**武汉大学计算机学院**

**本科生课程设计报告**

**软件设计与体系结构**

专 业 名 称 ：计算机科学与技术

课 程 名 称 ：软件设计与体系结构

指 导 教 师 ：王翀

学 生 姓 名 ：陈昱文（2018302110077）

学 生 姓 名 ：廖宇阳（2018302110063）

学 生 姓 名 ：钟仁毅（2018302110057）

二○二○年十二月

# 项目简介

## 项目名称

体育赛事管理系统

## 项目总体需求描述

本系统旨在帮助体育赛事发起方对体育赛事中赛事、运动员、裁判、比赛结果等相关事项进行科学管理与统计。

运动员方面，功能包括运动员进行注册，对所有运动员总体信息进行储存管理，允许运动员报名符合条件的比赛并查看相关情况，运动员可以对比赛结果进行申诉等。

赛事方面，允许赛事主办方进行注册并提交相关信息，对所有赛事信息进行统一管理，赛事方可以对运动员、裁判员的报名进行审核并负责分组工作，且可以对运动员申诉进行处理和审查裁判员上传的得分。

裁判员方面，功能包括裁判员进行注册，对所有裁判员总体信息进行储存管理，允许裁判员报名符合条件的比赛并进行相应的分数填写工作且可以对运动员进行处罚。

## 项目具体实现功能

* 运动员注册登录系统并提交相关信息
* 对运动员身份数据进行存储管理
* 对赛事进行注册并提交相关信息
* 对赛事数据进行管理
* 运动员对赛事进行报名
* 对特定赛事的报名信息进行管理
* 完成特定比赛的分组工作
* 裁判员对比赛结果进行记录，并提交由裁判组审核
* 裁判注册登录系统并提交相关信息
* 对裁判员身份数据进行存储管理
* 运动员能够对比赛结果进行申诉，提交裁判组处理
* 记录并广播比赛结果

## 具体模块成员分析

### 运动员模块

#### 属性

* 姓名
* 年龄
* 性别
* 身高
* 体重
* 院系
* 学号
* 身体状况
* 禁赛情况

#### 方法

* 报名
* 退赛
* 更改个人信息
* 查看赛事部分信息（分组、个人得分）
* 申诉

### 赛事模块

#### 属性

* 赛事名称
* 赛事负责人、主管单位
* 赛事等级
* 赛事时间
* 赛事地点
* 参赛人数
* 淘汰赛制
* 参赛人员
* 裁判组

#### 方法

* 更改赛事信息
* 同意\拒绝选手报名
* 进行抽签分组
* 审核比赛结果
* 处理申诉

### 裁判员模块

#### 属性

* 姓名
* 年龄
* 所属单位
* 工号
* 职称

#### 方法

* 注册赛事
* 更改个人信息
* 填写得分
* 进行判罚
* 报名比赛

### 运动员\裁判员\赛事列表（合三为一进行阐述）

#### 属性

* 运动员\赛事\裁判员实体

#### 方法

* 增加运动员\赛事\裁判员
* 删除运动员\赛事\裁判员
* 查询运动员\赛事\裁判员
* 更改运动员\赛事\裁判员

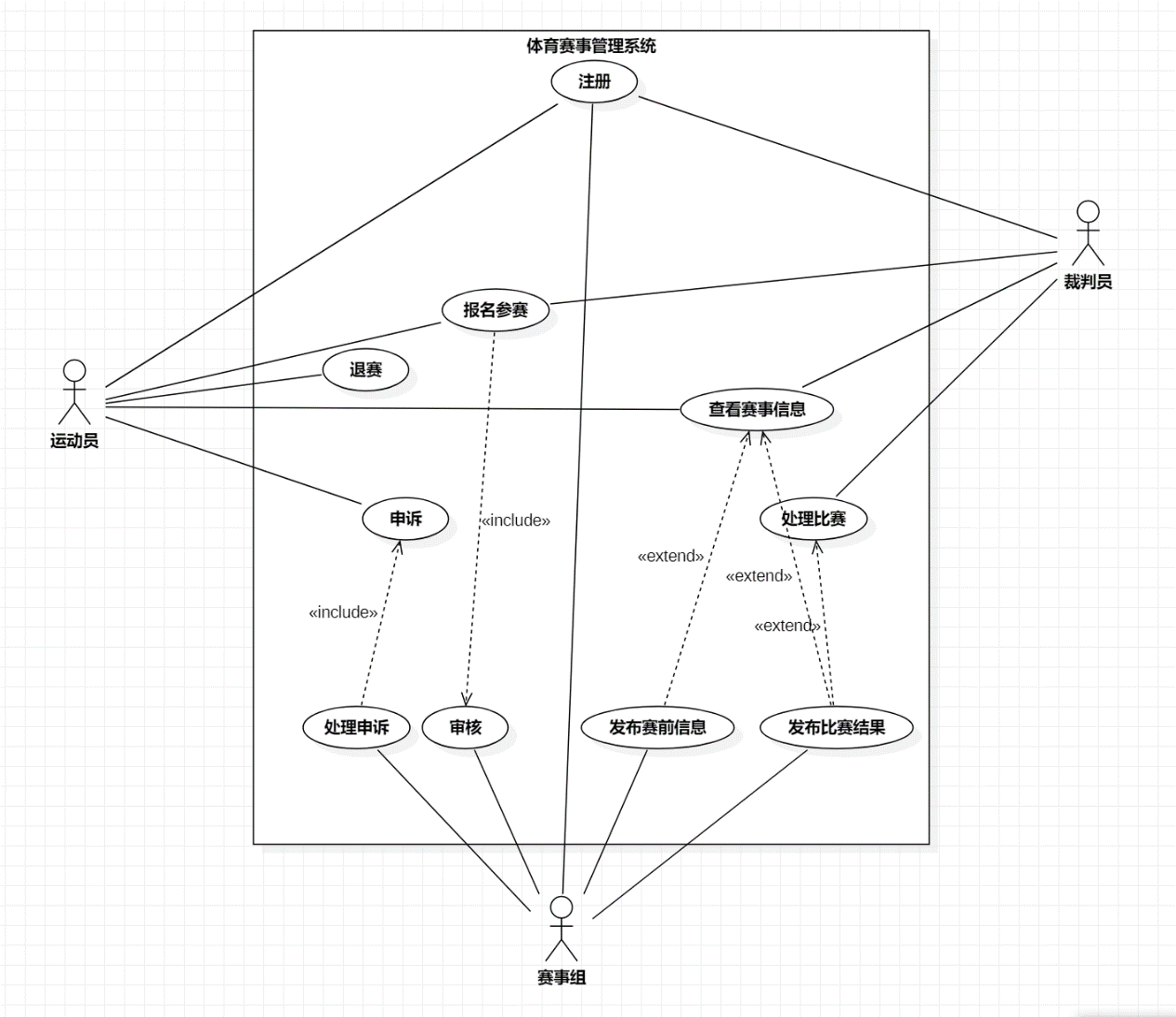
# 实验一

## 实验要求

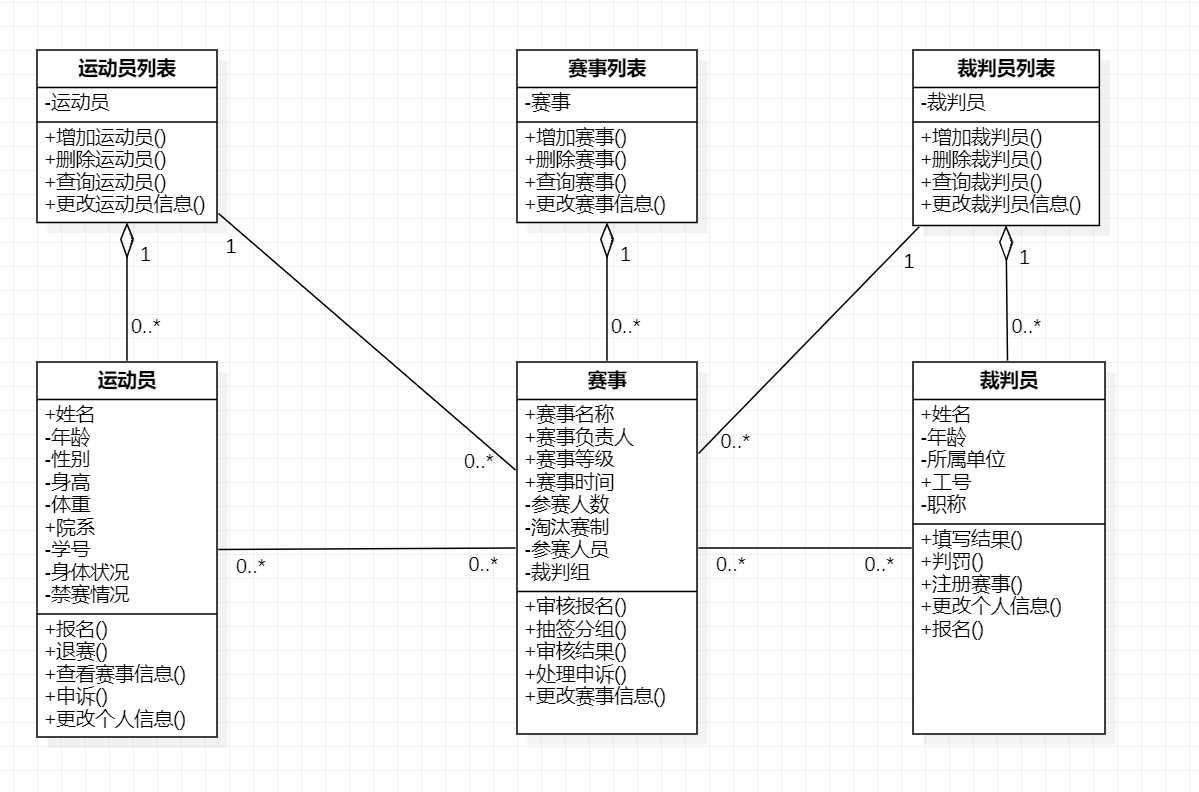
基于UML的软件分析与设计模型建模实验（含用例图、类图和时序图），只需考虑软件中的一个模块或场景，提交报告。

## 实验结果

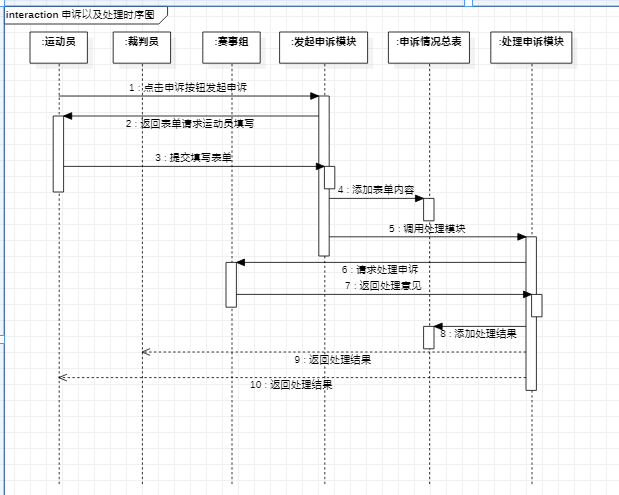
### 用例图



### 类图



### 时序图（申诉部分）



# 实验二

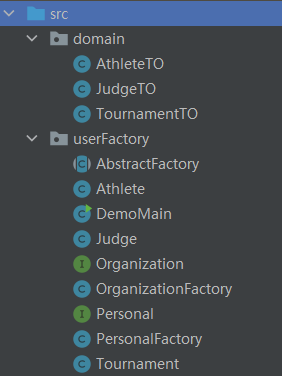
## 实验要求

设计模式实验，实现上述模块中的一部分功能，应用至少2种设计模式。

## 实验结果

### 设计模式一：抽象工厂模式

* 总体文件结构：



* 文件结构描述：

Domain包中：

TO层用于将部分信息传递给上层或者其他应用程序；

UserFactory包中：

DemoMain为程序主窗口，包含启动程序的Main函数；

AbstractFactory为抽象工厂类；

PersonalFactory和OrganizationFactory为两个实体工厂类；

PersonalFactory负责实例化实现Personal接口的类对象；

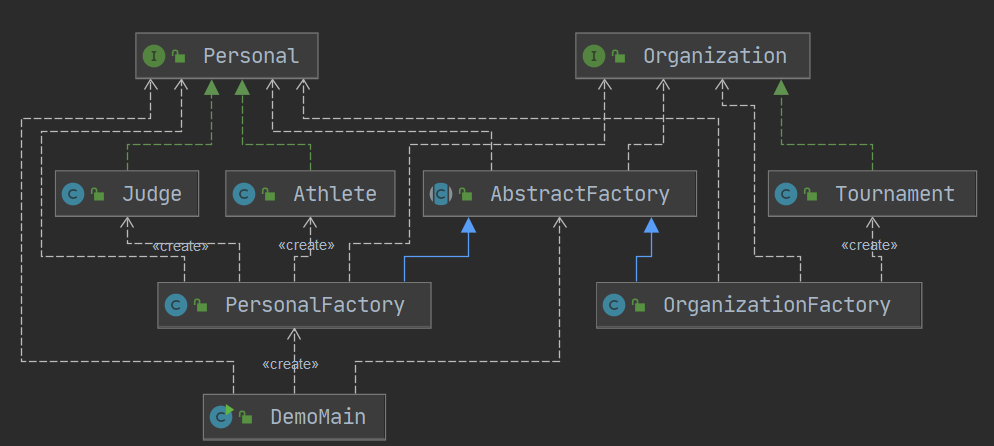
OrganizationFactory负责实例化实现Organization接口的类对象；

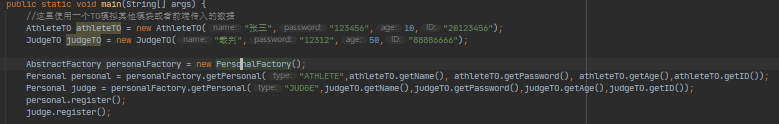
Personal和Organization分别是两个抽象的实体接口；

其中Personal的具体实现是Athlete和Judge；

Organization的具体实现是Tournament类；

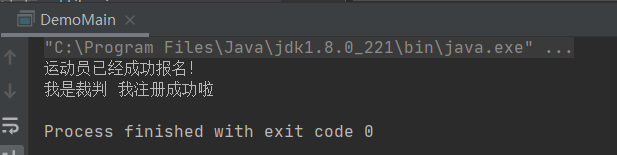
具体的依赖关系如下面总体依赖关系图所示。

* 总体依赖关系：
* 整体调用：



可以看到，在实例化工厂阶段。我们可以将工厂作为一个Singleton来进行实现，且所有的工厂对外展示都为AbstractFactory，对外部工厂的具体信息是隐藏的。再者，抽象工厂模式将产品对象的创建延迟到了他的ConcreteFactory子类中。

* 运行结果：



运行这段代码，可以看到personalFactory工厂成功的对运动员和裁判进行了注册，且对外界而言athlete和judge都被显示为personal类型的对象。

### 设计模式二：组合模式

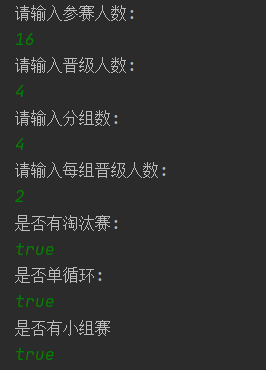
###### 概念

组合模式（Composite Pattern），又叫部分整体模式，是用于把一组相似的对象当作一个单一的对象。组合模式依据树形结构来组合对象，用来表示部分以及整体层次。这种类型的设计模式属于结构型模式，它创建了对象组的树形结构。这种模式创建了一个包含自己对象组的类。该类提供了修改相同对象组的方式。组合模式的意图是将对象组合成树形结构以表示"部分-整体"的层次结构，使得用户对单个对象和组合对象的使用具有一致性。它在处理树型结构的问题中，模糊了简单元素和复杂元素的概念，客户程序可以像处理简单元素一样来处理复杂元素，从而使得客户程序与复杂元素的内部结构解耦。组合模式的优点是高层模块调用简单，并且节点可自由增加。

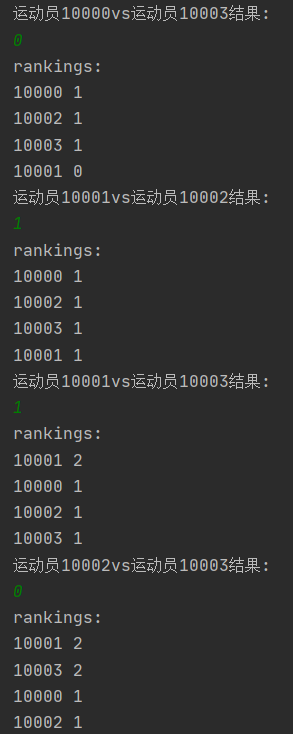
###### 使用原因

实现体育赛事管理平台的一大难点在于赛制的设计。不同类型的比赛项目适用于不同的赛制，如足球联赛一般采取积分赛制，篮球联赛则采取小组赛加淘汰赛的赛制。为了方便起见，我将主流的赛制分为两大类：积分赛制和淘汰赛制。积分赛制即一组队伍以单循环或双循环形式轮流比赛，取积分最高的几支球队获胜或晋级；淘汰赛制即每轮队伍两两捉对厮杀，败者淘汰胜者晋级，直到角逐出冠军。值得一提的是，目前主流的赛制还有一种是积分赛与淘汰赛制结合的情况。各支队伍先分组进行积分赛，积分靠前的队伍晋级，再通过淘汰赛决出最后的冠军。这种带有组合性质的赛制给了我使用组合模式的想法：组合赛制可作为容器构件，而积分赛和淘汰赛则作为叶子构件。这样可以有效地包装赛制，使主办方输入几个参数即可完成不同类型的赛事的建立，而不用考虑赛制的内部实现。

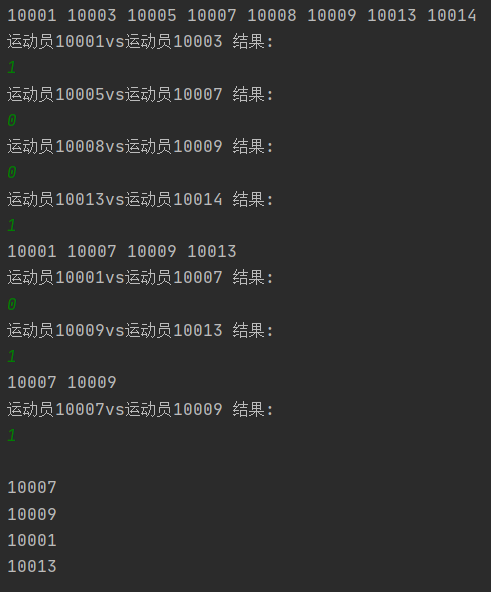
###### 实现结果



用户输入以上参数。



用户依次输入小组赛结果，实时显示出排名。

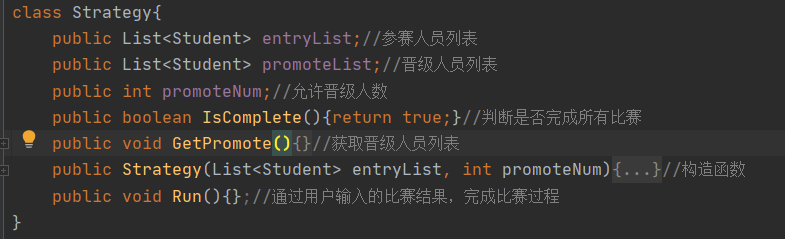


小组赛结果输入完毕，显示晋级名单。再输入淘汰赛结果，最后显示前四名。

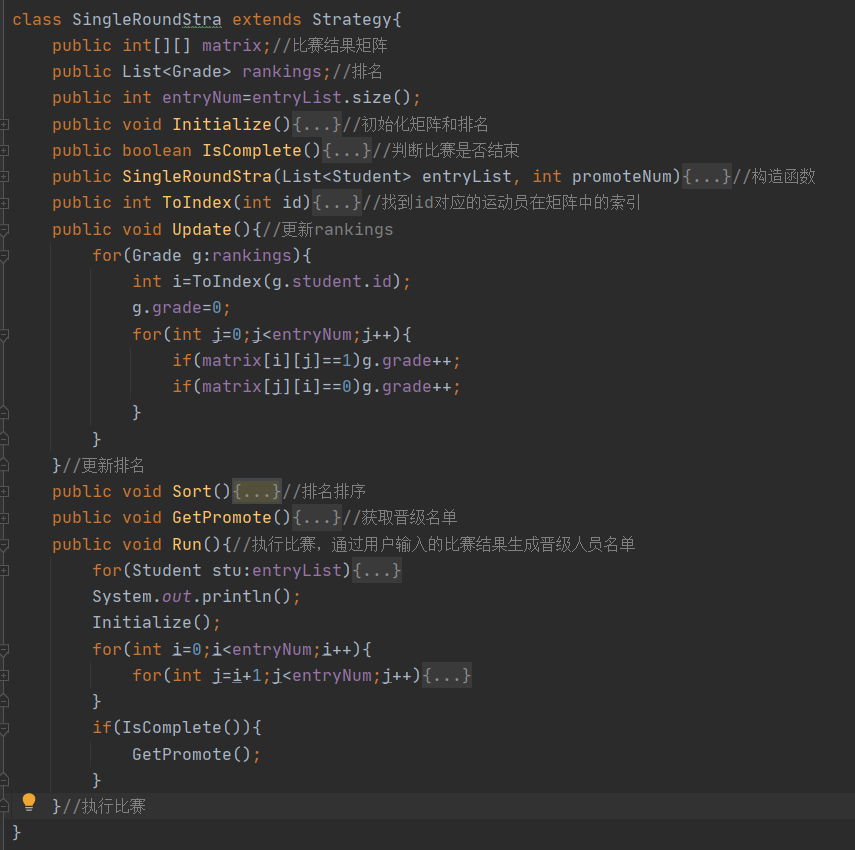
###### 文件结构

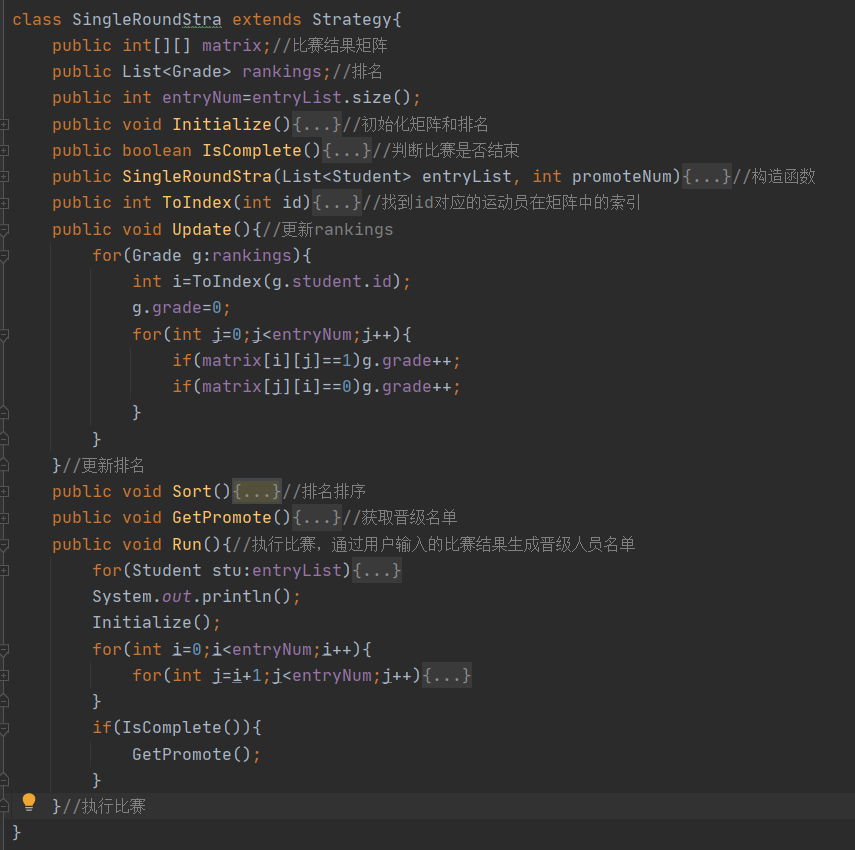
###### 代码分析

Competition类作为抽象构建类，是所有具体构件类的父类。

****

SingleRoundStra类是一种叶子构件，通过Run函数能执行一个小组的单循环积分赛，并生成晋级名单。

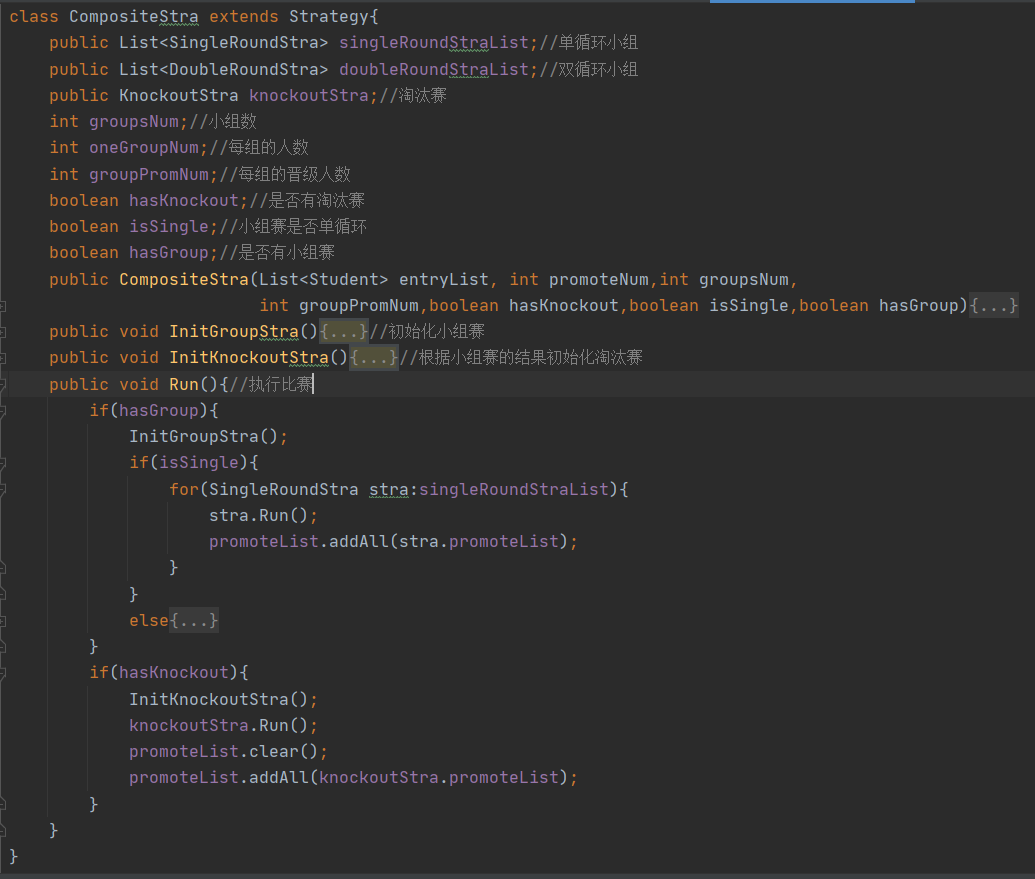
****

****

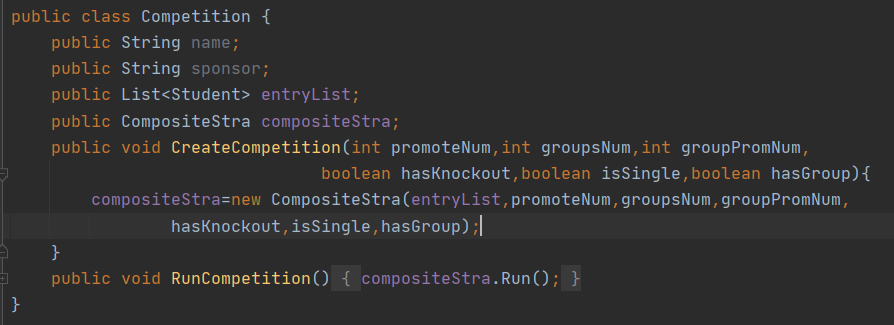
KnockoutStra是另一个叶子构件，通过Run函数执行一个淘汰赛流程，并生成获奖名单。

****

CompositeStra是一个容器构件，有两个list存放可能的叶子构件，还有一个淘汰赛构件，有一系列参数来决定使用哪些叶子构件组成比赛

****

Competition类用来生成赛制和运行比赛。



###### 不足之处

虽然总体实现了组合模式，但细节上仍与经典的组合模式有出入，最大的问题是只能通过容器构件来产生和组织好一个赛制，而不能直接使用叶子构件。此外，由于不同赛制区别较大，容器构件中仍存在不少条件语句，不简洁。

###### 总结

通过学习组合模式并尝试将其运用到项目中，我意识到对一个项目进行有计划的组织和布局的重要性。以往都是想到哪儿写到哪儿，缺乏整体感和大局观，但现在虽然写代码钱需要考虑的东西多了，但一旦设计好，再实现会容易许多，也少了很多推翻重写的情况。

# 实验三

## 实验要求

Web服务开发实验，针对上述模块中的任意一个功能，用任意的编程语言实现任意一个SOAP或REST API并能用客户端进行调用。

## 实验结果

### 技术选型

* 前端：Vue
* 后端：Spring Boot+Mybatis框架
* 另：Nginx进行跨域处理

### 实现功能

经过综合考虑，决定实现运动员注册这一个功能。

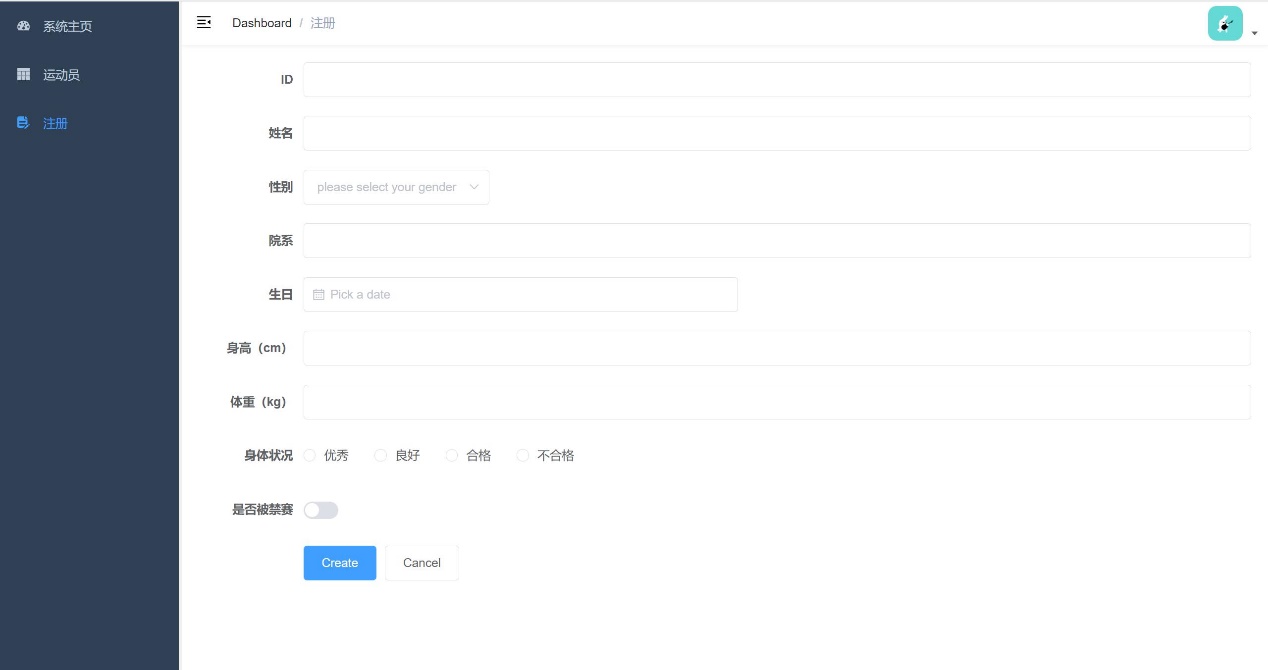
### 数据库设计



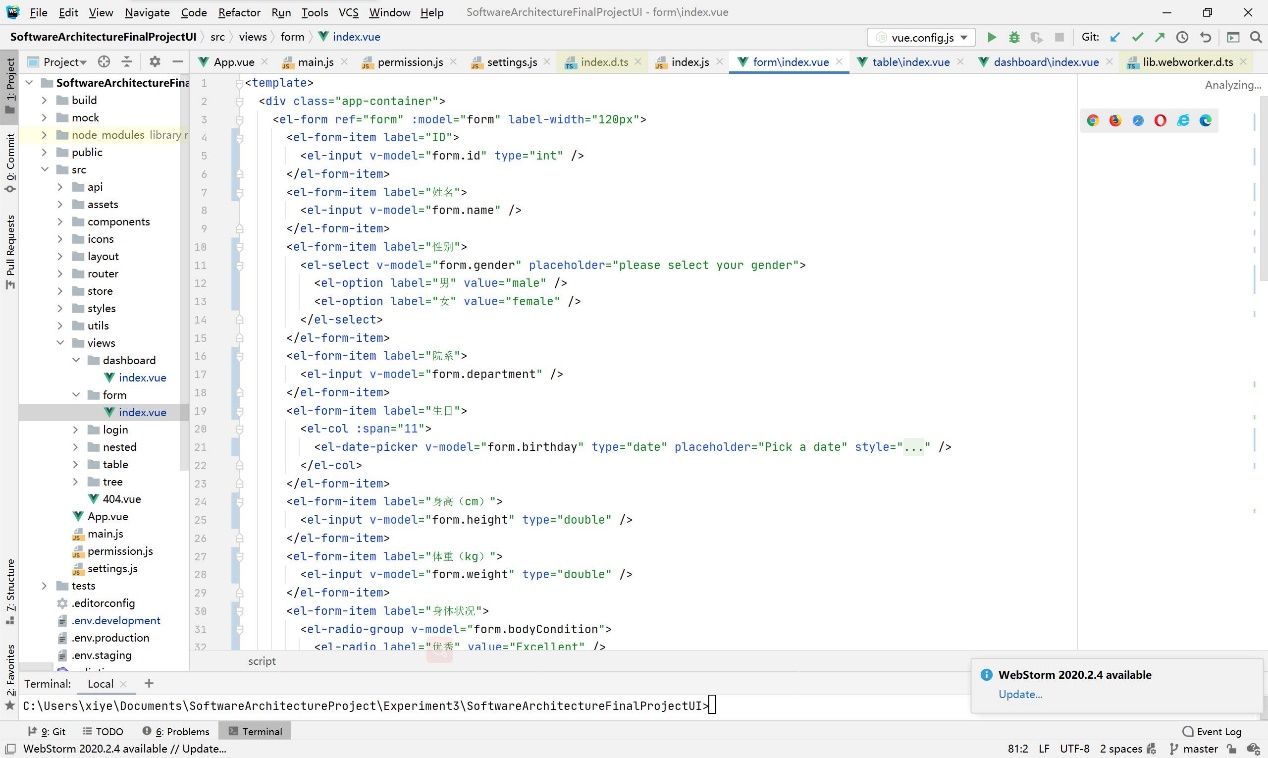
如第一部分项目简介中的具体模块成员分析中所列属性，我们在Mysql中建立了如上图所示的Table来存储运动员数据。

### 前端设计

最终表单填写界面如下图所示

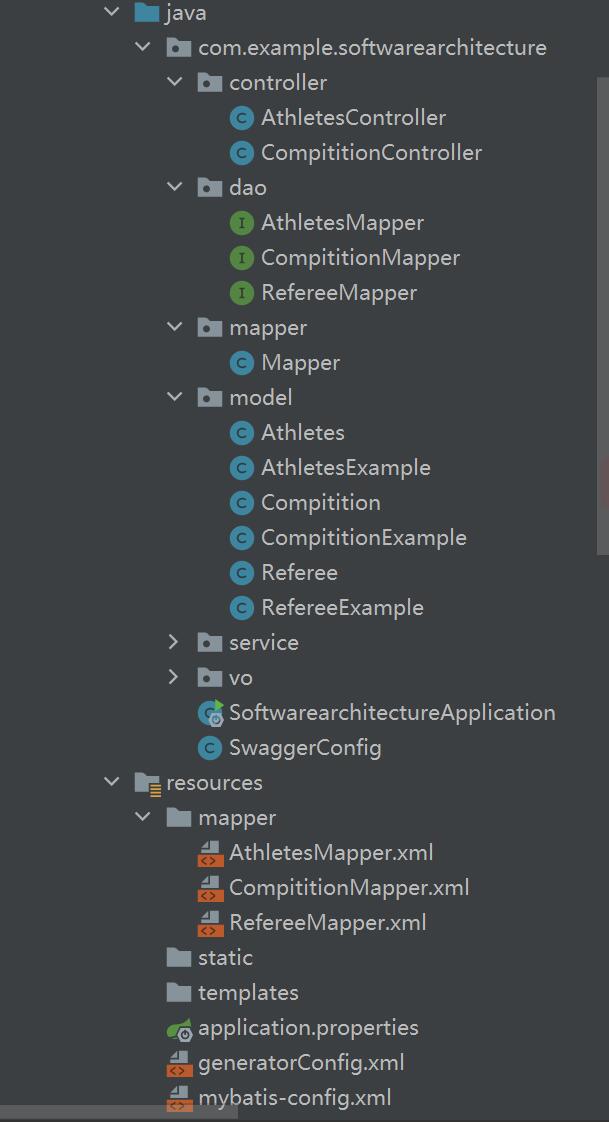


前端部分核心代码如下图所示：



由于前端是在vue-admin-template这一个极简的后台框架上进行的稍微的简单修改，所以无太多可以赘述的地方。

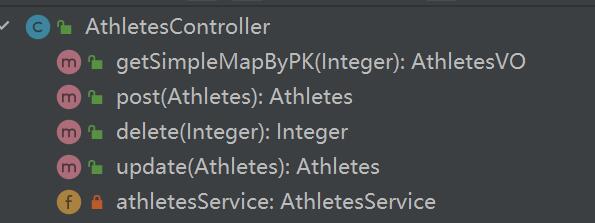
### 后端设计



后端代码整体架构如上图所示。

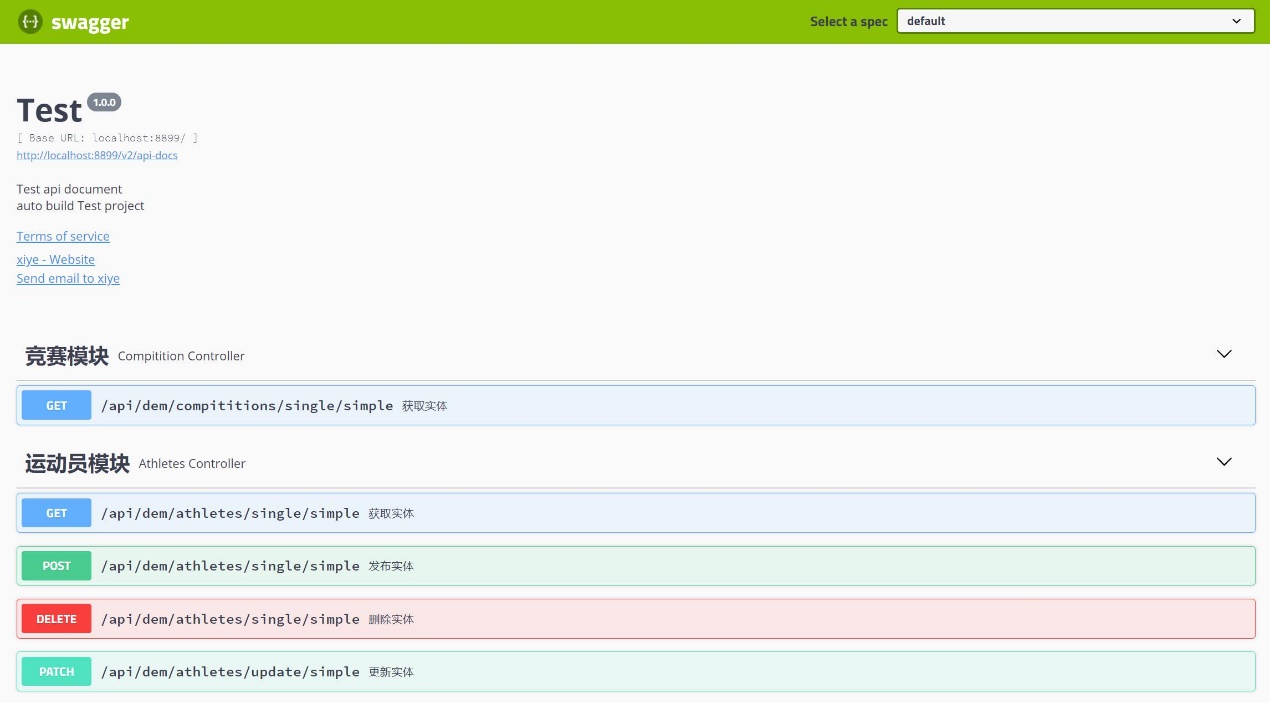
Mybatis层面，我们使用Mybatis Generator自动生成了每个文件的Mapper和相应的POJO类以及Example。Dao层中的mapper接口负责将resources中的xml格式的mapper文件与相应的model层中的实体类进行对应。Service层负责实现相应的功能。Controller层是留给前端的相应接口。

Controller层主要实现了下属几项接口：

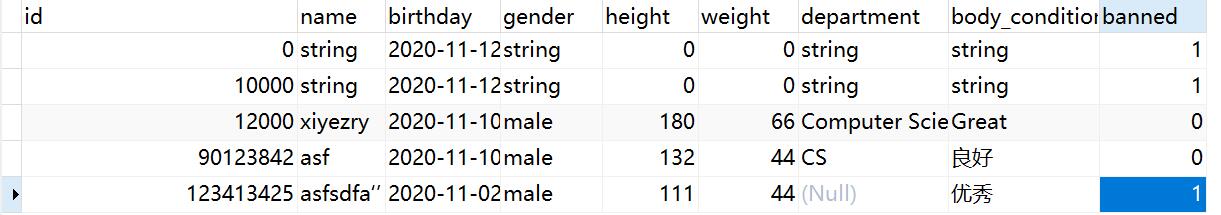


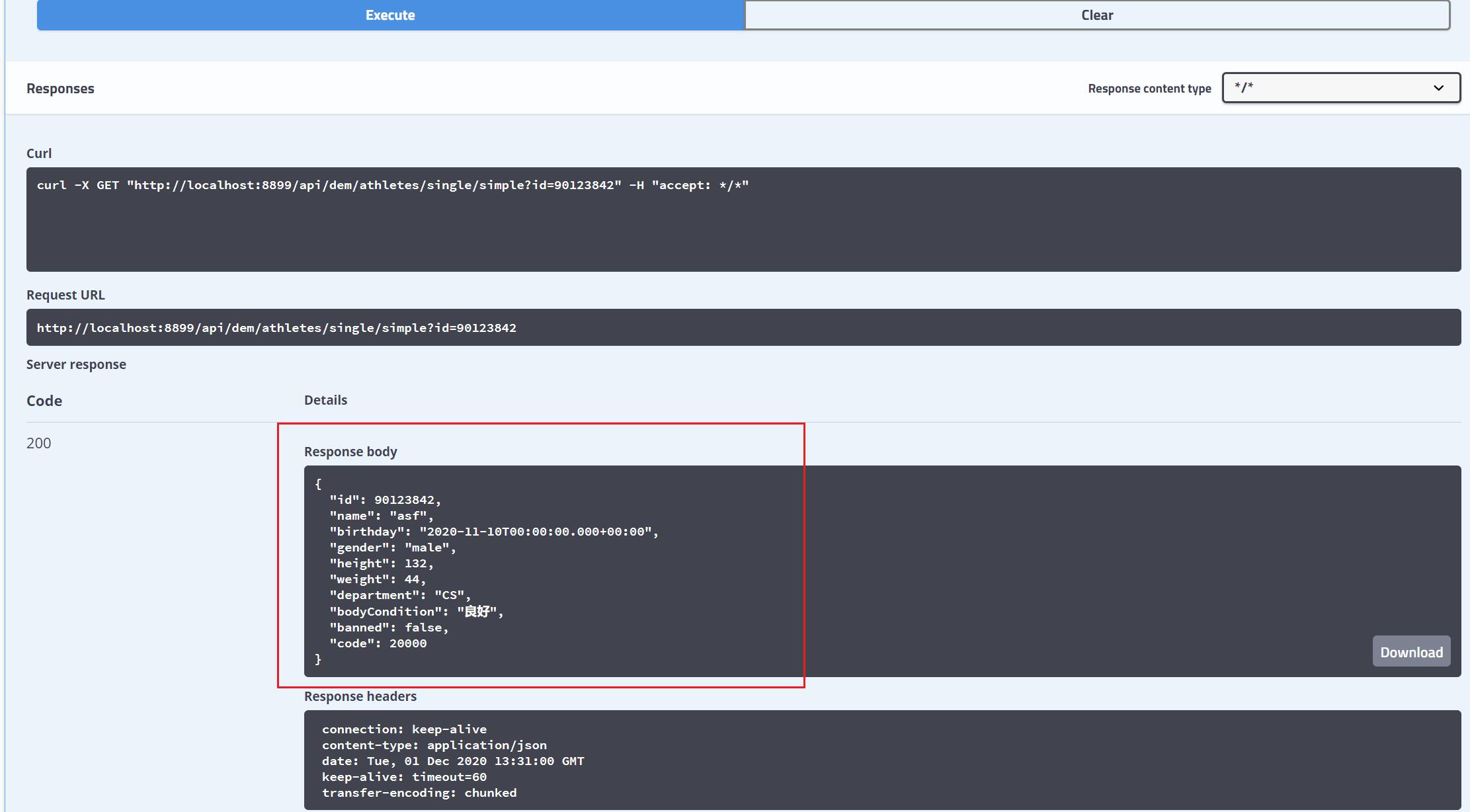
可以看到，已经实现了对运动员这一对象的基本CURD操作（虽然前端只调用了其中一个接口……）。

我们使用Swagger这一框架来测试接口功能是否完整，我们的接口在Swagger中如下所示：



可以看到 当数据库如下图所示时，我们已经可以用swagger获取到其中一个实体的详细信息。





其余接口不在具体赘述

### 前后端连接时的跨域问题

当前后端分离时，如果在协议、域名、端口三者之间的任意一个与当前url不同就会构成跨域问题。我们在实现过程中就遇到了Chrome浏览器报出CROS跨域问题。虽然教程告诉我们vue框架和后端spring boot都有比较简便的解决跨域的方法，但是经过失败了很多次很多次的尝试后，我们终于用Nginx成功解决了跨域问题。

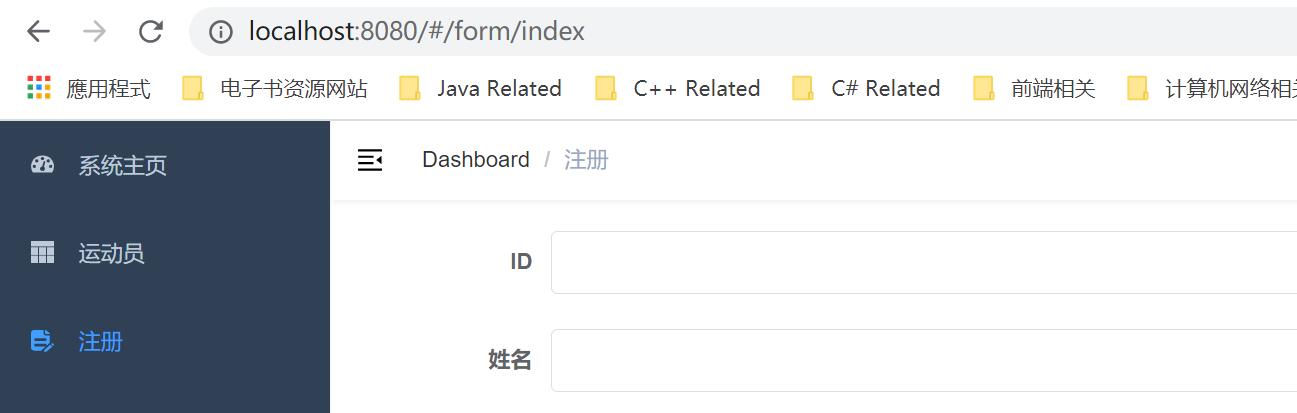
Nginx的具体原理在此处不再赘述，Nginx的具体配置文件如下：

![C:\Users\xiye\Documents\Tencent Files\469292748\Image\Group2\X4\](\X4](O{WXRW9BJE4%@YR22FD.jpg](data:image/jpeg;base64,)

简单来说，我们把本地的8899端口和9528端口都反代理到了8080端口。这样浏览器就会误以为前后端是一个端口啦。

### 前后端整合后效果

可以看到，当我们打开nginx后，从8080接口也能直接进入原先是9528接口的前端：

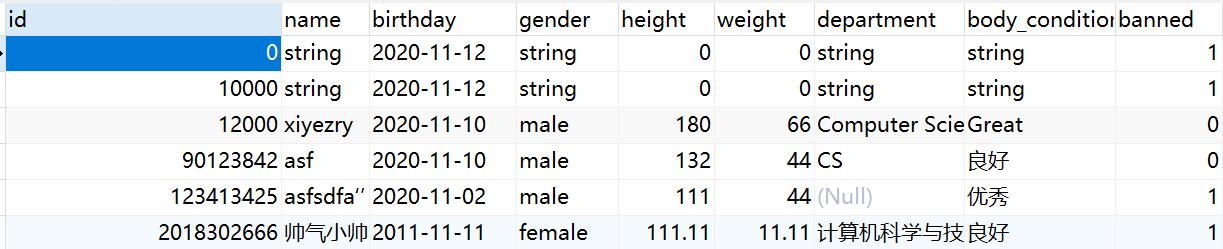


下面，我们来测试这时候对前端提交的表单能否成功地存入数据库。

我们来测试一下往表单中填入数据：



前端表单中的数据填入并点击Create键后，我们查看数据库，此时的数据库信息如下：



可以看到，我们的帅气小帅同学的基本信息已经被插入数据库中的Athletes表中了！