

数据库课程设计一

数据库后端

一、后端的开发与设计

使用 Python 作为开发语言，通过 Django 库进行后台框架搭建。主要对三个文件进行设计：

- ①通过 `Urls.py` 来解析前端发送过来的 `Http` 请求，并执行相应的方法
- ②在 `Views.py` 里完成各种方法的实现，并进行对数据库的操作
- ③在 `Models.py` 里实现数据库模型的构建，供 `Views.py` 使用；或者同样使用 `MySQLdb` 进行数据库的建立，两种方式均可。

二、数据库表的定义和展示

数据库的定义较为简单，首先我们定义以下实体：球员姓名，球员生涯开始时间，球队名，球队所在城市。然后用两个表定义关系模式：球员信息表和球队信息表。球员信息表有如下属性：球员姓名（Prime Key 此次作业先把球员名当做主码使用）、生涯起始时间、所在球队（外码）。球队信息表包括球队名（Prime Key）、球队所在城市。此次数据库仅供实现基本操作，所以较为简单。

player_name	begin_date	team_id
Curry	2009-09-01	Golden State Warriors
Durant	2007-09-01	Golden State Warriors
James	2018-10-12	Laker
James Harden	2009-09-01	Rockets
Klay Thompson	2011-09-01	Golden State Warriors
Westbrook	2008-09-01	Thunder

球员信息表

team_name	team_city
Golden State Warriors	Oakland
Laker	Los Angeles
Rockets	Houston
Thunder	Oklahoma

球队信息表

三、数据库的连接方法与基本操作

①使用 Python 的 MySQLdb 的 connect 进行数据库连接，并使用 cursor 游标进行操作。

```
config = {
    'host': '127.0.0.1',
    'port': 3306,
    'user': 'root',
    'passwd': 'change',
    'db': 'nba_database',
}
conn = mdb.connect(**config)

conn.autocommit(True)
```

②基本操作的实现：

1.“增加球员”操作使用 MySQL 的 INSERT，具体实现中为添加球员信息，先查找是否已经有了同名球员，如果有了返回相应信息，如果没有，再查找是否有该球队信息。如果没有球队信息则先创建新球队，然后再根据新球队名最为外码创建新球员。

```
def add_player(request, player_name, team_name, begin_date = datetime.date.today()):
    try:
        db_name = 'nba_database'
        conn.select_db(db_name)

        table_name = 'nbasite_player'

        player_exist = cursor.execute('SELECT * from %s where player_name = \'%s\'' % (table_name, player_name))

        if(player_exist != ZERO):
            print("Player is Already in the Table.")
            raise()

        else:
            table_name = 'nbasite_team'

            team_exist = cursor.execute('SELECT * from %s where team_name = \'%s\'' % (table_name, team_name))

            if(team_exist == ZERO):
                print("Not Found Team.")
                #先创建新的队伍
                team_city = ''
                cursor.execute('INSERT into nbasite_team(`team_name`, `team_city`) values(\'%s\', \'%s\')' % (team_name, team_city))

                #再创建新的队员
                begin_date_str = begin_date.strftime("%Y-%m-%d")

                cursor.execute('INSERT into nbasite_player(`player_name`, `begin_date`, `team_id`) values(\'%s\', \'%s\', \'%s\')' % (player_name, begin_date_str, team_name))

            else:
                print("Found Team.")
                #直接创建队员
                begin_date_str = begin_date.strftime("%Y-%m-%d")
                cursor.execute('INSERT into nbasite_player(`player_name`, `begin_date`, `team_id`) values(\'%s\', \'%s\', \'%s\')' % (player_name, begin_date_str, team_name))

            return HttpResponse("Player %s has been added into %s." % (player_name, team_name))

    except:
        return HttpResponse("Player %s is already in the Table." % player_name)
```

2.“删除球员”操作使用 MySQLdb 的 DELETE，具体实现为，先在表中找同名球员，如果没有给出相应的反馈，如果有则将其从数据库中删除并给出回应。

```
def delete_by_name(request, player_name):
    try:
        db_name = 'nba_database'
        conn.select_db(db_name)

        table_name = 'nbasite_player'

        try:
            result = cursor.execute('DELETE from %s where player_name = \'%s\'' %(table_name, player_name))

            if(result == ZERO):
                raise()
            else:
                return HttpResponse("Player %s has been deleted successfully." % player_name)

        except:
            print("Player Not Found.")
            raise()

    except:
        return HttpResponse("Player %s is Not in the Table!" % player_name)
```

3.“更改队伍”操作使用 MySQLdb 的 UPDATE，具体实现为，先在表中找同名球员，如果没有则报错并返回消息，如果有则在表中查找同名球队。如果没有则先建立该球队，如果有则将球员的球队信息更新。

```
def change_team_by_name(request, player_name, team_name):
    try:
        db_name = 'nba_database'
        conn.select_db(db_name)

        table_name = 'nbasite_player'

        number = cursor.execute("SELECT * from %s where player_name = \'%s\'' %(table_name, player_name))

        if(number == ZERO):
            print("Player is Not in the Table.")
            return HttpResponse("Change Fail. Player %s is Not in the Table!" % player_name)

        table_name = 'nbasite_team'
        team_exist = cursor.execute("SELECT * from %s where team_name = \'%s\'' %(table_name, team_name))

        if(team_exist == ZERO):
            team_city = ''
            cursor.execute('INSERT into nbasite_team(`team_name`, `team_city`) values(\'%s\', \'%s\')' %(team_name, team_city))
            print("Team Add Succeed.")

        table_name = 'nbasite_player'
        cursor.execute("UPDATE %s set `team_id` = \'%s\' where `player_name` = \'%s\'' %(table_name, team_name, player_name))

        return HttpResponse('Change %s\'s team succeed.' % player_name)

    except:
        return HttpResponse("Change Fail. Player %s is Not in the Table!" % player_name)
```

4.“查询球员”操作使用 MySQLdb 的 SELECT，查找同名球员，如果存在，再根据外部键找出球队信息，并将球员和球队信息一起输出。如果不存在，则返回查询失败。

```
def search_by_name(request, player_name):
    try:
        db_name = 'nba_database'
        conn.select_db(db_name)

        table_name = 'nbasite_player'

        cursor.execute('SELECT * FROM %s where player_name=\'%s\'' %(table_name, player_name))

        try:
            player_info = cursor.fetchone()

            begin_date = player_info[1]
            team_name = player_info[2]

            return HttpResponse("Team:" + '%s' %(team_name) + " Begin Date:" + '%s' %(begin_date))

        except:
            print("Fetch Player Failed")
            raise()

    except:
        return HttpResponse("Player %s is Not in the Table!" % player_name)
```

Web 前端

前端使用 html 构建整体页面，并嵌入 JavaScript 函数将网页表单数据传给后台服务器。这里网页界面比较简陋，后续会改进，本次作业主要学习了如何实现前后台交互，这里我们使用了 ajax 方式实现交互，具体实现方式见如下代码。这里只列举了查找球员信息操作的实现方式，其余几种操作原理相同。

Html 网页的主要分为两个部分，<head></head>即头部分，和<body></body>主题部分。在头部分我们定义了 JavaScript 函数，页面标题以及编码方式等。<body>标签内的部分是浏览器解析后生成的网页界面，html 中每个标签都有许多属性，通过某些属性的值可以确定该标签的位置，或绑定一个函数等。

本次作业我们只定义了几个块<div>，将数据库访问操作分隔开，然后在每个块内定义了一个表单，实现交互。在表单内我们定义了一个 button，并用标签属性 onclick 绑定了 JavaScript 函数。当用户输入球员数据，并点击按钮，即可调用 JavaScript 函数。

```
<div id="container">
  <div id="header" style="background-color:rgb(35, 147, 221);">
    <h1 style="margin-bottom:0;">NBA球员数据库</h1>
  </div>

  <div id="search" style="background-color:rgb(236, 217, 109); height:100px;">
    <b>查询球员信息</b>
    <form name="search-form">
      球员名: <input type="text" name="player">
      <button type="button" onclick="searchPlayerMsg()">确认</button>
      <p id="search-msg"> </p>
    </form>
  </div>
```

这里解释一下我们绑定的 JavaScript 函数，函数通过 html 的标签查找到改表单内填写的内容，使用 JavaScript 的 XMLHttpRequest 对象，它的 open 函数可以设置访问服务器的方式，访问的 url，以及同步或异步访问。然后使用 send 函数发送请求。在收到回复后该对象的状态会改变，这会触发我们预先绑定的一个函数，用于在 html 的一个名为 search-msg 的段内输出服务器返回的内容。

```
<script>
function searchPlayerMsg() {
  // var player = document.getElementById("player").value;
  var player = document.forms["search-form"]["player"].value;
  var xmlhttp = new XMLHttpRequest();
  var url = "search_player="+player;
  xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState==4 && xmlhttp.status==200) {
      document.getElementById("search-msg").innerHTML=xmlhttp.responseText;
    }
  }
  xmlhttp.open("GET",url,true);
  xmlhttp.send();
}
```

展示效果

NBA球员数据库

查询球员信息

球员名:

添加新球员

球员名: 球队名:

删除查询球员数据

球员名:

更改·球员数据

球员名: 球队名:

1.查询操作存在

查询球员信息

球员名:

Team:Laker Begin Date:2018-10-12

2.查找球员不存在

查询球员信息

球员名:

Player Kobe is Not in the Table!

3.添加新球员成功

添加新球员

球员名: 球队名:

Player Kobe Bryant has been added into Laker.

4.添加已存在球员失败

添加新球员

球员名: 球队名:

Player Kobe Bryant is already in the Table.

5.删除球员成功

删除查询球员数据

球员名:

Player Kobe Bryant has been deleted successfully.

6.删除不存在球员失败

删除查询球员数据

球员名:

Player Kobe Bryant is Not in the Table!

7.更换球员球队成功

更改·球员数据

球员名: 球队名:

Change Durant's team succeed.

8.更换不存在球员失败

更改·球员数据

球员名: 球队名:

Change Fail. Player Kobe is Not in the Table!