**淘淘商城**

**第二天**

# 课程计划

1. Dubbo连接方式
   1. Monitor(了解)
2. MyBatis回顾
3. 通用Mapper
4. 分页插件
5. 集成通用Mapper和分页插件
6. svn版本管理

# 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，但是这种方式有很多问题。

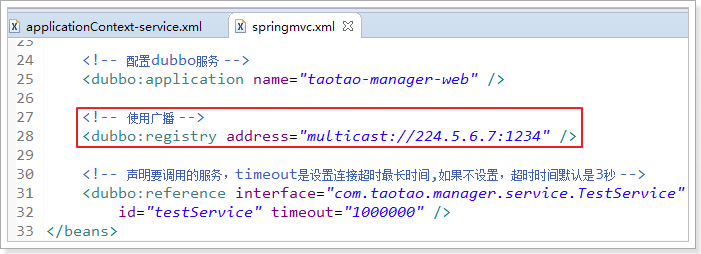
在企业开发中，不使用广播的方式。

taotao-manager服务端配置：



客户端配置：

taotao-manager-web的配置修改如下：



## Dubbo直连

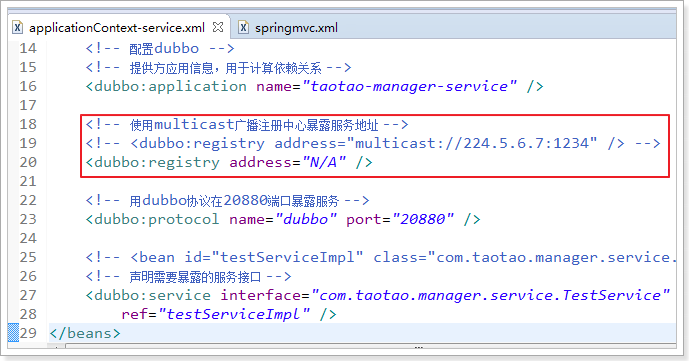
Dubbo直连，首先要取消广播，然后客户端直接到指定的url获取服务即可。

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。

服务端配置：

taotao-manager的修改如下，取消广播，注册地址为N/A



客户端配置：

taotao-manager-web配置如下，取消广播，从指定的url中获取服务



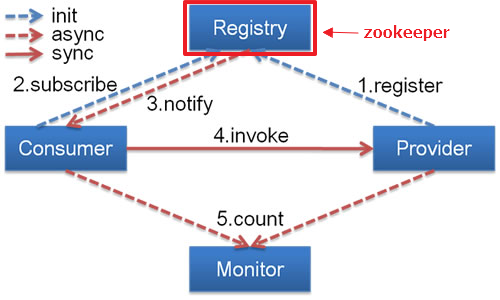
## Dubbo注册中心

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，这个时候就不是把服务信息进行广播了，而是告诉给注册中心进行管理，这个时候我们就需要有一个注册中心。

官方推荐使用zookeeper作为注册中心。

### Zookeeper介绍

下图是zookeeper在dubbo所处的位置：



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Apacahe 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

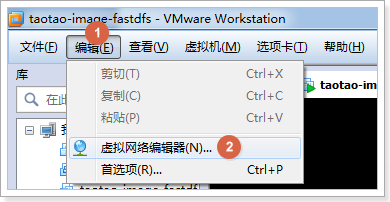
Zookeeper：

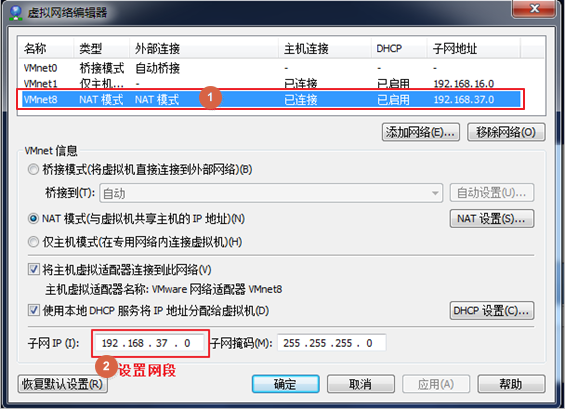
1. 可以作为集群的管理工具使用，和注册中心。
2. 可以集中管理配置文件。

### 准备Linux环境

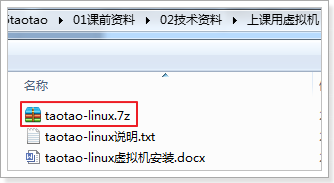
导入的虚拟机必须设置ip为192.168.37.161才可以使用FastDFS(后面的课程会用)

如果虚拟机不是37网段，需要手动设置网段，设置方式如下图

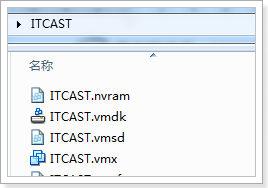




课程使用已经准备好的虚拟机，位置如下图：

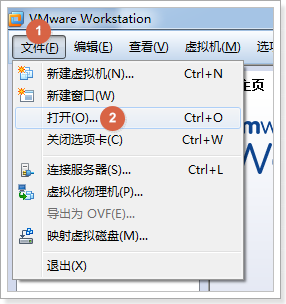


### 解压虚拟机

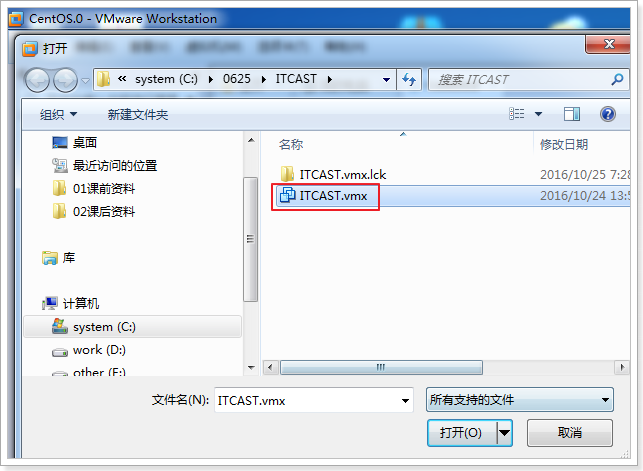


### 导入虚拟机到VMWare中

选择打开虚拟机

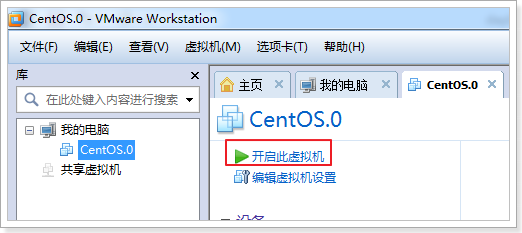


选中vmx文件

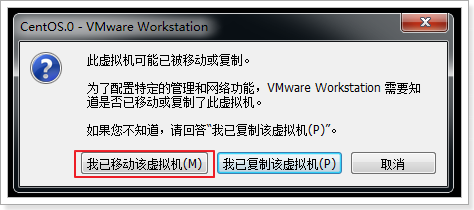


### 开启导入的虚拟机

如下图开启虚拟机：



选择移动虚拟机

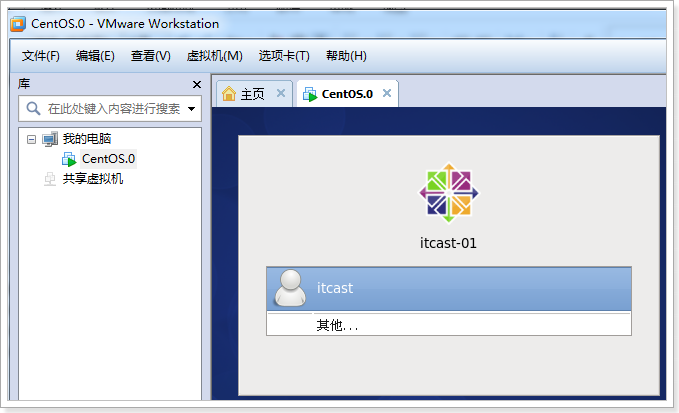


如果使用192.168.37.161作为IP地址，这里选择移动虚拟机。

移动虚拟机的作用是网卡和原来的网卡信息一样，ip是192.168.37.161

复制虚拟机的作用是，创建一个新网卡，ip是自动分配ip

进入登录页面，表示成功开启



Linux系统

账号:root 密码:root

### Zookeeper的安装

安装环境：

Linux：CentOS6.5

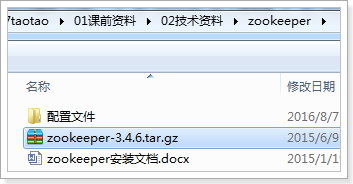
Jdk:1.7以上版本

Zookeeper是java开发的，可以运行在windows、linux环境。

需要先安装jdk（课前资料的虚拟机，已经安装好JDK）

1. 上传zookeeper

把下图所在的zookeeper上传到Linux



1. 移动到/usr/local

[root@itcast-01 ~]# mv zookeeper-3.4.6.tar.gz /usr/local/

1. 解压

[root@itcast-01 local]# tar -zxvf zookeeper-3.4.6.tar.gz

1. 进入解压目录，创建data

[root@itcast-01 local]# cd zookeeper-3.4.6

[root@itcast-01 zookeeper-3.4.6]# mkdir data

1. 进入conf目录，把zoo\_sample.cfg复制一份叫zoo.cfg

[root@itcast-01 zookeeper-3.4.6]# cd conf/

[root@itcast-01 conf]# cp zoo\_sample.cfg zoo.cfg

1. 修改修改zoo.cfg

[root@itcast-01 conf]# vim zoo.cfg

修改属性：

dataDir=/usr/local/zookeeper-3.4.6/data（新建的data目录所在的位置）

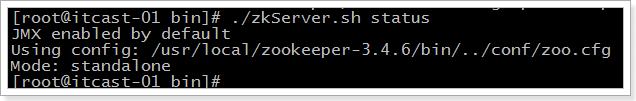
1. 进入bin目录启动zookeeper

[root@itcast-01 bin]# pwd

/usr/local/zookeeper-3.4.6/bin

[root@itcast-01 bin]# ./zkServer.sh start

查看状态：[root@localhost bin]# ./zkServer.sh status



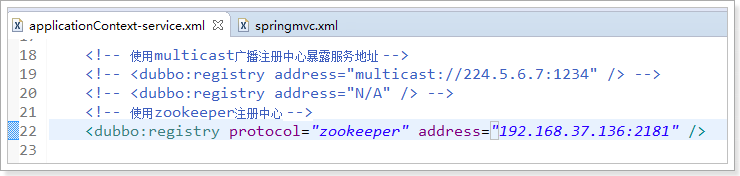
注意：zookeeper使用2181端口号，为了能正常使用zookeeper，我们需要打开2181端口号，或者关闭防火墙

关闭防火墙：[root@itcast-01 bin]# service iptables stop

### zookeeper使用

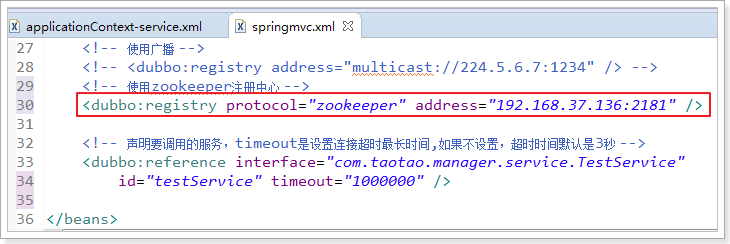
在服务端设置：

taotao-manager使用zookeeper注册中心：



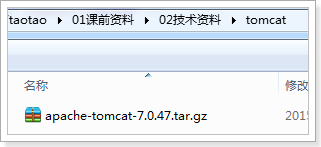
在客户端设置：

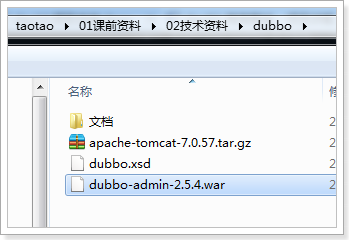
taotao-manager-web指定zookeeper注册中心，删除之前配置的url



## Dubbo的Monitor（了解）

使用了注册中心后，我们就可以很方便的使用Monitor监控中心了。监控中心其实就是一个web应用，我们直接使用Tomcat进行部署即可





配置monitor的方法：

1. 把dubbo-admin.war上传到linux服务器

1. 安装tomcat后（解压即可，参考Linux课程），把war包部署到tomcat中。

[root@itcast-01 webapps]# pwd

/usr/local/web/tomcat/webapps

删除原来的应用，创建ROOT，并移动

[root@itcast-01 webapps]# rm -rf \*

[root@itcast-01 webapps]# mkdir ROOT

[root@itcast-01 webapps]# mv /root/dubbo-admin-2.5.4.war ROOT/

解压war包，并删除

[root@itcast-01 ROOT]# jar -xf dubbo-admin-2.5.4.war

[root@itcast-01 ROOT]# rm -f dubbo-admin-2.5.4.war

3. 修改/dubbo-admin/WEB-INF/dubbo.properties属性文件。

dubbo.registry.address=zookeeper://127.0.0.1:2181

为注册中的实际地址。

1. 启动tomcat，

访问http://192.168.37.161:8080

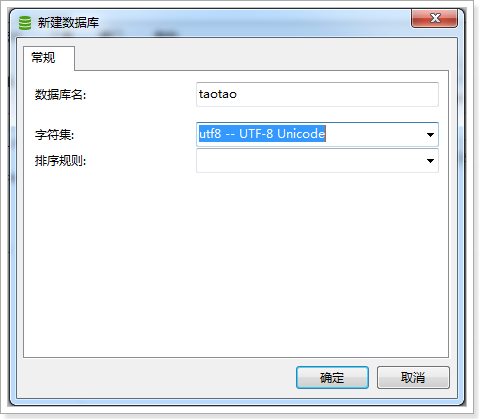
用户名：root 密码：root



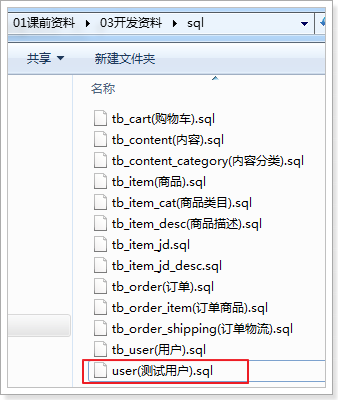
# mybatis回顾

## 数据库准备

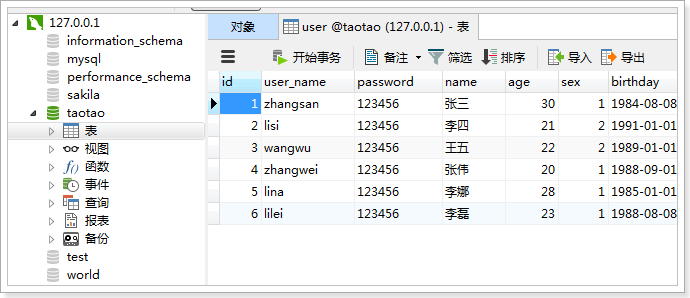
在MySQL中创建数据库taotao



导入课前资料中的User

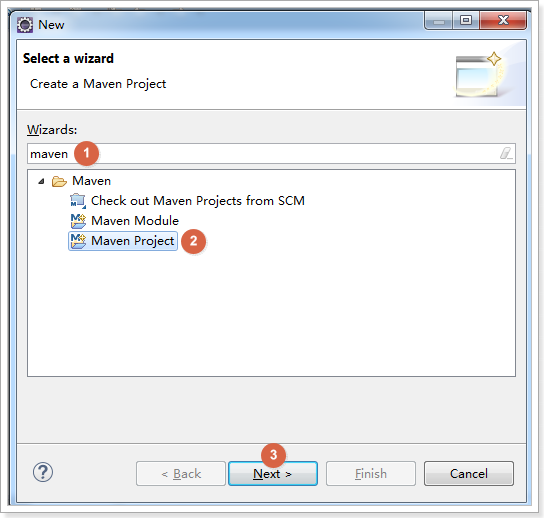


效果

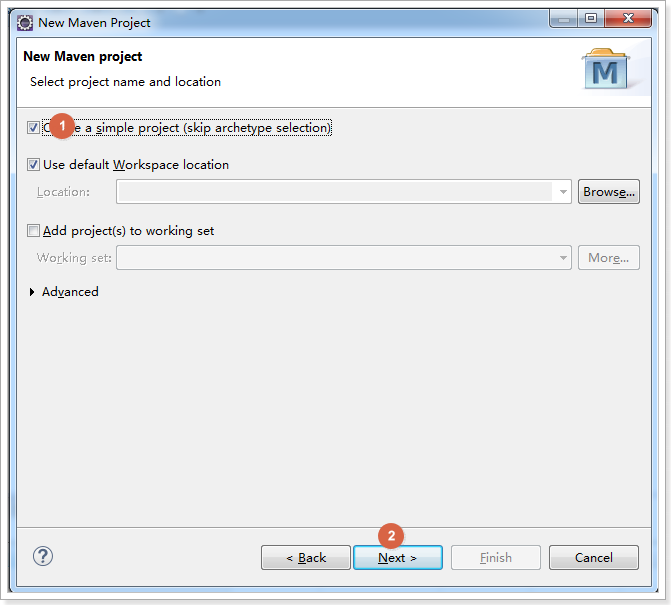


## 创建itcast-mybatis

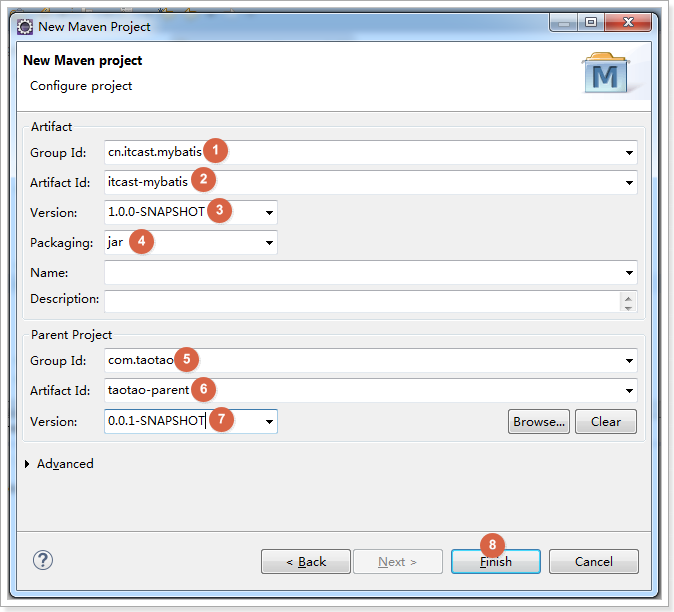
### 创建Maven工程



### 选择简单的Maven工程



### 填写信息



## 使用java视图模式



## 加入依赖

需要加入的依赖

1. Junit依赖
2. Mybatis依赖
3. Mysql依赖

4. 日志依赖

pom.xml

<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>

<parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>cn.itcast.mybatis</groupId>

<artifactId>itcast-mybatis</artifactId>

<version>1.0.0-SNAPSHOT</version>

<dependencies>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

<!-- Mybatis -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

</dependency>

<!-- MySql -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

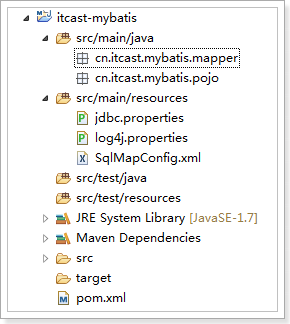
</dependency>

</dependencies>

</project>

## 加入配置文件

### 配置文件结构



jdbc.properties

jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://127.0.0.1:3306/taotao?useUnicode=true&characterEncoding=utf8&autoReconnect=true

jdbc.username=root

jdbc.password=root

log4j.properties

log4j.rootLogger=DEBUG,A1

log4j.logger.com.taotao = DEBUG

log4j.logger.org.mybatis = DEBUG

log4j.appender.A1=org.apache.log4j.ConsoleAppender

log4j.appender.A1.layout=org.apache.log4j.PatternLayout

log4j.appender.A1.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss,SSS} [%t] [%c]-[%p] %m%n

SqlMapConfig.xml

<?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>

<!-- 引入外部资源配置文件 -->

<properties resource=*"jdbc.properties"* />

<settings>

<!-- 开启驼峰自动映射 -->

<setting name=*"mapUnderscoreToCamelCase"* value=*"true"* />

</settings>

<typeAliases>

<package name=*"cn.itcast.mybatis.pojo"* />

</typeAliases>

<!-- 配置环境，制定数据库连接信息 -->

<environments default=*"development"*>

<environment id=*"development"*>

<transactionManager type=*"JDBC"* />

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*"${jdbc.driver}"* />

<property name=*"url"* value=*"${jdbc.url}"* />

<property name=*"username"* value=*"${jdbc.username}"* />

<property name=*"password"* value=*"${jdbc.password}"* />

</dataSource>

</environment>

</environments>

<mappers>

<package name=*"cn.itcast.mybatis.mapper"* />

</mappers>

</configuration>

## 实现增删改查

### 创建pojo

**public** **class** User {

// 主键

**private** Long id;

// 用户名

**private** String userName;

// 密码

**private** String password;

// 姓名

**private** String name;

// 年龄

**private** Integer age;

// 性别，1男性，2女性

**private** Integer sex;

// 出生日期

**private** Date birthday;

// 创建时间

**private** Date created;

// 更新时间

**private** Date updated;

get/set。。。。。。

### 创建Mapper

**public** **interface** UserMapper {

/\*\*

\* 根据id查询用户

\*

\* **@param** id

\* **@return**

\*/

User queryUserById(Long id);

/\*\*

\* 新增

\*

\* **@param** user

\*/

**void** saveUser(User user);

/\*\*

\* 更新

\*

\* **@param** user

\*/

**void** updateUserById(User user);

/\*\*

\* 根据id删除

\*

\* **@param** id

\*/

**void** deleteUserById(Long id);

}

### 创建UserMapper.xml

<?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.UserMapper"*>

<!-- 根据id查询用户 -->

<select id=*"queryUserById"* parameterType=*"long"* resultType=*"user"*>

SELECT \* FROM `user` WHERE id=#{id}

</select>

<!-- 新增 -->

<insert id=*"saveUser"* useGeneratedKeys=*"true"*

keyColumn=*"id"* keyProperty=*"id"*>

INSERT INTO `user`

(

`user\_name`,

`password`,

`name`,

`age`,

`sex`,

`birthday`,

`created`,

`updated`

)

VALUES

(

#{userName},

#{password},

#{name},

#{age},

#{sex},

#{birthday},

NOW(),

NOW()

)

</insert>

<!-- 更新 -->

<update id=*"updateUserById"*>

UPDATE `user`

SET

`user\_name` = #{userName},

`password` = #{password},

`name` = #{name},

`age` = #{age},

`sex` = #{sex},

`birthday` = #{birthday},

`updated` = NOW()

WHERE

(`id` = #{id});

</update>

<!-- 根据id删除 -->

<delete id=*"deleteUserById"*>

DELETE FROM `user` WHERE id=#{id}

</delete>

</mapper>

### 测试

**public** **class** UserMapperTest {

**private** SqlSessionFactory sqlSessionFactory;

@Before

**public** **void** setUp() **throws** Exception {

SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();

InputStream inputStream = Resources.*getResourceAsStream*("SqlMapConfig.xml");

**this**.sqlSessionFactory = builder.build(inputStream);

}

@Test

**public** **void** testQueryUserById() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);

UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);

User user = userMapper.queryUserById(2l);

System.***out***.println(user);

sqlSession.close();

}

@Test

**public** **void** testSaveUser() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);

UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);

User user = **new** User();

user.setId(**null**);

user.setName("韩梅梅");

user.setSex(2);

user.setUserName("hanmeimei3");

userMapper.saveUser(user);

System.***out***.println(user);

sqlSession.close();

}

@Test

**public** **void** testUpdateUserById() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);

UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);

User user = **new** User();

user.setId(9l);

user.setName("李传文");

user.setSex(3);

user.setUserName("lichuanwen");

userMapper.updateUserById(user);

sqlSession.close();

}

@Test

**public** **void** testDeleteUserById() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);

UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);

userMapper.deleteUserById(6l);

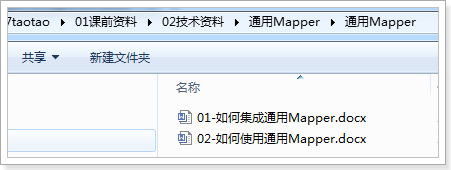
sqlSession.close();

}

}

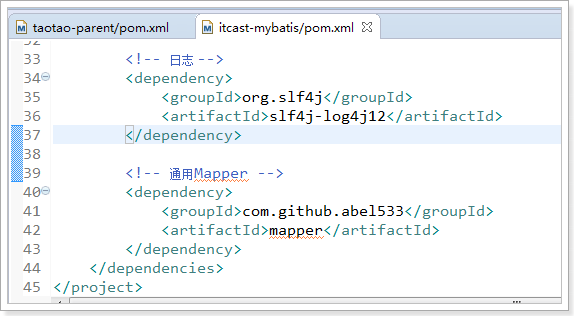
# 通用Mapper

文档地址



## 集成通用Mapper

### 加入通用Mapper依赖



### 在Mybatis配置文件中集成

添加plugins插件，注意在environments标签上面

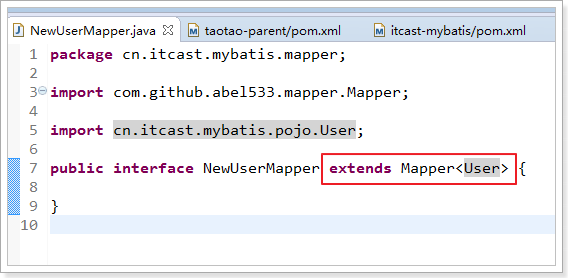


## 使用通用Mapper

### 创建Mapper

创建Mapper，继承通用Mapper即可，

通用Mapper的泛型为需要操作的JavaBean

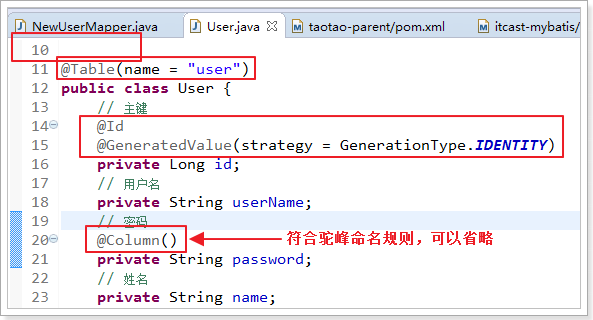


### 修改pojo

修改pojo，增加JPA注解

作用是把类名和表名进行映射

类中的属性名和表中的列名映射



## 测试用例

**public** **class** NewUserMapperTest {

**private** NewUserMapper newUserMapper;

@Before

**public** **void** init() **throws** Exception {

InputStream inputStream = Resources.*getResourceAsStream*("SqlMapConfig.xml");

SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(inputStream);

// 参数为true，设置事务是自动提交

SqlSession sqlSession = sqlSessionFactory.openSession(**true**);

**this**.newUserMapper = sqlSession.getMapper(NewUserMapper.**class**);

}

@Test

**public** **void** testSelectOne() {

User param = **new** User();

param.setId(1l);

// param.setSex(1);

User user = **this**.newUserMapper.selectOne(param);

System.***out***.println(user);

}

@Test

**public** **void** testSelect() {

User param = **new** User();

param.setSex(1);

// 查询所有数据参数为null

List<User> list = **this**.newUserMapper.select(param);

**for** (User user : list) {

System.***out***.println(user);

}

}

@Test

**public** **void** testSelectCount() {

User param = **new** User();

param.setSex(1);

// 查询所有数据参数为null

**int** count = **this**.newUserMapper.selectCount(**null**);

System.***out***.println(count);

}

@Test

**public** **void** testSelectByPrimaryKey() {

User user = **this**.newUserMapper.selectByPrimaryKey(1l);

System.***out***.println(user);

}

@Test

**public** **void** testInsert() {

User user = **new** User();

user.setId(**null**);

user.setUserName("caocao1");

user.setName("曹操");

user.setSex(1);

**this**.newUserMapper.insert(user);

System.***out***.println(user);

}

@Test

**public** **void** testInsertSelective() {

User user = **new** User();

user.setId(**null**);

user.setUserName("caocao");

user.setName("曹操");

user.setSex(1);

**this**.newUserMapper.insertSelective(user);

System.***out***.println(user);

}

@Test

**public** **void** testDelete() {

User param = **new** User();

param.setUserName("caocao1");

**this**.newUserMapper.delete(param);

}

@Test

**public** **void** testDeleteByPrimaryKey() {

*fail*("Not yet implemented");

}

@Test

**public** **void** testUpdateByPrimaryKey() {

User user = **new** User();

user.setId(11l);

user.setUserName("caocao2");

**this**.newUserMapper.updateByPrimaryKey(user);

}

@Test

**public** **void** testUpdateByPrimaryKeySelective() {

User user = **new** User();

user.setId(11l);

user.setUserName("caocao1");

**this**.newUserMapper.updateByPrimaryKeySelective(user);

}

// ---上面方法参数一般是user这个javabean，只能做单条件查询------

// 一下使用example条件查询

@Test

**public** **void** testSelectCountByExample() {

// 创建example,构造器为JavaBean的class

Example example = **new** Example(User.**class**);

// 设置条件

// 获取设置条件的对象

Criteria criteria = example.createCriteria();

// 设置

List<Object> ids = **new** ArrayList<>();

ids.add(1l);

ids.add(2l);

criteria.andIn("id", ids);

**int** count = **this**.newUserMapper.selectCountByExample(example);

System.***out***.println(count);

}

@Test

**public** **void** testDeleteByExample() {

*fail*("Not yet implemented");

}

@Test

**public** **void** testSelectByExample() {

// 创建example,构造器为JavaBean的class

Example example = **new** Example(User.**class**);

// 设置条件

// 获取设置条件的对象

Criteria criteria = example.createCriteria();

// 设置

List<Object> ids = **new** ArrayList<>();

ids.add(1l);

ids.add(2l);

criteria.andIn("id", ids);

List<User> list = **this**.newUserMapper.selectByExample(example);

**for** (User user : list) {

System.***out***.println(user);

}

}

@Test

**public** **void** testUpdateByExampleSelective() {

// 设置更新条件

Example example = **new** Example(User.**class**);

// 声明条件的list

List<Object> ids = **new** ArrayList<>();

ids.add(1l);

ids.add(2l);

// 设置条件

example.createCriteria().andIn("id", ids);

// 设置修改的数据

User user = **new** User();

user.setName("刘备");

**this**.newUserMapper.updateByExampleSelective(user, example);

}

@Test

**public** **void** testUpdateByExample() {

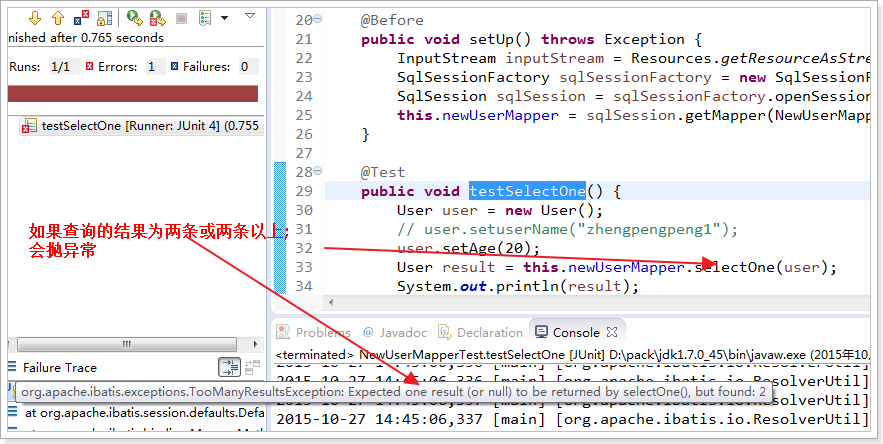
*fail*("Not yet implemented");

}

}

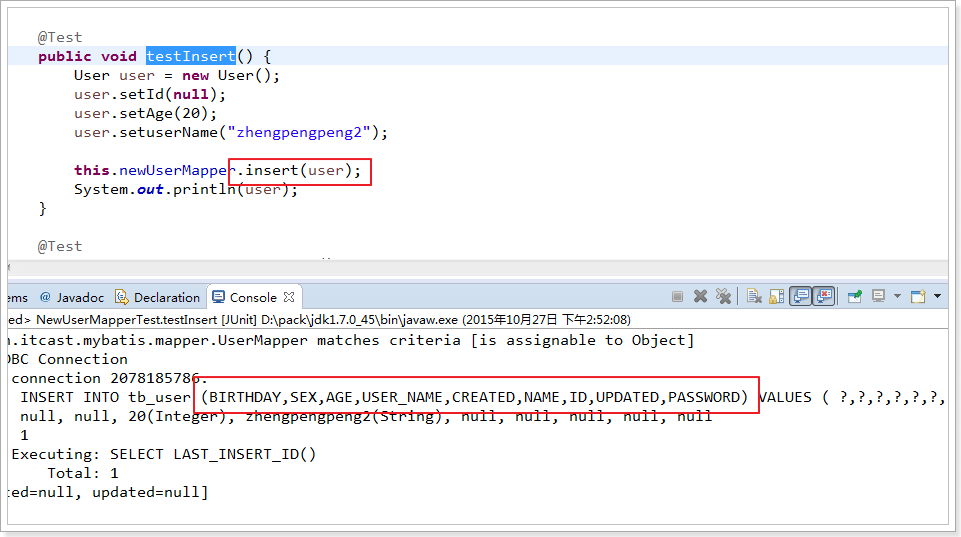
## 小结

### selectone

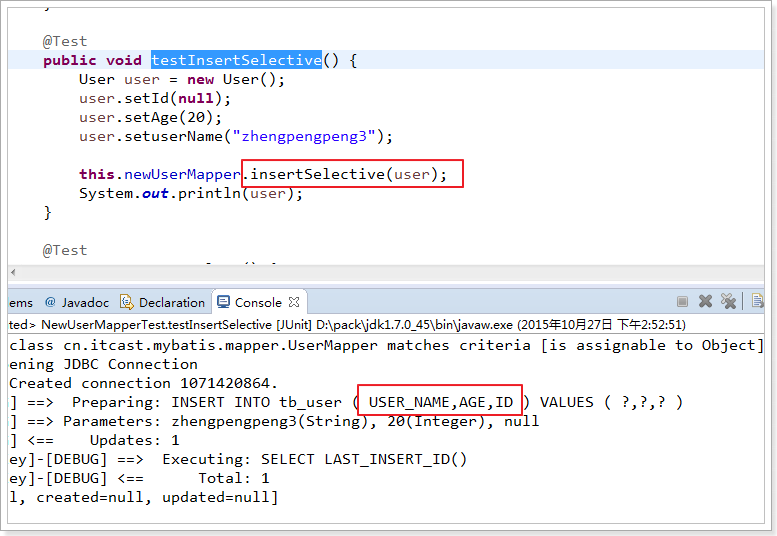


### insert和insertSelective的不同

insert会把null也保存

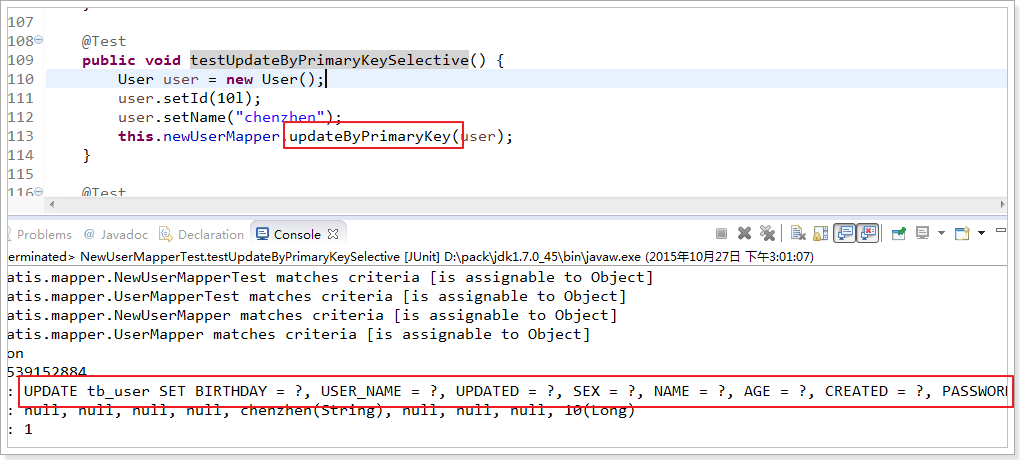


insertSelective忽略null

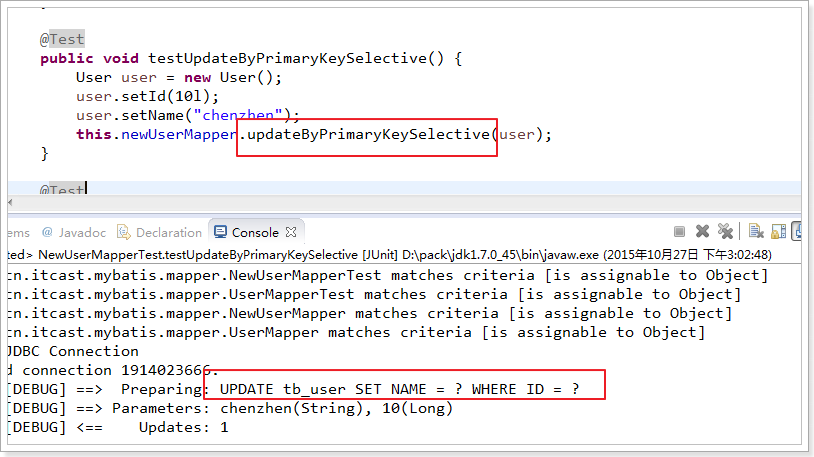


### updateByPrimaryKey和updateByPrimaryKeySelective不同

updateByPrimaryKey会把null也进行更新



updateByPrimaryKeySelective会忽略null



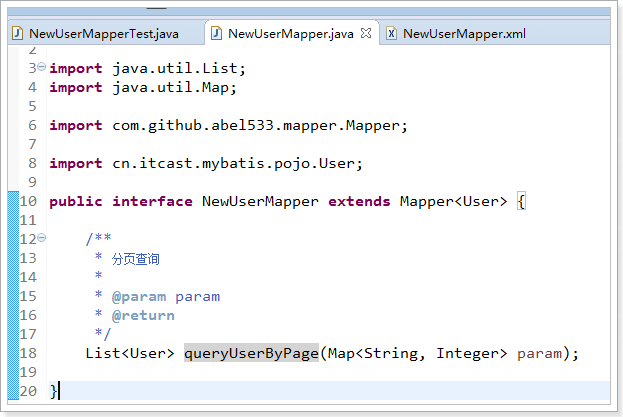
通用mapper和逆向工程一样，只能进行单表查询

多表查询怎么办？可以编写Mapper.xml，和以前的做法一样

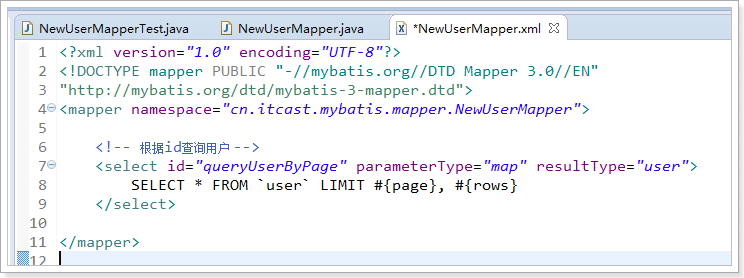
# 分页插件

## 原来的分页实现

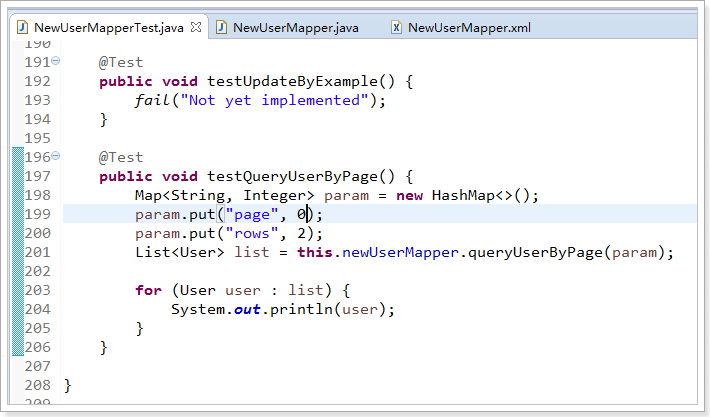
### Mapper增加分页查询方法



### Mapper.xml编写sql

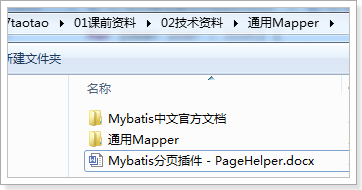


### 测试用例



## 使用分页插件

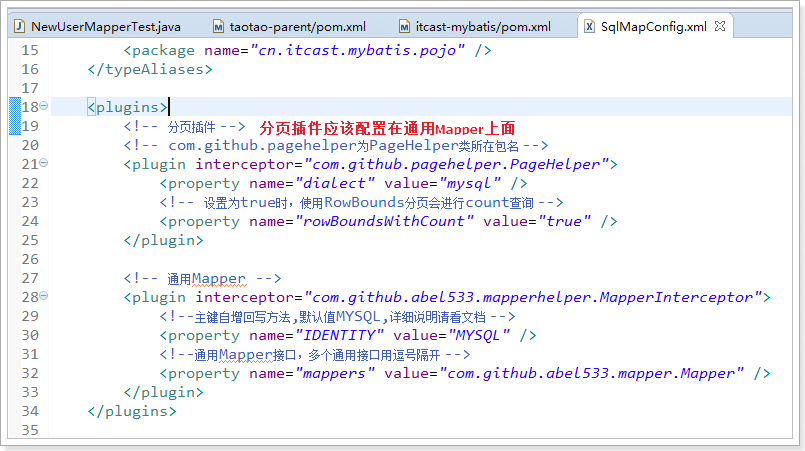
文档：



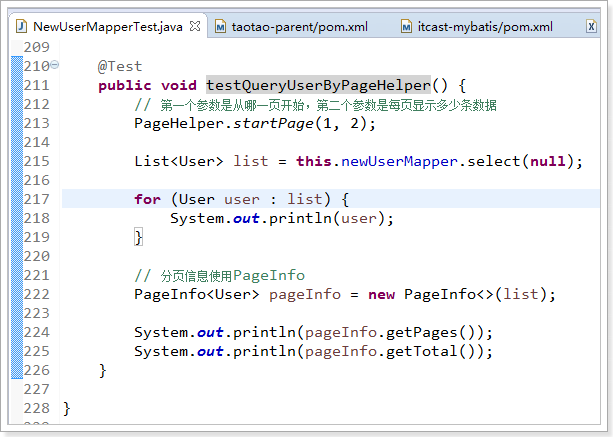
### 加入依赖



### 在MyBatis配置文件中集成



### 测试用例



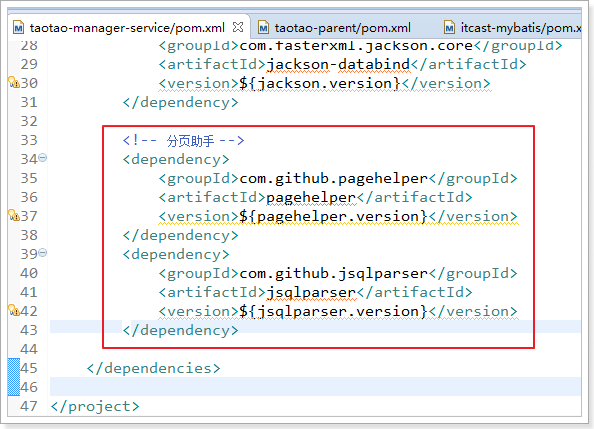
# 集成通用Mapper和分页插件

## 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-manager-pojo



分页逻辑是在service业务层进行，所以分页插件依赖加到taotao-manager-service



## 修改配置文件

修改mybatis配置文件SqlMapConfig.xml，加入分页插件和通用Mapper

<?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>

<settings>

<!-- 开启驼峰自动映射 -->

<setting name=*"mapUnderscoreToCamelCase"* value=*"true"* />

</settings>

<plugins>

<!-- 分页插件 -->

<!-- com.github.pagehelper为PageHelper类所在包名 -->

<plugin interceptor=*"com.github.pagehelper.PageHelper"*>

<property name=*"dialect"* value=*"mysql"* />

<!-- 设置为true时，使用RowBounds分页会进行count查询 -->

<property name=*"rowBoundsWithCount"* value=*"true"* />

</plugin>

<!-- 通用Mapper -->

<plugin interceptor=*"com.github.abel533.mapperhelper.MapperInterceptor"*>

<!--主键自增回写方法,默认值MYSQL,详细说明请看文档 -->

<property name=*"IDENTITY"* value=*"MYSQL"* />

<!--通用Mapper接口，多个通用接口用逗号隔开 -->

<property name=*"mappers"* value=*"com.github.abel533.mapper.Mapper"* />

</plugin>

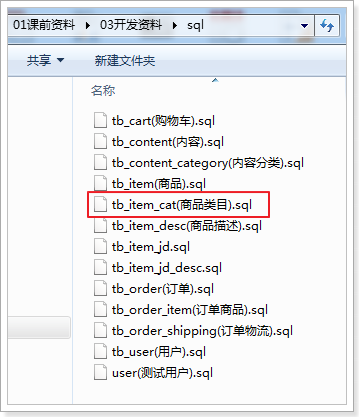
</plugins>

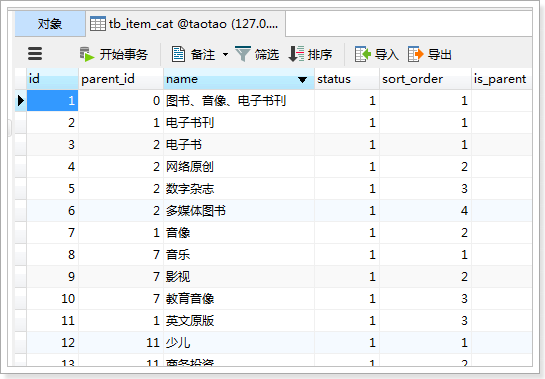
</configuration>

## 实现分页查询商品类目

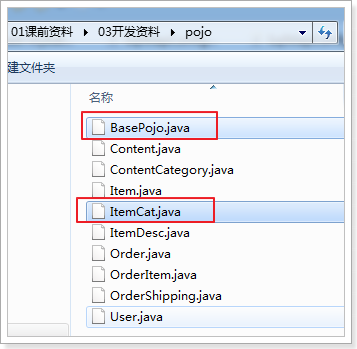
测试集成是否成功，实现浏览器地址栏输入url，分页查询商品类目数据

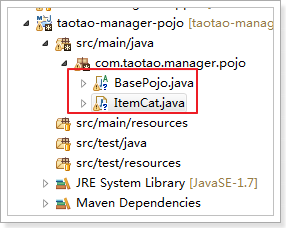
### 导入商品类目数据





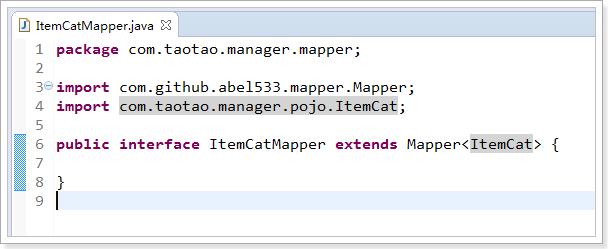
### 复制ItemCat和BasePojo





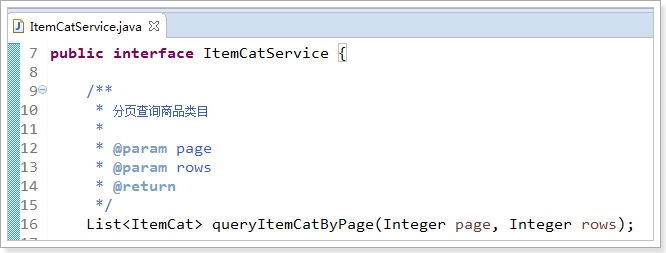
### 编写Mapper

在taotao-manager-mapper工程中编写ItemCatMapper接口

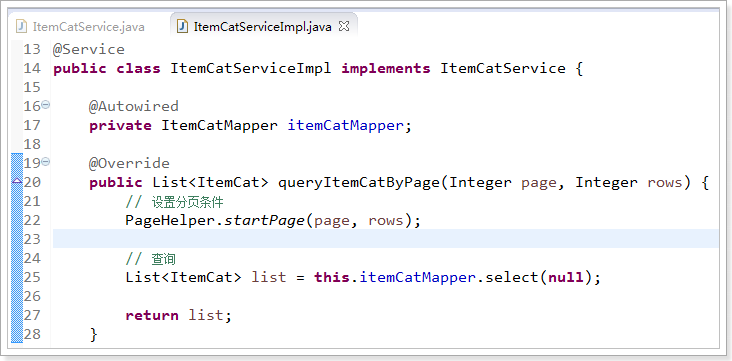


### 编写Service

在taotao-manager-interface编写接口



在taotao-manager-service编写实现类



### 编写controller

@Controller

@RequestMapping("item/cat")

**public** **class** ItemCatController {

@Autowired

**private** ItemCatService itemCatService;

// http://127.0.0.1:8081/rest/item/cat/query/1?rows=2

/\*\*

\* 分页查询商品类目

\* **@param** page

\* **@param** rows

\* **@return**

\*/

@RequestMapping("query/{page}")

@ResponseBody

**public** List<ItemCat> queryItemCatByPage(@PathVariable("page") Integer page, @RequestParam("rows") Integer rows) {

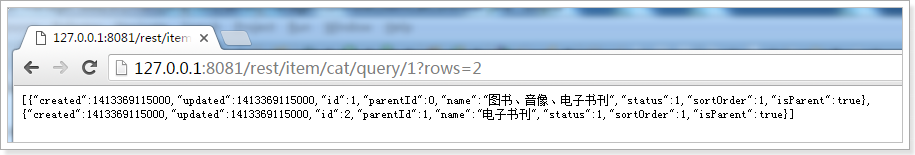
List<ItemCat> list = **this**.itemCatService.queryItemCatByPage(page, rows);

**return** list;

}

}

### 效果



# 使用svn管理代码

svn连接地址 https://192.168.84.16/svn/0729

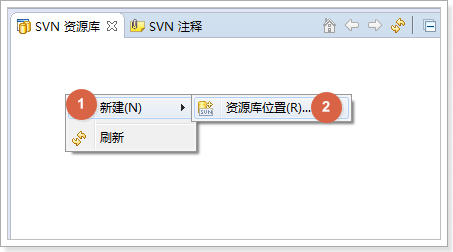
注意协议是https

svn账号：student

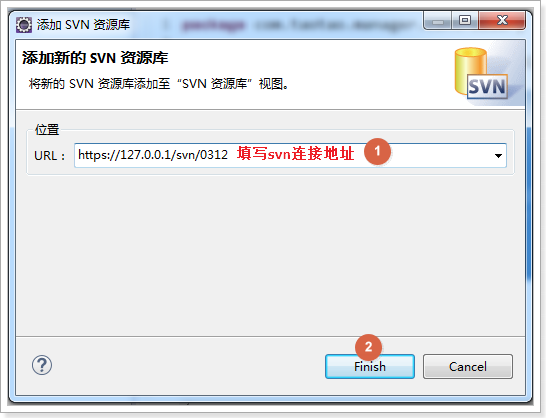
svn密码：123

## eclipse连接svn

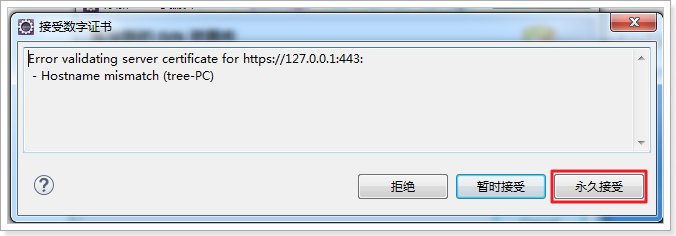
进入svn资源研究库视图，右键新建资源库位置



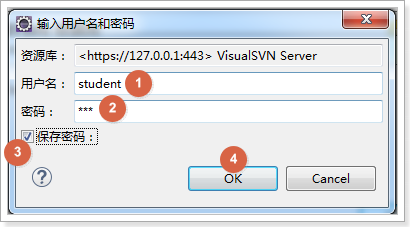
填写svn连接地址（注意填写实际svn服务器的地址)



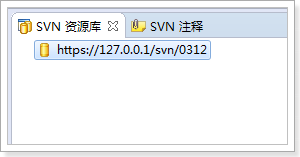
接受数字证书



输入账号密码



效果

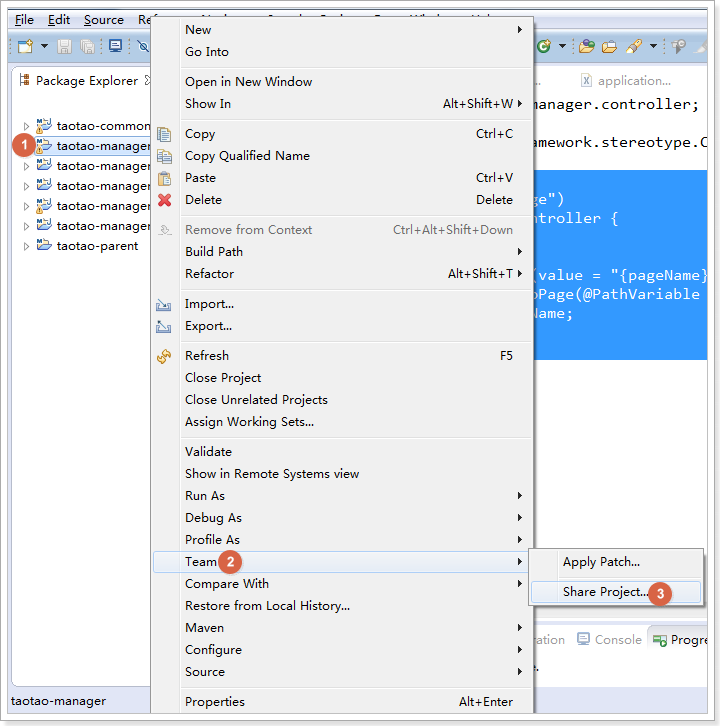


## 检入代码到svn

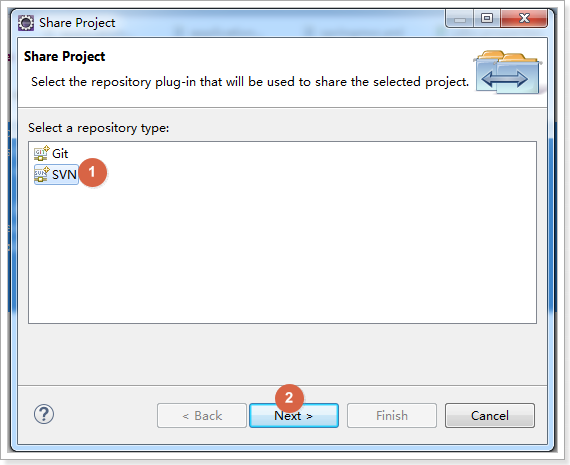
检入代码到svn，需要有svn的写入权限

1. 在需要检入的工程上点击右键，选择Share Project

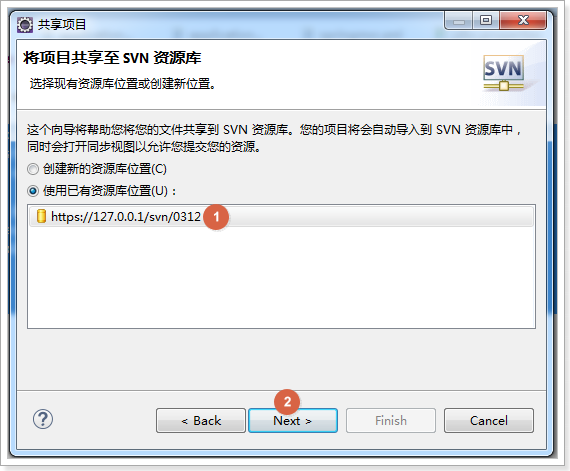
注意：聚合工程只检入父工程即可



1. 选中svn

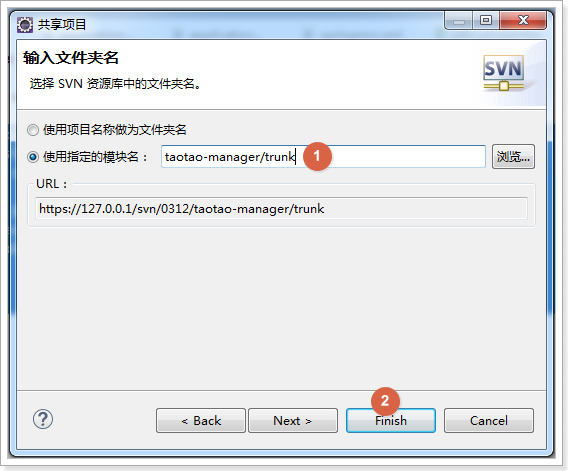


3. 选择对应的svn资源库



1. 指定模块名

模块名为： ${工程名}/trunk

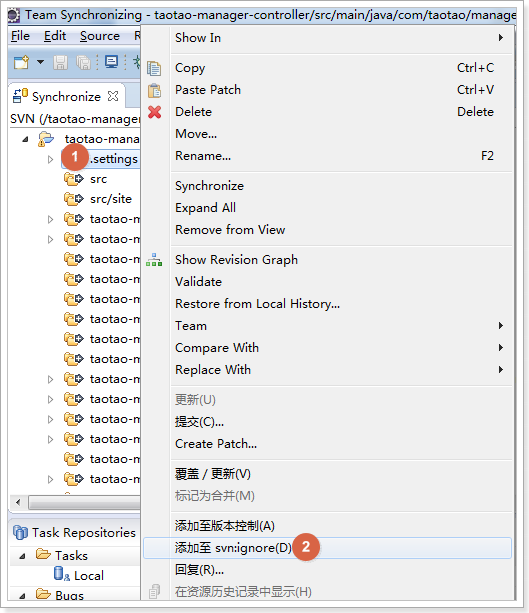


1. 在Synchronize视图设置忽略文件

检入到svn的只有src文件夹和pom.xml，其他的都忽略掉

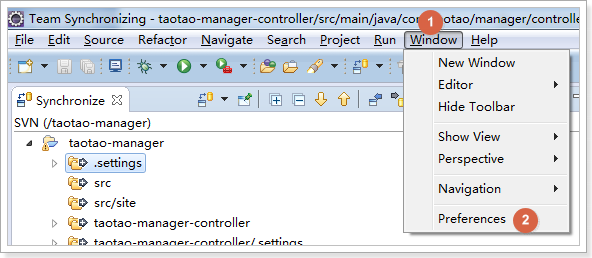
方法一：直接在需要忽略的文件点击右键进行设置

*只对当前文件生效*

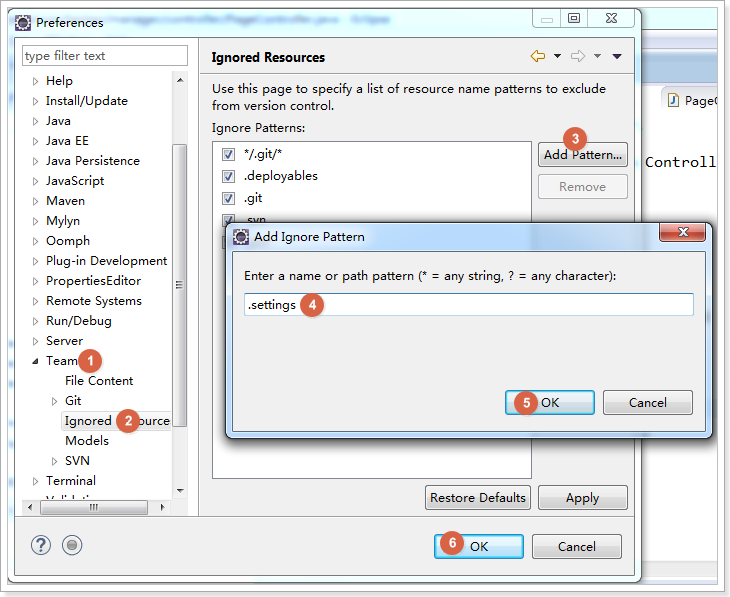


方法二：设置忽略规则

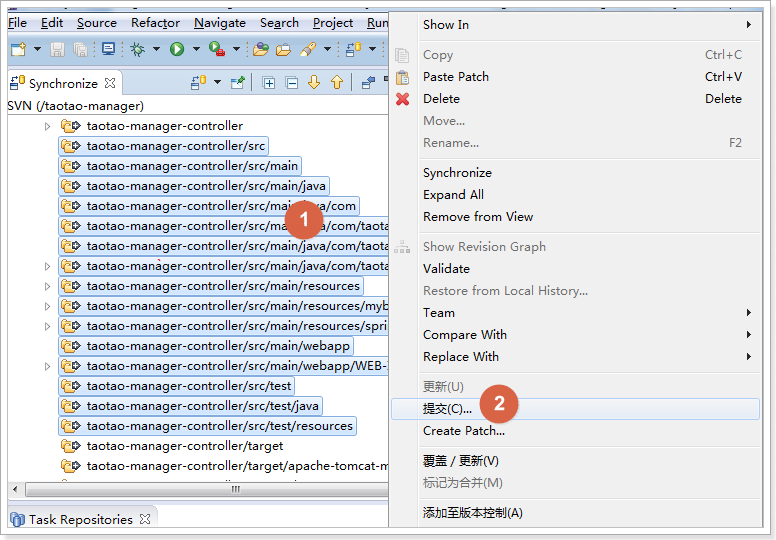
*对所有文件都生效*

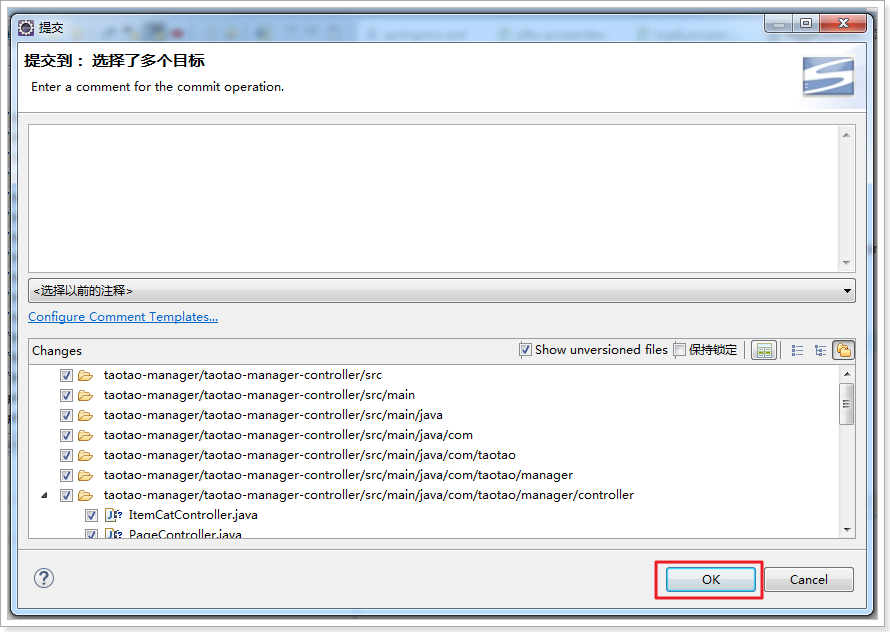


输入设置规则



1. 把需要检入的代码进行提交

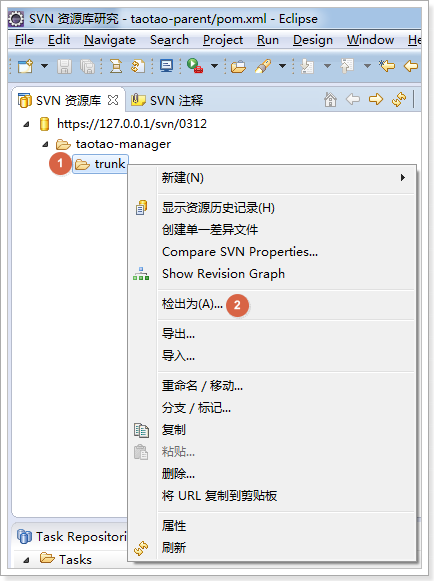




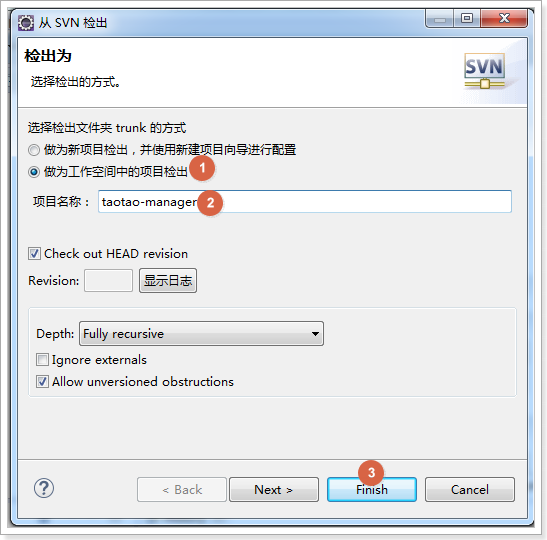
## 检出代码到本地

1. 检出代码

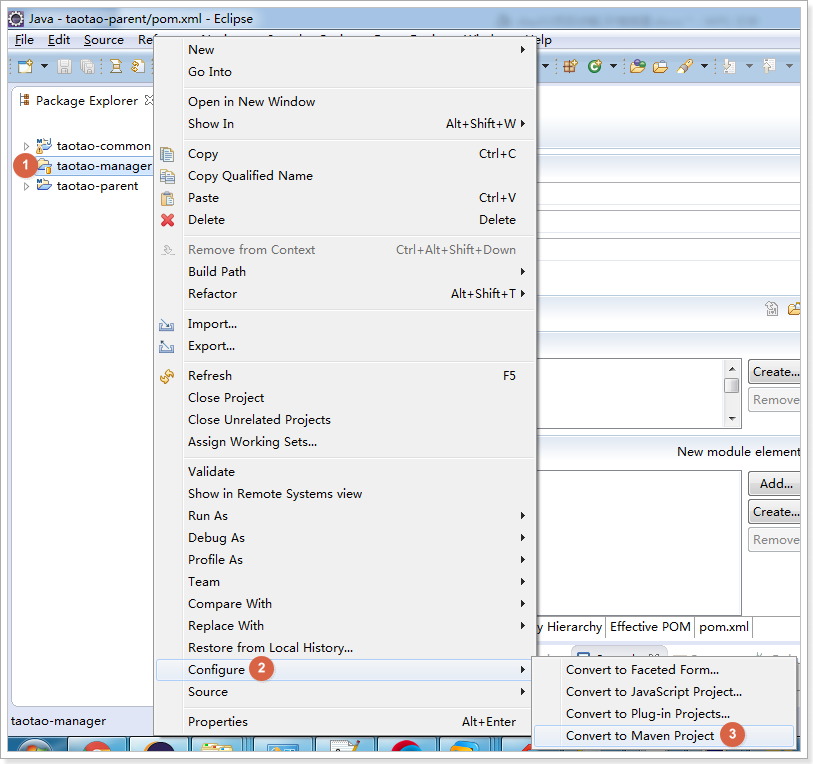
在trunk上点击右键



项目名称和需要检出的工程名一致

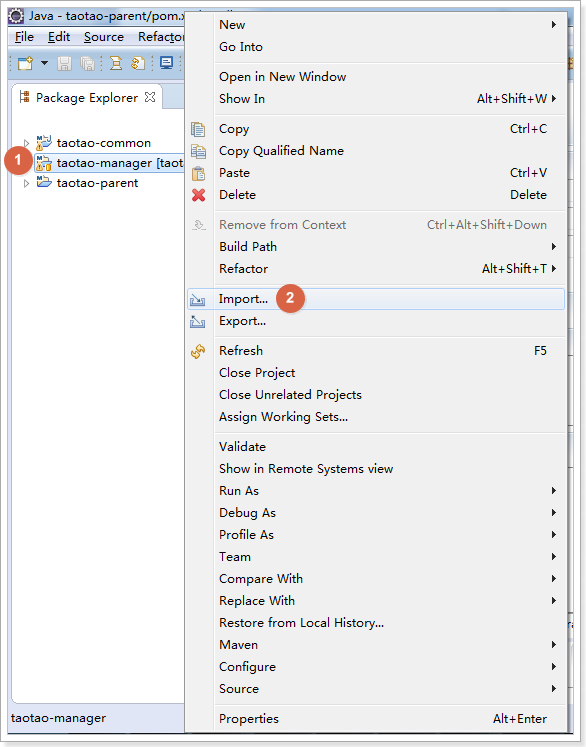


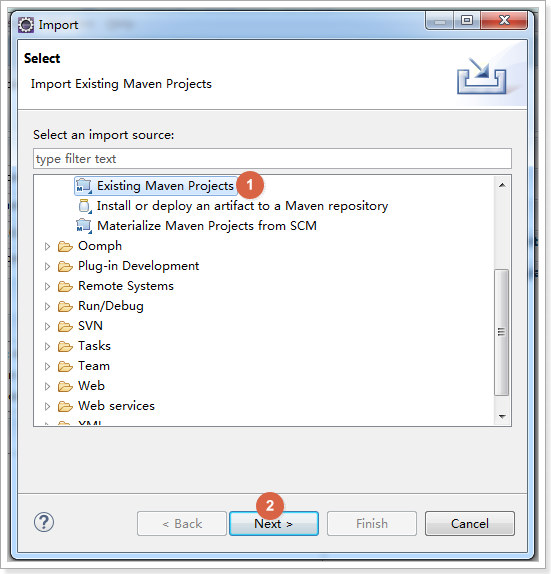
1. 将检出的工程转换为Maven工程

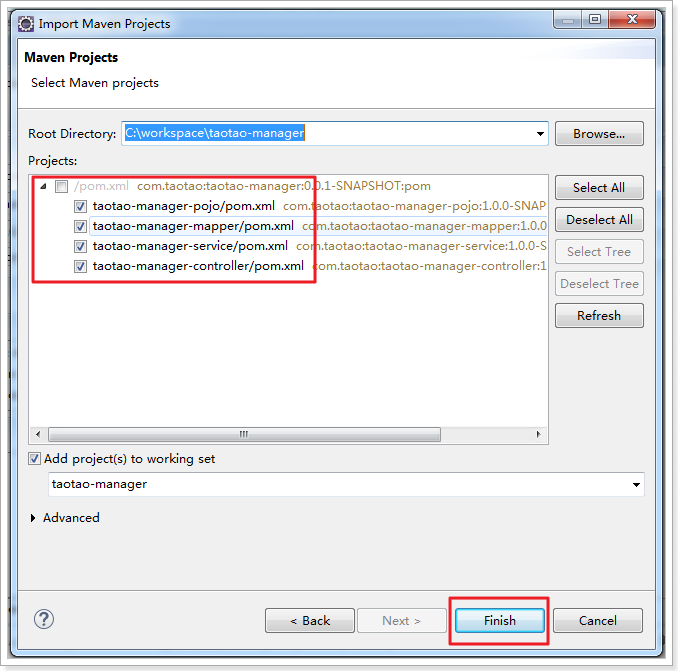


1. 导出子工程

只有聚合工程需要导出子工程，一般工程到第二步就检出成功了







导入成功

