Java学习之注解Annotation实现原理

前言：

最近学习了EventBus、BufferKinfe、GreenDao、Retrofit 等优秀开源框架，它们新版本无一另外的都使用到了注解的方式，我们使用在使用的时候也尝到不少好处，基于这种想法我觉得有必要对注解有个更深刻的认识，今天中午把公司的项目搞完了，晚上加个班学习总结一下Java的注解。

什么是注解？

对于很多初次接触的开发者来说应该都有这个疑问？Annontation是Java5开始引入的新特征，中文名称叫注解。它提供了一种安全的类似注释的机制，用来将任何的信息或元数据（metadata）与程序元素（类、方法、成员变量等）进行关联。为程序的元素（类、方法、成员变量）加上更直观更明了的说明，这些说明信息是与程序的业务逻辑无关，并且供指定的工具或框架使用。Annontation像一种修饰符一样，应用于包、类型、构造方法、方法、成员变量、参数及本地变量的声明语句中。

注解的用处：

1、生成文档。这是最常见的，也是java 最早提供的注解。常用的有@param @return 等

2、跟踪代码依赖性，实现替代配置文件功能。比如Dagger 2依赖注入，未来java开发，将大量注解配置，具有很大用处；

3、在编译时进行格式检查。如@override 放在方法前，如果你这个方法并不是覆盖了超类方法，则编译时就能检查出。

元注解：

java.lang.annotation提供了四种元注解，专门注解其他的注解：

@Documented –注解是否将包含在JavaDoc中

@Retention –什么时候使用该注解

@Target –注解用于什么地方

@Inherited – 是否允许子类继承该注解

1.) @Retention– 定义该注解的生命周期

- **RetentionPolicy.SOURCE**：在编译阶段丢弃。这些注解在编译结束之后就不再有任何意义，所以它们不会写入字节码。@Override, @SuppressWarnings都属于这类注解。
- **RetentionPolicy.CLASS**：在类加载的时候丢弃。在字节码文件的处理中有用。注解默认使用这种方式
- **RetentionPolicy.RUNTIME**：始终不会丢弃，运行期也保留该注解，因此可以使用反射机制读取该注解的信息。我们自定义的注解通常使用这种方式。

举例：bufferKnife 8.0 中@BindView 生命周期为CLASS

```
@Retention(CLASS) @Target(FIELD)
public @interface BindView {
  /** View ID to which the field will be bound. */
  @IdRes int value();
}
```

2.) Target – 表示该注解用于什么地方。默认值为任何元素，表示该注解用于什么地方。可用的ElementType参数包括

- **ElementType.CONSTRUCTOR**:用于描述构造器
- **ElementType.FIELD**:成员变量、对象、属性（包括enum实例）
- **ElementType.LOCAL_VARIABLE**:用于描述局部变量
- **ElementType.METHOD**:用于描述方法
- **ElementType.PACKAGE**:用于描述包
- **ElementType.PARAMETER**:用于描述参数
- **ElementType.TYPE**:用于描述类、接口(包括注解类型) 或enum声明

举例Retrofit 2 中@Field 作用域为参数

```
@Documented
@Target(PARAMETER)
```

IOS网络请求(2)
IOS学习总结(6)
Java日常总结(3)
Java设计模式(6)
Java数据结构(3)
React Native开发
工作随笔(3)

积分与排名

积分 - 238656
排名 - 797

最新评论

1. Re:Android混合开发之WebViewJavascriptBridge实现JS与java安全交互
给出对应的github的地址上传demo 正好我们都去关注一下
--全球顶尖卧底
2. Re:Android混合开发之WebViewJavascriptBridge实现JS与java安全交互
看了一些你的文章今天 个人觉得吧 文章还是不错 但是有个缺点 就是现在很多看文章的人出现的一个通病 就是光看文章感觉挺有道理 但是叫自己写的话就会出现难度 不知道博主能否分享在写文.....
--全球顶尖卧底
3. Re:Android图片缓存之初识Glide
@全球顶尖卧底谢谢支持...
--总李写代码

阅读排行榜

1. Android图片缓存之初识Glide(40722)
2. Android okHttp网络请求之Get/Post请求(33957)
3. Android数据存储之GreenDao 3.0 详解(30614)
4. Android消息传递之EventBus 3.0使用详解(30241)
5. Android okHttp网络请求之文件上传下载(26986)
6. Android混合开发之WebView与Javascript交互(24459)
7. Android动画效果之Frame Animation (逐帧动画)(20025)
8. Android自定义控件之自定义组合控件(19613)
9. Android图片缓存之Glide进阶(18642)
10. Android业务组件化之URL Scheme使用(16629)
11. Android自定义控件之基本原理(15829)
12. Android自定义控件之自定义属性(14893)
13. Android数据加密之MD5加密(13993)

```
@Retention(RUNTIME)
public @interface Field {
  String value();

  /** Specifies whether the {@linkplain #value() name} and value are already URL encoded. */
  boolean encoded() default false;
}
```

3.)@Documented–一个简单的Annotations标记注解，表示是否将注解信息添加在java文档中。

4.)@Inherited – 定义该注释和子类的关系

@Inherited 元注解是一个标记注解，@Inherited阐述了某个被标注的类型是被继承的。如果一个使用了@Inherited修饰的annotation类型被用于一个class，则这个annotation将被用于该class的子类。

常见标准的Annotation：

1.) Override

java.lang.Override是一个标记类型注解，它被用作标注方法。它说明了被标注的方法重载了父类的方法，起到了断言的作用。如果我们使用了这种注解在一个没有覆盖父类方法的方法时，java编译器将以一个编译错误来警示。

2.) Deprecated

Deprecated也是一种标记类型注解。当一个类型或者类型成员使用@Deprecated修饰的话，编译器将不鼓励使用这个被标注的程序元素。所以使用这种修饰具有一定的"延续性"：如果我们在代码中通过继承或者覆盖的方式使用了这个过时的类型或者成员，虽然继承或者覆盖后的类型或者成员并不是被声明为@Deprecated，但编译器仍然要报警。

3.) SuppressWarnings

SuppressWarning不是一个标记类型注解。它有一个类型为String[]的成员，这个成员的值为被禁止的警告名。对于javac编译器来讲，被-Xlint选项有效的警告名也同样对@SuppressWarings有效，同时编译器忽略掉无法识别的警告名。

```
@SuppressWarnings("unchecked")
```

自定义注解：

这里模拟一个满足网络请求接口，以及如何获取接口的注解函数，参数执行请求。

1.) **定义注解：**

@ReqType 请求类型

```
@Documented
@Target(METHOD)
@Retention(RUNTIME)
public @interface ReqType {

    /**
     * 请求方式枚举
     *
     */
    enum ReqTypeEnum{ GET,POST,DELETE,PUT};

    /**
     * 请求方式
     * @return
     */
    ReqTypeEnum reqType() default ReqTypeEnum.POST;
}
```

@ReqUrl 请求地址

```
@Documented
@Target(METHOD)
@Retention(RUNTIME)
public @interface ReqUrl {
```

14. Android数据加密之Aes加密(13699)
15. Java学习之注解Annotation实现原理(13051)
16. Android数据加密之Base64编码算法(12909)
17. Android混合开发之WebView使用总结(11735)
18. Android探索之HttpURLConnection网络请求(11255)
19. Android图片缓存之Bitmap详解(10821)
20. Android数据存储之Android 6.0运行时权限下文件存储的思考(10435)

推荐排行榜

1. Android业务组件化之现状分析与探讨(7)
2. Android业务组件化之URL Scheme使用(6)
3. Android自定义控件之自定义组合控件(6)
4. Android自定义控件之基本原理(6)
5. Android数据存储之GreenDao 3.0 详解(5)
6. Android注解使用之ButterKnife 8.0注解使用介绍(5)
7. Android图片缓存之初识Glide(5)
8. Android okHttp网络请求之Get/Post请求(4)
9. Android性能优化之利用LeakCanary检测内存泄漏及解决办法(4)
10. Android混合开发之WebViewJavascriptBridge实现JS与java安全交互(4)
11. Android UI体验之全屏沉浸式透明状态栏效果(3)
12. IOS缓存管理之YYCache使用(3)
13. Android线程管理之ThreadPoolExecutor自定义线程池(3)
14. Android消息传递之EventBus 3.0使用详解(3)
15. Java学习之注解Annotation实现原理(3)
16. Android混合开发之WebView使用总结(3)
17. Java学习之反射机制及应用场景(3)
18. Android动画效果之初识Property Animation (属性动画) (3)
19. Android动画效果之Frame Animation (逐帧动画) (3)
20. Android动画效果之Tween Animation (补间动画) (3)

```
    String reqUrl() default "";
}
```

@ReqParam 请求参数

```
@Documented
@Target(PARAMETER)
@Retention(RUNTIME)
public @interface ReqParam {
    String value() default "";
}
```

从上面可以看出注解参数的可支持数据类型有如下：

1.所有基本数据类型 (int,float,boolean,byte,double,char,long,short)
2.String类型
3.Class类型
4.enum类型
5.Annotation类型
6.以上所有类型的数组

而且不难发现@interface用来声明一个注解，其中的每一个方法实际上是声明了一个配置参数。方法的名称就是参数的名称，返回值类型就是参数的类型 (返回值类型只能是基本类型、Class、String、enum) 。可以通过default来声明参数的默认值。

2.) 如何使用自定义注解

```
public interface IReqApi {

    @ReqType(reqType = ReqType.ReqTypeEnum.POST)//声明采用post请求
    @ReqUrl(reqUrl = "www.xxx.com/openApi/login")//请求Url地址
    String login(@ReqParam("userId") String userId, @ReqParam("pwd") String pwd);//参数用户名 密码

}
```

3.) 如何获取注解参数

这里强调一下，Annotation是被动的元数据，永远不会有主动行为，但凡Annotation起作用的场合都是有一个执行机制/调用者通过反射获得了这个元数据然后根据它采取行动。

通过反射机制获取函数注解信息

```
      Method[] declaredMethods = IReqApi.class.getDeclaredMethods();
        for (Method method : declaredMethods) {
            Annotation[]  methodAnnotations = method.getAnnotations();
            Annotation[][] parameterAnnotationsArray = method.getParameterAnnotations();
        }
```

也可以获取指定的注解

```
ReqType reqType =method.getAnnotation(ReqType.class);
```

4.) 具体实现注解接口调用

这里采用Java动态代理机制来实现，将定义接口与实现分离开，这个后期有时间再做总结。

```
    private void testApi() {
        IReqApi api = create(IReqApi.class);
        api.login("whoislcj", "123456");
    }

    public <T> T create(final Class<T> service) {
        return (T) Proxy.newProxyInstance(service.getClassLoader(), new Class<?>[]{service},
                new InvocationHandler() {
```

```
                    @Override
                    public Object invoke(Object proxy, Method method, Object... args)
                            throws Throwable {// Annotation[]  methodAnnotations =
method.getAnnotations();//拿到函数注解数组
                        ReqType reqType = method.getAnnotation(ReqType.class);
                        Log.e(TAG, "IReqApi---reqType->" + (reqType.reqType() ==
ReqType.ReqTypeEnum.POST ? "POST" : "OTHER"));
                        ReqUrl reqUrl = method.getAnnotation(ReqUrl.class);
                        Log.e(TAG, "IReqApi---reqUrl->" + reqUrl.reqUrl());
                        Type[] parameterTypes = method.getGenericParameterTypes();
                        Annotation[][] parameterAnnotationsArray = method.getParameterAnnotations
();//拿到参数注解
                        for (int i = 0; i < parameterAnnotationsArray.length; i++) {
                            Annotation[] annotations = parameterAnnotationsArray[i];
                            if (annotations != null) {
                                ReqParam reqParam = (ReqParam) annotations[0];
                                Log.e(TAG, "reqParam---reqParam->" + reqParam.value() + "==" +
args[i]);
                            }
                        }
                        //下面就可以执行相应的网络请求获取结果 返回结果
                        String result = "";//这里模拟一个结果

                        return result;
                    }
                });
    }
```

打印结果：

```
07-15 06:59:56.308 14553-14553/com.whoislcj.testhttp E/MainActivity: IReqApi---reqType->POST
07-15 06:59:56.309 14553-14553/com.whoislcj.testhttp E/MainActivity: IReqApi---reqUrl->www.xxx.com/openApi/login
07-15 06:59:56.311 14553-14553/com.whoislcj.testhttp E/MainActivity: reqParam---reqParam->userId==whoislcj
07-15 06:59:56.311 14553-14553/com.whoislcj.testhttp E/MainActivity: reqParam---reqParam->pwd==123456
```

以上通过注解定义参数，通过动态代理方式执行函数，模拟了最基本的Retrofit 2中网络实现原理。

干我们这行，啥时候懈怠，就意味着长进的停止，长进的停止就意味着被淘汰，只能往前冲，直到凤凰涅槃的一天！

分类: Java日常总结

标签: Java注解, 注解

总李写代码
关注 - 0
粉丝 - 265
+加关注

3 0

« 上一篇：Android数据存储之GreenDao 3.0 详解
» 下一篇：Android注解使用之使用Support Annotations注解优化代码

posted @ 2016-07-15 08:14 总李写代码 阅读(13053) 评论(0) 编辑 收藏

最新IT新闻:

· 多次推迟后，SpaceX宣布将在今年11月发射猎鹰重型火箭
· 腾讯守护平台"查小号"功能正式上线
· 不信任比特币？微软推一种快速且安全的区块链技术
· 阿里将开线下购物中心猫茂 预计明年4月开业
· Ubuntu 17.10首个官方Beta版发布 各种风味版同步开放下载
» 更多新闻...

最新知识库文章:

· 做到这一点，你也可以成为优秀的程序员
· 写给立志做码农的大学生
· 架构腐化之谜
· 学会思考，而不只是编程
· 编写Shell脚本的最佳实践
» 更多知识库文章...