# 京淘项目第三天

## 完成数据库CRUD操作

### 业务需求

在controller中添加CRUD方法完成用户数据的操作.在测试类中注入Controller对象

在测试类中 新建user对象 利用controller方法完成入库/修改/删除操作

例子1:新增用户

User user = new User();

User.setId(XXX).setName(XXX)……..

userController.insertUser(user);

例子2:修改操作

User user = new User();

User.age = 18岁的 名称改为 苏妲己CXXXX\

userController.updateUser(user);

## 持久层框架发展

### 发展史

1. JDBC实现数据库操作
2. 数据库连接池
3. Hibernate 完全实现了自动化ORM(对象关系映射)
4. 使用对象完成数据库CRUD sql不需要编辑
5. 查询的结果集 自动的封装为对象 hql语句

例子: userDao.insert(user) hibernate自动将对象转化sql并且执行.

4.Mybatis:

Mybatis是一种半自动的ORM

1. sql语句用户自己编辑
2. 结果集自动映射为对象.

需求:能否将单表CRUD操作也实现自动化配置.用户操作对象能否自动的生成sql语句.

## 映射框架的实现原理

### 实现原理

核心思想:利用对象自动生成sql

实现思路:

1. 对象与数据表一一映射 注解
2. 对象中的属性与表中的字段一一映射
3. 编辑公共的接口,其中完成CURD全部操作
4. 将公共的接口方法最终转化sql语句!!!

例子1:新增数据

Insert into 表名(字段名……) **values**(属性值…..)

例子2:删除数据

接口方法: deleteById(100)

**Delete from** **表名** **where** **主键 = 100**

## Mybatis-plus介绍

### 介绍

[MyBatis-Plus](https://github.com/baomidou/mybatis-plus)（简称 MP）是一个 [MyBatis](http://www.mybatis.org/mybatis-3/" \t "_blank) 的增强工具，在 MyBatis 的基础上只做增强不做改变，为简化开发、提高效率而生。

### 导入jar包

说明:引入mybatis-plus包之后将原有mybatis包注释,否则jar包冲突

<!--spring整合mybatis-plus -->

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>3.0.6</version>

</dependency>

### 修改YML配置

#mybatis-plush配置

mybatis-plus:

#配置别名包

type-aliases-package: com.jt.pojo

#配置mapper映射

mapper-locations: classpath:/mybatis/mappers/\*.xml

#开启驼峰映射

configuration:

map-underscore-to-camel-case: true

## Mybatis-plus入门案例

### 添加日志输出项

#添加日志输出

logging:

level:

com.jt.mapper: debug

### 编辑pojo对象

@Data //set/get/toString

@Accessors(chain = **true**) //链式加载

@NoArgsConstructor //无参构造

@AllArgsConstructor //有参构造

@TableName("user") //对象与数据表完成映射

**public** **class** User **implements** Serializable{

@TableId(type=IdType.***AUTO***) //主键自动

**private** Integer id;

//@TableField("name")

**private** String name;

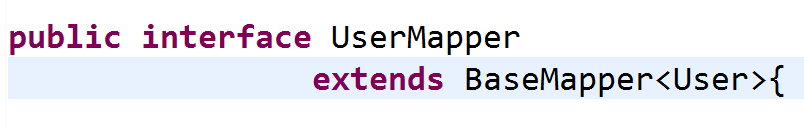
//@TableId("age")

**private** Integer age;

**private** String sex;

}

### 编辑Mapper接口



### 测试用户新增

@SpringBootTest

@RunWith(SpringRunner.**class**)

**public** **class** TestMybatisPlus {

@Autowired

**private** UserMapper userMapper;

@Test

**public** **void** insert() {

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

user.setName("mybatisplus");

user.setAge(19);

user.setSex("男");

**int** rows = userMapper.insert(user);

System.***out***.println("影响行数:"+rows);

}

}

### 查询测试

/\*\*查询用户列表信息

将对象中不为null的属性当做where条件

条件:age=18

关键字说明:

= eq,> gt,< lt,ge >=,le <=

\*/

@Test

**public** **void** testFind() {

//1.方式1.使用对象封装

//User user = new User();

//user.setAge(18);

//2.直接使用字段赋值

QueryWrapper<User> queryWrapper

= **new** QueryWrapper<User>();

//测试等于

//queryWrapper.eq("age",18);

//测试大于

queryWrapper.ge("age",18);

List<User> userList =

userMapper.selectList(queryWrapper);

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

}

### 数据修改测试

/\*\*

\* 3.数据更新 将年龄为18岁的用户改19岁

\* entity:需要修改数据的值

\* updateWrapper 条件构造器 充当where条件

\*/

@Test

**public** **void** testUpdate() {

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

user.setAge(19);

UpdateWrapper<User> updateWrapper =

**new** UpdateWrapper<>();

updateWrapper.eq("age", 18);

userMapper.update(user, updateWrapper);

}

### 删除空数据

//删除name字段为null的数据

@Test

**public** **void** deleteUser() {

QueryWrapper<User> queryWrapper = **new** QueryWrapper<User>();

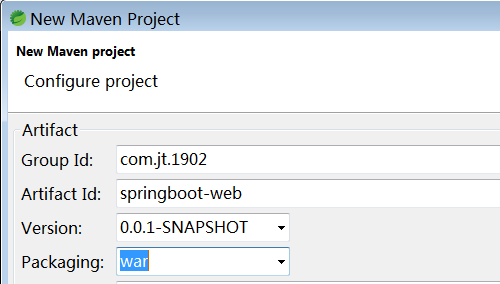
queryWrapper.isNull("name");

userMapper.delete(queryWrapper);

}

## SpringBoot整合JSP

### 构建项目



### 导入jar包

<!--springBoot整合JSP添加依赖 -->

<!--servlet依赖 -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

</dependency>

<!--jstl依赖 -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<!--使jsp页面生效 -->

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

### 编辑YML文件

server:

port: 8090

servlet:

context-path: /

#引入数据源配置

spring:

datasource:

#引入druid数据源

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3306/jtdb?serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true

username: root

password: root

#配置mvc页面

**mvc:**

**view:**

**prefix: /WEB-INF/**

**suffix: .jsp**

#mybatis-plush配置

mybatis-plus:

#配置别名包

type-aliases-package: com.jt.pojo

#配置mapper映射

mapper-locations: classpath:/mybatis/mappers/\*.xml

#开启驼峰映射

configuration:

map-underscore-to-camel-case: true

#添加日志输出

logging:

level:

com.jt.mapper: debug

### 编辑Controller

@Controller //需要跳转页面

**public** **class** UserController {

@Autowired

**private** UserService userService;

//用户通过localhost:8090/findAll获取用户数据

@RequestMapping("/findAll")

**public** String findAll(Model model) {

List<User> userList = userService.findAll();

//model底层将数据保存到了request域中

model.addAttribute("userList", userList);

**return** "userList"; //逻辑名称

}

}

### 页面JSP编辑

<table border=*"1px"* width=*"65%"* align=*"center"*>

<tr>

<td colspan=*"6"* align=*"center"*><h3>学生信息</h3></td>

</tr>

<tr>

<th>编号</th>

<th>姓名</th>

<th>年龄</th>

<th>性别</th>

<th></th>

</tr>

<!--el表达式取值从域获取 page request session context -->

<c:forEach items=*"${userList}"* var=*"u"*>

<tr>

<th>${u.id}</th>

<th>${u.name}</th>

<th>${u.age}</th>

<th>${u.sex}</th>

</tr>

</c:forEach>

</table>

### 页面效果展现



## 分布式

### 传统项目中存在问题

1. 传统项目将全部的功能模块都写到一个项目中,如果项目中模块报错,则可能影响整个项目运行.
2. 如果一个项目中有多个功能模块,则在开发时耦合性高,开发效率慢.

总结:传统项目架构设计上耦合性高.

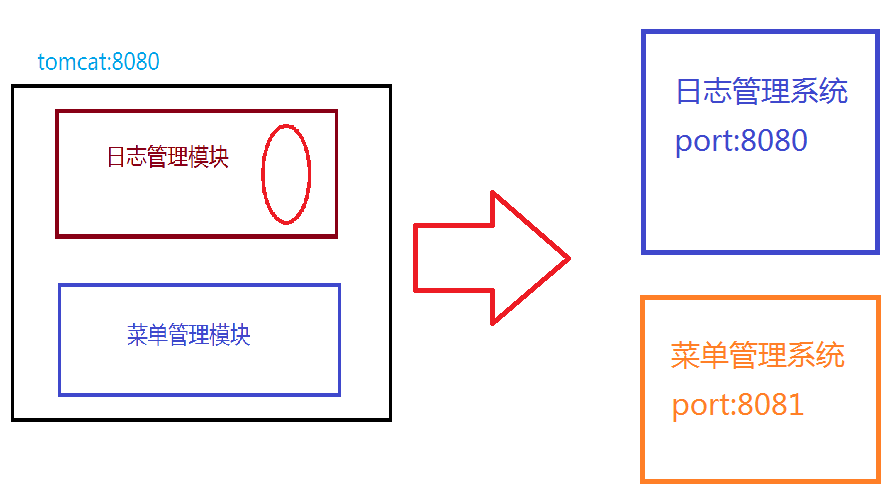
### 分布式说明

#### 分布式概念

将业务模块按照特定的规则进行**拆分**.分别部署到不同的服务器实现了架构的解耦.

#### 垂直拆分

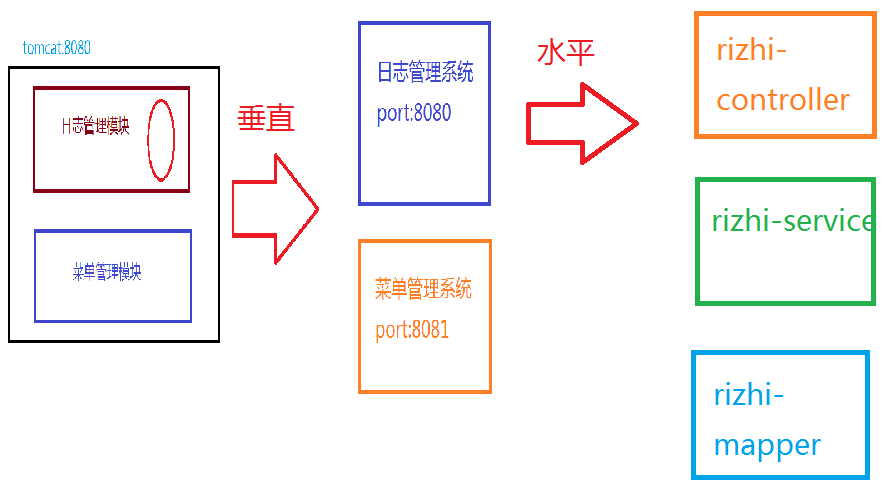
说明:按照项目的功能(**业务**)模块进行拆分.将不同的业务部署到不同的服务器中.



#### 水平拆分

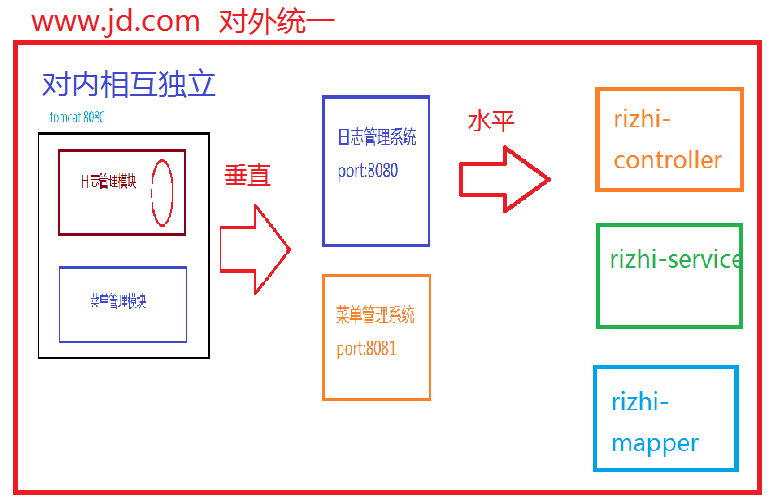
场景:一个service可能3000行代码由一个人完成controller-Service-mapper任务繁重.这时需要多个人开发!!

概念:按照调用的层级进行拆分.

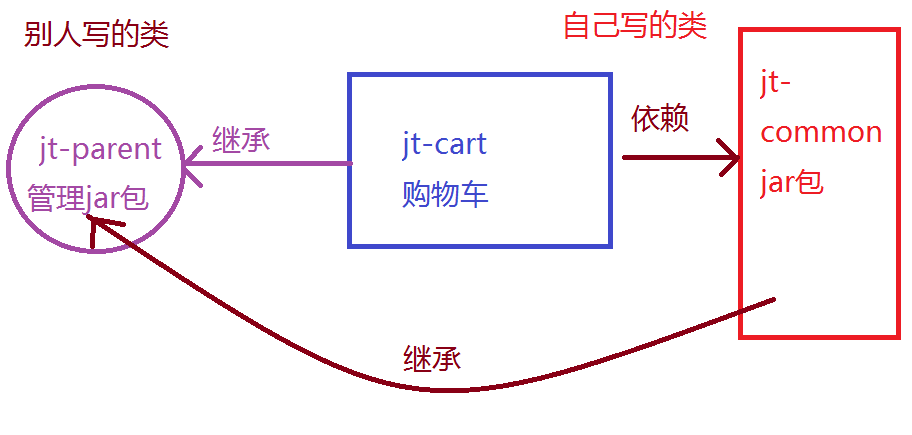


### 分布式项目总结

分布式项目:对外统一,对内独立.实现了架构的解耦



## 分布式项目引发问题



### Jar包如何管理

说明:利用单独的项目管理jar包,其他业务项目继承即可

### 工具类如何维护

说明:工具项目是架构师/高级程序员自己编辑java代码.这时类通过jar包的形式依赖到项目中.程序可以直接调用.

规定:继承只能继承jar包不能继承java类.

## 聚会项目构建

### 说明

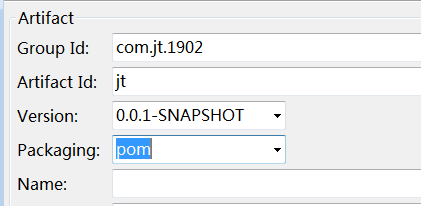
大型项目构建一般都会采用聚合项目.从整体看是一个项目,观察细节发现项目中又会分为不同的项目.

打包时使用POM类型.

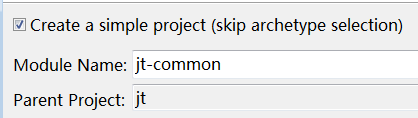
### 聚合项目构建

#### 创建项目

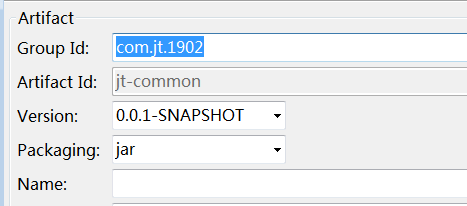
选择打包类型为pom



#### 创建工具类项目jt-common

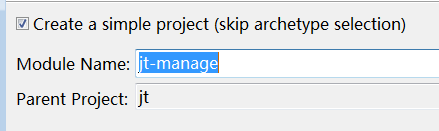
* 

选择项目类型

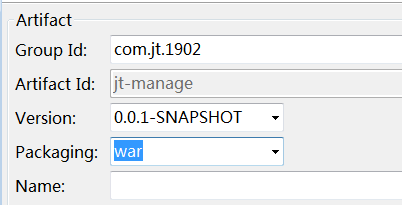


### 创建业务系统jt-manage

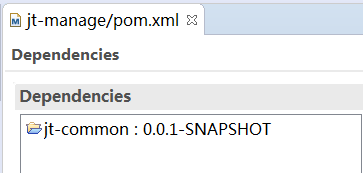
#### 创建项目



#### 选择war包



#### 添加依赖



### 导入项目文件

#### 编辑jt的pom.xml文件

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.1.3.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!--springBoot动态的引入springMVC全部的配置 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- 引入aop支持 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency>

<!--引入测试类 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<!--添加属性注入依赖 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

<!--支持热部署 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>springloaded</artifactId>

<version>1.2.8.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

</dependency>

<!--引入插件lombok 自动的set/get/构造方法插件 -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

<!--引入数据库驱动 -->

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.32</version>

<scope>runtime</scope>

</dependency>

<!--引入druid数据源 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.1.12</version>

</dependency>

<!--spring整合mybatis-plus -->

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>3.0.6</version>

</dependency>

<!--spring整合redis -->

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-redis</artifactId>

</dependency>

<!--springBoot整合JSP添加依赖 -->

<!--servlet依赖 注意与eureka整合时的问题 -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

</dependency>

<!--jstl依赖 -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<!--使jsp页面生效 -->

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

<!--添加httpClient jar包 -->

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpclient</artifactId>

</dependency>

<!--引入dubbo配置 -->

<!--<dependency>

<groupId>com.alibaba.boot</groupId>

<artifactId>dubbo-spring-boot-starter</artifactId>

<version>0.2.0</version>

</dependency>-->

<!--添加Quartz的支持 -->

<dependency>

<groupId>org.springframework.boot</groupId>

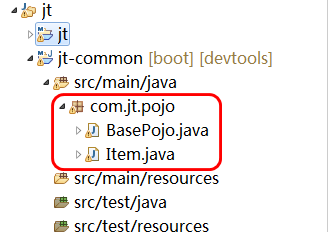
<artifactId>spring-boot-starter-quartz</artifactId>

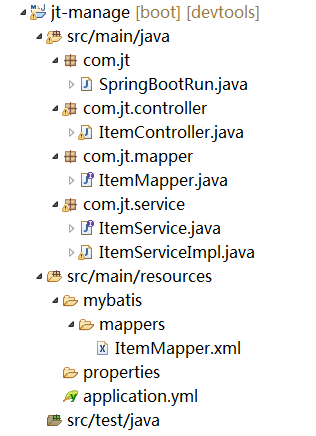
</dependency>

</dependencies>

#### 导入静态资源文件

说明:根据课前资料中3-jt-manage的模板代码导入





导入完成后,输入localhost:8091 检查页面是否正常运行.

