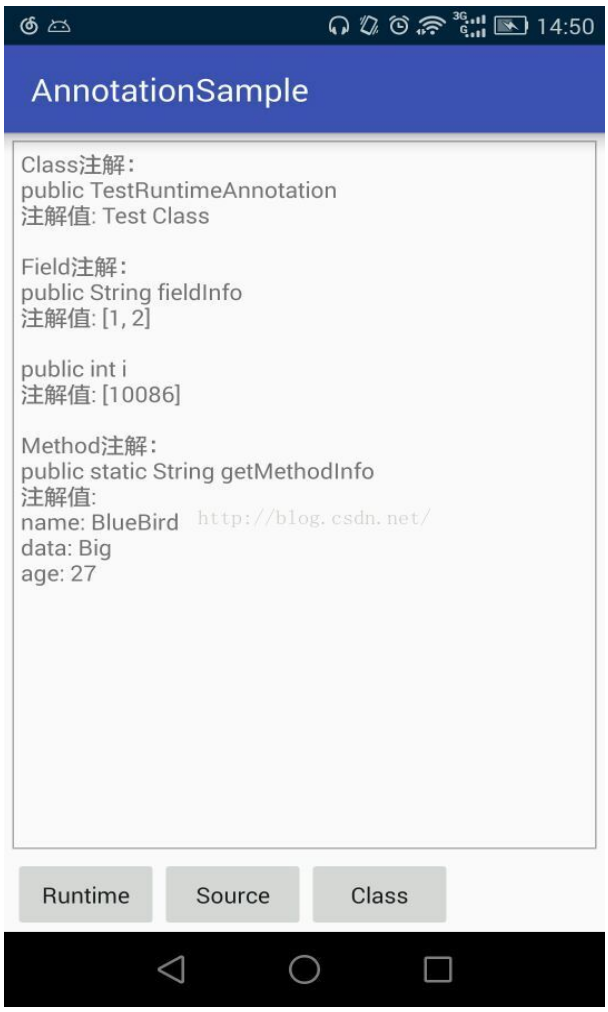
使用还是很简单的，最后来看怎么在代码中获取注解信息：

```java
/**
 * 测试运行时注解
 */
private void _testRuntimeAnnotation() {
    StringBuffer sb = new StringBuffer();
    Class<?> cls = TestRuntimeAnnotation.class;
    Constructor<?>[] constructors = cls.getConstructors();
    // 获取指定类型的注解
    sb.append("Class 注解: ").append("\n");
    ClassInfo classInfo = cls.getAnnotation(ClassInfo.class);
```

```
    if (classInfo != null) {
        sb.append(Modifier.toString(cls.getModifiers())).append(" ")
                .append(cls.getSimpleName()).append("\n");
        sb.append("注解值: ").append(classInfo.value()).append("\n\n");
    }

    sb.append("Field 注解: ").append("\n");
    Field[] fields = cls.getDeclaredFields();
    for (Field field : fields) {
        FieldInfo fieldInfo = field.getAnnotation(FieldInfo.class);
        if (fieldInfo != null) {
            sb.append(Modifier.toString(field.getModifiers())).append(" ")
                    .append(field.getType().getSimpleName()).append(" ")
                    .append(field.getName()).append("\n");
            sb.append("注解值: ").append(Arrays.toString(fieldInfo.value())).append("\n\n");
        }
    }

    sb.append("Method 注解: ").append("\n");
    Method[] methods = cls.getDeclaredMethods();
    for (Method method : methods) {
        MethodInfo methodInfo = method.getAnnotation(MethodInfo.class);
        if (methodInfo != null) {
            sb.append(Modifier.toString(method.getModifiers())).append(" ")
                    .append(method.getReturnType().getSimpleName()).append(" ")
                    .append(method.getName()).append("\n");
            sb.append("注解值: ").append("\n");
            sb.append("name: ").append(methodInfo.name()).append("\n");
            sb.append("data: ").append(methodInfo.data()).append("\n");
            sb.append("age: ").append(methodInfo.age()).append("\n");
        }
    }

    System.out.print(sb.toString());
}
```

所做的操作都是通过反射获取对应元素，再获取元素上面的注解，最后得到注解的属性值。

看一下输出情况，这里我直接显示在手机上：



这个自定义运行时注解是很简单的例子，有很多优秀的开源项目都有使用运行时注解来处理问题，有兴趣可以找一些来研究。因为涉及到反射，所以运行时注解的效率多少会受到影响，现在很多的开源项目使用的是编译时注解，关于编译时注解后面再来详细介绍。

相关内容：自定义注解之源码注解(RetentionPolicy.SOURCE)