彻底搞懂 Java 中的注解 Annotation

沉默王二 王磊的博客 5天前

> Java注解是一系列元数据，它提供数据用来解释程序代码，但是注解并非是所解释的代码本身的一部分。注解对于代码的运行效果没有直接影响。

网络上对注解的解释过于严肃、刻板，这并不是我喜欢的风格。尽管这样的解释听起来非常的专业。

为了缓解大家对“注解”的陌生感，我来说点有意思的。其实我对“注解”这个词的第一印象并不是Java的注解，而是朱熹的名作《四书章句集注》。为什么我会有这么大的脑洞呢？因为当我试着去翻译 Annotation 这个单词的时候，得到的结果是“注释”而不是“注解”。《四书章句集注》正是朱熹对《大学》、《中庸》、《论语》、《孟子》四书做出的重要的注释。要知道，该书可是明清以后科举考试的题库和标准答案！

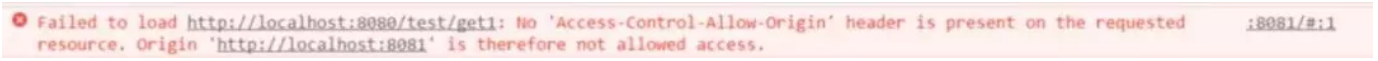
注解（ Annotation ）是在 Java SE 5.0 版本中开始引入的概念，同 class 和 interface 一样，也属于一种类型。很多开发人员认为注解的地位不高，但其实不是这样的。像 @Transactional 、 @Service 、 @RestController 、 @RequestMapping 、 @CrossOrigin 等等这些注解的使用频率越来越高。

01、为什么要使用注解呢？

为什么要使用注解呢？让我们从另外一个问题说起。

“跨域”这两个字就像一块狗皮膏药黏在每一个前端开发者的身上；我也不例外，虽然我并不是一个纯粹的前端开发者。

跨域问题的出现，源于浏览器的同源策略——限制一个源加载的脚本去访问另外一个源的资源，可有效地隔离潜在的恶意文件，是一种重要的安全机制。



跨域问题

跨域问题的解决方案也有很多，比如说：

1）JSONP

2）Nginx代理

3）"跨域资源共享"（Cross-origin resource sharing），简称 CORS ，可以说是处理跨域问题的标准做法。

记得第一次遇到跨域问题的时候，我特意向一个同学请教了解决方案，他告诉我的答案如下。

第一步，在web.xml添加filter。

```
<filter>
 <filter-name>contextfilter</filter-name>
 <filter-class>com.cmower.filter.WebContextFilter</filter-class>
</filter>
<filter-mapping>
 <filter-name>contextfilter</filter-name>
 <url-pattern>/*</url-pattern>
</filter-mapping>
```

第二步，实现WebContextFilter类。

```
public class WebContextFilter implements Filter {

   @Override
   public void destroy() {
   }

   @Override
   public void doFilter(ServletRequest request, ServletResponse response, FilterChain
        throws IOException, ServletException {
      HttpServletResponse httpServletResponse = (HttpServletResponse) response;
      httpServletResponse.setHeader("Access-Control-Allow-Origin", "*");
      httpServletResponse.setHeader("Access-Control-Allow-Headers", "accept,content
```

```
        httpServletResponse.setHeader("Access-Control-Allow-Methods", "OPTIONS,GET,POS
        chain.doFilter(request, httpServletResponse);

    }

    @Override
    public void init(FilterConfig arg0) throws ServletException {
    }

}
```

看到这样的解决方案，我真的是蛮崩溃的。不就一个跨域问题嘛，用得着这么多代码吗？

我对这样的解决方案非常的不满意。于是下定决心要好好的研究一番，大概花了半天的时间吧，我终于搞清楚了“跨域”问题，以及它的标准解决方案 CORS 。并且找到了一个极为简洁的解决方案—— @CrossOrigin ，只要在Controller类上加上这个注解，就可以轻松地解决跨域问题。

代码如下。

```
@RestController
@RequestMapping("course")
@CrossOrigin
public class CourseController {
}
```

如果没有找到 @CrossOrigin 这个注解，我真的就要按照同学提供的方案去解决跨域的问题了。但那样做就好像，我们卖掉家里的小汽车，然后出行的时候驾一辆马车一样。

这也正是我想告诉你的，为什么要使用注解的原因：它让我们的代码看起来更简洁，更有时代的进步感。

02、该如何定义注解呢？

注解需要通过 @interface 关键字（形式和接口非常的相似，只是前面多了一个 @ ）进行定义。我们可以打开 @CrossOrigin 的源码来看一下。

```
@Target({ ElementType.METHOD, ElementType.TYPE })
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface CrossOrigin {
    /**
     * List of allowed origins, e.g. {@code "http://domain1.com"}.
```

```
     * <p>These values are placed in the {@code Access-Control-Allow-Origin}
     * header of both the pre-flight response and the actual response.
     * {@code "*"} means that all origins are allowed.
     * <p>If undefined, all origins are allowed.
     * @see #value
     */
    @AliasFor("value")
    String[] origins() default {};
    /**
     * List of request headers that can be used during the actual request.
     * <p>This property controls the value of the pre-flight response's
     * {@code Access-Control-Allow-Headers} header.
     * {@code "*"}  means that all headers requested by the client are allowed.
     * <p>If undefined, all requested headers are allowed.
     */
    String[] allowedHeaders() default {};
    /**
     * List of supported HTTP request methods, e.g.
     * {@code "{RequestMethod.GET, RequestMethod.POST}"}.
     * <p>Methods specified here override those specified via {@code RequestMapping}.
     * <p>If undefined, methods defined by {@link RequestMapping} annotation
     * are used.
     */
    RequestMethod[] methods() default {};
}
```

从上面的代码可以看得出来，“注解”真的很“注解”，除了注释多和“元注解”多之外，真没有别的了。

“元注解”？什么是“元注解”呢？

“元注解”是用来注解（动词）注解（名词）的注解（名词）。请感受汉语的博大精深。 `@Target` 、 `@Retention` 和 `@Documented` 就是所谓的元注解。

1）@Target

Target是目标的意思， `@Target` 指定了注解运用的场景。都有哪些场景值呢？

- `ElementType.ANNOTATION_TYPE` ：可以给注解进行注解
- `ElementType.CONSTRUCTOR` ：可以给构造方法进行注解
- `ElementType.FIELD` ：可以给字段进行注解
- `ElementType.LOCAL_VARIABLE` ：可以给局部变量进行注解
- `ElementType.METHOD` ：可以给方法进行注解
- `ElementType.PACKAGE` ：可以给包进行注解
- `ElementType.PARAMETER` ：可以给方法内的参数进行注解
- `ElementType.TYPE` ：可以给类型进行注解，比如类、接口和枚举

2）@Retention

Retention这个单词的意思为保留期。也就是说，当 `@Retention` 应用到一个注解上的时候，它解释说明了这个注解的存活时间。来看它的取值范围。

- `RetentionPolicy.SOURCE` ：注解只在源码阶段保留，在编译器进行编译时它将被丢弃忽视。
- `RetentionPolicy.CLASS` ：注解只被保留到编译进行的时候，并不会被加载到 JVM 中。
- `RetentionPolicy.RUNTIME` ：注解可以保留到程序运行的时候，它会被加载进入到 JVM 中，所以在程序运行时可以获取到它们。

3）@Documented

`Documented` 就比较容易理解，它和文档有关。作用就是能够将注解中的元素包含到 Javadoc 中。

当我们了解了元注解的概念后，再回头看一下 `@CrossOrigin` 的源码，是不是感觉清晰多了呢？

如果能够细致地读一读源码中的注释，你就会看到WebContextFilter类中出现的关键字，诸如 `Access-Control-Allow-Origin` 、 `Access-Control-Allow-Headers` 、 `Access-Control-Allow-Methods` 。也就是说，当我们通过 `@CrossOrigin` 对Controller类注解后，SpringMVC就能够在运行时对这个类自动加上解决跨域问题的过滤器。

03、注解可以反射吗？

注解是可以通过反射获取的。

1）可以通过 Class 对象的 `isAnnotationPresent()` 方法判断该类是否应用了某个指定的注解。

2）通过 `getAnnotation()` 方法来获取注解对象。

3）当获取到注解对象后，就可以获取使用注解时定义的属性值。

示例如下：

```
@CrossOrigin(origins = "http://qingmiaokeji.com", allowedHeaders = "accept,content-t
public class TestController {
```

```
    public static void main(String[] args) {
        Class c = TestController.class;

        if (c.isAnnotationPresent(CrossOrigin.class)) {
            CrossOrigin crossOrigin = (CrossOrigin) c.getAnnotation(CrossOrigin.class);

            System.out.println(Arrays.asList(crossOrigin.allowedHeaders()));
            System.out.println(Arrays.asList(crossOrigin.methods()));
            System.out.println(Arrays.asList(crossOrigin.origins()));
        }

    }
}

// 输出：[accept,content-type]
// [GET, POST]
// [http://qingmiaokeji.com]
```

04、注解经常用在哪里呢？

1）`@Transactional` ：Spring 为事务管理提供的功能支持。

2）`@ Service` ：Spring在进行包扫描的时候，会自动将这个类注册到Spring容器中。

3）`@RestController` ：是 `@ResponseBody` 和 `@Controller` 的组合注解。

也就是说，下面这段代码与下下面的代码等同。

```
@RestController
public class HelloController {

    @RequestMapping(value="hello")
    public String sayHello(){
        return "hello";
    }
}

@Controller
@ResponseBody
public class HelloController {

    @RequestMapping(value="hello")
    public String sayHello(){
        return "hello";
    }
}
```

4）`@RequestMapping` ：Spring Web 应用程序中最常用到的注解之一，将 HTTP 请求映射到 MVC 和 REST 控制器的处理方法上。

5） `@Select` ：MyBatis提供的查询语句注解。示例如下：

```
@Select("select * from city")
List<City> getCitys();
```

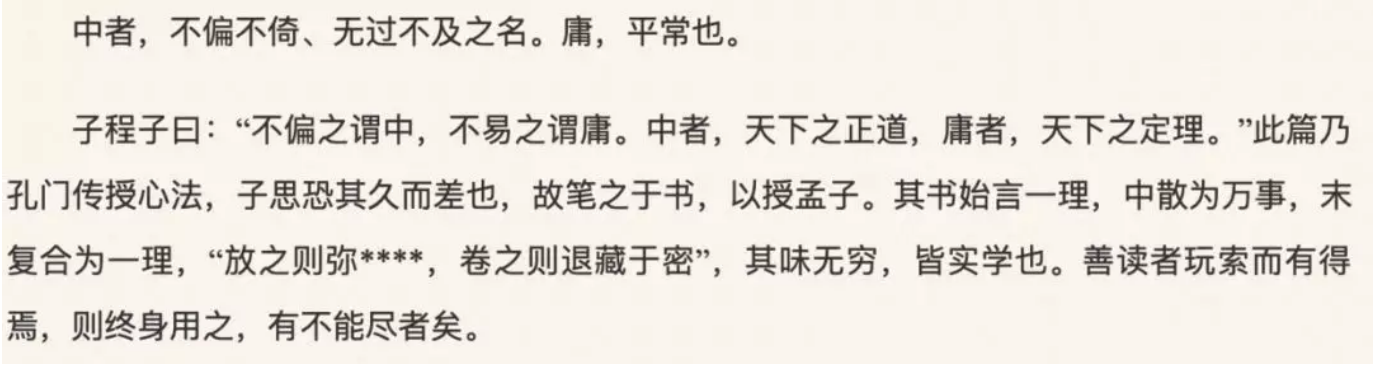
6）还有很多很多，就不再一一列举了。

最后

我想说的是，注解有许多用处，主要有：

- 提供信息给编译器： 编译器可以利用注解来探测错误和警告信息。
- 编译阶段时的处理： 软件工具可以利用注解信息来生成代码、HTML文档。
- 运行时的处理： 某些注解可以在程序运行的时候接受代码的提取。

别忘了：



四书章句集注

近期热点推荐

Java 最常见的 200+ 面试题

Java 200+ 面试题补充③ Dubbo 模块

Java 200+ 面试题补充② Netty 模块

Java 200+ 面试题补充 ThreadLocal 模块

面试题 == 和 equals 的区别

面试经验分享|精华篇

精美简历合集II

精美简历合集I