目录

[Day24 2017-10-27 4](#_Toc499622068)

[Spring引入properties文件 4](#_Toc499622069)

[Spring IOC和AOP 5](#_Toc499622070)

[IOC 控制反转（Inversion of Controller） 6](#_Toc499622071)

[AOP面向切面编程（Aspet Oriented Programming） 7](#_Toc499622072)

[Day25 2017-10-28 9](#_Toc499622073)

[Day26 2017-10-29 9](#_Toc499622074)

[常见HTTP错误代码 9](#_Toc499622075)

[Day27 2017-10-31 9](#_Toc499622076)

[Day28 2017-11-01 10](#_Toc499622077)

[Day29 2017-11-02 10](#_Toc499622078)

[UML类图 10](#_Toc499622079)

[SVN 10](#_Toc499622080)

[Day30 2017-11-03 11](#_Toc499622081)

[javaee项目开发流程 11](#_Toc499622082)

[Day31 2017-11-04 11](#_Toc499622083)

[整合最优雅SSM框架：SpringMVC + Spring + MyBatis: 11](#_Toc499622084)

[企业开发中为什么选择logback来记录日志，而不是log4j: 11](#_Toc499622085)

[HTTP请求行、请求头、请求体详解 12](#_Toc499622086)

[常见HTTP请求报文头属性 13](#_Toc499622087)

[HTTP响应报文 14](#_Toc499622088)

[常见的HTTP响应报文头属性 15](#_Toc499622089)

[@RequestMapping 用法详解之地址映射 15](#_Toc499622090)

[value的uri值为以下三类： 16](#_Toc499622091)

[parames和headers示例： 16](#_Toc499622092)

[@RequestBody, @ResponseBody 注解详解 17](#_Toc499622093)

[@RequestBody 17](#_Toc499622094)

[@ResponseBody 17](#_Toc499622095)

[Day32 2017-11-06 17](#_Toc499622096)

[mybatis中的CDATA标签的用法 17](#_Toc499622097)

[Day33 2017-11-07 18](#_Toc499622098)

[Java Web开发 之VO、PO、TO、SOA、DTO等收集 18](#_Toc499622099)

[为什么 Java 中要使用 Checked Exceptions 18](#_Toc499622100)

[Day34 2017-11-08 18](#_Toc499622101)

[Day35 2017-11-09 18](#_Toc499622102)

[Day36 2017-11-10 18](#_Toc499622103)

[设计模式-工厂模式 18](#_Toc499622104)

[Day37 2017-11-11 19](#_Toc499622105)

[Day38 2017-11-13 19](#_Toc499622106)

[Day39 2017-11-14 19](#_Toc499622107)

[MySQL CRUD 19](#_Toc499622108)

[插入 insert 19](#_Toc499622109)

[删除 20](#_Toc499622110)

[Day40 2017-11-15 21](#_Toc499622111)

[Shiro 21](#_Toc499622112)

[Shiro简介——《跟我学Shiro》 21](#_Toc499622113)

[${pageContext.request.contextPath} 23](#_Toc499622114)

[springMVC注解 23](#_Toc499622115)

[Day41 2017-11-16 24](#_Toc499622116)

[组合的@RequestMapping变种 24](#_Toc499622117)

[Spring常用注解汇总 24](#_Toc499622118)

[lombok 24](#_Toc499622119)

[Day42 2017-11-17 25](#_Toc499622120)

[Spring4.0 @RestController 25](#_Toc499622121)

[REST（*Representational State Transfer*）表述性状态转移 25](#_Toc499622122)

[304 25](#_Toc499622123)

[Hibernate ORM（对象关系映射框架） 25](#_Toc499622124)

[Hibernate使用步骤 26](#_Toc499622125)

[JAVA事务,JTA,JDBC,JDO,DAO,JNDI概念 26](#_Toc499622126)

[Day43 2017-11-20 27](#_Toc499622127)

[Day44 2017-11-21 27](#_Toc499622128)

[Day45 2017-11-22 28](#_Toc499622129)

[java stackoverflowerror与outofmemoryerror区别 28](#_Toc499622130)

[stackOverflower 28](#_Toc499622131)

[Outofmemory 28](#_Toc499622132)

[Day46 2017-11-23 29](#_Toc499622133)

[Day47 2017-11-24 29](#_Toc499622134)

[Day48 2017-11-25 29](#_Toc499622135)

[四种常见的 POST 提交数据方式 29](#_Toc499622136)

[application/x-www-form-urlencoded 29](#_Toc499622137)

[multipart/form-data 30](#_Toc499622138)

[applation/json 31](#_Toc499622139)

[test/xml 31](#_Toc499622140)

# 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，spring aop，aspectj区别与联系<http://blog.csdn.net/pingnanlee/article/details/38845955>

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。

主要功能为：日志记录，性能统计，安全控制，事务处理，异常处理等。对这些行为进行分离，将其独立到非指导业务逻辑的方法中，从而改进这些行为的时候不影响业务逻辑。

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

完成【人才管理系统】项目

# Day35 2017-11-09

完成【人才管理系统】项目

# Day36 2017-11-10

Idea Intellij多线程调试

<http://blog.csdn.net/fuzzytalker/article/details/50925218>

## 设计模式-工厂模式

<http://blog.csdn.net/jason0539/article/details/23020989>

简单工厂：只能生产具体的一种产品

工厂方法：抽象根工厂，导出更多具体生产某一种产品的工厂

抽象工厂：多个抽象产品类，每个抽象产品类可以派生出多个具体产品类。

一个抽象工厂类，可以派生出多个具体工厂类。

# Day37 2017-11-11

完成【人才管理系统】项目

# Day38 2017-11-13

完成【人才管理系统】项目

# Day39 2017-11-14

完成【人才管理系统】项目

## MySQL CRUD

<https://www.2cto.com/database/201605/509500.html>

MySQL.doc

### 插入 insert

1. 插入完整一行

Auto\_increment的列插入用null代替，对于允许null或设定了default的列，插入时可省略值（即传null）。

insert into tableName(…) values(…)

提高整体性能：降低插入语句优先级

如果数据库被多个用户访问，查询操作应优先于插入操作使可使用low\_priroity降低插入语句执行的优先级。

Insert low\_priroity into …

1. 插入行的一部分

insert into book(name,number) values(‘jom’,23)

1. 插入多行

insert into book(book\_id,name,number,start\_time,end\_time,create\_time)

-> values

-> (null,'jim',23,null,null,null),

-> (null,'jack',233,null,null,null);

MySQL使用单条insert插入多条数据比使用多条insert要快。

1. 插入某些查询结果（插入检索出的数据）

insert into book(name,number) select name,number from book where number >= 300

### 删除

delete from book where book\_id > 1007

1）除非确实打算更新和删除每一行，否则绝对不要使用不带 where 子句的update 和 delete语句；

2）保证每个表都有主键；

3）在对update 和 delete 语句使用 where子句前，应该先用 select 进行测试，保证它过滤的是正确记录，以防编写的where子句不正确；

4）使用强制实施引用完整性的数据库，这样MySQL将不允许删除具有与其他表相关联的数据的行；

# Day40 2017-11-15

## Shiro

### Shiro简介——《跟我学Shiro》

<http://jinnianshilongnian.iteye.com/blog/2018936/>

认证、授权、加密、会话管理、与Web集成、缓存等



Authentication：身份认证，登陆

Authorization：权限验证，授权

Session Manager：会话管理

Cryptography：加密，数据安全性

Web Support：Web支持，易于集成到Web环境

Caching：缓存

Concurrency：多线程并发验证

Testing，Run As，Remember Me。



Subject：主体，任何可以与应用交互的“用户”

SecurityManager：Subject是门面，SercurityManager才是正真做事的人

Realm：域，安全数据源

也就是说对于我们而言，最简单的一个Shiro应用：

1、应用代码通过Subject来进行认证和授权，而Subject又委托给SecurityManager；

2、我们需要给Shiro的SecurityManager注入Realm，从而让SecurityManager能得到合法的用户及其权限进行判断。



SessionDao：用户会话的dao（CRUD）

CacheManager：缓存管理器

## ${pageContext.request.contextPath}

Jsp中使用${pageContext.request.contextPath}获取绝对路径。

## @ModelAttribute

@ModelAttrbuite：<http://hbiao68.iteye.com/blog/1948380> <http://blog.csdn.net/hejingyuan6/article/details/49995987>

1 绑定请求参数到模型对象（Model），放在处理方法形参上时，用于将多个请求参数绑定到一个模型对象，从而简化绑定流程。

如：参数中传入 (Model model,int id)：model.addAttruibute(“id”,23);

可简化为：(@ModelAttribute(“id”) int id)

**将数据保存到request域中**

2 将@RequestMapping方法返回值绑定到模型对象中，暴露返回值。

@ModelAttribute(“data”)

public Map<String,String> getData(){

….

}

所在页面控制器的同级方法被访问时该方法将首先调用，在jsp中就可通过${data}使用返回的数据。

如果不指明添加到模型对象中数据的名字，则使用返回类型的首字母小写类名代替：

@ModelAttribute

Public List<String> getData(){

…

}

Jsp中使用${list}使用，**最好指定，也是将数据保存到request域中。**

使用@ModelAttribute将公用的取数据的方法返回值传到页面，不用在每一个controller方法通过Model将数据传到页面。

# Day41 2017-11-16

## 组合的@RequestMapping变种

<https://www.baidu.com/link?url=7UYXPYZVvY14sak36CAjFzHATUvRQFciEdQzVm4biUZ9rn_QcTTiWAhlE8H-5VHKgME_l-9KgREzPOK3eLafF_&wd=&eqid=d414aaae0000b214000000055a0d6189>

@GetMapping

@PostMapping

@DeleteMapping

@PatchMapping

## Spring常用注解汇总

<https://www.cnblogs.com/xingzc/p/5777814.html>

## lombok

<http://blog.csdn.net/zhglance/article/details/54931430>

<https://projectlombok.org/features/all>

项目中使用的bean，entity等类，绝大部分需要getter，setter，toString，equals和hashcode方法，虽然IDE能快速生成方法，但一旦类有所改动，就需要对这些代码进行大量修改，使用了lombok提供的注解就可以解决这些问题。

@Setter，@getter，@ToString，@RequiredArgsConstructor，@EqualsAndHashCode，@NoArgsConstructor或@Data。

@Data相当于其它6个注解的合集

# Day42 2017-11-17

## Spring4.0 @RestController

<http://wiselyman.iteye.com/blog/2002446>

## REST（Representational State Transfer）表述性状态转移

<https://www.cnblogs.com/loveis715/p/4669091.html>

spring4.0新增了@ResultController注解，继承自@Controller注解，该注解是专门为REST服务而设计的。

@RestController本身被@Controller和@ResponseBody注解。

## 304

<https://www.douban.com/note/161120791/>

304 的标准解释是：Not Modified 客户端有缓冲的文档并发出了一个条件性的请求（一般是提供If-Modified-Since头表示客户只想比指定日期更新的文档）。服务器告诉客户，原来缓冲的文档还可以继续使用。

如果客户端发起一个文件请求的时候，发现缓存的文件有Last modified，那么在请求中会包含if modified since，这个时间就是缓存文件的最后修改时间。因此，如果请求中包含 If Modified Since，就说明已经有缓存在客户端。只要判断这个时间和当前请求的文件的修改时间就可以确定是返回 304 还是 200 。

对于静态文件，如图片，css等，服务器会自动完成Last modified和if modified since的比较，完成缓存或者更新。

## Hibernate ORM（对象关系映射框架）

<http://hibernate.org/>

java应用程序与数据库交互



开源和轻量级、快速、数据库独立查询（HQL），自动创建表、简化复杂连接、提供查询统计和数据库状态。



JNDI：java命名目录接口

JTA：java事务API

SessionFactory是ConnectionProvider（连接提供者）的会话和客户端工厂，SessionFactory提供了方法来获得Session。

连接提供者(ConnectionProvider)：它是一个JDBC连接工厂。 它从DriverManager或DataSource抽象出来的应用程序。 它是一个可选项。

### Hibernate使用步骤

创建java项目，添加jar依赖，创建持久类，创建持久类的映射文件，创建配置文件，创建检索或存储持久对象的类，运行程序。

## JAVA事务,JTA,JDBC,JDO,DAO,JNDI概念

<http://dyldragon.iteye.com/blog/789374>

通常认为事务仅与数据库相关，事务必须服从ISO/IEC指定的ACDI原则：

ACDI：原子性，一致性，隔离性，持久性。

1. 什么是java事务
2. 为什么需要
3. java事务的类型

有三种：JDBC事务，JTA（java Transcation API）事务，容器事务

JDBC事务：由java.sql.Connection对象控制，可通过如下方法控制事务：

setAutoCommit，getAutoCommit，commit，rollback。

JDBC事务不能跨越多个数据库。

JTA事务：高层的，与实现无关，与协议无关的。JTA允许应用程序执行分布式事务处理。

容器事务：容器事务主要是J2EE应用服务器提供的，容器事务大多是基于JTA完成，这是一个基于JNDI的，相当复杂的API实现。这项功能由J2EE应用服务器提供。这使得我们可以简单的指定将哪个方法加入事务，一旦指定，容器将负责事务管理任务。

1. 几种事务的差异

1) JDBC事务控制的局限性在一个数据库连接内，但是其使用简单。

2) JTA事务的功能强大，事务可以跨越多个数据库或多个DAO，使用也比较复杂。

3) 容器事务，主要指的是J2EE应用服务器提供的事务管理，局限于EJB应用使用。

# Day43 2017-11-20

# Day44 2017-11-21

请假

# Day45 2017-11-22

设计模式：单例+装饰

## java stackoverflowerror与outofmemoryerror区别

<http://blog.csdn.net/chenchaofuck1/article/details/51144223>

### stackOverflower

java程序启动一个新的线程时，JVM会为其分配一个栈，java栈以帧为单位保持线程运行状态，当线程调用一个方法是，jvm压入一个新的栈帧到这个线程的栈中，只要这个方法还没返回，这个栈帧就存在。 如果方法的嵌套调用层次太多(如递归调用),随着java栈中的帧的增多，最终导致这个线程的栈中的所有栈帧的大小的总和大于-Xss设置的值，而产生StackOverflowError溢出异常。

### Outofmemory

JVM抛出OutOfMemoryError前，会尝试进行一次Full GC，如果GC后可用内存还是不足，才会抛出OutOfMemoryError。因此，这时程序猿必然无法主动处理这一问题，只能等程序崩溃后再去查证原因。

1. 栈内存溢出

java程序启动一个新线程时，没有足够的空间为该线程分配java栈，一个线程java栈的大小由-Xss设置决定；JVM则抛出OutOfMemoryError异常。

1. 堆内存溢出

java堆用于存放对象的实例，当需要为对象的实例分配内存时，而堆的占用已经达到了设置的最大值(通过-Xmx)设置最大值，则抛出OutOfMemoryError异常。

1. 方法区内存溢出

方法区用于存放java类的相关信息，如类名、访问修饰符、常量池、字段描述、方法描述等。在类加载器加载class文件到内存中的时候，JVM会提取其中的类信息，并将这些类信息放到方法区中。

当需要存储这些类信息，而方法区的内存占用又已经达到最大值（通过-XX:MaxPermSize）；将会抛出OutOfMemoryError异常对于这种情况的测试，基本的思路是运行时产生大量的类去填满方法区，直到溢出。这里需要借助CGLib直接操作字节码运行时，生成了大量的动态类。

# Day46 2017-11-23

设计模式：享元+代理

# Day47 2017-11-24

观察者模式

# Day48 2017-11-25

## 四种常见的 POST 提交数据方式

<https://imququ.com/post/four-ways-to-post-data-in-http.html>

HTTP/1.1 协议规定的 HTTP 请求方法有 OPTIONS、GET、HEAD、POST、PUT、DELETE、TRACE、CONNECT 这几种。其中 POST 一般用来向服务端提交数据.

### application/x-www-form-urlencoded

最常见的 POST 提交数据的方式了。浏览器的原生 <form> 表单，如果不设置 enctype 属性，那么最终就会以 application/x-www-form-urlencoded 方式提交数据。

请求类似于下面这样（无关的请求头在本文中都省略掉了）：

POST http://www.example.com HTTP/1.1

Content-Type: application/x-www-form-urlencoded;charset=utf-8

title=test&sub%5B%5D=1&sub%5B%5D=2&sub%5B%5D=3

提交的数据按照 key1=val1&key2=val2 的方式进行编码，key 和 val 都进行了 URL 转码。

很多时候，我们用 Ajax 提交数据时，也是使用这种方式。

### multipart/form-data

使用表单上传文件时，必须让 <form> 表单的 enctype 等于 multipart/form-data。

POST http://www.example.com HTTP/1.1

Content-Type:multipart/form-data; boundary=----WebKitFormBoundaryrGKCBY7qhFd3TrwA

------WebKitFormBoundaryrGKCBY7qhFd3TrwA

Content-Disposition: form-data; name="text"

title

------WebKitFormBoundaryrGKCBY7qhFd3TrwA

Content-Disposition: form-data; name="file"; filename="chrome.png"

Content-Type: image/png

PNG ... content of chrome.png ...

------WebKitFormBoundaryrGKCBY7qhFd3TrwA—

首先生成了一个 boundary 用于**分割**不同的字段，为了避免与正文内容重复，boundary 很长很复杂。然后 Content-Type 里指明了数据是以 **multipart/form-data** 来编码，本次请求的 boundary 是什么内容。消息主体里按照字段个数又分为多个结构类似的部分，每部分都是以 --boundary 开始，紧接着是**内容描述信息**，然后是回车，最后是字段具体内容（**文本或二进制**）。如果传输的是文件，还要包含文件名和文件类型信息。消息主体最后以 --boundary-- 标示结束。

这种方式一般用来上传文件。

上面提到的这两种 POST 数据的方式，都是浏览器原生支持的，而且现阶段标准中原生 <form> 表单也只支持这两种方式（通过 <form> 元素的 enctype 属性指定，默认为 application/x-www-form-urlencoded。其实 enctype 还支持 text/plain，不过用得非常少）。

### applation/json

JSON 格式支持比键值对复杂得多的结构化数据，这一点也很有用。可以方便的提交复杂的结构化数据，特别适合 RESTful 的接口。

POST http://www.example.com HTTP/1.1 Content-Type: application/json;charset=utf-8

{"title":"test","sub":[1,2,3]}

### test/xml

XML-RPC（XML Remote Procedure Call）。它是一种使用 HTTP 作为传输协议，XML 作为编码方式的远程调用规范。

# Day49 2017-11-28

## JSP 9大内置对象

1. request请求对象，类型 javax.servlet.ServletRequest，作用域 Request。
2. response响应对象，类型 javax.servlet.SrvletResponse，作用域 Page。
3. pageContext页面上下文对象，类型 javax.servlet.jsp.PageContext，作用域 Page。
4. session 会话对象，类型 javax.servlet.http.HttpSession，作用域 Session。
5. application应用程序对象，类型 javax.servlet.ServletContext，作用域 Application。
6. out 输出对象，类型 javax.servlet.jsp.JspWriter，作用域 Page。
7. config 配置对象，类型 javax.servlet.ServletConfig，作用域 Page。
8. page，页面对象，类型 javax.lang.Object，作用域 Page。
9. exception例外对象，类型 javax.lang.Throwable，作用域 page。

## Spring IOC原理

<https://www.cnblogs.com/ITtangtang/p/3978349.html>



BeanFactory作为最顶层的一个接口类，它定义了IOC容器的基本功能规范。

最终的默认实现类是 DefaultListableBeanFactory，他实现了所有的接口。

要知道工厂是如何产生对象的，我们需要看具体的IOC容器实现，spring提供了许多IOC容器的实现。比如XmlBeanFactory，ClasspathXmlApplicationContext等。如果说XmlBeanFactory是容器中的屌丝，ApplicationContext应该算容器中的高帅富。

Bean 的解析主要就是对 Spring 配置文件的解析。这个解析过程主要通过下图中的类完成：

