1. **创建项目**

Taotao-parent（父工程管理jar包的版本）（pom包）

|--Taotao-common（把通用的工具类打包）（jar包）

|--Taotao-manager（继承父工程）（pom包）

|-taotao-manager-pojo（jar包）

|-taotao-manager-dao（jar包）

|-taotao-manager-service（jar包）

|-taotao-manager-web（war包）

|--taotao-sso （war包）

|--taotao-rest （war包）

|--taotao-order （war包）

|--taotao-portal（war包）

|--Taotao-search（war包）

父项目中打包方式必须是pom 如 <packaging>pom</packaging>

**jar包和war包的区别：**

war是一个web模块，其中需要包括WEB-INF，是可以直接运行的WEB模块。而jar一般只是包括一些class文件，在声明了Main\_class之后是可以用java命令运行的.

它们都是压缩的包,拿Tomcat来说,将war文件包放置它的\webapps\目录下，启动Tomcat,这个包可以自动进行解压，也就是你的web目录，相当于发布了。

war包:是做好一个web应用后，通常是网站，打成包部署到容器中。

jar包：通常是开发时要引用通用类，打成包便于存放管理。

个人理解:有页面jsp等用war，没有用jar

1. **Spring和SpringMVC**

Web.xml

<!-- 初始化spring容器 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/applicationContext-\*.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 解决post乱码 -->

<filter>

<filter-name>CharacterEncodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CharacterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- springmvc的前端控制器 -->

<servlet>

<servlet-name>taotao-manager</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- contextConfigLocation不是必须的， 如果不配置contextConfigLocation， springmvc的配置文件默认在：WEB-INF/servlet的name+"-servlet.xml" -->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/springmvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>taotao-manager</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

spring和springMVC扫描包的问题（为什么spring配置了扫描，springmvc还要配置？）：http://blog.csdn.net/lhpnba/article/details/77988616

1. **Nginx、Fastdfs安装 图片上传**

在springmvc.xml中配置多媒体解析器

|  |
| --- |
| <bean id=*"multipartResolver"*  class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>  <!-- 设定默认编码 -->  <property name=*"defaultEncoding"* value=*"UTF-8"*></property>  <!-- 设定文件上传的最大值5MB，5\*1024\*1024 -->  <property name=*"maxUploadSize"* value=*"5242880"*></property>  </bean> |

官方提供一个jar包fastdfs\_client.jar

|  |
| --- |
| **public** **void** testUpload() **throws** Exception {  // 1、把FastDFS提供的jar包添加到工程中  // 2、初始化全局配置。加载一个配置文件。  ClientGlobal.*init*("D:\\workspaces-itcast\\JaveEE18\\taotao-manager\\taotao-manager-web\\src\\main\\resources\\properties\\client.conf");  // 3、创建一个TrackerClient对象。  TrackerClient trackerClient = **new** TrackerClient();  // 4、创建一个TrackerServer对象。  TrackerServer trackerServer = trackerClient.getConnection();  // 5、声明一个StorageServer对象，null。  StorageServer storageServer = **null**;  // 6、获得StorageClient对象。  StorageClient storageClient = **new** StorageClient(trackerServer, storageServer);  // 7、直接调用StorageClient对象方法上传文件即可。  String[] strings = storageClient.upload\_file("D:\\Documents\\Pictures\\images\\2f2eb938943d.jpg", "jpg", **null**);  **for** (String string : strings) {  System.***out***.println(string);  }  } |

Client.conf

|  |
| --- |
| tracker\_server=192.168.25.133:22122 |

1. **Ajax跨域使用jsonp实现**
2. **HttpClient**

Get请求

|  |
| --- |
| **public** **void** testHttpGet() **throws** Exception {  // 第一步：把HttpClient使用的jar包添加到工程中。  // 第二步：创建一个HttpClient的测试类  // 第三步：创建测试方法。  // 第四步：创建一个HttpClient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();  // 第五步：创建一个HttpGet对象，需要制定一个请求的url  HttpGet get = **new** HttpGet("http://www.itheima.com");  // 第六步：执行请求。  CloseableHttpResponse response = httpClient.execute(get);  // 第七步：接收返回结果。HttpEntity对象。  HttpEntity entity = response.getEntity();  // 第八步：取响应的内容。  String html = EntityUtils.*toString*(entity);  System.***out***.println(html);  // 第九步：关闭response、HttpClient。  response.close();  httpClient.close();  } |

post请求

|  |
| --- |
| **public** **void** testHttpPost() **throws** Exception {    // 第一步：创建一个httpClient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();  // 第二步：创建一个HttpPost对象。需要指定一个url  HttpPost post = **new** HttpPost("http://localhost:8082/posttest.html");  // 第三步：创建一个list模拟表单，list中每个元素是一个NameValuePair对象  List<NameValuePair> formList = **new** ArrayList<>();  formList.add(**new** BasicNameValuePair("name", "张三"));  formList.add(**new** BasicNameValuePair("pass", "1243"));  // 第四步：需要把表单包装到Entity对象中。StringEntity  StringEntity entity = **new** UrlEncodedFormEntity(formList, "utf-8");  post.setEntity(entity);  // 第五步：执行请求。  CloseableHttpResponse response = httpClient.execute(post);  // 第六步：接收返回结果  HttpEntity httpEntity = response.getEntity();  String result = EntityUtils.*toString*(httpEntity);  System.***out***.println(result);  // 第七步：关闭流。  response.close();  httpClient.close();  } |

**提交表单和提交json数据，content-Type不同**

**表单的content-type：application/x-www-form-urlencoded**

**Json的content-type：application/json**

1. **Redis集群安装、jedis调用**

需要使用Jedis连接redis服务器。

先把jedis依赖的jar包添加到工程中。

|  |
| --- |
| 单机版：  //单机版测试  @Test  **public** **void** testJedisSingle() **throws** Exception {  //创建一个Jedis对象  Jedis jedis = **new** Jedis("192.168.25.153", 6379);  jedis.set("test", "hello jedis");  String string = jedis.get("test");  System.***out***.println(string);  jedis.close();  }  使用连接池：  //使用连接池  @Test  **public** **void** testJedisPool() **throws** Exception {  //创建一个连接池对象  //系统中应该是单例的。  JedisPool jedisPool = **new** JedisPool("192.168.25.153", 6379);  //从连接池中获得一个连接  Jedis jedis = jedisPool.getResource();  String result = jedis.get("test");  System.***out***.println(result);  //jedis必须关闭  jedis.close();  //系统关闭时关闭连接池  jedisPool.close();  } |

|  |
| --- |
| 集群版：  //连接redis集群  @Test  **public** **void** testJedisCluster() **throws** Exception {  //创建一个JedisCluster对象  Set<HostAndPort> nodes = **new** HashSet<>();  nodes.add(**new** HostAndPort("192.168.25.153", 7001));  nodes.add(**new** HostAndPort("192.168.25.153", 7002));  nodes.add(**new** HostAndPort("192.168.25.153", 7003));  nodes.add(**new** HostAndPort("192.168.25.153", 7004));  nodes.add(**new** HostAndPort("192.168.25.153", 7005));  nodes.add(**new** HostAndPort("192.168.25.153", 7006));  //在nodes中指定每个节点的地址  //jedisCluster在系统中是单例的。  JedisCluster jedisCluster = **new** JedisCluster(nodes);  jedisCluster.set("name", "zhangsan");  jedisCluster.set("value", "100");  String name = jedisCluster.get("name");  String value = jedisCluster.get("value");  System.***out***.println(name);  System.***out***.println(value);      //系统关闭时关闭jedisCluster  jedisCluster.close();  } |

|  |
| --- |
| Spring的配置：  <!-- 包扫描器，扫描带@Service注解的类 -->  <context:component-scan base-package=*"com.taotao.rest.service"*></context:component-scan>  <!-- 配置redis客户端单机版 -->  <bean id=*"jedisPool"* class=*"redis.clients.jedis.JedisPool"*>  <constructor-arg name=*"host"* value=*"192.168.25.153"*></constructor-arg>  <constructor-arg name=*"port"* value=*"6379"*></constructor-arg>  </bean>  <!-- 配置redis客户端实现类 -->  <bean id=*"jedisClientSingle"* class=*"com.taotao.rest.component.impl.JedisClientSingle"*/>    <!-- 配置redis客户端集群版 -->  <!-- <bean id="jedisCluster" class="redis.clients.jedis.JedisCluster">  <constructor-arg>  <set>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"/>  <constructor-arg name="port" value="7001"/>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"/>  <constructor-arg name="port" value="7002"/>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"/>  <constructor-arg name="port" value="7003"/>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"/>  <constructor-arg name="port" value="7004"/>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"/>  <constructor-arg name="port" value="7005"/>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"/>  <constructor-arg name="port" value="7006"/>  </bean>  </set>  </constructor-arg>  </bean>  <bean id="jedisClientCluster" class="com.taotao.rest.component.impl.JedisClientCluster"/> -->  </beans> |

|  |
| --- |
| 测试：  @Test  **public** **void** testJedisClientSpring() **throws** Exception {  //创建一个spring容器  ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("classpath:spring/applicationContext-\*.xml");  //从容器中获得JedisClient对象  JedisClient jedisClient = applicationContext.getBean(JedisClient.**class**);  //jedisClient操作redis  jedisClient.set("cliet1", "1000");  String string = jedisClient.get("cliet1");  System.***out***.println(string);  } |

1. **Solr服务（待补充）**

|  |
| --- |
| **Solr单机版安装**：  第一步：安装jdk、安装tomcat  第二步：解压solr压缩包。  第三步：把dist/solr-4.10.3.war部署到tomcat下。  第四步：解压缩war包。启动tomcat解压。  第五步：需要把/root/solr-4.10.3/example/lib/ext目录下的所有的jar包添加到solr工程中。  第六步：创建solrhome。把/root/solr-4.10.3/example/solr文件夹复制一份作为solrhome。  第七步：告诉solr服务solrhome的位置。需要修改web.xml    第八步：启动tomcat。 |

|  |
| --- |
| **Solr集群版安装**： |

1. **FreeMarker**

|  |
| --- |
| **使用方法：**  第一步：把freemarker的jar包添加到工程中  第二步：freemarker的运行不依赖web容器，可以在java工程中运行。创建一个测试方法进行测试。  第三步：创建一个Configration对象  第四步：告诉config对象模板文件存放的路径。  第五步：设置config的默认字符集。一般是utf-8  第六步：从config对象中获得模板对象。需要制定一个模板文件的名字。  第七步：创建模板需要的数据集。可以是一个map对象也可以是一个pojo，把模板需要的数据都放入数据集。  第八步：创建一个Writer对象，指定生成的文件保存的路径及文件名。  第九步：调用模板对象的process方法生成静态文件。需要两个参数数据集和writer对象。  第十步：关闭writer对象。 |

|  |
| --- |
| **实现：**  **public** **class** FreeMarkerTest {  @Test  **public** **void** testFreeMarker() **throws** Exception {  // 第一步：把freemarker的jar包添加到工程中  // 第二步：freemarker的运行不依赖web容器，可以在java工程中运行。创建一个测试方法进行测试。  // 第三步：创建一个Configuration对象  Configuration configuration = **new** Configuration(Configuration.*getVersion*());  // 第四步：告诉config对象模板文件存放的路径。  configuration.setDirectoryForTemplateLoading(**new** File("D:\\workspaces-itcast\\JaveEE18\\taotao-portal\\src\\main\\webapp\\WEB-INF\\ftl"));  // 第五步：设置config的默认字符集。一般是utf-8  configuration.setDefaultEncoding("utf-8");  // 第六步：从config对象中获得模板对象。需要制定一个模板文件的名字。  Template template = configuration.getTemplate("first.ftl");  // 第七步：创建模板需要的数据集。可以是一个map对象也可以是一个pojo，把模板需要的数据都放入数据集。  Map root = **new** HashMap<>();  root.put("hello", "hello freemarker");  // 第八步：创建一个Writer对象，指定生成的文件保存的路径及文件名。  Writer out = **new** FileWriter(**new** File("D:\\temp\\html\\hello.html"));  // 第九步：调用模板对象的process方法生成静态文件。需要两个参数数据集和writer对象。  template.process(root, out);  // 第十步：关闭writer对象。  out.flush();  out.close();  }  } |

**前端使用使用EL表达式。**

|  |
| --- |
| Spring配置文件  <bean id=*"freemarkerConfig"*  class=*"org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer"*>  <property name=*"templateLoaderPath"* value=*"/WEB-INF/ftl/"* />  <property name=*"defaultEncoding"* value=*"UTF-8"* />  </bean> |

|  |
| --- |
| **模板：** |

1. **SSO单点登录系统 使用redis、cookie、jedis**
2. **Tomcat热部署**
3. **Mycat分表分库**
4. **Keepalive保证集群高可用**
5. **Lvs**
6. **秒杀、倒计时等**
7. **Quartz(下单不付钱，多长时间不付钱就自动取消订单)**