1. **springmvc 上传图片的处理。**

a.前端jsp form 窗体指定 enctype = ‘multipart/form’.

b.对应接收参数的dto中。图片的类型为 private MultipartFile imgFile;

c.要在配置文件中配置springmvc的图片上传解析器

<!-- 配置文件上传解析器 -->  
<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">  
 <!-- 指定所上传文件的总大小不能超过20M。注意maxUploadSize属性的限制不是针对单个文件，而是所有文件的容量之和 -->  
 <property name="maxUploadSize" value="20000000"/>  
 <property name="defaultEncoding" value="utf-8"></property>  
</bean>

D.在service中解析文件并保存。

public boolean adAdd(AdDto adDto) {  
 Ad ad = new Ad();  
 ad.setTitle(adDto.getTitle());  
 ad.setLink(adDto.getLink());  
 ad.setWeight(adDto.getWeight());

if(null != adDto.getImgFile() && adDto.getImgFile().getSize() >0){  
 String filename = System.*currentTimeMillis*() + "\_"+adDto.getImgFile().getOriginalFilename();  
 String fileSavePath = "D:\\workspace\\ssm\\upload\\ad";  
 File adFile = new File(fileSavePath+filename);  
 File derectory = new File(fileSavePath);  
 if(!derectory.exists()){  
 derectory.mkdirs();  
 }  
 try {  
 adDto.getImgFile().transferTo(adFile);  
 ad.setImgFileName(filename);  
 adDao.insert(ad);  
 return true;  
 } catch (IOException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }else{  
 return false;  
 }  
  
}

主要用 MultiPartFile的 transferto 将接收的图片保存到指定文件中。

adDto.getImgFile().transferTo(adFile);

对应资源访问路径和保存路径，由于访问的方式有多种。

通过前台，后台管理程序，cdn等多种方式。

通过request是不能知道的，所以将路径保证在配置文件中，并对应测试环境，现网环境，要分不同的配置文件。在打包时，选择不同的文件发布。

1. spring 加载配置文件，只在第一次加载，后面再配置就不加载了。

所以将配置文件放在一起，用\*通配符去第一次加载所有的。

1. spring读配置文件的信息。@Value

@Value(${key})(peopertites的key值。)

Private String path;

1. 返回码和返回信息可以用枚举
2. 用封装类型的好处。

不用INT，用INTERGER

主要是能保存null，且不会报错。

1. 共同js类，可以用命令空间的写法，防止与第三方js插件同名的命名空间进行污染

Var common = window.commom || {}; js立即执行，闭包

7．Spring 提供BeanUtils.copyProperties(adDto,conditon);实现将addto中的属性copy到condition中。

8．分页标签的制作，自定义标签。

分页功能，首先需要一个Page 类 定义当前页，总页数，总记录数，每页展示的数据等。

然后可以用一个interrupt拦截器，拦截分页查询，自动sql语句加上分页限制条件。

对于第几页的，总页数，前端的页脚等问题，可以用一个自定义标签来制作。

这些都是底层自己手动 实现分页功能。

前端提供了很多 分页插件 来实现该功能。

9.jsp加载顺序

Jsp会从上到下加载 整个文件。

对应引用的js,不会自动执行的，只有在调用方法时才会执行JS。

$(**function**() {  
 //common.showMessage($("#message").val());  
 alert(2);  
});

该写法会在加载完jsp页面后，自执行该函数。

10.对于公用的属性，可以封装起来，用其他的类去继承。

11.自定义标签 page.tag 完成分页页脚

<%@**tag** language="java" pageEncoding="UTF-8" %>  
<%@**attribute** name="page" type="com.cxy.bean.Page" required="true" %>  
<%@**attribute** name="jsMethodName" type="java.lang.String" required="true" %>  
  
<script>  
 **function** transCurrentPage(currentPage) {  
 **var** rule = /^[0-9]\*[1-9][0-9]\*$/;  
 **if**(!rule.test(currentPage)){  
 currentPage = 1;  
 }  
 eval("**${**jsMethodName**}**(currentPage)");  
 }  
</script>  
<div class="page fix">  
 <a href="javascript:transCurrentPage('1');" class="first">首页</a>  
 <a href="javascript:transCurrentPage(**${**page.currentPage**}** - 1);" class="pre">上一页</a>  
 当前第<span>**${**page.currentPage**}**/**${**page.totalPage**}**</span>页  
 <a href="javascript:transCurrentPage('**${**page.currentPage + 1**}**');" class="next">下一页</a>  
 <a href="javascript:transCurrentPage('**${**page.totalPage**}**');" class="last">末页</a>  
 跳至 &nbsp;<input id="currentPageText" value="1" class="allInput w28" type="text"/>&nbsp;页 &nbsp;  
 <a href="javascript:transCurrentPage($('#currentPageText').val());" class="go">GO</a>  
</div>

可以自己传参数<%@**attribute** name="jsMethodName" type="java.lang.String" required="true" %>

调用该tag的时候，必须传入该参数。

Jsp界面使用tagdir引入自定义tag

<%@ **taglib** prefix="t" tagdir="/WEB-INF/tags"%>

调用标签。

<**t:page** jsMethodName="search" page="**${**searchParam.page**}**"></**t:page**>

对于分页的每页显示多少数据。

可以从前台传入，还可以放在配置文件当中。

JsonInclude （Include.NOT\_NULL）

空的不转化成json数据中的内容。

12.mybatis分页拦截器

要在mybatis.xml文件中加入配置

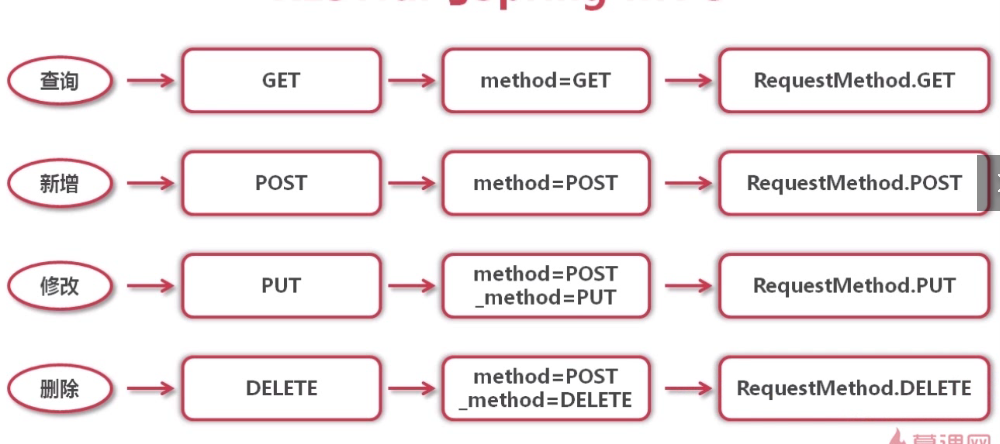
<plugins>  
 <plugin interceptor="org.imooc.dao.interceptor.PageInterceptor"/>  
</plugins>

然后编写拦截器

@Intercepts({@Signature(type=StatementHandler.class,method="prepare",args={Connection.class})})  
public class PageInterceptor implements Interceptor{  
  
 public Object intercept(Invocation arg0) throws Throwable {  
 StatementHandler statementHandler = (StatementHandler)arg0.getTarget();  
 MetaObject metaObject = MetaObject.*forObject*(statementHandler, SystemMetaObject.*DEFAULT\_OBJECT\_FACTORY*, SystemMetaObject.*DEFAULT\_OBJECT\_WRAPPER\_FACTORY*,new DefaultReflectorFactory());  
 MappedStatement mappedStatement = (MappedStatement)metaObject.getValue("delegate.mappedStatement");  
 String id = mappedStatement.getId();  
 if(id.endsWith("ByPage")) {  
 BoundSql boundSql = statementHandler.getBoundSql();  
 String sql = boundSql.getSql();  
 String countSql = "select *count*(\*) from(" + sql + ")t";  
 Connection conn = (Connection)arg0.getArgs()[0];  
 PreparedStatement statement = conn.prepareStatement(countSql);  
 ParameterHandler parameterHandler = (ParameterHandler)metaObject.getValue("delegate.parameterHandler");  
 parameterHandler.setParameters(statement);  
 ResultSet rs = statement.executeQuery();  
 BaseBean bean = (BaseBean)boundSql.getParameterObject();  
 Page page = bean.getPage();  
 if(rs.next()) {  
 page.setTotalNumber(rs.getInt(1));  
 }  
 String pageSql = sql + " limit " + (page.getCurrentPage() - 1) \* page.getPageNumber() + "," + page.getPageNumber();  
 metaObject.setValue("delegate.boundSql.sql", pageSql);  
 }  
 return arg0.proceed();  
 }  
  
 public Object plugin(Object arg0) {  
 return Plugin.*wrap*(arg0, this);  
 }  
  
 public void setProperties(Properties arg0) {  
   
 }  
  
}

13.restFul风格的URL，代表的是资源。

操作 方法 form表单 springmvc指定的方法



Put 和delete是form表单要加一个hidden的标签，指定\_method 为put/delete。

要想映射 form表单的delete和put到对应的controller ,还需要加过滤器fiter，过滤请求，映射form中的delete和put请求到相应的接口方法。

在web.xml中 配置 HideenHttpmethodFilter.

对于get方法，提交表单的时候，参数会拼接在路径中，很可能出现乱码，这是需要设定Tomcat中的 connector 的URIEncoding = “UTF-8.

Springmvc 在对应restful接口时。

如果 put 修改 的带有文件的内容时。会先用HideenHttpmethodFilter解析参数，由于带有文件multipart的，解析不出来。

过滤器会在文件上传解析器前执行，

这时在解析文件就没有意义了。

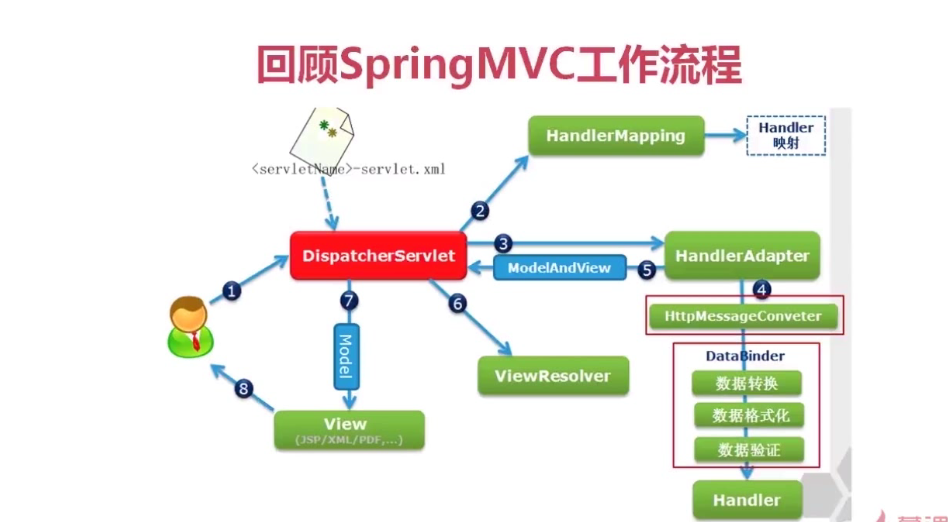
在HideenHttpmethodFilter过滤器执行之前，在加一个解析上传文件的解析文件。

MultiPartFilter。在该过滤器中初始化参数 解析文件的拦截器。

示例；

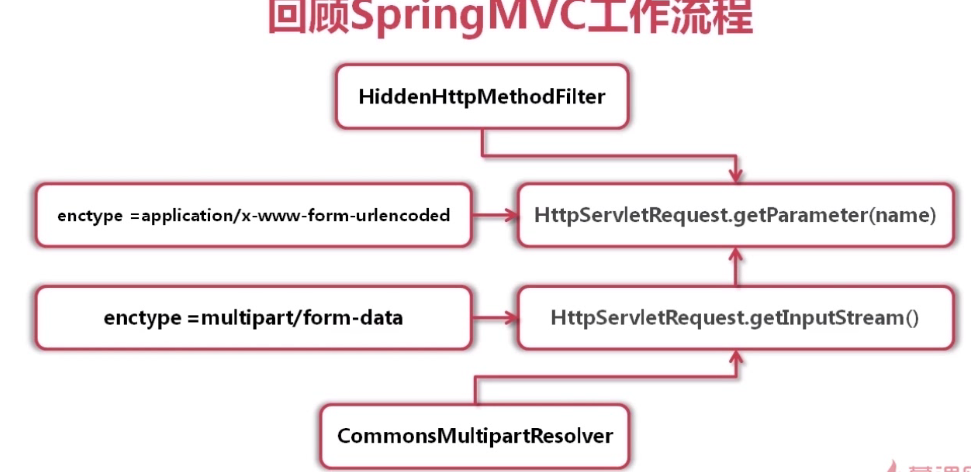
<filter>  
 <filter-name>MultipartFilter</filter-name>  
 <filter-class>org.springframework.web.multipart.support.MultipartFilter</filter-class>  
 <init-param>  
 <param-name>multipartResolverBeanName</param-name>  
 <param-value>multipartResolver</param-value>  
 </init-param>  
</filter>  
<filter-mapping>  
 <filter-name>MultipartFilter</filter-name>  
 <url-pattern>/\*</url-pattern>  
</filter-mapping>  
  
<filter>  
 <filter-name>hiddenHttpMethodFilter</filter-name>  
 <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>  
</filter>  
<filter-mapping>  
 <filter-name>hiddenHttpMethodFilter</filter-name>  
 <url-pattern>/\*</url-pattern>  
</filter-mapping>

此时，依然不能转化方法，结合过滤器，解决问题。



在1的时候，如果web.xml中有fitler过滤器时，会先执行过滤器的内容。而且fitlter是按顺序执行的。

在3步骤后，获得handleradapter后，会执行  
interceptor拦截器的内容。



14.结合源码解决问题。

过滤器有doFilter 方法。

FiterChain.doFiter(processedRequeset,response);

每执行完一个过滤器后，需要调用该方法，执行之后的过滤器，相当于过滤器链，如果不执行，之后的过滤器就不再执行了。

过滤器肯定实现了Filter接口，并重写了三个方法，初始化，doFilter，销毁。

一个虚拟机下，一个web应用，会对应一个servletcontext上下文，获取web应用相关知识。

框架都是基于servletContext来封装的。

SpringMVC封装了一个WebapplicationContext 来获取应用上下文。。

修改源码使其生效，直接重新新建一个同名文件，然后复制对应源码内容，自己在修改源码，编译运行，会使其生效的。

自己编译生成在classes文件下的class文件，会比jar包中的classes文件先加载。后面的会忽略。

源码也可以打断点。

通过源码分析，springMVC 在执行

org.springframework.web.multipart.support.MultipartFilter

过滤器时，会从根配置文件来获取 上下文信息。

即从spring监听器监听的配置文件获取上下文。

<context-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:spring/root-context.xml</param-value>  
</context-param>  
  
<listener>  
 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
</listener>

SpringMVC 的

org.springframework.web.servlet.DispatcherServlet

执行时，也会根据配置文件创建DispatcherServlet

的上下文。

由于文件上传解析器的过滤器，会在spring监听器中获取上下文，如果没有配置，就找不到对应的类。

框架中都会提供默认的处理类，还会有自定义的处理类，这时，就需要修改初始化参数，将自己的类传入。

<init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:spring/applicationContext.xml</param-value>  
</init-param>

Inti-param.

好的编码习惯。MVC分层，dao层的修改不应该修改到dto。

15.mybatis中 一个表中的存储的是城市的code，策略的code，转化关系，对应在dic字典表中。如何实现这种表与表中的转化。

对应一对一，多对一的关系，需要在原来的bean中加入转化的bean。

然后查询的返回值需要用resultMap对应。

<result column="comment\_total\_num" property="commentTotalNum" />  
<association property="cityDic" javaType="Dic">  
 <result column="city\_name" property="name"/>  
</association>  
<association property="categoryDic" javaType="Dic">  
 <result column="category\_name" property="name"/>  
</association>

参考business.

Column 查询出来的列名，property对应bean的字段。

association

是表与表字段转化的处理。

Property是bean对应的字段，javaType 对应的另一个表的bean。

Result 是映射关系，column查询的列名，property对应 bean中的属性。

Mysql中关键字desc，在mybatis中用`` 括起来就行了。

Mybatis中引用静态常量的方法。

${@类名@常量名}

16.springMVC参数绑定。

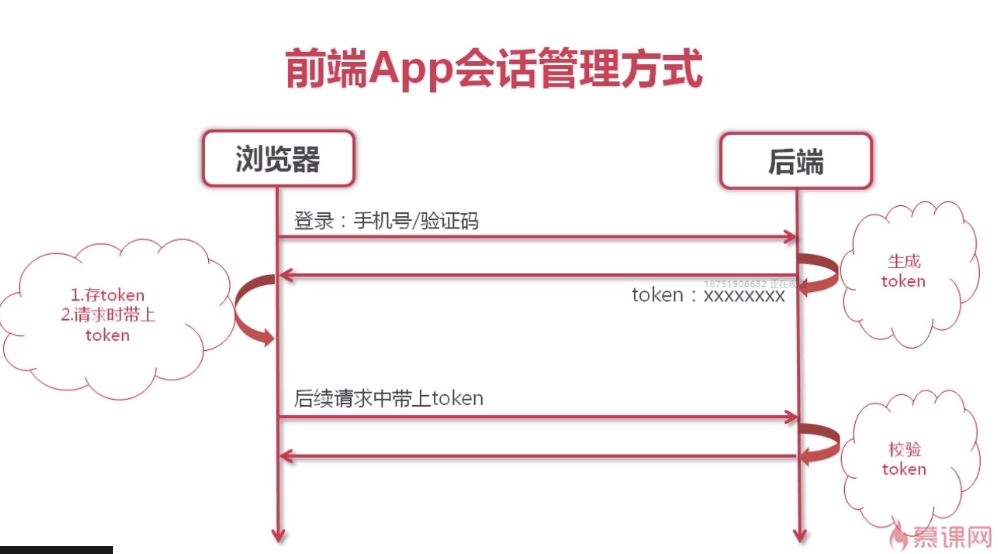
1、 @PathVariable   
当使用@RequestMapping [URI](https://www.baidu.com/s?wd=URI&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao) template 样式映射时， 即 someUrl/{paramId}, 这时的paramId可通过 @Pathvariable注解绑定它传过来的值到方法的参数上。  
示例代码：  
@Controller   
@RequestMapping("/owners/{ownerId}")   
public class RelativePath[Uri](https://www.baidu.com/s?wd=Uri&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)TemplateController {   
   
 @RequestMapping("/pets/{petId}")   
 public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {   
 }   
}   
上面代码把[URI](https://www.baidu.com/s?wd=URI&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao) template 中变量 ownerId的值和petId的值，绑定到方法的参数上。若方法参数名称和需要绑定的uri template中变量名称不一致，需要在@PathVariable("name")指定uri template中的名称。  
2、 @RequestHeader、@CookieValue  
  
@RequestHeader 注解，可以把Request请求header部分的值绑定到方法的参数上。  
示例代码：  
这是一个Request 的header部分：  
Host localhost:8080   
Accept text/html,application/xhtml+xml,application/xml;q=0.9   
Accept-Language fr,en-gb;q=0.7,en;q=0.3   
Accept-Encoding gzip,deflate   
Accept-Charset ISO-8859-1,utf-8;q=0.7,\*;q=0.7   
Keep-Alive 300   
@RequestMapping("/displayHeaderInfo.do")   
public void displayHeaderInfo(@RequestHeader("Accept-Encoding") String encoding, @RequestHeader("Keep-Alive") long keepAlive) {   
}   
上面的代码，把request header部分的 Accept-Encoding的值，绑定到参数encoding上了， Keep-Alive header的值绑定到参数keepAlive上。  
@CookieValue 可以把Request header中关于cookie的值绑定到方法的参数上。  
例如有如下Cookie值：  
JSESSIONID=415A4AC178C59DACE0B2C9CA727CDD84   
参数绑定的代码：  
  
@RequestMapping("/displayHeaderInfo.do")   
public void displayHeaderInfo(@CookieValue("JSESSIONID") String cookie) {   
}   
即把JSESSIONID的值绑定到参数cookie上。  
  
3、@RequestParam, @RequestBody  
@RequestParam   
A） 常用来处理简单类型的绑定，通过Request.getParameter() 获取的String可直接转换为简单类型的情况（ String--> 简单类型的转换操作由ConversionService配置的转换器来完成）；因为使用request.getParameter()方式获取参数，所以可以处理get 方式中queryString的值，也可以处理post方式中 body data的值；  
B）用来处理Content-Type: 为 application/x-www-form-urlencoded编码的内容，提交方式GET、POST；  
  
C) 该注解有两个属性： value、required； value用来指定要传入值的id名称，required用来指示参数是否必须绑定；  
示例代码：  
  
@Controller   
@RequestMapping("/pets")   
@SessionAttributes("pet")   
public class EditPetForm {   
 @RequestMapping(method = RequestMethod.GET)   
 public String setupForm(@RequestParam("petId") int petId, ModelMap model) {   
 Pet pet = this.clinic.loadPet(petId);   
 model.addAttribute("pet", pet);   
 return "petForm";   
 }   
@RequestBody  
该注解常用来处理Content-Type: 不是application/x-www-form-urlencoded编码的内容，例如application/json, application/xml等；  
  
它是通过使用HandlerAdapter 配置的HttpMessageConverters来解析post data body，然后绑定到相应的bean上的。  
  
因为配置有FormHttpMessageConverter，所以也可以用来处理 application/x-www-form-urlencoded的内容，处理完的结果放在一个MultiValueMap<String, String>里，这种情况在某些特殊需求下使用，详情查看FormHttpMessageConverter api;

17.两种回话模式

1.传统session 会话模式



Token









Nginx 转发请求。

如果在代码中用一个单例的类中的map存储公共信息，由于Nginx转发到服务集群当中，一份代码，重复布置到不同的机器，则不会保持单例、

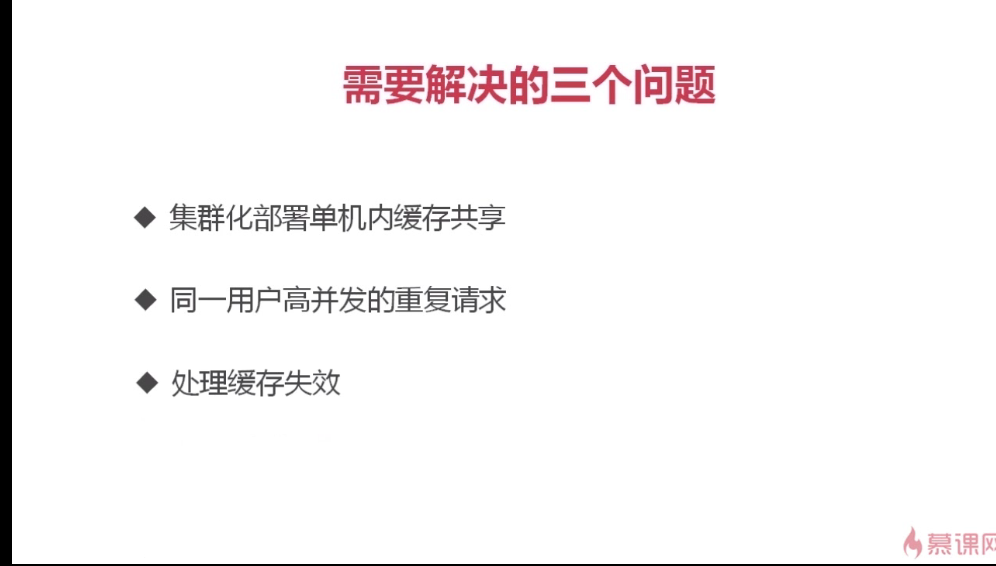
例如token登录。

用户请求验证码时，Nginx转发到机器一，单例map中存储 手机号和验证码信息。

用户登录时，Nginx转发到机器二，则取不到之前的信息。

处理方法，Nginx 转发规则，用户访问的IP没变，则始终访问同一台机器。

2,。将用户信息放在缓存服务器中。



使用redis。

Guava cache

短信通道

18.日志

Logback.xml配置文件

<appender>

19.验证码

后端返回验证码的MD5值给前台，前台在校验的时候，先比较验证码和MD5是否一致，不一致，就不用提交了。

UUID

20.表结构的设计

少使用主外键关系，如果一个系统中，有很多主外键关系，设计的灵活性降低，各表之间关系复杂，当数据多的时候，对数据库进行拆分，部署在多台服务器上会有问题。

保证数据的一致性

不一定要主外键关系。

一般用无意义的主键来就行各表之间的关联。

在买单的时候如果继续查询数据库统计已售数量，可能会影响效率，用户买单的效率可能下降。

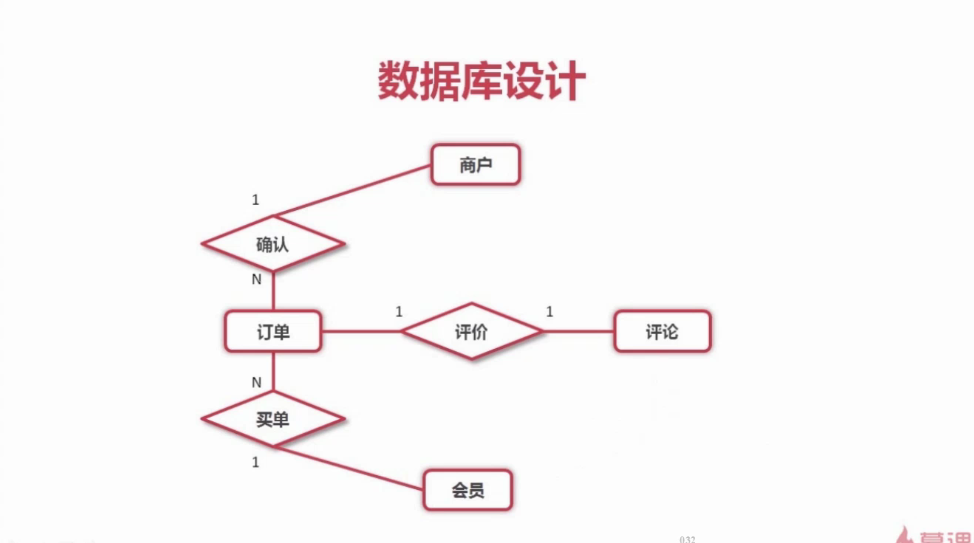
实时数据，将为准实时，延后更新。

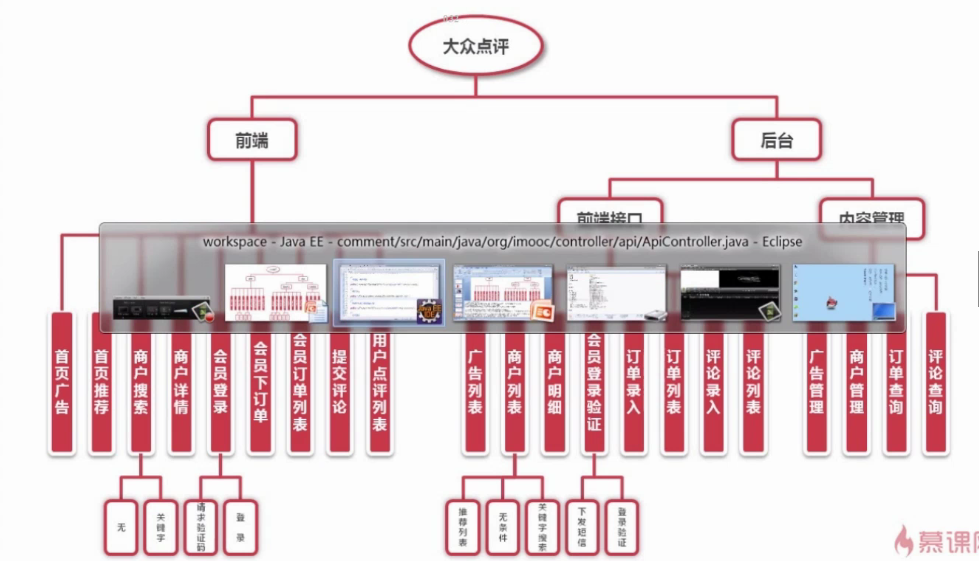
定时任务

对应用户订单提交后，需要更新商户的已售数量，用定时任务，如果每次都全表更新的话，越到后面，效率越低。

所以可以给订单加一个时间，每次同步时同步 创建的时间 大于 上次同步的时间，然后将数量累加。

评论





Spring解决重定向传参问题

RedirectAttributes attr

attr.addFlashAttribute(PageCodeEnum.*KEY*, PageCodeEnum.*LOGIN\_FAIL*);

向重定向页面传递参数，用完之后清除。

原理是利用session。

21.Spring拦截器

对用户登录请求校验。

<mvc:interceptor>  
 <mvc:mapping path="/\*\*"/>  
 <mvc:exclude-mapping path="/index"/>  
 <mvc:exclude-mapping path="/session"/>  
 <mvc:exclude-mapping path="/session/menus"/>  
 <mvc:exclude-mapping path="/login/\*\*"/>  
 <mvc:exclude-mapping path="/css/\*\*"/>  
 <mvc:exclude-mapping path="/js/\*\*"/>  
 <mvc:exclude-mapping path="/images/\*\*"/>  
 <mvc:exclude-mapping path="/api/\*\*"/>  
 <bean class="org.imooc.controller.interceptor.AuthInterceptor"></bean>  
</mvc:interceptor>

<mvc:interceptor>定义拦截器

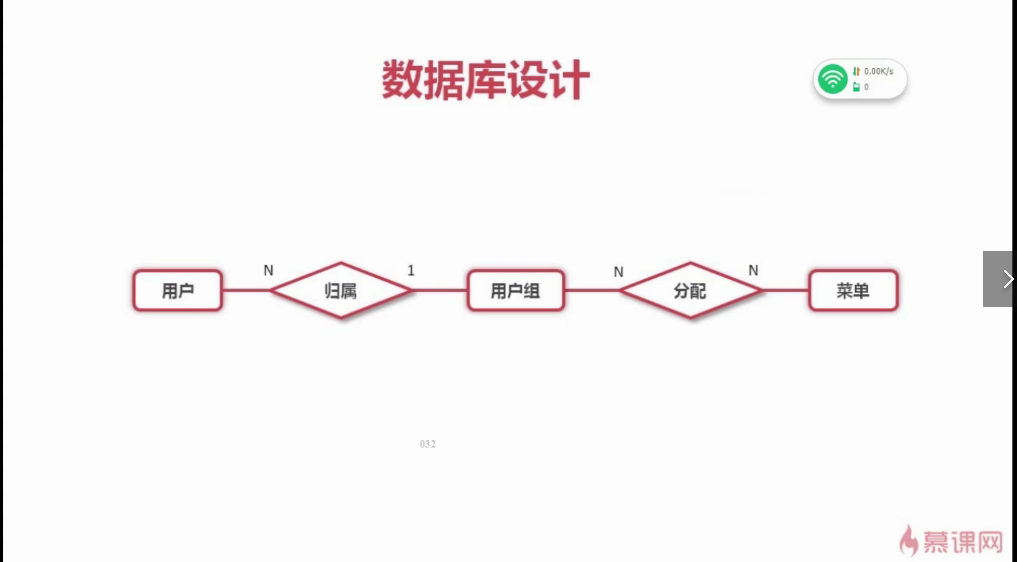
<mvc:mapping path="/\*\*"/>  
拦截的请求，

<mvc:exclude-mapping path="/api/\*\*"/>  
排除的请求

利用拦截器，进行是否登录过的校验。

*/\*\*  
 \* session拦截器  
 \*/*public class SessionInterceptor implements HandlerInterceptor {  
  
 */\*\*  
 \* 在进入Handler方法执行之前执行本方法  
 \*   
 \** ***@return*** *true:执行下一个拦截器，直到所有拦截器都执行完，再执行被拦截的Controller  
 \* false:从当前的拦截器往回执行所有拦截器的afterCompletion(),再退出拦截器链  
 \*/* public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {  
 if (request.getSession().getAttribute(SessionKeyConst.*USER\_INFO*) != null) {  
 return true;  
 }  
 // 针对ajax请求处理  
 if (request.getHeader("x-requested-with") != null) {  
 String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.getServerPort() + request.getContextPath();  
 response.setHeader("url", basePath + "/login/sessionTimeout");  
 } else {  
 request.getRequestDispatcher("/login/sessionTimeout").forward(request, response);  
 }  
 return false;  
 }  
  
 */\*\*  
 \* 在进入Handler方法之后，返回ModelAndView之前执行  
 \*/* @Override  
 public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,  
 ModelAndView modelAndView) throws Exception {  
 // *TODO Auto-generated method stub* }  
  
 */\*\*  
 \* 在Handler方法执行完之后执行  
 \*/* @Override  
 public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)  
 throws Exception {  
 // *TODO Auto-generated method stub* }  
  
}

22.基于角色的权限管理



用户表 角色表 菜单表

用户表 属于某一个组，可以在用户表中加一个 角色表的主键，表示属于这个组。

角色表 id，角色名

菜单表

url，id，order，name.

角色表——菜单表的关联表

角色id，菜单id，多对多的关系。

使用ztree插件展示界面。

Ajax请求使用restful规范。

Post方法提交参数加一个”\_method” =”PUT” 即可。

Ztree 拖拽事件，对于拖拽菜单，然后拖到主菜单下，注意要将子菜单的节点全部加一，然后将新加的菜单的节点挂在父节点下。

23.Ajax session超时。

在用户验证登录后，将服务器断掉后，此时再再之前页面进行ajax请求，应该要拦截为session超时，并且要跳转到登录界面。

如何判断ajax请求？

request.getHeader("x-requested-with") != null

请求头中有x-requested-with 的，即为ajax请求。

然后用请求头 返回要跳转的地址。然后再js中ajax请求完成时加载地址。

complete : **function**(response) {  
 **var** url = response.getResponseHeader("url");  
 **if**(url) {  
 location.href = url;  
 } **else** {  
 **if**(param.complete && **typeof** param.complete == "function") {  
 param.complete();  
 }  
 }  
}

使用js loacation.load 加载地址页面。会发现是在frame中跳转的，要想回到最顶层跳转，需要在js中处理。

$(**function** () {  
 common.showMessage('**${**pageCode.msg**}**');  
 **var** topWindow = window;  
 **while**(topWindow.parent != topWindow) {  
 topWindow = topWindow.parent;  
 }  
 topWindow.location.href = "**${**basePath**}**/login";  
});

使用最顶层window打开页面。

所以不直接跳转到login，需要跳转到timeout界面，在里面用顶层窗体打开界面。

jQuery.extends(true, ,)

第一个参数，false为浅拷贝，true为深拷贝。

例如

Date:{

A:a,

B:b

}

Date2:{

C:c

}

浅拷贝合并成date{

C:c

}

深拷贝

Data：{

A:a,

B:b,

C:c

}

Set 去重，treeset 排序

String.format(“%02d”)固定两位，没有用0补齐。

Echarts 数据库取值，格式化数据。

都是一些数据的读取，格式化，多思考。

24.总结

权限管理-用户-角色-菜单-菜单功能的增删改的权限

，ztree，echarts，登录，session拦截器，ajax请求，过滤器，spring文件上传，restful,分页拦截器，自定义标签，源码解析，缓存，定时器，logback。

待做

异常拦截器 自定义异常hhhhhhhh好

服务端验证

Spring 标签库

数据权限控制

Linux对文件和目录访问权限的控制

全文检索

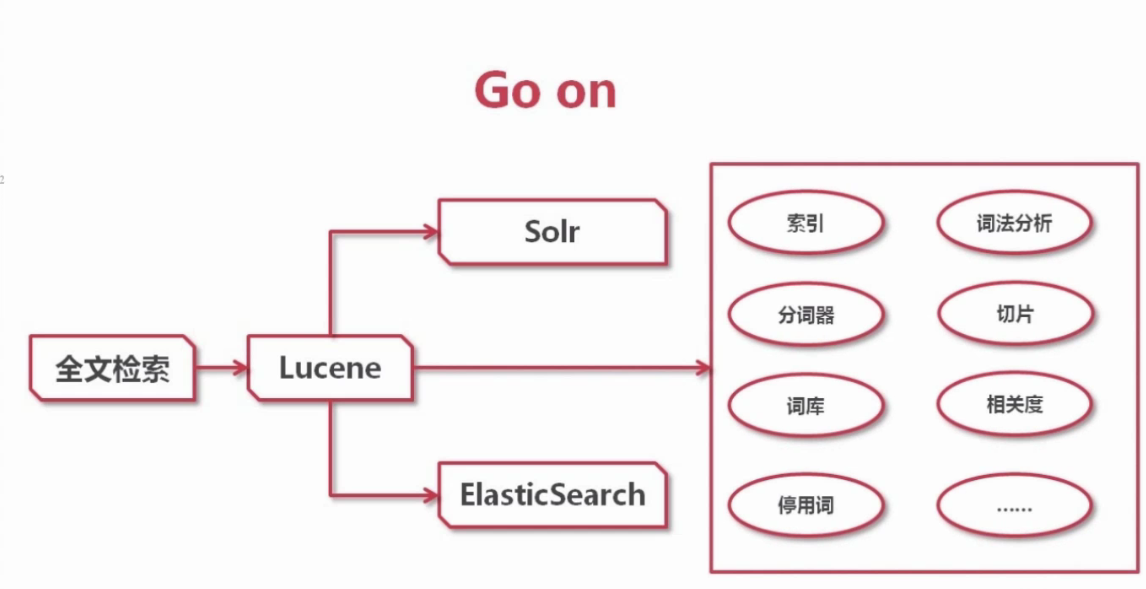
推荐系统 推荐算法

数据挖掘

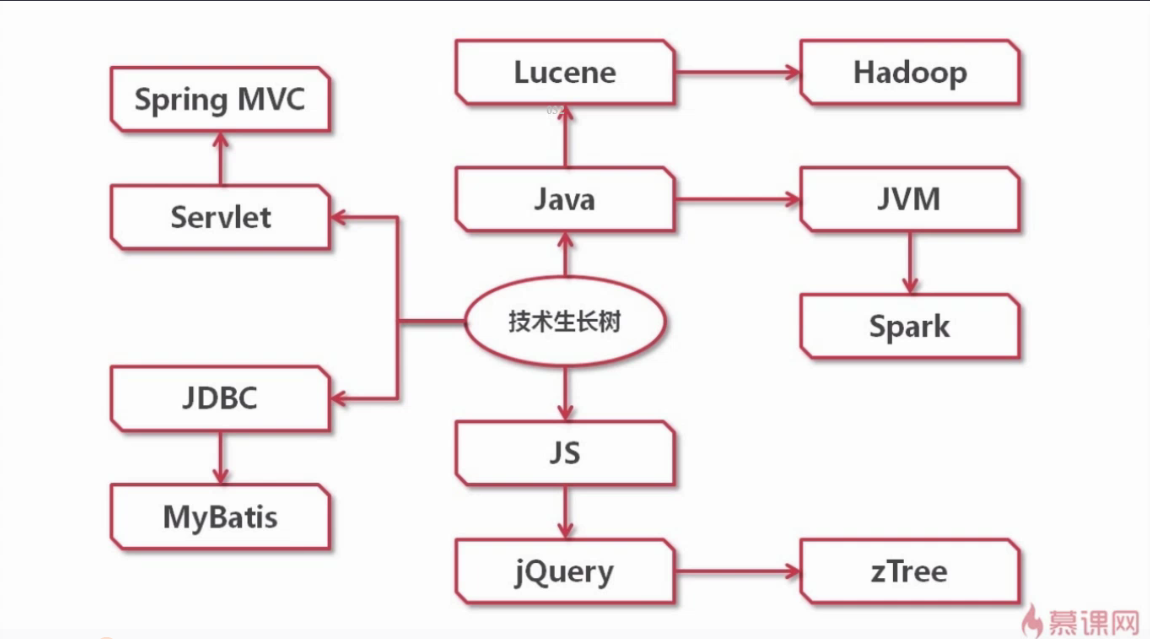
Hadoop hive hdfs hbase storm

spark









技术的实现原理

选这个技术的原因？

来源于生活，多动脑，多思考。

25.权限

用户 组 菜单 动作

用户属于哪个组 用户表中有组的ID

组

菜单 属于哪个组 存放在 组和 菜单的关系

动作属于哪个菜单 动作有对应菜单的ID

组有哪些动作 存放动作与组的对应关系

一对多的关系，关联id在多的那个实体中存

多对多需要重新弄一个关联表

权限拦截器

查出用户的所用动作权限，当用户请求一个界面时，根据url看，该用户有没有该动作url，如果存在，显示界面。这部分操作可以在拦截器中做。

还有界面控制 是否有对应的修改，删除按钮的权限，可以用jsp标签来判断，如果session中，才显示按钮。

1. session超时
2. 没权限
3. 资源被拦截