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

# Day29 2017-11-02

数据库建表，界面

## UML类图

<http://www.uml.org.cn/oobject/201104212.asp>

## SVN

visualSVN: <http://blog.csdn.net/lincyang/article/details/5658274>



开发注意事项：

实体类需要实现Serializable

svn提交要写message

实体类空参构造函数

# Day30 2017-11-03

## javaee项目开发流程

<https://zhidao.baidu.com/question/433244940.html>

# Day31 2017-11-04

## 整合最优雅SSM框架：SpringMVC + Spring + MyBatis:

<https://www.2cto.com/kf/201606/518341.html>

D:\JetBrains\IdeaProjects\SsmTest

## 企业开发中为什么选择logback来记录日志，而不是log4j:

<http://www.cnblogs.com/rollenholt/p/3525822.html>

crul：

curl是利用URL语法在命令行方式下工作的开源文件传输工具。它被广泛应用在Unix、多种Linux发行版中，并且有DOS和Win32、Win64下的移植版本。

## HTTP请求行、请求头、请求体详解

<http://blog.csdn.net/u010256388/article/details/68491509>

http请求由三部分组成：请求行+请求头+请求体





①是请求方法，GET和POST是最常见的HTTP方法，除此以外还包括DELETE、HEAD、OPTIONS、PUT、TRACE。不过，当前的大多数浏览器只支持GET和POST，Spring 3.0提供了一个HiddenHttpMethodFilter，允许你通过“\_method”的表单参数指定这些特殊的HTTP方法（实际上还是通过POST提交表单）。服务端配置了HiddenHttpMethodFilter后，Spring会根据\_method参数指定的值模拟出相应的HTTP方法，这样，就可以使用这些HTTP方法对处理方法进行映射了。

②为请求对应的URL地址，它和报文头的Host属性组成完整的请求URL

③是协议名称及版本号。

④是HTTP的报文头，报文头包含若干个属性，格式为“属性名:属性值”，服务端据此获取客户端的信息。

⑤是报文体，它将一个页面表单中的组件值通过param1=value1&param2=value2的键值对形式编码成一个格式化串，它承载多个请求参数的数据。不但报文体可以传递请求参数，请求URL也可以通过类似于“/chapter15/user.html? param1=value1&param2=value2”的方式传递请求参数。

### 常见HTTP请求报文头属性

1. Accept：请求报文可通过一个“Accept”报文头属性告诉服务端 客户端接受什么类型的响应。

Accept属性的值可为一个或多个MIME类型（MIME：<http://en.wikipedia.org/wiki/MIME_type>  ）

1. Cookie：客户端的Cookie就是通过这个报文头属性传给服务端的，如：

Cookie: $Version=1; Skin=new;jsessionid=5F4771183629C9834F8382E23BE13C4C

服务端怎么知道客户端的请求属于哪一个session，就是通过jsessionid判断的。

1. Referer：表示这个请求是从哪个URL过来的
2. Cache-Control：对缓存进行控制，如一个请求希望响应返回的内容在客户端要被缓存一年，或不希望被缓存就可以通过这个报文头达到目的。如：

Cache-Control：no-cache （服务端将对请求返回的响应内容不再客户端进行缓存）

1. Accept-Language：客户端本地化信息
2. Content-Length：请求报文体长度
3. Content-Type：客户端发送的数据格式
4. User-Agent：产生请求的浏览器类型

HttpServletRequest.getSession()的内部机制就是获得请求报文头中Cookie属性的JSESSIONID值。

## HTTP响应报文

由三部分组成：响应行+响应头+响应体



①报文协议及版本；

②状态码及状态描述；

③响应报文头，也是由多个属性组成；

④响应报文体，即我们真正要的“干货”。

和请求报文相比，响应报文多了一个“响应状态码”，它用于告诉客户端此次请求处理结果。

HTTP响应状态码包含5中状态：

1. 1xx 消息，一般是告诉客户端，请求已经收到了，正在处理，别急...
2. 2xx 处理成功，一般表示：请求收悉、我明白你要的、请求已受理、已经处理完成等信息.
3. 3xx 重定向到其它地方。它让客户端再发起一个请求以完成整个处理。
4. 4xx 处理发生错误，**责任在客户端**，如客户端的请求一个不存在的资源，客户端未被授权，禁止访问等。
5. 5xx 处理发生错误，**责任在服务端**，如服务端抛出异常，路由出错，HTTP版本不支持等。

状态码参见：<http://en.wikipedia.org/wiki/List_of_HTTP_status_codes>

HttpServletResponse.setStatus(int sc)：可设置响应状态码。

### 常见的HTTP响应报文头属性

1. Cache-Control：响应输出到客户端后，服务端通过该报文头属告诉客户端如何控制响应内容的缓存
2. ETag
3. Location：我们在JSP中让页面Redirect到一个某个A页面中，其实是让客户端再发一个请求到A页面，这个需要Redirect到的A页面的URL，其实就是通过响应报文头的Location属性告知客户端的，如下的报文头属性，将使客户端redirect到iteye的首页中，如：

Location: <http://www.iteye.com>，这个响应报文属性将使客户端再次发送请求到指定url（即重定向）

1. Set-Cookie：服务端可以设置客户端的Cookie，其原理就是通过这个响应报文头属性实现的，如：

Set-Cookie: UserID=JohnDoe; Max-Age=3600; Version=1

更多其它的HTTP响应头报文，参见：<http://en.wikipedia.org/wiki/List_of_HTTP_header_fields>

HttpServletResponse.setHeader(String name,String value)：设置响应报文头属性。

addCookie，sendRedirect，方法内部都是修改了响应报文头。

## @RequestMapping 用法详解之地址映射

<http://www.cnblogs.com/qq78292959/p/3760560.html>

### value的uri值为以下三类：

1. 可以指定为普通的具体值；
2. 可以指定为含有某变量的一类值(URI Template Patterns with Path Variables)；

@RequestMapping(value="/owners/{ownerId}", method=RequestMethod.*GET*)  
public String findOwner(@PathVariable String ownerId, Model model) {  
 Owner owner = ownerService.findOwner(ownerId);  
 model.addAttribute("owner", owner);  
 return "displayOwner";  
}

1. 可以指定为含正则表达式的一类值( URI Template Patterns with Regular Expressions);

@RequestMapping("/spring-web/{symbolicName:[a-z-]+}-{version:\d\.\d\.\d}.{extension:\.[a-z]}")  
 public void handle(@PathVariable String version, @PathVariable String extension) {  
 // ...  
 }  
}

### parames和headers示例：

仅处理请求中包含了名为“myParam”，值为“myValue”的请求；

@RequestMapping(value = "/pets/{petId}", method = RequestMethod.*GET*, params="myParam=myValue")  
public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {  
 // implementation omitted  
}

仅处理request的header中包含了指定“Referer”请求头和对应值为“http://www.ifeng.com/”的请求。即请求来自于指定网址。

@RequestMapping(value = "/pets", method = RequestMethod.*GET*, headers="Referer=http://www.ifeng.com/")  
public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {  
 // implementation omitted  
}

## @RequestBody, @ResponseBody 注解详解

<http://www.cnblogs.com/qq78292959/p/3760651.html>

### @RequestBody

作用：

该注解用于读取Request请求的body（**请求体）部分数据**，使用系统默认配置的HttpMessageConverter进行解析，然后把相应的数据绑定到要返回的对象上；再把HttpMessageConverter返回的对象数据绑定到 **controller中方法的参数**上。

说明：request的body部分的数据编码格式由header请求头部分的Content-Type指定；

### @ResponseBody

作用：

该注解用于将**Controller的方法返回的对象**，通过适当的HttpMessageConverter转换为**指定格式**后，写入到**Response对象的body数据区**。

# Day32 2017-11-06

Java WEB高并发秒杀系统: <https://github.com/codingXiaxw/seckill>

## mybatis中的CDATA标签的用法

<http://kisuntech.blog.51cto.com/8003773/1313797>

术语 CDATA 指的是不应由 XML 解析器进行解析的文本数据（Unparsed Character Data）。CDATA 部分由 "<![CDATA[" 开始，由 "]]>" 结束。

# Day33 2017-11-07

## Java Web开发 之VO、PO、TO、SOA、DTO等收集

<http://blog.csdn.net/scholar_man/article/details/48287571>

## 为什么 Java 中要使用 Checked Exceptions

<http://www.iteye.com/topic/2038>

# Day34 2017-11-08

# Day35 2017-11-09