原 AspectJ 切面注解中五种通知注解：@Before、@After、@AfterRunning @AfterThrowing、@Around

2017年12月16日 23:52:27 苍鹰蛟龙 阅读数：15985

要在 Spring 中声明 AspectJ 切面, 只需要在 IOC 容器中将切面声明为 Bean 实例. 当在 Spring IOC 容器中初始化 Asp Spring IOC 容器就会为那些与 AspectJ 切面相匹配的 Bean 创建代理。
在切面类中需要定义切面方法用于响应响应的目标方法，切面方法即为通知方法，通知方法需要用注解标识，Aspect 知注解:

- @Before: 前置通知, 在方法执行之前执行
- @After: 后置通知, 在方法执行之后执行 。
- @AfterRunning: 返回通知, 在方法返回结果之后执行
- @AfterThrowing: 异常通知, 在方法抛出异常之后
- @Around: 环绕通知, 围绕着方法执行

下面分别举例5中通知方法的使用

首先建立一个目标接口ArithmeticCalculator：

```
1  package lzj.com.spring.aop;
2
3  public interface ArithmeticCalculator {
4      int add(int i, int j);
5      int div(int i, int j);
6  }
```

然后创建接口的实现类ArithmeticCalculatorIml：

```
1  package lzj.com.spring.aop;
2  import org.springframework.stereotype.Component;
3
4  @Component("arithmeticCalculator")
5  public class ArithmeticCalculatorIml implements ArithmeticCalculator {
6
7      @Override
8      public int add(int i, int j) {
9          int result = i + j;
10         System.out.println("add->result:" + result);
11         return result;
12     }
13
14     @Override
15     public int div(int i, int j) {
16         int result = i / j;
17         System.out.println("div->result:" + result);
18         return result;
19     }
20
21 }
```

配置文件bean-aop.xml：

```
<context:component-scan base-package="lzj.com.spring.aop"></context:component-scan>
```

创建测试类：

```
package lzj.com.spring.aop;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

    public static void main(String[] args) {
        ApplicationContext ctx = new ClassPathXmlApplicationContext("bean-aop.xml");
        ArithmeticCalculator arithmetic = (ArithmeticCalculator) ctx.getBean("arithmeticCalculator
        arithmetic.add(3, 2);
        arithmetic.div(4, 2);

    }

}
```

运行结果：

```
add->result:5
div->result:2
```

上面的例子把目标类注入到IOC容器中，执行时从容器中获取目标类的bean，然后调用目标方法。
下面要在目标方法的前后等执行其它操作，打印日志，不需要改变任何目标方法，只需要增加切面类，新建切面类Log
注入到IOC中，然后在切面类中定义要执行的切面方法即可。

在执行下面切面方法之前，需要先启动五种注解，配置文件中定义如下：

```
<context:component-scan base-package="lzj.com.spring.aop"></context:component-scan>
<aop:aspectj-autoproxy></aop:aspectj-autoproxy>
```

一、@Before前置通知

用@Before标识的方法为前置方法，在目标方法的执行之前执行，即在连接点之前进行执行。
示例如下：

```
package lzj.com.spring.aop;
import java.util.Arrays;
import java.util.List;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class LogProxy {

    @Before("execution(public int lzj.com.spring.aop.ArithmeticCalculator.*(int, int))")
    public void beforMethod(JoinPoint point){
```

```
        String methodName = point.getSignature().getName();
        List<Object> args = Arrays.asList(point.getArgs());
        System.out.println("调用前连接点方法为：" + methodName + ",参数为：" + args);
    }

}
```

执行测试类，输出结果如下：

```
调用前连接点方法为：add,参数为：[3, 2]
add->result:5
调用前连接点方法为：div,参数为：[4, 2]
div->result:2
```

在目标方法add和div之前分别执行了前置通知方法。

二、@After后置通知方法

后置方法在连接点方法完成之后执行，无论连接点方法执行成功还是出现异常，都将执行后置方法。示例如下：

```
@Aspect
@Component
public class LogProxy {

    @After(("execution(public int lzj.com.spring.aop.ArithmeticCalculator.*(int, int))"))
    public void afterMethod(JoinPoint point){
        String methodName = point.getSignature().getName();
        List<Object> args = Arrays.asList(point.getArgs());
        System.out.println("调用后连接点方法为：" + methodName + ",参数为：" + args);
    }
}
```

执行测试类，输出结果如下：

```
add->result:5
调用后连接点方法为：add,参数为：[3, 2]
div->result:2
调用后连接点方法为：div,参数为：[4, 2]
```

发现add和div两个连接点方法执行之后都调用了后置方法。如果目标连接点方法出现异常时，也会执行后置通知方法。下：

```
public class Main {

    public static void main(String[] args) {
        ApplicationContext ctx = new ClassPathXmlApplicationContext("bean-aop.xml");
        ArithmeticCalculator arithmetic = (ArithmeticCalculator) ctx.getBean("arithmeticCalculat
        arithmetic.add(3, 2);
        /*被除数为0,会抛出异常*/
        arithmetic.div(4, 0);
    }
```

执行测试方法，输出结果如下：

```
add->result:5
调用后连接点方法为：add,参数为：[3, 2]
调用后连接点方法为：div,参数为：[4, 0]
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at lzj.com.spring.aop.ArithmeticCalculatorIml.div(ArithmeticCalculatorIml.java:17)
    ……
```

从输出结果中可以看出，即使目标方法出现异常，后置通知方法依然执行。但后置通知拿不到目标方法执行后的结果，能出现异常。如果要拿目标方法的执行结果，要用下面的通知方法。

三、@AfterRunning返回通知方法

当连接点方法成功执行后，返回通知方法才会执行，如果连接点方法出现异常，则返回通知方法不执行。返回通知方功后才会执行，所以，返回通知方法可以拿到目标方法(连接点方法)执行后的结果。切面类中定义返回通知方法，示例

```
@Aspect
@Component
public class LogProxy {

    /*通过returning属性指定连接点方法返回的结果放置在result变量中，在返回通知方法中可以从result变量中获取连接点方
    @AfterReturning(value="execution(public int lzj.com.spring.aop.ArithmeticCalculator.*(int, int))",
            returning="result")
    public void afterReturning(JoinPoint point, Object result){
        String methodName = point.getSignature().getName();
        List<Object> args = Arrays.asList(point.getArgs());
        System.out.println("连接点方法为：" + methodName + ",参数为：" + args + ",目标方法执行结果为:
    }
}
```

运行测试方法，输出结果如下：

```
add-->result:5
连接点方法为：add,参数为：[3, 2],目标方法执行结果为：5
div-->result:2
连接点方法为：div,参数为：[4, 2],目标方法执行结果为：2
```

当连接点方法出现异常时，不执行返回通知方法，把测试方法该为如下：

```
public class Main {

    public static void main(String[] args) {
        ApplicationContext ctx = new ClassPathXmlApplicationContext("bean-aop.xml");
        ArithmeticCalculator arithmetic = (ArithmeticCalculator) ctx.getBean("arithmeticCalculat
        arithmetic.add(3, 2);
        arithmetic.div(4, 0);

    }

}
```

运行测试方法，输出结果如下：

```
add-->result:5
连接点方法为：add,参数为：[3, 2],目标方法执行结果为：5
Exception in thread "main" java.lang.ArithmeticException: / by zero
……
```

从输出结果可以看出，div(4,0)出现异常，因此该连接点对应的返回通知方法也不执行。

四、@AfterThrowing异常通知

异常通知方法只在连接点方法出现异常后才会执行，否则不执行。在异常通知方法中可以获取连接点方法出现的异常，知方法，示例如下：

```
/*通过throwing属性指定连接点方法出现异常信息存储在ex变量中，在异常通知方法中就可以从ex变量中获取异常信息了*/
@AfterThrowing(value="execution(public int lzj.com.spring.aop.ArithmeticCalculator.*(int, int))",
            throwing="ex")
    public void afterReturning(JoinPoint point, Exception ex){
        String methodName = point.getSignature().getName();
        List<Object> args = Arrays.asList(point.getArgs());
        System.out.println("连接点方法为：" + methodName + ",参数为：" + args + ",异常为：" + ex);
    }
```

测试方法为：

```
public class Main {
    public static void main(String[] args) {
        ApplicationContext ctx = new ClassPathXmlApplicationContext("bean-aop.xml");
        ArithmeticCalculator arithmetic = (ArithmeticCalculator) ctx.getBean("arithmeticCalculat
        arithmetic.add(3, 2);
        arithmetic.div(4, 0);
    }
}
```

执行测试方法，输出结果如下：

```
add-->result:5
连接点方法为：div,参数为：[4, 0],异常为：java.lang.ArithmeticException: / by zero
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

从输出结果中可以看出，add方法没有异常，因此不执行异常通知方法，div方法出现异常，执行科异常通知方法。
上面的例子中，异常类型设置的是Exception，表示捕获连接点方法的所有异常信息，也可以指定捕获指定类型的信息

```
@AfterThrowing(value="execution(public int lzj.com.spring.aop.ArithmeticCalculator.*(int, int))",
            throwing="ex")
    /*只捕获连接点方法中的NullPointerException 类型的异常信息*/
    public void afterReturning(JoinPoint point, NullPointerException ex){
        String methodName = point.getSignature().getName();
        List<Object> args = Arrays.asList(point.getArgs());
        System.out.println("连接点方法为：" + methodName + ",参数为：" + args + ",异常为：" + ex);
    }
```

五、@Around环绕通知

环绕通知方法可以包含上面四种通知方法，环绕通知的功能最全面。环绕通知需要携带 ProceedingJoinPoint 类型的参
须有返回值, 返回值即为目标方法的返回值。在切面类中创建环绕通知方法，示例如下：

```
@Around("execution(public int lzj.com.spring.aop.ArithmeticCalculator.*(int, int))")
    public Object aroundMethod(ProceedingJoinPoint pdj){
        /*result为连接点的放回结果*/
        Object result = null;
        String methodName = pdj.getSignature().getName();
```

```
6
7           /*前置通知方法*/
8           System.out.println("前置通知方法>目标方法名：" + methodName + ",参数为：" + Arrays.asList(pdj.getA
9
10          /*执行目标方法*/
11          try {
12              result = pdj.proceed();
13
14              /*返回通知方法*/
15              System.out.println("返回通知方法>目标方法名" + methodName + ",返回结果为：" + result);
16          } catch (Throwable e) {
17              /*异常通知方法*/
18              System.out.println("异常通知方法>目标方法名" + methodName + ",异常为：" + e);
19          }
20
21          /*后置通知*/
22          System.out.println("后置通知方法>目标方法名" + methodName);
23
24          return result;
25      }
26  }
```

测试方法为：

```
1   public class Main {
2       public static void main(String[] args) {
3           ApplicationContext ctx = new ClassPathXmlApplicationContext("bean-aop.xml");
4           ArithmeticCalculator arithmetic = (ArithmeticCalculator) ctx.getBean("arithmeticCalculat
5           arithmetic.add(3, 2);
6           arithmetic.div(4, 0);
7       }
8   }
```

运行测试方法：

```
1   public class Main {
2       public static void main(String[] args) {
3           ApplicationContext ctx = new ClassPathXmlApplicationContext("bean-aop.xml");
4           ArithmeticCalculator arithmetic = (ArithmeticCalculator) ctx.getBean("arithmeticCalculat
5           arithmetic.add(3, 2);
6           arithmetic.div(4, 0);
7       }
8   }
```

运行测试方法，输出结果：

```
1   前置通知方法>目标方法名：add,参数为：[3, 2]
2   add->result:5
3   返回通知方法>目标方法名add,返回结果为：5
4   后置通知方法>目标方法名add
5   前置通知方法>目标方法名：div,参数为：[4, 0]
6   异常通知方法>目标方法名div,异常为：java.lang.ArithmeticException: / by zero
7   后置通知方法>目标方法名div
8   Exception in thread "main" org.springframework.aop.AopInvocationException: Null return value from adv
9   iv(int,int)
10      at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:219)
11      at com.sun.proxy.$Proxy7.div(Unknown Source)
        at lzj.com.spring.aop.Main.main(Main.java:12)
```

从输出结果中可以看出，环绕通知实现了上面几种通知的结合。
当div目标方法出现异常时，在环绕通知方法中已经用try…catch方法进行捕捉了，为什么最后输出结果中还出现了一个错误：

```
Exception in thread "main" org.springframework.aop.AopInvocationException: Null return value from adv
    at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:219)
    at com.sun.proxy.$Proxy7.div(Unknown Source)
    at lzj.com.spring.aop.Main.main(Main.java:12)
```

那是因为在环绕通知方法中开始就定义了目标方法的返回结果
`Object result = null`。当目标方法出现异常时，`result = pdj.proceed();`执行时出现异常，此时result中还是n
知方法最后`return result;`时，返回的result就是null，但是环绕通知的返回类型我们定义的是Object类型的，null不
型，所以抛出了个类型转换的错误。我们可以在环绕通知方法中把异常抛出去，即为：

```
@Around("execution(public int lzj.com.spring.aop.ArithmeticCalculator.*(int, int))")
    public Object aroundMethod(ProceedingJoinPoint pdj){
        /*result为连接点的放回结果*/
        Object result = null;
        String methodName = pdj.getSignature().getName();

        /*前置通知方法*/
        System.out.println("前置通知方法>目标方法名：" + methodName + ",参数为：" + Arrays.asList(p

        /*执行目标方法*/
        try {
            result = pdj.proceed();

            /*返回通知方法*/
            System.out.println("返回通知方法>目标方法名" + methodName + ",返回结果为：" + result);
        } catch (Throwable e) {
            /*异常通知方法*/
            System.out.println("异常通知方法>目标方法名" + methodName + ",异常为：" + e);
            /*当环绕通知方法本身还有其它异常时，非连接点方法出现的异常，此时抛出来*/
            throw new RuntimeException();
        }

        /*后置通知*/
        System.out.println("后置通知方法>目标方法名" + methodName);

        return result;
    }
}
```

在输出结果中会抛出一个运行时异常`java.lang.RuntimeException`

插曲：不可以在执行目标方法时在定义result变量：

```
        ……
        /*执行目标方法*/
        try {
            Object result = pdj.proceed();
            ……
        } catch (Throwable e) {
            ……
        }
```

```
9           ……
10          return result;
```

这种方法是行不通的，在 `Object result = pdj.proceed();` 中，如果 `pdj.proceed()` 执行失败，就会被try ...catch会就不会执行定义result变量那一步了，即 `Object result` 不会执行，所以在 `return result;` 就会出现错误。

spring AOP中的AfterThrowing增强处理不能完全处理异常 阅读数 1726

原文链接：http://hi.baidu.com/skychongrichie/item/3d0ef3636ccb82037cdecc75springAOP… 博文 来自： 雕…

05 Spring Aop实例（AOP 如此简单）@Aspect、@Around 注解… 阅读数 297

导语没有什么是不可以改变的，换个角度看世界，截然不同！IoC相关的基本内容告一段落… 博文 来自： qq…

Spring（12）：使用注解（@AfterThrowing/@After/@Around）… 阅读数 838

Spring（12）：使用注解实现AOP异常抛出增强与实例 博文 来自： qq…

SpringAOP之注入AspectJ切面 阅读数 1836

SpringInAction中说：“SpringAOP构建在动态代理之上，因此，Spring对AOP的支持局限于… 博文 来自： 刘…

Spring在service层事物和@AfterThrowing添加日志冲突 阅读数 453

因为@AfterThrowing方法在service事物rollback之前执行（具体看源码），添加日志成功以… 博文 来自： s3…

基于Annotation的零配置方式--AspectJ 阅读数 5290

复习一下SpringAOP利用注解的形式，配置一个简单的@Before事前通知首先定义一个切面… 博文 来自： bol…

[Spring实战系列]（18）注解切面 阅读数 2830

使用注解来创建切面是AspectJ5所引入的关键特性。在AspectJ5之前，编写AspectJ切面需… 博文 来自： Ying

spring AOP 事务 与 Afterthrowing 冲突的解决办法 阅读数 2786

今天在开发过程中发现一个很奇怪的问题，在模拟事务回滚过程中，发现事务回滚没问题，… 博文 来自： z2…

Spring框架学习-深入理解AOP02----AOP简介，AspectJ，AOP基… 阅读数 232

Spring框架学习-深入理解AOP—AOP简介，AspectJ，AOP基于注解和XML配置（5种通知… 博文 来自： yx…

北京陈女士辞掉国企工作，用手机赚钱，全款买车买房

福溢 · 猎媒

Web前端面试指导(十二)：::before 和:before有什么区别? 阅读数 142

题目点评这个问题看来很简单，但如果之前没有琢磨这个问题，给人感觉也是门头一垂，听… 博文 来自： Ro…

:after/::after和:before/::before的异同 阅读数 1298

相同点都可以用来表示伪类对象，用来设置对象前的内容:before和::before写法是等效的;:af… 博文 来自： 若…

::after和::before的使用 阅读数 160

今天遇到个小问题要实现一个每一个词后面加上“/”但是最后一个字段不加的需求但是按照平… 博文 来自： we…

CSS 巧用 :before和:after 阅读数 320

前几天的晚上较全面的去看了下css的一些文档和资料，大部分的样式运用都没什么大问题… 博文 来自： 周…

CSS -::before 和:before有什么区别? 阅读数 260

相同点 都可以用来表示伪类对象，用来设置对象前的内容 :befor和::before写法是等效的 … 博文 来自： 半…

【Spring4.0】基于注解方式配置SpringAOP 阅读数 238

一、什么是AOP？面向侧面的程序设计（aspect-orientedprogramming，AOP，又译作面向… 博文 来自： Ch…

Spring 9：aop:config标签 阅读数 1147

aop:config标签 使用aop的专用标签来完成相关的配置.（AOP标签库） 其中主要表现是使… 博文 来自： ZJ…

spring基础知识 （20）：返回通知&异常通知&环绕通知 阅读数 5714

AspectJ支持5种类型的通知注解:@Before:前置通知,在方法执行之前执行@After:后置通知,… 博文 来自： Yel…

Spring学习总结（9）——Spring AOP总结 阅读数 7912

spring IOC和AOP是Spring框架的两大核心基石，本文将对SpringAOP做一个系统的总结。… 博文 来自： 一…

苍鹰蛟龙 关注

原创	粉丝	喜欢	评论
209	58	66	69

等级： 博客 5 访问： 27万+

积分： 4968 排名： 9492

勋章：

ing AOP 中 @Aspect 的高级用法 阅读数 1848

运算支持在切点定义中加入以下运算符进行复合运算：运算符说明&&a… 博文 来自： 生…

pring mvc配置文件中配置AOP，竟然会在执行两次before… 阅读数 433

p:aspectj-autoproxy/> <!--定义切面--> <beanid=&… 博文 来自： tia…

习笔记四（AOP中的通知参数和注解开发） 阅读数 5437

篇博客介绍了如何通过AOP来切入我们想实现的公共性的功能，这篇博客来讲一… 博文

oot（二十一）@Aspect 切面注解使用 阅读数 4682

AOP面向切面编程，可以用来配置事务、做日志、权限验证、在用户请求时做一… 博文 来自： 徐…

op by aspectj注解 (3) 万能的环绕通知 阅读数 234

以把前置后置异常都涵盖，甚至权限要更大些，可以改方法返回值甚至选择不调… 博文 来自： hai…

与基于@AspectJ的AOP 阅读数 333

点元注解：注解的注解@Retention(生命范围：源代码，class，runtime)、@Inh… 博文 来自： my…

AOP @AfterThrowing没有返回值的原因 阅读数 4489

util.Map;importjavax.servlet.http.HttpServletRequest;importorg.apache.log4j.Lo… 博文 来自： Lz…

，before,beforeClass,after, afterClass的执行顺序 阅读数 1690

Java5中的注解（annotation），以下是JUnit4常用的几个annotation： @Before… 博文 来自： 追…

AOP基于注解和配置文件方式实现前通知，后通知，环绕通… 阅读数 451

基于注解和配置文件方式实现前通知，后通知，环绕通知，异常通知，最终通知… 博文 来自： 侯…

元测试 @Before @After。。。 阅读数 7896

使用Java5中的注解（annotation），常用的几个annotation介绍： @BeforeClas… 博文 来自： u0…

最新文章

解决unable to locate package net-tools
dockerfile命令
bufio缓存读写
IO读取
fmt输出格式化IO

个人分类

spring boot 21篇
SpringMVC 19篇
spring 31篇
Mybaits 26篇
spring batch 29篇

展开

归档

2019年4月 1篇
2019年3月 2篇
2019年2月 5篇
2018年12月 6篇
2018年11月 3篇

展开

热门文章

4、@ConfigurationProperties和@EnableConfigurationProperties配合使用
阅读数 23495

AspectJ 切面注解中五种通知注解：@Before、@After、@AfterRunning、
阅读数 15850

16、动态SQL之<where>、<if>条件判断
阅读数 14517

Integer.parseInt(s)与Integer.valueOf(s)的区别详解
阅读数 10957

5、@Async注解配合@EnableAsync注解使用
阅读数 9765

最新评论

16、动态SQL之<wher...
weixin_41195786：<trim prefix="and" suffixOverrides=","></trim>了解一下，不需要额外的 1=1

10.7、spring boot的...
u010502101：[reply]qq_36759663[/reply] 兄弟，你的spring boot还有一段路要学的

10.7、spring boot的...
qq_36759663：我信你个鬼，你个糟老头子坏滴很！我看别人的Spring boot工程静态资源就是放：...

SpringCloud微服务搭建详解
u010502101：[reply]zhouixi[/reply] 只是springcloud入门文章，具体应用根据公司要求。只改一份 ...

SpringCloud微服务搭建详解
zhouixi：想问下，这跟实际的企业架构是一样的吗。这算不算上伪微服务，运维怎么部署，然后，...

最后更新：

@Aspectj注解切入无效 阅读数 474
...gmvc后，如果把<aop:aspectj-autoproxyproxy-target-class="true"/>放在ap... 博文 来自：Mr...

仅需三步，大幅度提升CTR，帮助您快速提升业务目标！
第四范式先荐帮助客户从0到1搭建推荐系统,显著提升用户活跃,留存,观看时长等重要业务指标

va注解（三） 实现Junit中的@Test、@Before、@After 阅读数 1万+
自定义注解，都只是解析了一个注解，今天要讲的junit需要三个注解，而且解析... 博文 来自：e...

spring AOP @Around @Before @After 区别 阅读数 4306
演示了springaop中@Around@Before@After三个注解的区别@Before是在所拦... 博文 来自：不...

efore和beforeclass区别 阅读数 2421
每个测试方法之前都会运行一次，只需声明成public@beforeclass在类中只运行... 博文 来自：万...

e相关 阅读数 193
中，有@BeforeClass、@AfterClass、@Before和@After几个注解，查了一下... 博文 来自：El...

识Junit基本注解@Before、@After、@Test、@BeforeC... 阅读数 63
://blog.csdn.net/zixiao217/article/details/52951679Junit官网：http://junit.org/jun... 博文 来自：xie...

如何快速提升个性化推荐效果?
先荐推荐系统适合首页,信息流,内容详情页等场景,支持网页, APP,小程序,快应用,1分钟安装5分钟上线, ...工干预推

Aspectj 切面获取方法参数并拿到具体的值 阅读数 4557
法参数的值(需要配合切面,单单的反射是静态的获取不到方法参数的值,所以需要... 博文 来自：Da...

Framework中的AOP之around通知 阅读数 9806
，您看到了如何使用SpringAOP来实现跟踪和记录方面。跟踪和记录都是“消极”... 博文 来自：xia...

AOP之AspectJ的注解方式使用 阅读数 2816
jar包，请看上一篇博客。注解：如果使用注解进行aop开发，必须进行aspectj自... 博文 来自：不...

注解基本介绍 阅读数 2万+
uallocateexternalresourcesinaBefore methodyouneedtoreleasethemafterthetestr... 博文 来自：怀...

面向切面的编程AOP、AspectJ、Spring 阅读数 2467
一个开源框架，Spring是于2003年兴起的一个轻量级的Java开发框架。Spring提... 博文 来自：我...

第四范式发布先荐推荐系统，帮助300+媒体平台实现内容升级
先荐推荐系统由国际顶尖智能推荐团队研发,三步即可完成接入,毫秒级数据更新,支持PC,WAP, APP全平台接入

笔记——spring之aop、切面类中五种通知的使用、存在... 阅读数 245
P：（AspectOrientedProgramming）面向切面编程；OOP：（ObjectOriented... 博文 来自：qq...

op 执行两次 阅读数 7523
shiro框架后，发现方法本体执行一次，aop执行两次！经过研究，是因为系统中... 博文 来自：wa...

修复工具增强版 阅读数 193万+
2018-12-20 DirectX修复工具最新版：DirectX Repair V3.8 增强版 NEW! 版本... 博文 来自：VB...

Fsmtp与pop3实现收发邮件的功能 阅读数 1万+
实现了收发邮件的功能，网上虽然已经有很多资料，但实际开发过程中还是遇到… 博文 来自：二…

x下退格时，出现^H^H^H^H的问题 阅读数 5037
去：使用stty命令修改。 2、stty命令语法 stty是linux下改变和打印终端设置的常… 博文 来自：pyt…

]工具face_recognition的安装与应用 阅读数 5万+
具face_recognition的安装与应用 博文 来自：ro…

listview，spinner控件字体大小的解决办法 阅读数 4469
ner和listview中的字体大小 需要重写这2个控件的getView函数 只有这样 才能修… 博文 来自：努…

ng in Algorithm》12.详解十一种排序算法 阅读数 1万+

test 登录 注册 ×

gson报错找不到类 阅读数 1175
一个项目的包在eclipse看可能已经被导进去了 但是可能没有添加到路径去，应该去看一下… 博文 来自：he…

【深度剖析HMM（附Python代码）】1.前言及隐马尔科夫链HMM… 阅读数 1万+
1. 前言 隐马尔科夫HMM模型是一类重要的机器学习方法，其主要用于序列数据的分析，广… 博文 来自：tos…

微信小程序（七）：仿找事吧APP附近三公里Demo 阅读数 1万+
功能点：轮播；列表，下拉刷新上拉加载更多；地图；网络请求；数据绑定等 文本仿照了 … 博文 来自：阿东

bsgs算法 阅读数 1万+
bsgs算法 bsgs算法，又称大小步算法（某大神称拔山盖世算法）。主要用来解决 A^x=B(… 博文 来自：clo…

DataTables 的 实例 《一》 阅读数 1万+
1.加载需要的js/css文件 2. function del(id){ alert(id); } var table; $(document).ready(function… 博文 来自：辛…

mybatis一级缓存(session cache)引发的问题 阅读数 2万+
mybatis一级缓存(session cache)引发的问题 博文 来自：fly…

全国省市区数据SQL - 省市区 阅读数 5万+
全国省市区SQL 博文 来自：hic…

图解opengl曲线和曲面绘制 阅读数 6350
VC6 下载 http://blog.csdn.net/bcbobo21cn/article/details/44200205 opengl环境配置 http://b… 博文 来自：bc…

centos 查看命令源码 阅读数 8万+
yum install yum-utils 设置源: [base-src] name=CentOS-5.4 - Base src - baseurl=http://va… 博文 来自：lin…

linux上安装Docker(非常简单的安装方法) 阅读数 20万+
最近比较有空，大四出来实习几个月了，作为实习狗的我，被叫去研究Docker了，汗汗！ … 博文 来自：我…

NAT方式主机与VM虚拟机ping不通解决办法 阅读数 5万+
从百度里边搜到的常规解决方法都是说什么防火墙影响，nat方式本来就ping不通等，今天… 博文 来自：…

搭建图片服务器《二》-linux安装nginx 阅读数 3万+

nginx是个好东西，Nginx (engine x) 是一个高性能的HTTP和反向代理服务器，也是一个IM… 博文 来自： m…

QEMU 简单使用 阅读数 8325

安装： yum install qemu 创建image文件： qemu-img create -f qcow2 guest.qcow2 3G 安… 博文 来自： 务…

将Excel文件导入数据库（POI+Excel+MySQL+jsp页面导入）第一… 阅读数 3万+

本篇文章是根据我的上篇博客，给出的改进版，由于时间有限，仅做了一个简单的优化。相… 博文 来自： Ly…

关于SpringBoot bean无法注入的问题（与文件包位置有关） 阅读数 17万+

问题场景描述整个项目通过Maven构建，大致结构如下： 核心Spring框架一个module sprin… 博文 来自： 开…

强连通分量及缩点tarjan算法解析 阅读数 57万+

强连通分量： 简言之 就是找环（每条边只走一次，两两可达） 孤立的一个点也是一个连通… 博文 来自： 九…

LSTM简介以及数学推导(FULL BPTT) 阅读数 8万+

前段时间看了一些关于LSTM方面的论文，一直准备记录一下学习过程的，因为其他事儿，… 博文 来自： 天…

jquery/js实现一个网页同时调用多个倒计时(最新的) 阅读数 44万+

jquery/js实现一个网页同时调用多个倒计时(最新的) 最近需要网页添加多个倒计时. 查阅网… 博文 来自： W…

servlet+jsp实现过滤器，防止用户未登录访问 阅读数 2万+

我们可能经常会用到这一功能，比如有时，我们不希望用户没有进行登录访问后台的操作页… 博文 来自： 沉…

SODBASE CEP学习（四）：类SQL语言EPL与Storm或jStorm集成 阅读数 3475

互联网+的影响力就是大，storm框架最初是设计用来做互联网文本处理和一些统计工作的工… 博文 来自： ha…

魔兽争霸3冰封王座1.24e 多开联机补丁 信息发布与收集点 阅读数 2万+

畅所欲言！ 博文 来自： S…

oracle数据库导出ORA-39127错误解决方案 阅读数 4638

错误类型及描述: expdp 导出表在表分析是开始出现报错。 ORA-39127: unexpected error fr… 博文

利用STM32F103RB进行AD采样，并且利用了DMA控制器，并且… 阅读数 1571

void ADC1_init(void) { #if auto_samp==1 ADC_InitTypeDef ADC_InitStructure; GPIO_InitT… 博文 来自： liu…

thymeleaf模板实现html5标签的非严格检查 阅读数 4万+

一、概述最近在springboot项目引入thymeleaf模板时，使用非严格标签时，运行会报错。默… 博文 来自： Lu…

Chrome的下一个里程碑 阅读数 1558

Aura 是为HTML Canvas提供的混合框架，是适用于 Chrome 和 ChromeOS 的新一代窗口… 博文 来自： Ch…

SSM框架——详细整合教程（Spring+SpringMVC+MyBatis） 阅读数 48万+

使用SSM（Spring、SpringMVC和Mybatis）已经有三个多月了，项目在技术上已经没有什… 博文 来自： 在…