目录

[Day24 2017-10-27 2](#_Toc496890301)

[Spring引入properties文件 2](#_Toc496890302)

[Spring IOC和AOP 3](#_Toc496890303)

[IOC 控制反转（Inversion of Controller） 3](#_Toc496890304)

[AOP面向切面编程（Aspet Oriented Programming） 4](#_Toc496890305)

# Day24 2017-10-27

## Spring引入properties文件

<http://blog.csdn.net/eson_15/article/details/51365707>

1. Xml文件中引用

<context:property-placeholder location=””/>

如：

<**context:property-placeholder** location="db-config.properties"/>

或者<bean/>标签

如：

<!-- 用于读取properties文件，此处将数据库连接的用户名密码等存在db-config.properties文件中 -->  
<**bean** id="propertyConfigurer"  
 class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">  
 <**property** name="locations">  
 <**list**>  
 <**value**>db-config.properties</**value**>  
 </**list**>  
 </**property**>  
</**bean**>

1. 注解方式加载

## Spring IOC和AOP

<http://blog.csdn.net/luoshenfu001/article/details/5816408/>

用注解配置Spring bean

1. Dao层

@Repository

1. 服务层

@Service

1. 控制层（页面控制器）

@Controller

@Scope：指明对象创建后的作用范围。

@Resource：在属性的setter方法上加注解指定注入的对象来源：@Resource(name = "noteDaoImpl2")，这些来源会在xml文件的包扫描（或定义的bean）中寻找。

也可以只使用@Component来指定对象的所在层，Spring会自动归类，但应该首选更具体的注解来指明组件所属层。

### IOC 控制反转（Inversion of Controller）

1. 深入理解IOC和DI（依赖注入）

<http://www.importnew.com/13619.html>

IOC和DI中包含的面向对象思想：开闭原则（O），里氏替换原则（L），依赖倒置原则（D），迪米特原则。

普通的编程模式：

public class A {

private InterfaceB cb;

public A(){

**cb = new ClassBImplementsInterfaceB();**

}

public void method(){

cd.doThings();

}

}

其中InterfaceB接口定义了doThings方法。此时类A的依赖情况为：

A即依赖于接口（InterfaceB），又同时依赖实现（ClassBImplementsInterfaceB），**因为它需要在编译阶段就确定使用哪种实现。**

**使类A对于选择哪个具体InterfaceB实现类的权利从其中移除，转交给外部决定（Spring）（控制反转）。**即它所依赖的对象由外部注入（决定）**（依赖注入）**

1. 传统编程和IoC的对比

传统编程：决定使用哪个具体的实现类的控制权在类本身，在编译阶段就确定了。

IoC模式：调用类只依赖接口，而不依赖具体的实现类，减少了耦合。控制权交给了容器，在运行的时候才由容器决定将具体的实现，动态的“注入”到调用类的对象中。

1. IOC目的

降低类间耦合，增强代码复用性；

面向接口编程，实施依赖倒置原则；

提高系统可插入、可测试和可修改的特性。

1. Spring中的IOC

IOC容器负责容纳bean，并对bean进行管理，在Spring中，BeanFactory是IOC的核心接口，其职责包括：实例化、定位、配置引用程序中的对象，并建立这些对象间的依赖。常见的BeanFactory有XMLBeanFactory。

### AOP面向切面编程（Aspet Oriented Programming）

<http://blog.csdn.net/luoshenfu001/article/details/5816408/>

AOP可以处理一些具有横切性质的系统服务，如事务处理、安全检查、缓存、对象池管理等。

面向对象编程（OOP）解决问题的重点在于对具体**领域模型的抽象**，而面向切面编程（AOP）解决问题的关键则在于对**关注点**的抽象。也就是说，系统中对于**一些需要分散在多个不相关的模块中解决的共同问题**，则交由AOP来解决；来实现松散耦合。

1. PointCut中的execution

execution(modifiers-pattern? ret-type-pattern declaring-type-pattern? name-pattern(param-pattern) throws-pattern?)

execution中可指定方法可见性、返回值、所在包、方法名、参数名、方法声明的异常，其中除了name-pattern外，其它都是可选的。

1. Adivce通知

各种类型的通知包括：

@Component  
@Aspect  
@Pointcut()  
@Around()  
@Before()  
@After()  
@AfterReturning  
@AfterThrowing

etc..

1. 使用注解配置：
2. @Component  
   @Aspect  
   public class PointCuts2 {  
     
    @Pointcut("execution(public \* com.duan.springweb.controller.\*.\*(..))")  
    public void pointcut() {  
    }  
      
    @Around("pointcut()")  
    public Object around(ProceedingJoinPoint pjp) throws Throwable {  
    Utils.*ol*.accept("Around 环绕执行之前");  
    Object obj = pjp.proceed(pjp.getArgs());  
    Utils.*ol*.accept("Around 环绕执行之后");  
    return obj;  
    }  
      
   }
3. 使用xml配置

使用AOP需要开启动态代理

<!--开启动态代理-->  
<**aop:aspectj-autoproxy**/>  
  
<**bean** id="aspect" class="com.duan.springweb.aspect.PointCuts">  
</**bean**>  
<**aop:config**>  
 <**aop:aspect** ref="aspect">  
 <**aop:pointcut** id="testAop"  
 expression="execution(public \* com.duan.springweb.controller.NoteController2.testAop(int,String))"/>  
 <**aop:after** method="after"  
 pointcut-ref="testAop"/>  
 <**aop:before** method="before"  
 pointcut-ref="testAop"/>  
 <**aop:after-returning** method="afterReturn"  
 pointcut-ref="testAop"/>  
 <**aop:after-throwing** method="afterThrowing"  
 pointcut-ref="testAop"/>  
 </**aop:aspect**>  
</**aop:config**>

# Day25 2017-10-28

周六Spring实践

# Day26 2017-10-29

## 常见HTTP错误代码

<http://blog.csdn.net/xinxin19881112/article/details/6565823>

404 请求的页面不存在

200 服务器成功返回网页

503 服务不可用

1XX 临时响应

2XX 成功处理了请求的状态代码

# Day27 2017-10-31

分配项目，分组

IT培训机构管理系统-招生子系统（组长），招聘一站式（企业，用户，伯乐）

# Day28 2017-11-01

需求分析

功能，分工，进度表，架构图，命名规范，UML图