



淘淘商城

第二天

1. 课程计划

第二天：商品列表功能实现

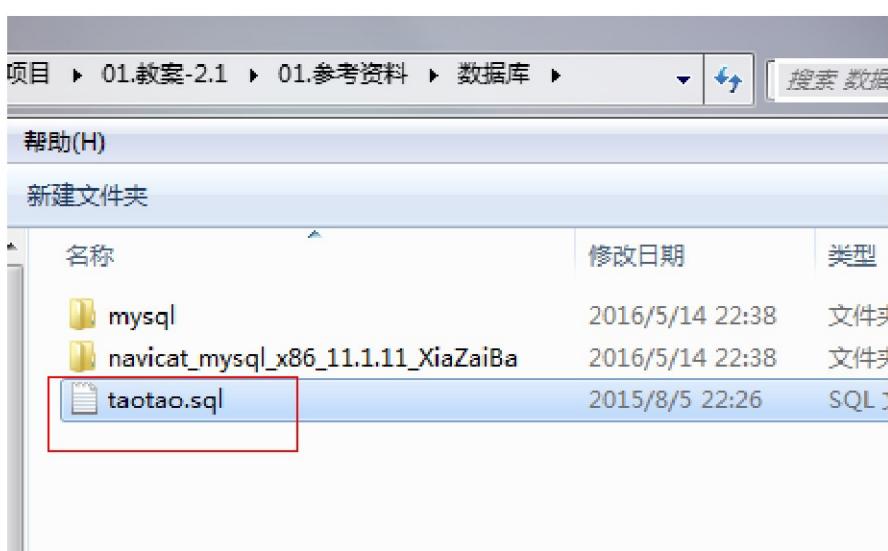
- 1、Ssm 框架整合。
- 2、服务中间件 dubbo
- 3、整合测试
- 4、商品列表查询功能实现。

2. Ssm 框架整合

2.1. 数据库

数据库使用 mysql 数据库，要求 5.5 以上版本。

- 1、在 mysql 数据库中创建数据库 taobao
- 2、将创建数据库的脚本导入到 taobao 中。



2.2. Mybatis 逆向工程

使用 mybatis 官方提供的 mybatis-generator 生成 pojo、mapper 接口及映射文件。

并且将 pojo 放到 taobao-manager-pojo 工程中。

将 mapper 接口及映射文件放到 taobao-manager-dao 工程中。

2.3. 整合思路

1、Dao 层：

mybatis 整合 spring，通过 spring 管理 SqlSessionFactory、mapper 代理对象。需要 mybatis 和 spring 的整合包。

2、Service 层：

所有的 service 实现类都放到 spring 容器中管理。由 spring 创建数据库连接池，并有 spring 管理实务。

3、表现层：

Springmvc 框架，由 springmvc 管理 controller。

2.4.Dao 整合

2.4.1. 创建 SqlMapConfig.xml 配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

</configuration>
```

2.4.2. Spring 整合 mybatis

创建 applicationContext-dao.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.2.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-4.2.xsd
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-4.2.xsd">

    <!-- 数据库连接池 -->
    <!-- 加载配置文件 -->
    <context:property-placeholder location="classpath:properties/*.properties" />
    <!-- 数据库连接池 -->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
        destroy-method="close">
```

```

<property name="url" value="${jdbc.url}" />
<property name="username" value="${jdbc.username}" />
<property name="password" value="${jdbc.password}" />
<property name="driverClassName" value="${jdbc.driver}" />
<property name="maxActive" value="10" />
<property name="minIdle" value="5" />

</bean>
<!-- 让 spring 管理 sqlSessionFactory 使用 mybatis 和 spring 整合包中的 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <!-- 数据库连接池 -->
    <property name="dataSource" ref="dataSource" />
    <!-- 加载 mybatis 的全局配置文件 -->
    <property name="configLocation" value="classpath:mybatis/SqlMapConfig.xml" />
</bean>
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.taobao.mapper" />
</bean>
</beans>

```

db.properties

```

jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/taobao?characterEncoding=utf-8
jdbc.username=root
jdbc.password=root

```

备注：

Druid 是目前最好的数据库连接池，在功能、性能、扩展性方面，都超过其他数据库连接池，包括 DBCP、C3P0、BoneCP、Proxool、JBoss DataSource。
Druid 已经在阿里巴巴部署了超过 600 个应用，经过多年多生产环境大规模部署的严苛考验。

2.5. Service 整合

2.5.1. 管理 Service

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-4.2.xsd"/>

```

```
http://www.springframework.org/schema/context/spring-context-4.2.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.2.xsd http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.2.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.2.xsd">

<context:component-scan base-package="com.taobao.service"></context:component-scan>
</beans>
```

2.5.2. 事务管理

创建 applicationContext-trans.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.2.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.2.xsd http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.2.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.2.xsd">

    <!-- 事务管理器 -->
    <bean id="transactionManager"
          class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <!-- 数据源 -->
        <property name="dataSource" ref="dataSource" />
    </bean>
    <!-- 通知 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <!-- 传播行为 -->
            <tx:method name="save*" propagation="REQUIRED" />
            <tx:method name="insert*" propagation="REQUIRED" />
            <tx:method name="add*" propagation="REQUIRED" />
            <tx:method name="create*" propagation="REQUIRED" />
            <tx:method name="delete*" propagation="REQUIRED" />
        </tx:attributes>
    </tx:advice>
</beans>
```

```

<tx:method name="update*" propagation="REQUIRED" />
<tx:method name="find*" propagation="SUPPORTS" read-only="true" />
<tx:method name="select*" propagation="SUPPORTS" read-only="true" />
<tx:method name="get*" propagation="SUPPORTS" read-only="true" />
</tx:attributes>
</tx:advice>
<!-- 切面 -->
<aop:config>
    <aop:advisor advice-ref="txAdvice"
        pointcut="execution(* com.taobao.service.*.*(..))" />
</aop:config>
</beans>

```

2.5.3. Web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="2.5">
    <display-name>taotao-manager</display-name>
    <!-- 加载 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>
</web-app>

```

2.6. 表现层整合

2.6.1. Springmvc.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/p"
    >

```

```
http://www.springframework.org/schema/mvc/spring-mvc-4.2.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.2.xsd">

<context:component-scan base-package="com.taobao.controller" />
<mvc:annotation-driven />
<bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>
</beans>
```

2.6.2. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="2.5">
    <display-name>taobao-manager-web</display-name>
    <welcome-file-list>
        <welcome-file>login.html</welcome-file>
    </welcome-file-list>
    <!-- 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>taobao-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" --&gt;
&lt;init-param&gt;
    &lt;param-name&gt;contextConfigLocation&lt;/param-name&gt;
    &lt;param-value&gt;classpath:spring/springmvc.xml&lt;/param-value&gt;
&lt;/init-param&gt;
&lt;load-on-startup&gt;1&lt;/load-on-startup&gt;
&lt;/servlet&gt;
&lt;servlet-mapping&gt;
    &lt;servlet-name&gt;taotao-manager&lt;/servlet-name&gt;
    &lt;url-pattern&gt;/&lt;/url-pattern&gt;
&lt;/servlet-mapping&gt;
&lt;/web-app&gt;</pre>
```

2.7. 系统间通信

2.7.1. 分析

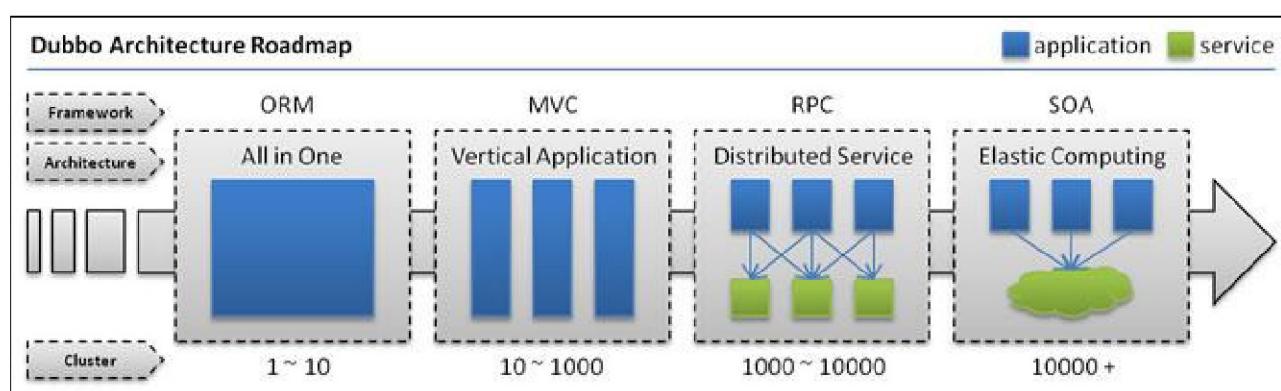
由于淘宝商城是基于 soa 的架构，表现层和服务层是不同的工程。所以要实现商品列表查询需要两个系统之间进行通信。

如何实现远程通信？

- 1、Webservice：效率不高基于 soap 协议。项目中不推荐使用。
- 2、使用 restful 形式的服务：http+json。很多项目中应用。如果服务太多，服务之间调用关系混乱，需要治理服务。
- 3、使用 dubbo。使用 rpc 协议进行远程调用，直接使用 socket 通信。传输效率高，并且可以统计出系统之间的调用关系、调用次数。

2.7.2. 什么是 dubbo

随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，分布式服务架构以及流动计算架构势在必行，亟需一个治理系统确保架构有条不紊的演进。

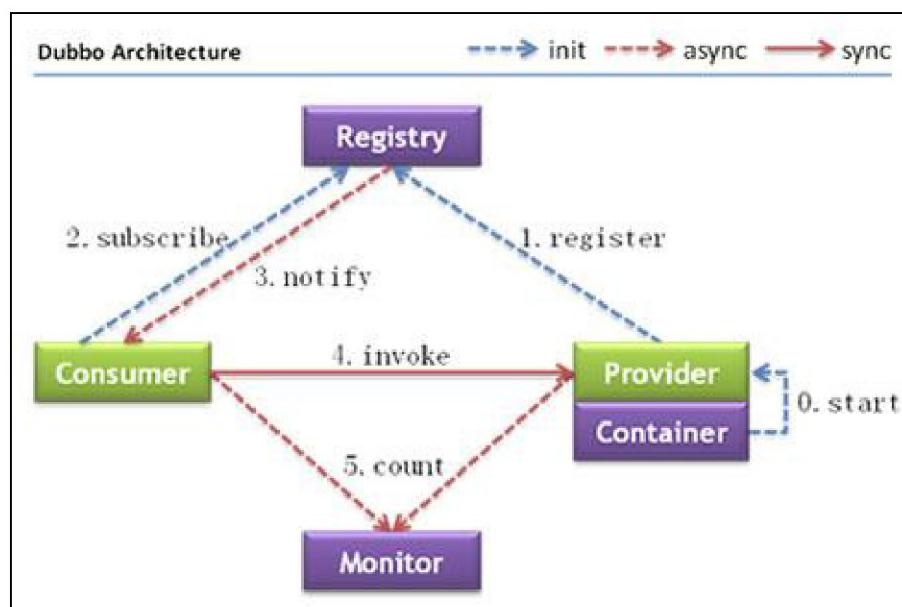


- 单一应用架构

- 当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。
- 此时，用于简化增删改查工作量的 数据访问框架(**ORM**) 是关键。
- 垂直应用架构
 - 当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。
 - 此时，用于加速前端页面开发的 **Web 框架(MVC)** 是关键。
- 分布式服务架构
 - 当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。
 - 此时，用于提高业务复用及整合的 **分布式服务框架(RPC)** 是关键。
- 流动计算架构
 - 当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。
 - 此时，用于提高机器利用率的 **资源调度和治理中心(SOA)** 是关键。

Dubbo 就是资源调度和治理中心的管理工具。

2.7.3. Dubbo 的架构



节点角色说明：

- **Provider**: 暴露服务的服务提供方。
- **Consumer**: 调用远程服务的服务消费方。
- **Registry**: 服务注册与发现的注册中心。
- **Monitor**: 统计服务的调用次调和调用时间的监控中心。
- **Container**: 服务运行容器。

调用关系说明：

- 0. 服务容器负责启动，加载，运行服务提供者。
- 1. 服务提供者在启动时，向注册中心注册自己提供的服务。
- 2. 服务消费者在启动时，向注册中心订阅自己所需的服务。
- 3. 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。
- 4. 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。
- 5. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

2.7.4. 使用方法

Dubbo 采用全 Spring 配置方式，透明化接入应用，对应用没有任何 API 侵入，只需用 Spring 加载 Dubbo 的配置即可，Dubbo 基于 Spring 的 Schema 扩展进行加载。

单一工程中 **spring** 的配置

```
<bean id="xxxService" class="com.xxx.XxxServiceImpl" />
<bean id="xxxAction" class="com.xxx.XxxAction">
    <property name="xxxService" ref="xxxService" />
</bean>
```

远程服务：

在本地服务的基础上，只需做简单配置，即可完成远程化：

将上面的 local.xml 配置拆分成两份，将服务定义部分放在服务提供方 remote-provider.xml，将服务引用部分放在服务消费方 remote-consumer.xml。

并在提供方增加暴露服务配置 `<dubbo:service>`，在消费方增加引用服务配置 `<dubbo:reference>`。

发布服务：

```
<!-- 和本地服务一样实现远程服务 -->
<bean id="xxxService" class="com.xxx.XxxServiceImpl" />
<!-- 增加暴露远程服务配置 -->
<dubbo:service interface="com.xxx.XxxService" ref="xxxService" />
```

调用服务：

```
<!-- 增加引用远程服务配置 -->
<dubbo:reference id="xxxService" interface="com.xxx.XxxService" />
<!-- 和本地服务一样使用远程服务 -->
<bean id="xxxAction" class="com.xxx.XxxAction">
    <property name="xxxService" ref="xxxService" />
</bean>
```

2.7.5. 注册中心

注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，压力较小。使用 dubbo-2.3.3 以上版本，建议使用 zookeeper 注册中心。

Zookeeper 是 Apache Hadoop 的子项目，是一个树型的目录服务，支持变更推送，适合作为 Dubbo 服务的注册中心，工业强度较高，可用于生产环境，并推荐使用

Zookeeper 的安装：

第一步：安装 jdk

第二步：解压缩 zookeeper 压缩包

第三步：将 conf 文件夹下 zoo_sample.cfg 复制一份，改名为 zoo.cfg

第四步：修改配置 dataDir 属性，指定一个真实目录

第五步：

启动 zookeeper: bin/zkServer.sh start

关闭 zookeeper: bin/zkServer.sh stop

查看 zookeeper 状态: bin/zkServer.sh status

注意要关闭 linux 的防火墙。

3. 整合测试

3.1. 需求

根据商品 id 查询商品信息，并将商品信息使用 json 数据返回。

3.2. 分析

请求的 url: /item/{itemId}

参数：商品 id，从请求的 url 中获得

返回值：TbItem 对象，逆向工程生成的 pojo（响应 json 数据）

3.3. Dao 层

根据商品 id 查询商品信息，单表查询可以使用逆向工程生成的代码。

3.4. Service 层

1、在 taotao-manager-interface 工程中创建一个 ItemService 接口

2、在 taotao-manager-Service 工程中创建一个 itemServiceImpl 的实现类。

```
@Service
public class ItemServiceImpl implements ItemService {

    @Autowired
    private TbItemMapper itemMapper;

    @Override
    public TbItem getItemById(long itemId) {
```

```
TbItem tbItem = itemMapper.selectByPrimaryKey(itemId);
return tbItem;
}
```

3.5. 发布服务

1、在 taotao-manager-Service 工程中添加 dubbo 依赖的 jar 包。

```
<!-- dubbo 相关 -->

<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>dubbo</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.jboss.netty</groupId>
            <artifactId>netty</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.apache.zookeeper</groupId>
    <artifactId>zookeeper</artifactId>
</dependency>
<dependency>
    <groupId>com.github.sgroschupf</groupId>
    <artifactId>zkclient</artifactId>
</dependency>
```

2、在 spring 的配置文件中添加 dubbo 的约束，然后使用 dubbo:service 发布服务。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-4.2.xsd"/>
```

```
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.2.xsd http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.2.xsd
http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.2.xsd">

<context:component-scan base-package="com.taobao.service"></context:component-scan>

<!-- 使用 dubbo 发布服务 -->
<!-- 提供方应用信息，用于计算依赖关系 -->
<dubbo:application name="taobao-manager" />
<dubbo:registry protocol="zookeeper"
    address="192.168.25.154:2181,192.168.25.154:2182,192.168.25.154:2183" />
<!-- 用 dubbo 协议在 20880 端口暴露服务 -->
<dubbo:protocol name="dubbo" port="20880" />
<!-- 声明需要暴露的服务接口 -->
<dubbo:service interface="com.taobao.service.ItemService" ref="itemServiceImpl" />

</beans>
```

3.6. 引用服务

1、在 taobao-manager-web 工程中添加 dubbo 依赖的 jar 包

```
<!-- dubbo 相关 -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>dubbo</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.jboss.netty</groupId>
            <artifactId>netty</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.apache.zookeeper</groupId>
    <artifactId>zookeeper</artifactId>
</dependency>
<dependency>
    <groupId>com.github.sgroschupf</groupId>
```

```
<artifactId>zkcclient</artifactId>
</dependency>
```

2、在 springmvc 的配置文件中添加服务的引用

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.2.xsd
http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.2.xsd">

<context:component-scan base-package="com.taobao.controller" />
<mvc:annotation-driven />
<bean
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>

<!-- 引用 dubbo 服务 -->
<dubbo:application name="taobao-manager-web"/>
<dubbo:registry protocol="zookeeper"
address="192.168.25.154:2181,192.168.25.154:2182,192.168.25.154:2183"/>
<dubbo:reference interface="com.taobao.service.ItemService" id="itemService" />
```

```
</beans>
```

3.7 Controller

```
@Controller
public class ItemController {

    @Autowired
    private ItemService itemService;

    @RequestMapping("/item/{itemId}")
    @ResponseBody
    public TbItem getItemById(@PathVariable Long itemId) {
```

```
//根据商品 id 查询商品信息
TbItem tbItem = itemService.getItemById(itemId);
return tbItem;
}
```

3.8. 解决 mapper 映射文件不发布问题

在 taobao-manager-dao 工程的 pom 文件中添加如下内容：

```
<!-- 如果不添加此节点 mybatis 的 mapper.xml 文件都会被漏掉。 -->
<build>
    <resources>
        <resource>
            <directory>src/main/java</directory>
            <includes>
                <include>**/*.properties</include>
                <include>**/*.xml</include>
            </includes>
            <filtering>false</filtering>
        </resource>
    </resources>
</build>
```

4. Dubbo 监控中心



需要安装 tomcat，然后部署监控中心即可。

1、部署监控中心：

```
[root@localhost ~]# cp dubbo-admin-2.5.4.war apache-tomcat-7.0.47/webapps/dubbo-admin.war
```

2、启动 tomcat

3、访问 <http://192.168.25.167:8080/dubbo-admin/>

用户名：root

密码：root

如果监控中心和注册中心在同一台服务器上，可以不需要任何配置。

如果不在同一台服务器，需要修改配置文件：

/root/apache-tomcat-7.0.47/webapps/dubbo-admin/WEB-INF/dubbo.properties



```
dubbo.registry.address=zookeeper://127.0.0.1:2181
dubbo.admin.root.password=root
dubbo.admin.guest.password=guest
```

5. 商品列表查询

5.1. 展示后台首页

5.1.1. 功能分析

请求的 url: /

参数: 无

返回值: 逻辑视图 String

5.1.2. Controller

```
@Controller
public class PageController {

    @RequestMapping("/")
    public String showIndex() {
        return "index";
    }

    @RequestMapping("/{page}")
    public String showPage(@PathVariable String page) {
        return page;
    }
}
```

5.2. 功能分析

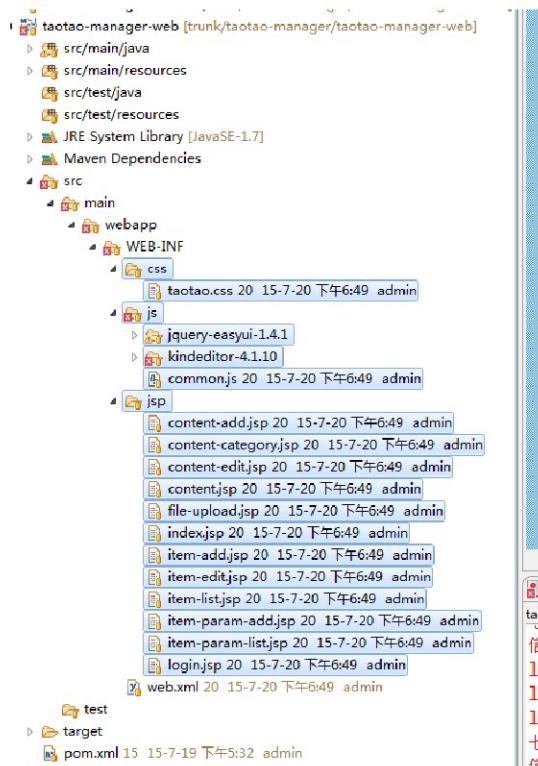
5.2.1. 整合静态页面

静态页面位置：02.第二天（三大框架整合，后台系统搭建）\01.参考资料\后台管理系统静态页面

名称	修改日期	类型	大小
css	2015/7/20 13:38	文件夹	
js	2015/7/20 13:38	文件夹	
content.jsp	2015/1/7 9:30	JSP 文件	4 KB
content-add.jsp	2015/1/7 9:31	JSP 文件	3 KB
content-category.jsp	2015/1/7 9:31	JSP 文件	4 KB
content-edit.jsp	2015/1/7 9:31	JSP 文件	11 KB
file-upload.jsp	2015/1/7 9:31	JSP 文件	3 KB
index.jsp	2015/7/20 16:54	JSP 文件	5 KB
item-add.jsp	2015/1/7 9:31	JSP 文件	5 KB
item-edit.jsp	2015/1/7 9:31	JSP 文件	5 KB
item-list.jsp	2015/1/7 9:31	JSP 文件	7 KB
item-param-add.jsp	2014/11/12 9:23	JSP 文件	4 KB
item-param-list.jsp	2015/1/7 9:31	JSP 文件	3 KB
login.jsp	2015/7/20 16:15	JSP 文件	2 KB

使用方法：

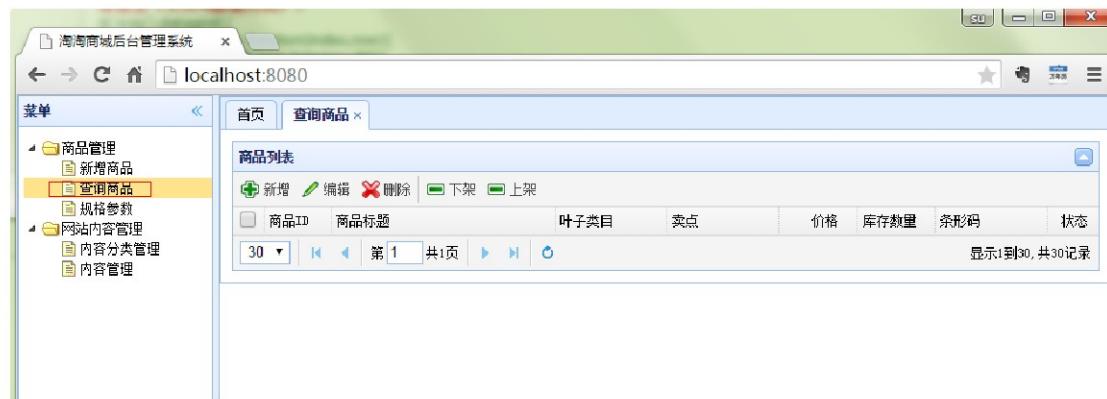
把静态页面添加到 taobao-manager-web 工程中的 WEB-INF 下：



由于在 web.xml 中定义的 url 拦截形式为 “/” 表示拦截所有的 url 请求，包括静态资源例如 css、js 等。所以需要在 springmvc.xml 中添加资源映射标签：

```
<mvc:resources location="/WEB-INF/js/" mapping="/js/**"/>
<mvc:resources location="/WEB-INF/css/" mapping="/css/**"/>
```

5.2.2. 商品列表页面



对应的 **jsp** 为：

item-list.jsp

请求的 **url**：

/item/list

请求的参数：

page=1&rows=30

响应的 **json** 数据格式：

Easyui 中 datagrid 控件要求的数据格式为：

```
{total:2,rows:[{"id":1,"name":"张三"}, {"id":2,"name":"李四"}]}
```

返回过滤数据显示。该函数带一个参数'data'用来指向源数据（即：获取的数据源，比如Json对象）。您可以改变源数据的标准数据格式。这个函数必须返回包含'total'和'rows'属性的标准数据对象。

5.2.3. 响应的 json 数据格式 EasyUIResult

```
public class EasyUIDataGridResult {

    private Integer total;

    private List<?> rows;

    public EasyUIResult(Integer total, List<?> rows) {
        this.total = total;
        this.rows = rows;
    }

    public EasyUIResult(Long total, List<?> rows) {
        this.total = total.intValue();
        this.rows = rows;
    }

    public Integer getTotal() {
        return total;
    }
}
```

```
public void setTotal(Integer total) {  
    this.total = total;  
}  
  
public List<?> getRows() {  
    return rows;  
}  
  
public void setRows(List<?> rows) {  
    this.rows = rows;  
}  
}
```

5.2.4. 分页处理

逆向工程生成的代码是不支持分页处理的，如果想进行分页需要自己编写 mapper，这样就失去逆向工程的意义了。为了提高开发效率可以使用 mybatis 的分页插件 PageHelper。

5.3. 分页插件 PageHelper

5.3.1. Mybatis 分页插件 - PageHelper 说明

如果你也在用 Mybatis，建议尝试该分页插件，这个一定是最方便使用的分页插件。
该插件目前支持 Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL 六种数据库分页。

5.3.2. 使用方法

第一步：把 PageHelper 依赖的 jar 包添加到工程中。官方提供的代码对逆向工程支持的不好，
使用参考资料中的 pagehelper-fix。



\01. 教案-2.1\01. 参考资料\mybatis

帮助 (H)

共享 ▾ 刻录 新建文件夹

名称	修改日期	类型	大小
mybatis3-dtd	2016/5/14 22:38	文件夹	
pagehelper-fix	2016/5/14 22:38	文件夹	
逆向工程	2016/5/14 22:38	文件夹	
druid配置.docx	2014/11/14 15:31	Microsoft Word...	18 KB
Mybatis分页插件 - PageHelper.docx	2014/12/31 15:55	Microsoft Word...	32 KB
pagehelper-fix.zip	2015/7/21 18:47	zip Archive	132 KB

第二步：在 Mybatis 配置 xml 中配置拦截器插件：

```
<plugins>
    <!-- com.github.pagehelper 为 PageHelper 类所在包名 -->
    <plugin interceptor="com.github.pagehelper.PageHelper">
        <!-- 设置数据库类型 Oracle, Mysql, MariaDB, SQLite, Hsqldb, PostgreSQL 六种数据库-->
        <property name="dialect" value="mysql"/>
    </plugin>
</plugins>
```

第二步：在代码中使用

1、设置分页信息：

```
//获取第 1 页, 10 条内容, 默认查询总数 count
PageHelper.startPage(1, 10);

//紧跟着的第一个 select 方法会被分页
List<Country> list = countryMapper.selectIf(1);
```

2、取分页信息

```
//分页后, 实际返回的结果 list 类型是 Page<E>, 如果想取出分页信息, 需要强制转换为 Page<E>,
Page<Country> listCountry = (Page<Country>)list;
listCountry.getTotal();
```

3、取分页信息的第二种方法

```
//获取第 1 页, 10 条内容, 默认查询总数 count
PageHelper.startPage(1, 10);

List<Country> list = countryMapper.selectAll();

//用 PageInfo 对结果进行包装
PageInfo page = new PageInfo(list);

//测试 PageInfo 全部属性
assertEquals(1, page.getPageNum());
assertEquals(10, page.getPageSize());
assertEquals(1, page.getStartRow());
assertEquals(10, page.getEndRow());
assertEquals(183, page.getTotal());
assertEquals(19, page.getPages());
assertEquals(1, page.getFirstPage());
assertEquals(8, page.getLastPage());
assertEquals(true, page.isFirstPage());
assertEquals(false, page.isLastPage());
assertEquals(false, page.isHasPreviousPage());
assertEquals(true, page.isHasNextPage());
```

5.3.3. 分页测试

```
@Test
public void testPageHelper() throws Exception {
    //初始化 spring 容器
```

```
ApplicationContext applicationContext = new ClassPathXmlApplicationContext("classpath:spring/applicationContext-*.xml");

//获得 Mapper 的代理对象
TbItemMapper itemMapper = applicationContext.getBean(TbItemMapper.class);

//设置分页信息
PageHelper.startPage(1, 30);

//执行查询
TbItemExample example = new TbItemExample();
List<TbItem> list = itemMapper.selectByExample(example);

//取分页信息
PageInfo<TbItem> pageInfo = new PageInfo<>(list);
System.out.println(pageInfo.getTotal());
System.out.println(pageInfo.getPages());
System.out.println(pageInfo.getPageNum());
System.out.println(pageInfo.getPageSize());

}
```

5.4. Service 层

参数: int page , int rows

业务逻辑: 查询所有商品列表, 要进行分页处理。

返回值: EasyUIDataGridResult

```
@Override
public EasyUIDataGridResult getItemList(int page, int rows) {

    //设置分页信息
    PageHelper.startPage(page, rows);
    //执行查询
    TbItemExample example = new TbItemExample();
    List<TbItem> list = itemMapper.selectByExample(example);
    //取分页信息
    PageInfo<TbItem> pageInfo = new PageInfo<>(list);

    //创建返回结果对象
    EasyUIDataGridResult result = new EasyUIDataGridResult();
    result.setTotal(pageInfo.getTotal());
    result.setRows(list);

    return result;
}
```

5.4.1. 发布服务

```

13
14      <!-- 使用dubbo发布服务 -->
15      <!-- 提供方应用信息，用于计算依赖关系 -->
16      <dubbo:application name="taotao-manager" />
17      <!-- 使用multicast广播注册中心暴露服务地址 -->
18      <!-- <dubbo:registry address="multicast://224.5.6.7:1234" /> -->
19      <dubbo:registry protocol="zookeeper" address="192.168.25.167:2181" />
20      <!-- 用dubbo协议在20880端口暴露服务 -->
21      <dubbo:protocol name="dubbo" port="20880" />
22      <!-- 声明需要暴露的服务接口 -->
23      <dubbo:service interface="com.taotao.service.ItemService" ref="itemServiceImpl" />
24
25  
```

5.5. 表现层

引用服务：

```

17      <property name="suffix" value=".jsp" />
18  
```

- 19 <!-- 静态资源映射 -->
- 20 <mvc:resources location="/css/" mapping="/css/**"/>
- 21 <mvc:resources location="/js/" mapping="/js/**"/>
- 22
- 23 <!-- 引用dubbo服务 -->
- 24 <dubbo:application name="taotao-manager-web"/>
- 25 <!-- <dubbo:registry address="multicast://224.5.6.7:1234" /> -->
- 26 <dubbo:registry protocol="zookeeper" address="192.168.25.167:2181" />
- 27 <dubbo:reference interface="com.taotao.service.ItemService" id="itemService" />
- 28
- 29

1、初始化表格请求的 url: /item/list

2、Datagrid 默认请求参数：

- 1、page：当前的页码，从 1 开始。
- 2、rows：每页显示的记录数。
- 3、响应的数据：json 数据。EasyUIDataGridResult

```

@RequestMapping("/item/list")
@ResponseBody
public EasyUIDataGridResult getItemList(Integer page, Integer rows) {
    EasyUIDataGridResult result = itemService.getItemList(page, rows);
    return result;
}

```

可以设置服务超时时间：

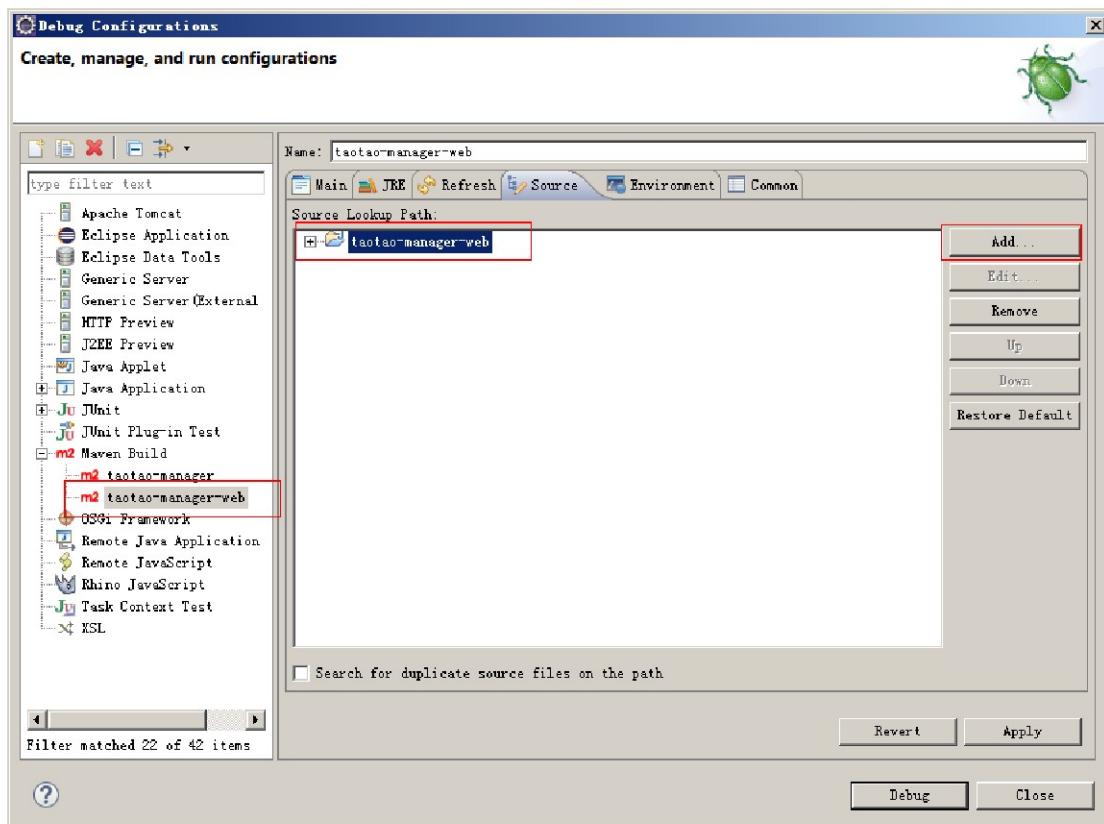
服务调用超时时间默认 1 秒，

```

11
12      <context:component-scan base-package="com.taotao.service"></context:component-scan>
13
14      <!-- 使用dubbo发布服务 -->
15      <!-- 提供方应用信息，用于计算依赖关系 -->
16      <dubbo:application name="taotao-manager" />
17      <!-- 使用multicast广播注册中心暴露服务地址 -->
18      <!-- <dubbo:registry address="multicast://224.5.6.7:1234" /> -->
19      <dubbo:registry protocol="zookeeper" address="192.168.25.167:2181" />
20      <!-- 用dubbo协议在20880端口暴露服务 -->
21      <dubbo:protocol name="dubbo" port="20880" />
22      <!-- 声明需要暴露的服务接口 -->
23      <dubbo:service interface="com.taotao.service.ItemService" ref="itemServiceImpl" timeout="300000" />
24
25  
```



Debug 设置源代码：



5.6. 安装 maven 工程跳过测试

clean install -DskipTests