# 课程计划

* Dubbo连接方式
* MyBatis回顾
* 分页插件
* 通用Mapper
* 集成通用Mapper和分页插件

# Dubbo连接方式

使用Dubbo进行远程调用实现服务交互，它支持多种协议，如Hessian、HTTP、RMI、Memcached、Redis等等。由于Dubbo将这些协议的实现进行了封装了，无论是服务端（开发服务）还是客户端（调用服务），都不需要关心协议的细节，只需要在配置中指定使用的协议即可，从而保证了服务提供方与服务消费方之间的透明。

Dubbo的客户端和服务端有三种连接方式，分别是：广播，直连和使用zookeeper注册中心。

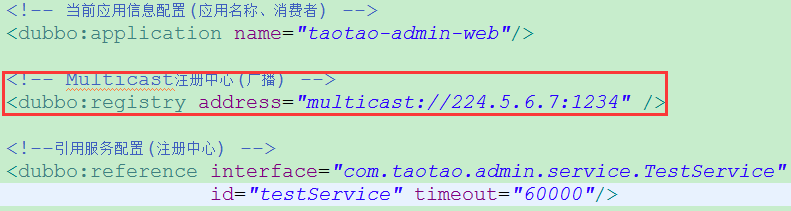
## Dubbo广播

这种方式是dubbo官方入门程序所使用的连接方式，但是这种方式有问题。在企业开发中，不使用广播的方式。

服务端(taotao-admin)配置：



客户端(taotao-admin-web)配置：

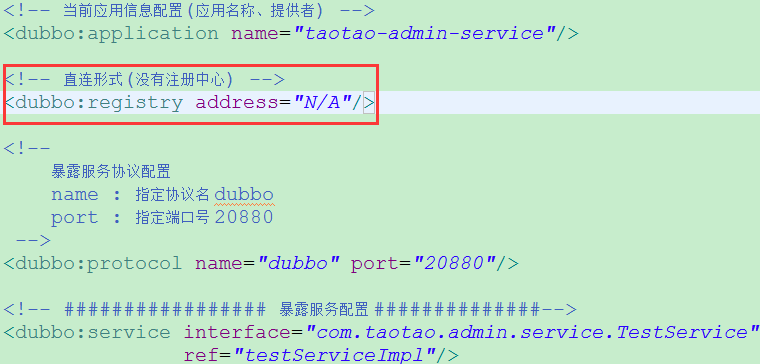


## Dubbo直连

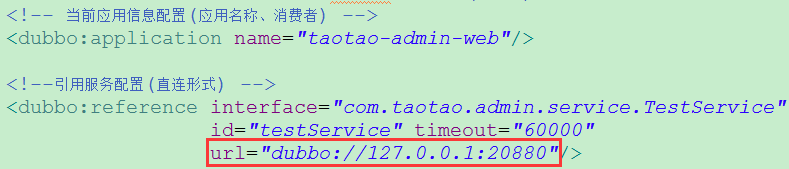
Dubbo直连，首先要取消广播，然后客户端直接到指定的url获取服务即可。

这种方式在企业中一般在开发中环境中使用，但是生产环境很少使用，因为服务是直接调用，没有使用注册中心，很难对服务进行管理。

服务端(taotao-admin)配置；修改如下，取消广播，注册地址为N/A



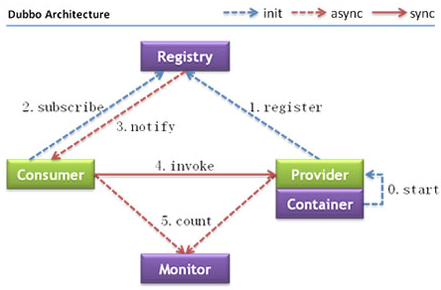
客户端(taotao-admin-web)配置；修改如下，取消广播，从指定的url中获取服务



## Dubbo注册中心

Dubbo注册中心和广播配置类似，不过需要指定注册中心类型和注册中心地址，这时不是把服务信息进行广播，而是将服务给注册中心进行管理，这个时候就需要有一个注册中心。官方推荐使用zookeeper作为注册中心。

### Zookeeper



注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者在启动时与注册中心交互，消费者不断的发起请求获取服务信息，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。

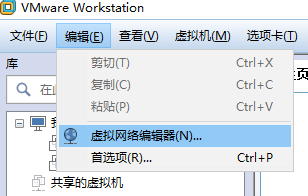
ZooKeeper是一个分布式的，开放源码的分布式应用程序协调服务，是apache组织的一个开源的框架，是Hadoop和Hbase的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

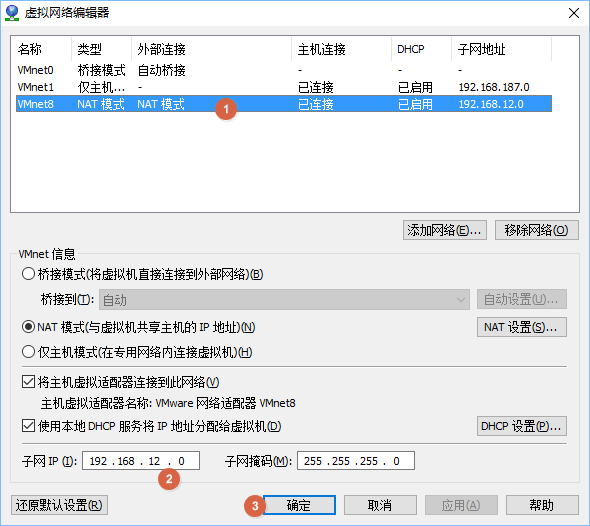
Zookeeper：

1. 可以作为集群的管理工具使用和注册中心；
2. 可以集中管理配置文件。

### 准备Linux环境

导入的虚拟机必须设置ip为192.168.12.128才可以使用FastDFS(后面的课程会用)；如果虚拟机不是12网段，需要手动设置网段，设置方式如下图：

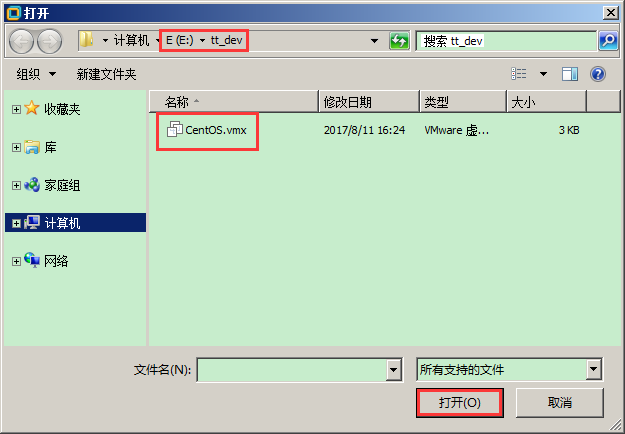




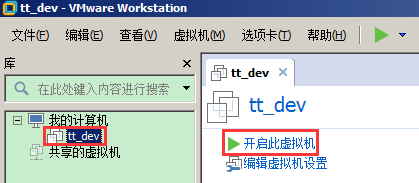
### 打开虚拟机到VMWare

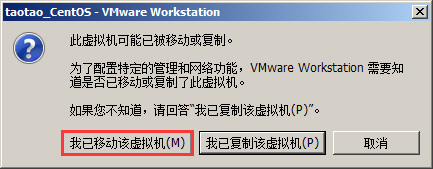
找到下发的虚拟机压缩文件“tt\_dev.7z；解压导入虚拟机内，步骤如下：





### 启动虚拟机





**注意：只能选择我已移动该虚拟机，因为我安装的软件ip不能变**

**否则无法启动该虚拟机。**

启动成功；则如下



上述安装的CentOS6.5系统；

帐号：root，密码：123456

### 安装zookeeper

详见“资料\zookeeper\ Linux安装zookeeper.docx”

### 使用zookeeper

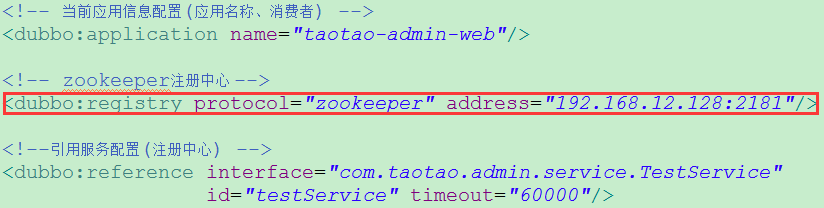
服务端与消费端都需要添加zookeeper依赖jar包，如下：

|  |
| --- |
| <!-- zookeeper分布式协调服务 -->  <dependency>  <groupId>org.apache.zookeeper</groupId>  <artifactId>zookeeper</artifactId>  </dependency>  <!-- zkclient(zookeeper客户端) -->  <dependency>  <groupId>com.github.sgroschupf</groupId>  <artifactId>zkclient</artifactId>  </dependency> |

服务端(taotao-admin)设置；使用zookeeper注册中心



客户端(taotao-admin-web)设置；指定zookeeper注册中心，删除之前配置的url



## Dubbo监控中心Monitor

使用了注册中心后，就可以很方便的使用Monitor监控中心查看服务提供方和消费方。而监控中心其实就是一个web应用，直接使用Tomcat进行部署即可。

监控中心Monitor的安装详见“资料\dubbo\安装部署dubbo的监控中心monitor.docx”

# MyBatis回顾

## 创建数据库

创建mybatis\_db数据库并导入： 资料\sql\notice测试数据.sql。

## 搭建项目

创建mybatis-pager Maven项目，注意packaging类型为**jar**即可，在这个项目中主要回顾mybatis的基本操作和使用分页助手、MBG、通用Mapper，写测试代码。

引入项目的依赖jar包；具体pom.xml文件如下：

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>cn.itcast</groupId>  <artifactId>mybatis-page</artifactId>  <version>0.0.1-SNAPSHOT</version>    <!-- 配置依赖关系 -->  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.4.1</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.38</version>  </dependency>  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>  </dependencies>    <!-- 构建部分 -->  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.1</version>  <configuration>  <source>1.7</source>  <target>1.7</target>  <encoding>utf-8</encoding>  </configuration>  </plugin>  </plugins>  </build>  </project> |

创建Mybatis的全局配置文件到resources文件夹：

|  |
| --- |
| <?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>  <!-- 加载属性文件 -->  <properties resource=*"jdbc.properties"*></properties>    <!-- 全局的设置信息-->  <settings>  <!-- 开启缓存 -->  <setting name=*"cacheEnabled"* value=*"true"*/>  <!-- 启用延迟加载功能 -->  <setting name=*"lazyLoadingEnabled"* value=*"true"*/>  <!-- 按需要延迟加载-->  <setting name=*"aggressiveLazyLoading"* value=*"false"*/>  <!-- 开启驼峰命名(方便自动映射) -->  <setting name=*"mapUnderscoreToCamelCase"* value=*"true"*/>  </settings>    <!-- 配置MyBatis3的的环境 -->  <environments default=*"project"*>  <environment id=*"project"*>  <!-- 配置事务管理器-->  <transactionManager type=*"JDBC"*/>  <!-- 配置数据源(用MyBatis3自带的连接池)-->  <dataSource type=*"POOLED"*>  <!-- 驱动类名 -->  <property name=*"driver"* value=*"${driver}"*/>  <!-- URL -->  <property name=*"url"* value=*"${url}"*/>  <!-- 用户名-->  <property name=*"username"* value=*"${username}"*/>  <!-- 密码 -->  <property name=*"password"* value=*"${password}"*/>  </dataSource>  </environment>  </environments>    <!-- 配置数据库表访问接口或映射文件 -->  <mappers>  <mapper resource=*"mapper/NoticeMapper.xml"*/>  </mappers>  </configuration> |

添加日志配置文件log4j.properties和jdbc.properties数据库配置文件到resources资源文件夹；

log4j.properties

|  |
| --- |
| log4j.rootLogger=DEBUG,stdout  log4j.appender.stdout=org.apache.log4j.ConsoleAppender  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  log4j.appender.stdout.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss,SSS} [%t] [%c]-[%p] %m%n |

jdbc.properties

|  |
| --- |
| driver=com.mysql.jdbc.Driver  url=jdbc:mysql://localhost:3306/mybatis\_db  username=root  password=root |

在cn.itcast.mybatis.pojo包下开发文实体类Notice.java，提供“NoticeMapper.xml” mybatis映射配置文件到resources/mapper/资源文件夹。

## 基本操作

### Mapper和映射文件

创建cn.itcast.mybatis.mapper.NoticeMapper接口，这里测试查询公告列表

|  |
| --- |
| **package** cn.itcast.mybatis.mapper;  **import** java.util.List;  **import** org.apache.ibatis.annotations.Select;  **import** cn.itcast.mybatis.pojo.Notice;  **public** **interface** NoticeMapper {  /\*\* 查询所有的公告 \*/  @Select("select \* from notice")  List<Notice> findAll();  } |

NoticeMapper.xml文件：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace=*"cn.itcast.mybatis.mapper.NoticeMapper"*>    </mapper> |

### 测试

测试代码中，创建NoticeMapperTest测试接口中的各个方法；测试类如下：

|  |
| --- |
| **public** **class** NoticeMapperTest {  /\*\* 定义NoticeMapper数据访问接口代理对象 \*/  **private** NoticeMapper noticeMapper;  /\*\* 获取SqlSession \*/  **private** SqlSession sqlSession;  @Before  **public** **void** before() **throws** Exception {  /\*\* 加载mybatis-config.xml文件产生输入流 \*/  InputStream inputStream = Resources.*getResourceAsStream*("mybatis-config.xml");  /\*\* 创建SqlSessionFactory \*/  SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(inputStream);  /\*\* 获取SqlSession \*/  sqlSession = sqlSessionFactory.openSession();  /\*\* 获取代理对象 \*/  noticeMapper = sqlSession.getMapper(NoticeMapper.**class**);  }  /\*\* 查询所有 \*/  @Test  **public** **void** findAll() {  List<Notice> notices = noticeMapper.findAll();  System.***out***.println(notices);  }  @After  **public** **void** after(){  sqlSession.commit();  sqlSession.close();  }  } |

# 分页助手

在开发过程中对数据进行分页查询是很常见的功能；这里引入一个第三方开发的Mybatis分页插件PageHelper。

详细见《mybatis框架相关.xmind》

## 配置

1. 引入依赖，将下面的依赖信息加入到项目pom.xml文件中；

|  |
| --- |
| <!-- pagehelper分页插件 -->  <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  <version>5.0.0</version>  </dependency> |

1. 在mybatis-config.xml文件中添加配置信息；在environments标签之前添加如下配置信息：

|  |
| --- |
| <!-- 配置插件 -->  <plugins>  <plugin interceptor=*"com.github.pagehelper.PageInterceptor"*>  <!-- 配置方言 -->  <property name=*"helperDialect"* value=*"mysql"*/>  <!-- 配置统计总录数 -->  <property name=*"rowBoundsWithCount"* value=*"true"*/>  </plugin>  </plugins> |

## 应用

PageHelper分页助手在使用过程中比较简单；只需要在查询之前调用如下的代码即可：PageHelper.startPage(page, rows);

上面的page表示 页号，rows 表示 每页数据大小。对其测试如下：

|  |
| --- |
| /\*\* 分页查询 \*/  @Test  **public** **void** findByPage(){  /\*\* 设置开始分页 \*/  PageHelper.*startPage*(1, 3);  /\*\* 查询 \*/  List<Notice> lists = noticeMapper.findAll();  /\*\* 创建PageInfo封装数据 \*/  PageInfo<Notice> pageInfo = **new** PageInfo<>(lists);  **for** (Notice notice : pageInfo.getList()){  System.***out***.println(notice);  }  System.***out***.println("总共 " + pageInfo.getTotal() + " 条数据；当前第 " + pageInfo.getPageNum()  + " 页；总共 " + pageInfo.getPages() + " 页；。");  } |

# 逆向工程

Mybatis的逆向工程需要添加一个Eclipse插件Mybatis Generator；该插件可以进行逆向工程。详见 资料\mybatis\ 中的《mybatis框架相关.xmind》

# 通用Mapper

通用Mapper是什么？是一个分页助手的作者编写的api，主要目的是：使用了这组api后，我们在开发过程中常见的CRUD代码可以省略不写，因为这些操作基本上比较固定和通用，所以对这些常用的操作进行了封装；于是有了通用Mapper。简而言之；通用Mapper就是为了提高开发效率而存在的。

详见 资料\mybatis\ 中的《mybatis框架相关.xmind》

## 配置

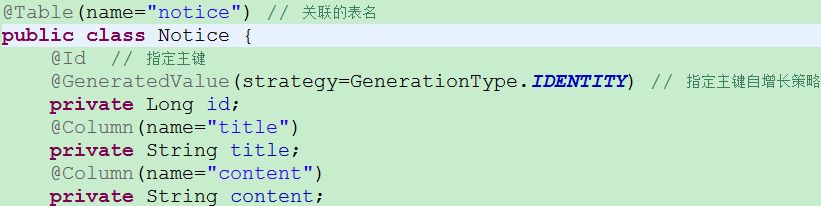
在未使用Spring之前，对通用Mapper的配置只能在代码中进行。Spring中集成Mapper在淘淘商城项目中再配置。这里只需要引入通用Mapper的依赖即可。

|  |
| --- |
| <!-- 配置通用mapper -->  <dependency>  <groupId>tk.mybatis</groupId>  <artifactId>mapper</artifactId>  <version>3.4.0</version>  </dependency> |

## 应用

通用Mapper所进行的也是将查询结果与具体的实体类进行映射，但如果是一般的数据库操作，它是不再需要写\*Mapper.xml文件的；它通过jpa注解实现数据库表与实体类的结果映射。搭建mybatis\_mapper maven项目做测试

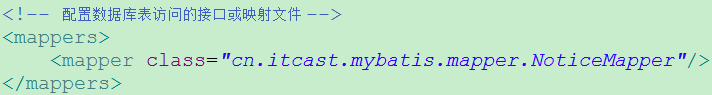
1. 写Notice类，并对各个属性添加相应的JPA注解；



1. 编写NoticeMapper继承Mapper；

|  |
| --- |
| **package** cn.itcast.mybatis.mapper;  **import** tk.mybatis.mapper.common.Mapper;  **import** cn.itcast.mybatis.pojo.Notice;  **public** **interface** NoticeMapper **extends** Mapper<Notice> {    } |

1. 将NoticeMapper添加到mybatis-config.xml总配置文件



1. 编写NoticeMapperTest测试基本的操作。

|  |
| --- |
| **public** **class** NoticeMapperTest {    /\*\* 定义NoticeMapper数据访问接口代理对象 \*/  **private** NoticeMapper noticeMapper;  /\*\* 获取SqlSession \*/  **private** SqlSession sqlSession;    @Before  **public** **void** before() **throws** Exception {  /\*\* 加载mybatis-config.xml文件产生输入流 \*/  InputStream inputStream = Resources.*getResourceAsStream*("mybatis-config.xml");  /\*\* 创建SqlSessionFactory \*/  SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(inputStream);  /\*\* 获取SqlSession \*/  sqlSession = sqlSessionFactory.openSession();  /\*\* 获取代理对象 \*/  noticeMapper = sqlSession.getMapper(NoticeMapper.**class**);    /\*\* 创建MapperHelper \*/  MapperHelper mapperHelper = **new** MapperHelper();  /\*\* 注册自己项目中使用的通用Mapper接口，这里没有默认值，必须手动注册 \*/  mapperHelper.registerMapper(NoticeMapper.**class**);  /\*\* 执行配置信息 \*/  mapperHelper.processConfiguration(sqlSession.getConfiguration());  }    /\*\* 根据主键id查询 \*/  @Test  **public** **void** findOne() {  Notice notice = noticeMapper.selectByPrimaryKey(10L);  System.***out***.println(notice);  }  /\*\* 查询所有 \*/  @Test  **public** **void** findAll() {  List<Notice> notices = noticeMapper.selectAll();  System.***out***.println(notices);  }  /\*\* 分页查询 \*/  @Test  **public** **void** findByPage() {  /\*\* 设置开始分页 \*/  PageHelper.*startPage*(1, 3);  /\*\* 查询 \*/  List<Notice> lists = noticeMapper.selectAll();  /\*\* 创建PageInfo封装数据 \*/  PageInfo<Notice> pageInfo = **new** PageInfo<>(lists);  **for** (Notice notice : pageInfo.getList()){  System.***out***.println(notice);  }  System.***out***.println("总共 " + pageInfo.getTotal() + " 条数据；当前第 " + pageInfo.getPageNum()  + " 页；总共 " + pageInfo.getPages() + " 页；。");  }  /\*\* 添加 \*/  @Test  **public** **void** save() {  Notice n = **new** Notice();  n.setTitle("李小华");  n.setContent("李中华");  noticeMapper.insertSelective(n);  System.***out***.println(n.getId());  }  /\*\* 修改 \*/  @Test  **public** **void** update() {  Notice n = **new** Notice();  n.setId(22l);  n.setTitle("李大华");  noticeMapper.updateByPrimaryKeySelective(n);  System.***out***.println(n.getId());  }  /\*\* 删除 \*/  @Test  **public** **void** delete() {  noticeMapper.deleteByPrimaryKey(22l);  }  @After  **public** **void** after(){  sqlSession.commit();  sqlSession.close();  }  } |

## 小结

在通用Mapper的方法中；insert和insertSelective或者其它方法带有Selective的与没有带的是有区别的；区别就是：带有Selective的方法进行操作时如果属性为null则对应的属性不会进行操作。如：insertSelective插入一个对象时，如果对象中的name属性为null那么它在转为sql语句时，name字段是不会出现在sql语句中；而没有带Selective的则会出现。

# 集成通用Mapper和分页助手

## 加入依赖

集成通用Mapper，需要给pojo加JPA注解，所以通用Mapper依赖需要加到taotao-admin-pojo项目的pom.xml

|  |
| --- |
| <dependencies>  <!-- 配置通用mapper -->  <dependency>  <groupId>tk.mybatis</groupId>  <artifactId>mapper</artifactId>  </dependency>  </dependencies> |

分页逻辑是在service业务层进行，所以分页插件依赖本应该加到taotao-admin-service但是有可能表现层项目需要依赖它的分页实体所以加入taotao-admin-interface的pom.xml更合适。

|  |
| --- |
| <!-- 配置分页插件 -->  <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  </dependency> |

## 修改配置文件

### 分页助手配置

打开/taotao-admin-service/src/main/resources/mybatis-config.xml

添加如下配置信息：

|  |
| --- |
| <!-- 配置插件 -->  <plugins>  <plugin interceptor=*"com.github.pagehelper.PageInterceptor"*>  <!-- 配置方言 -->  <property name=*"helperDialect"* value=*"mysql"*/>  <!-- 配置统计总录数 -->  <property name=*"rowBoundsWithCount"* value=*"true"*/>  </plugin>  </plugins> |

### 通用Mapper配置

打开/taotao-admin-service/src/main/resources/applicationContext.xml

添加如下配置信息：

|  |
| --- |
| <!-- 配置通用Mapper集成Spring4 -->  <bean class=*"tk.mybatis.spring.mapper.MapperScannerConfigurer"*>  <!-- 设置数据访问接口类 -->  <property name=*"basePackage"* value=*"com.taotao.admin.mapper"*/>  </bean> |

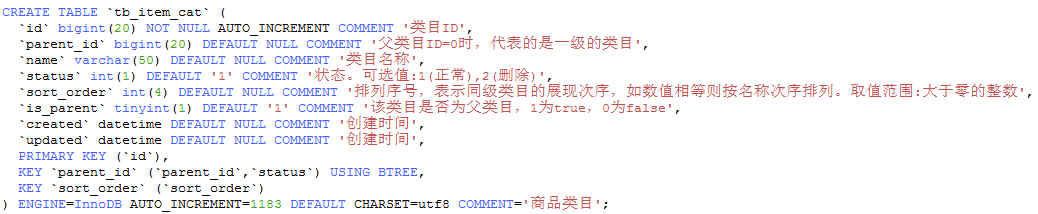
## 实现分页查询商品类目

验证集成分页助手和通用Mapper是否成功；

测试地址为：[http://localhost:9091/itemcat/1?rows=5](http://localhost:9091/rest/item/cat/query/1?rows=5)

### 导入商品类目数据库表

将“资料\sql\tb\_item\_cat(商品类目).sql”文件导入到数据库中。



### 引入ItemCat

引入“资料\pojo”文件夹下的ItemCat.java文件到taotao-admin-pojo项目中。

### 编写Mapper

在taotao-admin-mapper项目编写如下接口：

|  |
| --- |
| **package** com.taotao.admin.mapper;  **import** com.taotao.admin.pojo.ItemCat;  **import** tk.mybatis.mapper.common.Mapper;  **public** **interface** ItemCatMapper **extends** Mapper<ItemCat> {  } |

### 编写Service接口

在taotao-admin-interface项目编写如下接口：

|  |
| --- |
| **package** com.taotao.admin.service;  **import** java.util.List;  **import** com.taotao.admin.pojo.ItemCat;  **public** **interface** ItemCatService {  List<ItemCat> findItemCatByPage(Integer page, Integer rows);  } |

### 编写Service实现类

在taotao-admin-service项目编写如下实现类：

|  |
| --- |
| **package** com.taotao.manage.service.impl;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** com.github.pagehelper.PageHelper;  **import** com.taotao.admin.mapper.ItemCatMapper;  **import** com.taotao.admin.pojo.ItemCat;  **import** com.taotao.admin.service.ItemCatService;  @Service  **public** **class** ItemCatServiceImpl **implements** ItemCatService {  @Autowired  **private** ItemCatMapper itemCatMapper;  @Override  **public** List<ItemCat> findItemCatByPage(Integer page, Integer rows) {  //设置分页  PageHelper.*startPage*(page, rows);  //查询  List<ItemCat> list = itemCatMapper.selectAll();  **return** list;  }  } |

### 服务配置

在/taotao-admin-service/src/main/resources/applicationContext-dubbo.xml

文件中暴露服务：

|  |
| --- |
| <dubbo:service interface=*"com.taotao.admin.service.ItemCatService"*  ref=*"itemCatServiceImpl"*/> |

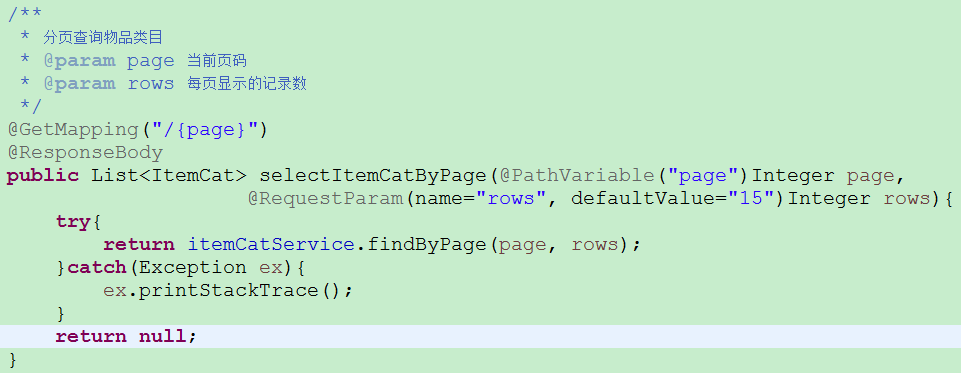
在/taotao-admin-web/src/main/resources/taotao-admin-web-servlet.xml

文件中声明要调用的服务

|  |
| --- |
| <dubbo:reference interface=*"com.taotao.admin.service.ItemCatService"*  id=*"itemCatService"* timeout=*"60000"*/> |

### 编写Controller

在taotao-admin-web项目编写如下类：



### 测试

安装taotao-admin项目后，分别启动taotao-admin和taotao-admin-web两个项目；测试结果如下：

