南京大学

本 科 毕 业 设 计

院　系 软件学院

专　业 软件工程

题　目 NBA 数据分析系统服务器模块的设计与实现

年　级 2012 级　学　号 121250133

学生姓名 孙凡

指导教师 刘钦　职　称 讲师

论文提交日期 2016 年 5 月 20 日

南京大学本科生毕业论文（设计）中文摘要

毕业论文题目： NBA 数据分析系统服务器模块的设计与实现

软件学院 院系 软件工程 专业 2012 级本科生姓名：

指导教师（姓名、职称）： 刘钦 讲师

摘要：

现有的 NBA 应用更多地关注球迷群体，功能点着重于比赛数据的呈现，但是缺乏对球队教练和球探需求的重视。在各类数据统计和数据分析在 NBA 发展已经十分成熟的情况下，利用这些数据帮助球队教练和球探作出合 理的决策显得很有必要。NBA 数据分析系统便致力于通过对数据的组织处理为各类角色提供所需的信息。

NBA 数据分析系统基于 Spring MVC 和 Hibernate 设计开发了自身的服务器框架，客户端则是通过 Android 实现。在实现从网页获取 NBA 数据时，主要使用了 Java 正则表达式对网页源码进行匹配，并使用 Spring 的定时任务 Task 实现数据的定时爬取和更新。项目组整体实现了一个能够为球迷、球探和教练提供 NBA 不同数据展示的 Android 应用。该应用为球迷提供基本的球员、球队和比赛的数据查询等功能，为球探则提供联盟球员的检索和自定义的筛选，为教练提供球队下球员检索和球队与其他球队的交锋数据分析等功能。

在项目开发过程中，作者独立完成了数据库的设计、服务器架构、数据爬取与定时任务的设计以及主要数据接口的实现。

关键词：NBA；数据分析；Spring MVC 框架；Hibernate 框架；数据爬虫；Spring 定时任务

南京大学本科生毕业论文（设计）英文摘要

THESIS: Design and implementation of server module of NBA Data Analysis System

DEPARTMENT: Software Institute

SPECIALIZATION: Software Engineering

UNDERGRADUATE: Fan Sun

MENTOR: Qin Liu

ABSTRACT:

Most of the applications about NBA just focus on providing information about games to users, ignoring the requirements of coaches and scouts for advanced data. Our project, the NBA Data System tries to provide more organized and advanced data to help coaches and scout making reasonable decisions.

The NBA Data Analysis System is an Android application, and we design the server module based on Spring MVC Framework and Hibernate Framework. By using java regular expression, we are able to grab data from web pages storing data of NBA. We also design the Spring Timing Task to deal with the automatic data maintenance .Apart from basic information about players、teams and matches, the NBA Data Analysis System also provides searching and self-defined filtering of NBA players for scouts and data analysis of team players as well as team data comparison for coaches.

The author's duty includes design of database and server module as well as the implementation of data crawler、the timing task and the main data interface.

KEY WORDS: NBA;Data Analysis;Spring MVC;Hibernate;Data Crawler;Spring Timing Task

目 录

图目录

表目录

第一章 引言

1.1 项目背景

NBA（美国职业篮球联赛）于 1946 年 6 月 6 日在纽约成立，是由北美三十支队伍组成的男子职业篮球联盟，美国四大职业体育联盟之一。汇集了全世界最顶级的球员，是世界上水平最高的篮球赛事。[1]自成立以来，NBA 由最初的 11 支球队发展壮大为现在遍布全美的 30 支球队，各种制度与规则也日益完善，缔造了许多大众熟知的篮球巨星，在全世界吸引了数以亿计的球迷，仅仅在中国，NBA 在社交媒体上的粉丝就超过了一亿。除了通过观看比赛了解最新赛程与联盟动态，广大关注 NBA 的球迷群体也通过各种篮球论坛或是 NBA 应用了解着 NBA 球队或球员的最新动态。

1.2 国内（外）NBA 应用现状

由于 NBA 球迷群体的庞大，目前已经有许多知名的 NBA 应用，例如虎扑篮球、腾讯体育，新浪 NBA，甚至 NBA 官方（美国职业篮球协会）也在 2016 年 4 月 15 日推出了首个 NBA 官方应用——NBA APP，为中国球迷量身打造。这些应用基本都是以球迷为主要的受众对象，球迷可以从 APP 及时接收推送消息，了解球员和球队新闻、比分、数据、排名、重要活动和比赛日程，并且可以在比赛期间通过应用的图文或视频直播了解比赛最新动态与数据统计。

但是现有的 NBA 应用在关注广大球迷群体的同时，忽视了另一部分人—球探和教练，对于 NBA 信息更高阶层的需求。教练和球探需要的不仅仅是基础的数据统计，而需要更具有针对性的数据，对于数据的组织与生成有着更高的需求。

1.3 论文的组织架构

本文的组织结构如下所示：

第一章 引言部分。介绍了项目背景与现有的 NBA 应用提供的主要功能。

第二章 技术概述。对服务器架构所使用的 Spring MVC 和 Hibernate 技术，以及数据爬取和数据维护所使用的 Java 正则表达式与 Spring 定时任务 Task 进行了一定的阐述。

第三章 NBA 数据分析系统需求分析与概要设计。首先从整体对应用的需求进行了分析，然后对服务器架构和数据库设计进行了阐述。

第四章 NBA 数据分析系统服务器模块的详细设计与实现。对服务器的四个主要模块的详细设计与实现进行了分析与阐述。

第五章 总结与展望。总结论文期间所做的工作，并进一步展望 NBA 数据分析系统未来的发展方向。

第二章：技术概述

2.1 Spring MVC 框架

NBA 数据分析系统服务器的主体架构采用了 Spring MVC 框架，通过实现 Model-View-Controller 模式来很好地将数据、业务与对象进行分离。在设计上，Spring MVC 是围绕 DispatcherServlet 实现的，DispatcherServlet 负责将请求派发到特定的 handler，通过可配置的处理器映射（handler mappings）、视图解析（view resolution）、本地化（locale）以及主题解析（theme resolution）来处理请求并且转到对应的视图。[2]DispatcherServlet 的配置是在 web.xml 中声明的：

```
<servlet>
        <servlet-name>springmvc</servlet-name>

    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:spring-mvc.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>*.do</url-pattern>
  </servlet-mapping>
```

图 2.1 DispatcherServlet 配置图

上面声明了一个名为 springmvc 的 DispatcherServlet,该 Servlet 将处理所有以.do 结尾的请求，在初始化 DispatchServlet 的时候，SpringMVC 会到项目的 classpath 下寻找名为 spring-mvc.xml 的配置文件，并初始化其中的 bean 对象。

NBA 数据分析系统服务器端使用的是基于注解的 Spring MVC 架构，通过注解来实现注入。处理器 Handler 是基于@Controller 和@RequestMapping 这两个注解的，@Controller 声明一个处理器类，@RequestMapping 声明对应请求的映射关系，这样就可以提供一个非常灵活的匹配和处理方式。

```
@Controller
public class MatchController {
    @Resource
    private MatchService matchService;
```

```
        /**
         * @return  最近50场比赛的列表
         */
        @RequestMapping("/getLatestMatchList")
        public @ResponseBody List<Match> getLatestMatches(){
                return matchService.getLatestMatches();
        }
}
```

图 2.2 基于注解的 SpringMVC 示例图

在上面的代码中，@Controller 注解标识了这个类是一个控制器，接下来通过@RequestMapping 注解指定了方法处理 URL 为 getLatestMatchList 的请求，在 RequestMapping 中还可以指定一些其他属性：通过 method 属性，可以严格控制某一方法只能被标记的请求路径对应的请求方法才能访问，如指定 method 的值为 GET，则表示只有通过 GET 方式才能访问该方法；通过 params 属性，可以通过该属性指定请求参数中必须包含某一参数，或必须不包含某一参数，或某参数的值必须是什么，以此来缩小指定的映射范围。[3]

依赖注入是 Spring 的一大特性，能够有效促进松耦合，一个对象不需要自己去创建或获取自己依赖的对象，而是被动的获取，从而在切换依赖的时候不需要改变使用依赖的对象的代码。除了上文提到的 Controller,Spring MVC 中和注入相关的常见注解还有 Autowired、Resource、Qualifier、Service、Repository、Component。[4]Autowired 是自动注入，自动从 spring 的上下文找到合适的 bean 来注入；Resource 用来指定名称注入；Qualifier 和 Autowired 配合使用，指定 bean 的名称；Service，Controller，Repository 分别标记类是 Service 层类，Controller 层类，数据存储层的类，spring 扫描注解配置时，会标记这些类要生成 bean；Component 是一种泛指，标记类是组件，spring 扫描注解配置时，会标记这些类要生成 bean。在上图的代码中，@Resource 注解注入了一个 Service 层对象 matchService，在方法 getLatestMatches()中调用了注入对象 matchService 的方法获取所需的数据。

2.2 Hibernate 技术

Hibernate 是 NBA 数据分析系统所使用的另一大框架，Hibernate 是一个开源的对象关系映射框架，它对 JDBC 进行了轻量级的对象封装，使得程序员可以使用面向对象编程思维来操纵数据库，完成数据持久化的任务。[5]

Hibernate 的核心组件包括以下几个部分[6]：

（1）Session 接口：

用来操纵持久化对象，有 get()、save()、update()和 delete()等方法用来对 PO

进行加载，保存，更新及删除等操作,是 Java 应用程序和 Hibernate 进行交互所使用的主要接口，也是持久化操作的核心 API。

（2）SessionFactory 接口：

SessionFactory 接口负责初始化 Hibernate。它充当数据数据源的代理，并负责创建 Session 对象。SessionFactory 是线程安全的，多个并发线程可以同时访问一个 SessionFactory 并从中获取 Session 实例。

（3）Transaction 接口：

用来管理 Hibernate 事务，它主要方法有 commit()和 rollback()，用于提交操作结果到数据库或是在异常情况下回滚数据库操作，可以从 Session 的 beginTrancation()方法生成。

（4）Configuration 接口：

Configuration 的作用是对 Hibernate 进行配置，并对它进行启动。在 Hibernate 启动的过程中，Configuration 类的实例会从配置文件中读取 Hibernate 配置，然后创建 SessionFactory 对象。

（5）Query 接口：

用于对持久化对象进行查询操作，可以从 Session 的 createQuery()方法生成。

Hibernate 的运行过程：

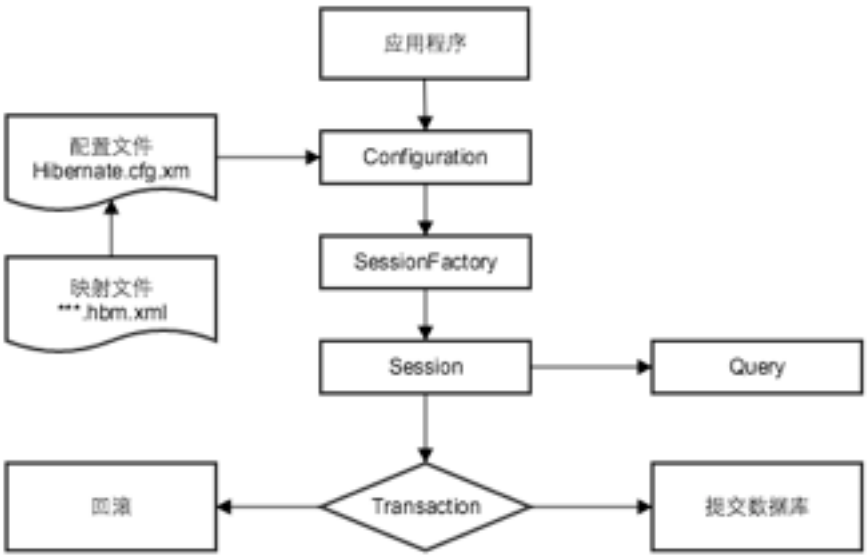

图 2.3 Hibernate 运行过程图

如上图所示，在 Hibernate 运行过程中，应用程序首先调用 Configration 类，该类读取 Hibernate 的配置文件及映射文件中的信息（使用注解配置可以代替映射文件），并根据配置文件的信息生成 SessionFactory 对象。然后从 SessionFactory 对象生成 Session 对象，在需要对持久化对象进行查询的情况下，可通过 Session

对象的 createQuery()方法生成一个 Query 对象，然后利用 Query 对象执行查询操作；在需要对持久化对象进行增删改操作的情况下，Session 对象会调用 beginTransaction()方法生成 Transaction 对象，然后通过 Session 对象的 get()、load()、save()、update()、saveOrUpdate()、delete()等方法对 PO 进行加载，保存，更新，删除等操作，如果没有异常，Transaction 对象会将操作结果提交到数据库并清理 Session 的缓存，如果存在异常，Transaction 对象会回滚操作。

Hibernate 注解配置：

Hibernate 实现有两种配置方式，采用配置文件进行 xml 配置或是采用注解配置，在 NBA 数据分析系统中，对 model 采用的是注解配置方式，下面介绍项目中所使用的注解配置。

```
@Entity
@Table(name = "team")
@JsonIgnoreProperties(value = { "handler", "hibernateLazyInitializer" })
public class Team {
	@Id
	private int id;

	@Column(name="name")
	private String name;

	private String city;

}
```

图 2.4 Hibernate 注解配置示例图

上面给出的是球队类的部分代码，@Entity 和@Table 注解表明当前的类是一个持久化类，映射数据表 team，@Column 注解标识了数据表中的字段，@Id 注解标识了数据表中的主键，需要注意的是持久化类中只能有一个属性标识为@Id，对于拥有联合主键的表，可以简单地再添加一个字段作为主键，或者可以使用联合主键类。例如对于项目中的球队比赛数据类，一场球赛对应两只球队，所以需要球队的 id 和比赛的 id 才能唯一的确定一个球队某一场比赛的数据统计，因此，可以设置球队 id 和比赛 id 作为球队比赛数据的联合主键类：

```
@Entity
@Table(name = "team_match_statistics")
@IdClass(net.nba.model.TeamMatchStatisticsPK.class)
@JsonIgnoreProperties(value = { "handler", "hibernateLazyInitializer" })
public class TeamMatchStatistics {
```

```
        @Id
        private int matchId;
        @Id
        private int teamId;
}
```

图 2.5 Hibernate 下联合主键使用图

```
package net.nba.model;

import java.io.Serializable;

/**
 * @author sunfan314
 *      TeamMatchStatistics主键类
 */
public class TeamMatchStatisticsPK implements Serializable{
        private int matchId;
        private int teamId;
        public TeamMatchStatisticsPK() {
                super();
        }
        public TeamMatchStatisticsPK(int matchId, int teamId) {
                super();
                this.matchId = matchId;
                this.teamId = teamId;
        }
        public int getMatchId() {
                return matchId;
        }
        public void setMatchId(int matchId) {
                this.matchId = matchId;
        }
        public int getTeamId() {
                return teamId;
        }
        public void setTeamId(int teamId) {
                this.teamId = teamId;
    }
```

图 2.6 Hibernate 主键类示例图

作为主键类，必须符合以下几点要求[7]：

（1）：必须实现 Serializable 接口

（2）：必须有默认的 public 无参数的构造方法

（3）：必须覆盖 equals 和 hashCode 方法。equals 方法用于判断两个对象是

否相同，EntityManger 通过 find 方法来查找 Entity 时，是根据 equals 的返回值来判断的。本例中，只有对象的 matchId 和 teamId 值完全相同时返回 true，否则返回 false。hashCode 方法返回当前对象的哈希码，生成的 hashCode 相同的概率越小越好，算法可以进行优化。

如上面的代码所示，在编写完主键类 TeamMatchStatisticsPK 之后，在持久化类 TeamMatchStatistics 中通过@IdClass 注解对联合主键进行标注，并在类中同时标注主键属性 matchId 和 teamId 表明主键类使用这两个属性就完成了对联合主键的声明。此时通过如下代码就可以获取某个球队某场比赛的数据统计：

```
TeamMatchStatisticsPK pk=new

TeamMatchStatisticsPK(teamId,matchId);

TeamMatchStatistics

statistics=dao.get(TeamMatchStatistics.class,pk);
```

Hibernate 关联映射[8]：

基于关系型数据库表之间的相互关联，Hibernate 也提供了关联映射的方式展现持久化类之间的关系。Hibernate 下常用的关联映射有七种：

（1）一对一单向关联映射：

适用于两个对象之间一对一的关系，例如项目中球员基本信息 Player 和球员详细信息 PlayerInfoDetail 之间的关系，有两种策略可以实现两者之间一对一的关联映射：

外键关联：外键关联，本来是用于多对一的配置，但是加上唯一的限制之后（采用@OneToOne 注解来映射，指定多的一端 unique 为 true，这样就限制了多的一端的多重性为一），也可以用来表示一对一关联关系

主键关联：让两个对象具有相同的主键值，以表明它们之间的一一对应的关系；数据库表不会有额外的字段来维护它们之间的关系，仅通过表的主键来关联。

（2）一对一外键双向关联映射：

对比一对一单向关联映射，需要在另一端也添加<one-to-one>标签，并且需要加上 mappedBy 属性，在一对一双向关联中，没有声明 mappedBy 属性的一端作为主体端，承担着维护这段关系的责任，而声明了 mappedBy 属性的一端则不需要维护这段关系

（3）多对一单向关联映射：在多的一端添加外键，指向一的一端，需要使用@ManyToOne 注解来实现映射

（4）一对多单向关联映射：使用@OneToMany 注解来实现关联映射，需要

区别于多对一关联映射的是，如果不添加@JoinColumn 注解，Hibernate 会为一对多关系建立中间表而不是为多的一端添加外键

（5）一对多双向关联映射：在一的一端使用@OneToMany 注解并声明 mappedBy 属性指定由多端对关系进行维护，在多的一端使用@ManyToOne 注解

（6）多对多单向关联映射：在一端使用@ManyToMany 注解，在默认情况下，Hibernate 会自动创建一张中间表来维护多对多关系，用户也可以通过@JoinTable 注解自行声明中间表

（7）多对多双向关联映射：在两端都使用@ManyToMany 注解，在一端声明 mappedBy 属性指定由另一多端维护这段关系

Spring MVC 下 Hibernate 的配置：

在 Spring MVC 下使用 Hibernate，需要在 spring 对应的配置文件中对 Hibernate 的数据源、SessionFactory 以及事务进行配置：

数据源配置：

```
<!-- 关联数据库properties文件 -->
    <bean

    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
            <list>
                <value>classpath:db.properties</value>
            </list>
        </property>
    </bean>

    <!-- 配置DataSource数据源 -->

    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="${driverClassName}" />
        <property name="url" value="${url}" />
        <property name="username" value="${username}" />
        <property name="password" value="${password}" />
    </bean>
```

图 2.7 SpringMVC 下数据源配置图

SessionFactory 配置：

```
<!-- 配置hibernate -->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
```

```
        <property name="dataSource">
            <ref bean="dataSource" />
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <!-- 设置懒加载不受事务管理，解决懒加载和session关闭的冲突
问题 -->
                <prop key="hibernate.enable_lazy_load_no_trans">true</prop>
            </props>
        </property>
        <!-- 自动扫描注解方式配置的hibernate类文件 -->
        <property name="packagesToScan">
            <list>
                <value>net.nba.model</value>
            </list>
        </property>
    </bean>
```

图 2.8 SpringMVC 下 SessionFactory 配置图

Transaction 配置：

```
<!-- 事务管理 -->
    <bean id="transactionManager"

    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <tx:annotation-driven transaction-manager="transactionManager" />
    <context:annotation-config />

    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="list*" propagation="REQUIRED" read-only="true" />
            <tx:method name="get*" propagation="REQUIRED" read-only="true" />
            <tx:method name="find*" propagation="REQUIRED" read-only="true"
            />
            <tx:method name="query*" propagation="REQUIRED" read-only="true"
            />
            <tx:method name="*" propagation="REQUIRED" />
        </tx:attributes>
```

```
        </tx:advice>
        <aop:config>
                <aop:pointcut            id="serviceMethods"            expression="execution(*
net.nba.service.*.*(..))" />
                <aop:advisor advice-ref="txAdvice" pointcut-ref="serviceMethods" />
        </aop:config>
        <aop:config>
                <aop:pointcut              id="taskMethods"              expression="execution(*
net.nba.data.task.*.*(..))" />
                <aop:advisor advice-ref="txAdvice" pointcut-ref="taskMethods" />
      </aop:config>
```

图 2.9 SpringMVC 下 Transaction 配置图

2.3 Java 正则表达式

NBA 数据分析系统需要获取 NBA 球队、球员以及各场比赛的详细数据统计，为此需要从网页爬取数据（在本项目中选择的是新浪 NBA 数据库）。在进行数据爬取时需要在网页源码中使用正则表达式获取需要的数据。

Java 正则表达式包 java.util.regex 包主要包括以下三个类[9]：

Pattern 类：

pattern 对象是一个正则表达式的编译表示。Pattern 类没有公共构造方法，想要创建一个 pattern 对象，必须首先调用其公共静态编译方法，它返回一个 pattern 对象，该方法接收一个正则表达式作为它的参数:

Pattern pattern=Pattern.compile(regular expression);

Matcher 类：

Matcher 对象是对输入字符串进行解释和匹配操作的引擎。与 Pattern 类一样，Matcher 也没有公共构造方法。你需要调用 Pattern 对象的 matcher 方法来获得一个 Matcher 对象:

Matcher matcher = pattern.matcher(inputStr);

PatternSyntaxException：

PatternSyntaxException 是一个非强制异常类，它表示一个正则表达式模式中的语法错误。

Matcher 类下提供了索引方法、研究方法与替换方法，在查找匹配的数据模式时，主要使用的 Matcher 类的研究方法：

public boolean find()：尝试查找与该模式匹配的输入序列的下一个子序列；

public boolean matches()：尝试将整个区域与模式匹配；

public boolean lookingAt()：尝试将从区域开头开始的输入序列与该模式匹配

public boolean find(int start)：重置此匹配器，然后尝试查找匹配该模式、从指定索引开始的输入序列的下一个子序列。

2.4 服务器定时任务技术

由于赛程数据的不断更新，服务器也需要对数据进行相应的更新与维护，为此需要设置定时任务，在赛季期间定时爬取网页数据并更新数据库数据。在项目中，所使用的是 Spring MVC 框架下注解执行的定时任务。

配置 Spring Task，首先需要在 Spring 的配置文件头中添加命名空间及描述：

```
xmlns:task=“http://www.springframework.org/schema/task"

http://www.springframework.org/schema/task
http://www.springframework.org/schema/task/spring-task-3.0.xsd
```

图 2.10 SpringMVC 下 Task 命名空间及描述图

由于使用的是注解方式执行任务，需要在配置文件 中开启定时器：

```
<task:annotation-driven />
```

图 2.11 SpringMVC 下配置开启 Task 图

只有添加了这句配置，Spring 才能识别@Scheduled 注解

最后是添加定时任务：

```
    /**
     * 更新球队信息，更新频率较低
     * 更新时间：在赛季期间（每年的10月至12月，一月至6月），每月的1日15日凌晨更新
     */
    @Scheduled(cron = "0 0 0 1,15 1-6,10-12 ?")
    public void updateTeamInfo(){
        List<Team> list = teamInfoSpider.getTeamInfoList();
        for (Team team : list) {
            teamDao.saveOrUpdate(team);
        }
        String info="Update Team Success!";
        MyLog.log(info);
    }
```

图 2.12 Spring 下定时任务实现图

@Scheduled 注解在需要执行的定时任务上，cron 表达式表明定时任务触发的时间，语法格式为[10]：

Seconds Minutes Hours DayOfMonth Month DayOfWeek

在每个域中都使用数字，但也可以出现特殊字符，例如上面的代码中出现了,和-两个特殊字符，其中,表示枚举值，DayOfMonth 域中以逗号间隔的 1 和 15 表示一月中的 1 日和 15 日；而-表示范围，Month 域的 1-6 和 10-12 表示每年的 1 月至 6 月，10 月至 12 月（由于 NBA 赛季一般在 10 月开始进入季前赛和常规赛，4 月进入季后赛，6 月决出总冠军）

2.5 本章小结

本章主要介绍了本项目所使用的关键技术。首先对服务器架构所使用的 Spring MVC 和 Hibernate 技术进行了简要的介绍，然后介绍了在 NBA 数据爬取过程中所使用的 Java 正则表达式，由于服务器的定时数据维护也是系统实现的一个难点，所以对于 Spring 定时任务 Task 作出了一定的介绍。

第三章 NBA 数据分析系统需求分析与概要设计

3.1 NBA 数据分析系统整体概述

现有的 NBA 应用更多的以广大球迷为主要的受众群体，忽视了教练和球探对于数据的更高层次的需求。为此我们开发了一款 Android 应用—NBA 数据分析系统，其服务器采用 J2EE（Spring MVC+Hibernate）进行架构。在 NBA 数据分析系统中，我们的受众对象除了广大球迷，还着重强调了球队教练和球探的需求，希望通过对于 NBA 球队、球员以及比赛数据的组织处理，为球迷提供基本的 NBA 动态的同时，也能满足球队教练与球探对于球队信息、球员信息更深入的需求。

3.2 NBA 数据分析系统需求分析

NBA 数据分析系统主要面向三类用户：球迷、球探与教练，三者的用户需求有一定的交集，但是都有各自着重的部分。对于球迷而言，球迷更注重 NBA 最新的动态信息，包括最近的比赛数据以及最新的球队与球员排名，当然也不乏对于球员或球队详细信息获取的需要，但是综合而言所需的都是 NBA 基础的球员、球队与比赛数据；相比球迷，球队教练所需的不仅仅是 NBA 的最新动态，对于某只球队的教练而言，关于球队自身一些更为详尽有针对性的高阶数据可能更为需要，在面对一场比赛的情况下，为教练提供一些双方球队与对位球员数据对比，以及对于球队球员进攻防守得分等能力值的衡量对于教练的决策更有帮助；而对球探而言，对于球队的关注点可能比较少，球探致力于通过各种数据统计寻找在某些能力上满足自身需要的球员，因此为球探提供可自定义的球员筛选条件能够对他们选择球员提供一定的帮助。

NBA 数据分析系统用例图如下所示：

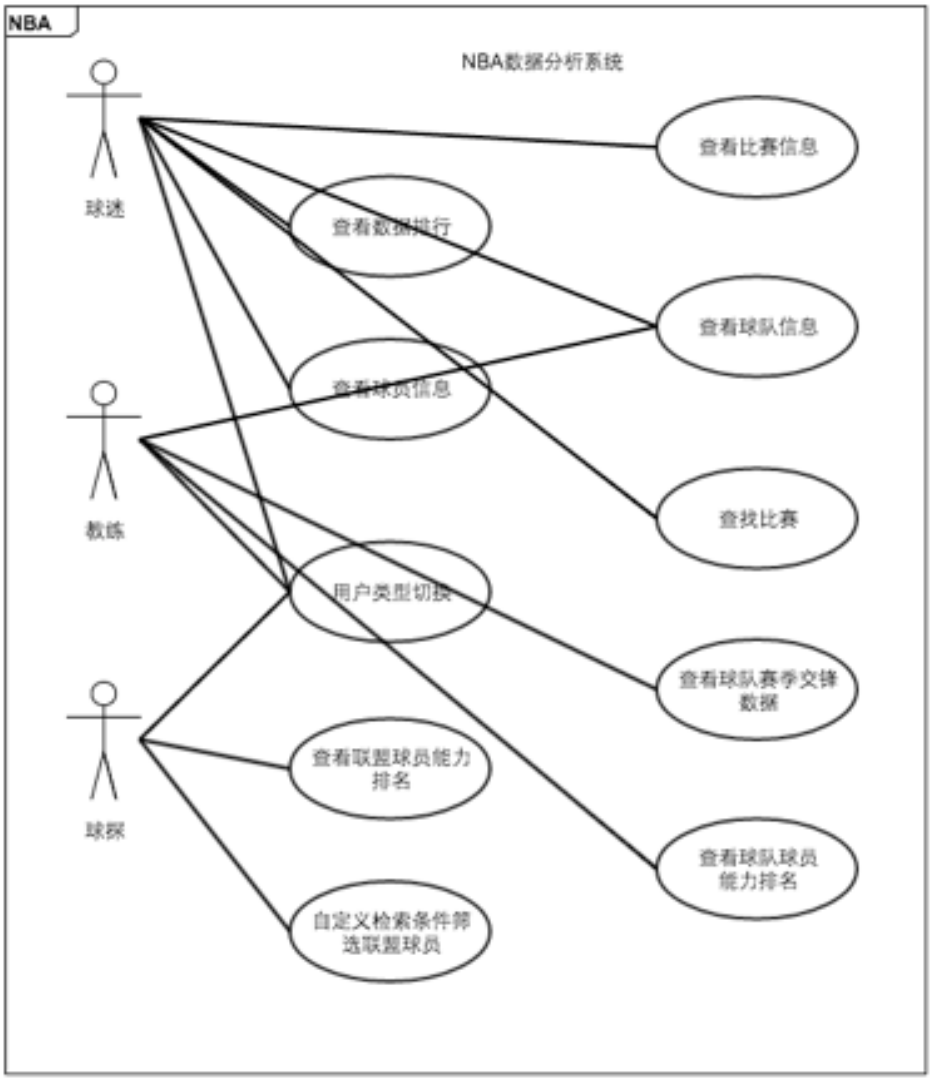

图 3.1 NBA 数据分析系统用例图

详细用例描述：

表 3.1 用户类型切换用例描述

ID	1
名称	用户类型切换
参与者	用户
触发条件	用户在侧栏抽屉中选择切换用户
前置条件	无
后置条件	系统存储用户类型，并根据用户选择的角色类型，切换应用主界面
正常流程	1. 用户选择切换用户类型 2. 系统提供用户类型选项并显示当前用户类型 3. 用户选择类型并进行切换 4. 系统存储用户类型，并根据用户选择的角色类型，切换应

	用主界面
扩展流程	3a.用户所选类型和原有类型相同 1.系统提示用户所选类型和原类型相同 2.返回至正常流程第一步
特殊需求	无

表 3.2 查看比赛信息用例描述

ID	2
名称	查看比赛信息
参与者	用户
触发条件	用户浏览比赛列表
前置条件	无
后置条件	无
正常流程	1.用户在比赛列表中选择一场比赛进行查看 2.系统跳转至比赛详细信息，显示对战球队数据统计与球员数据统计 3.用户点击返回至比赛列表 4.系统返回比赛列表界面
扩展流程	无
特殊需求	1.比赛列表显示最近的 50 场比赛记录 2.比赛列表按日期分段显示，最近的比赛置于比赛列表的最顶部

表 3.3 查看球队信息用例描述

ID	3
名称	查看球队信息
参与者	用户
触发条件	用户点击球队图标
前置条件	无
后置条件	无
正常流程	1.用户点击球队图标 2.系统跳转至球队信息页面，显示球队基本信息（球队名称、球队所在城市、所处分区、进入 NBA 年份以及获取总冠军个数） 3.用户点击查看球队阵容信息 4.系统跳转显示球队阵容信息（球员名、球员位置、球员号码、球员身高、体重、年龄、出生日期、NBA 球龄） 5.用户点击查看球队最近比赛 6.系统显示球队当前赛季比赛列表
扩展流程	无
特殊需求	无

表 3.4 查看球员信息用例描述

ID	4
名称	查看球员信息
参与者	用户
触发条件	用户在球员列表中点击查看球员信息
前置条件	无
后置条件	无
正常流程	1.用户在球员列表中选择一位球员查看球员信息 2.系统跳转至球员信息页面，显示球员基本信息（球员名、球员位置、球员号码、球员身高、体重、年龄、出生日期、NBA 球龄）与球员赛季数据统计（首发次数、场均出场时间、场均得分、篮板、助攻、抢断、失误、犯规。。）
扩展流程	无
特殊需求	无

表 3.5 查看数据排行用例描述

ID	5
名称	查看数据排行
参与者	用户
触发条件	用户在比赛列表下点击日期右侧的数据排行
前置条件	无
后置条件	无
正常流程	1.用户在比赛列表下点击日期右侧的数据排行 2.系统跳转至排行界面，显示当前赛季东西部球队排行 3.用户点击选择查看赛季球员数据排行 4.系统显示当前赛季场均得分、篮板、助攻、抢断四项数据统计下的球员排行 5.用户点击选择查看某日数据王 6.系统根据所选日期显示当日得分、篮板、助攻、抢断四项数据统计下的球员排行
扩展流程	无
特殊需求	1.赛季球员数据排行与每日数据王在得分、篮板、助攻、抢断四项数据统计下每项选取排名前五的球员显示 5.日期的格式为（年-月-日），其中年份以四位数字表示，月份以两位数字表示，日以两位数字表示，不足两位的以 0 补足，例：2016-05-02

表 3.6 查看球队赛季交锋数据用例描述

ID	6
名称	查看球队赛季交锋数据
参与者	教练
触发条件	教练在球队信息中选择查看球队交锋数据

前置条件	用户类型为教练
后置条件	无
正常流程	1.教练在球队信息页面下选择查看球队交锋数据 2.系统显示交锋球队列表 3.教练选择球队查看交锋数据 4.系统显示两队赛季交锋比赛列表与球队数据对比
扩展流程	4a：两队暂无交锋数据 1：系统提示两队赛季尚未交锋
特殊需求	无

表 3.7 查看球队球员能力排名用例描述

ID	7
名称	查看球队球员能力排名
参与者	教练
触发条件	教练按得分能力、防守能力等数据指标查看球队球员能力排名
前置条件	用户类型为教练
后置条件	无
正常流程	1.球队教练在教练主界面下查看球队球员能力排名 2.系统默认返回球队球员得分能力排名 3.教练选择其他指标（防守能力、三分能力等）进行查询 4.系统显示该指标下球队球员能力排名
扩展流程	无
特殊需求	1.数据排名根据当前赛季球员数据进行统计排名，如果当前赛季球队比赛数据过少（少于 10 场），则根据球员上赛季数据进行排名

表 3.8 查找比赛用例描述

ID	8
名称	查找比赛
参与者	用户
触发条件	用户在侧栏抽屉中选择查找比赛
前置条件	无
后置条件	无
正常流程	1.用户选择查找比赛 2.系统提示用户选择日期 3.用户选择日期进行查询 4.系统返回该日比赛列表
扩展流程	4a：该日没有比赛数据 1：系统提示该日没有比赛进行 2：返回正常流程第二步
特殊需求	1.日期的格式为（年-月-日），其中年份以四位数字表示，月份以两位数字表示，日以两位数字表示，不足两位的以 0 补足，例：

	2016-05-02

表 3.9 查看联盟球员能力排名用例描述

ID	9
名称	查看联盟球员能力排名
参与者	球探
触发条件	球探检索
前置条件	球探按得分能力、防守能力等数据指标查看联盟球员能力排名
后置条件	无
正常流程	1.球探在球探主界面下查看联盟球员能力排行 2.系统默认返回联盟球员能力排行 3.球探选择其他（防守能力、三分能力等）进行查询 4.系统显示该指标下联盟球员能力排名
扩展流程	无
特殊需求	1.数据排名根据当前赛季球员数据进行统计排名，如果当前赛季球队比赛数据过少（总数少于 150 场，平均每只球队比赛数目小于 10 场），则根据球员上赛季数据进行排名

表 3.10 自定义检索条件筛选联盟球员用例描述

ID	10
名称	自定义检索条件筛选联盟球员
参与者	球探
触发条件	球探选择自定义公式进行查询
前置条件	用户类型为球探
后置条件	无
正常流程	1.球探选择自定义公式进行查询 2.系统提示用户输入自定义公式 3.球探输入自定义公式，选择查询 4.系统解析公式之后，对获取的联盟球员赛季数据进行检索，返回筛选后的球员排名
扩展流程	4a：自定义公式有语法错误 1：系统提示用户更改公式 2：返回至正常流程第三步
特殊需求	无

3.3 NBA 数据分析系统概要设计

3.3.1 服务器架构

服务器整体逻辑视图：

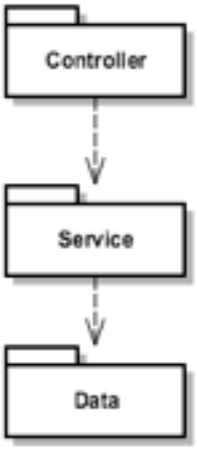

图 3.2 服务器整体逻辑视图

服务器体系结构逻辑设计：

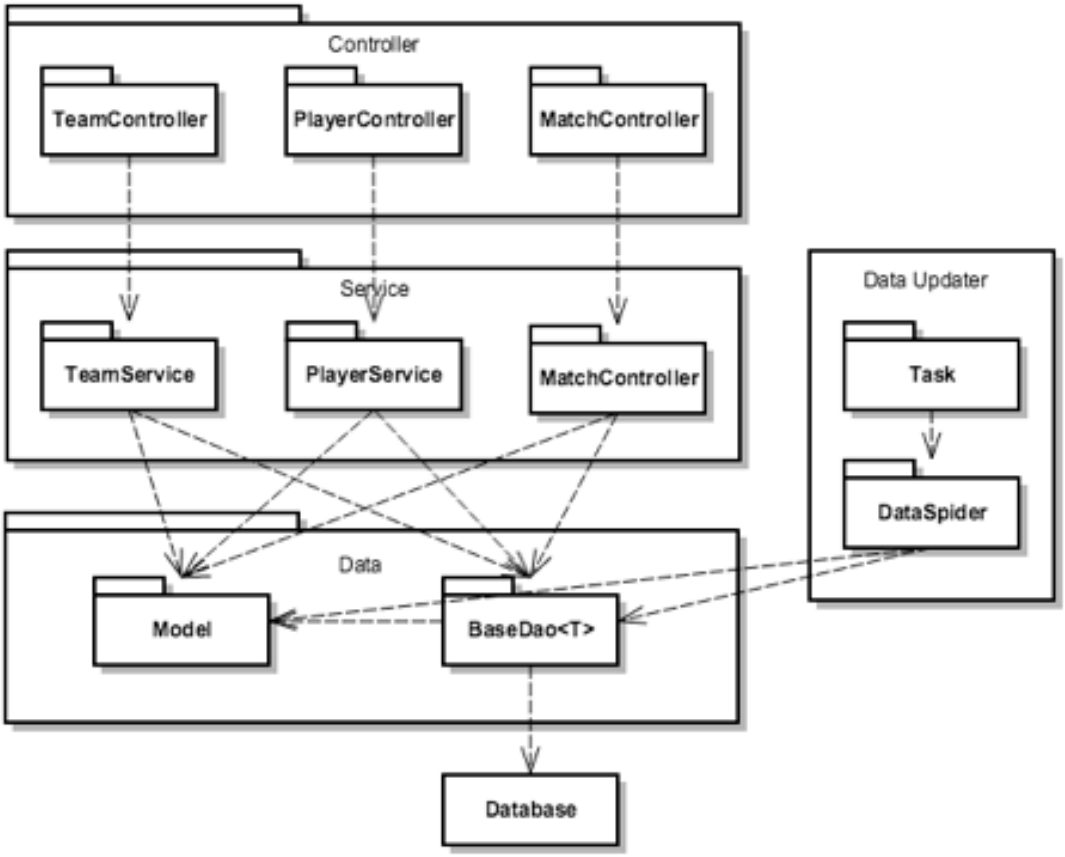

图 3.3 服务器体系结构逻辑设计图

以 Android 客户端发送请求获取某位球员个人信息为例，展示服务器从响应

请求到提供对应 json 数据的过程：

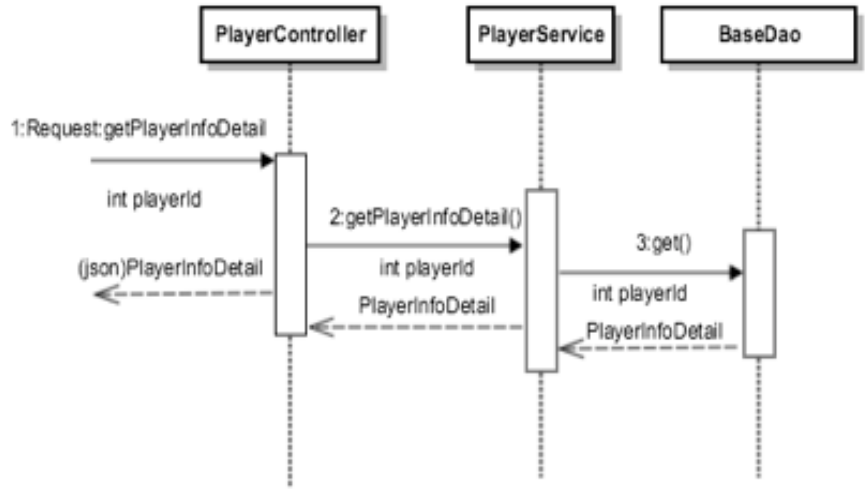

图 3.4 服务器请求响应图

3.3.2 数据库设计

数据库实体关系图：

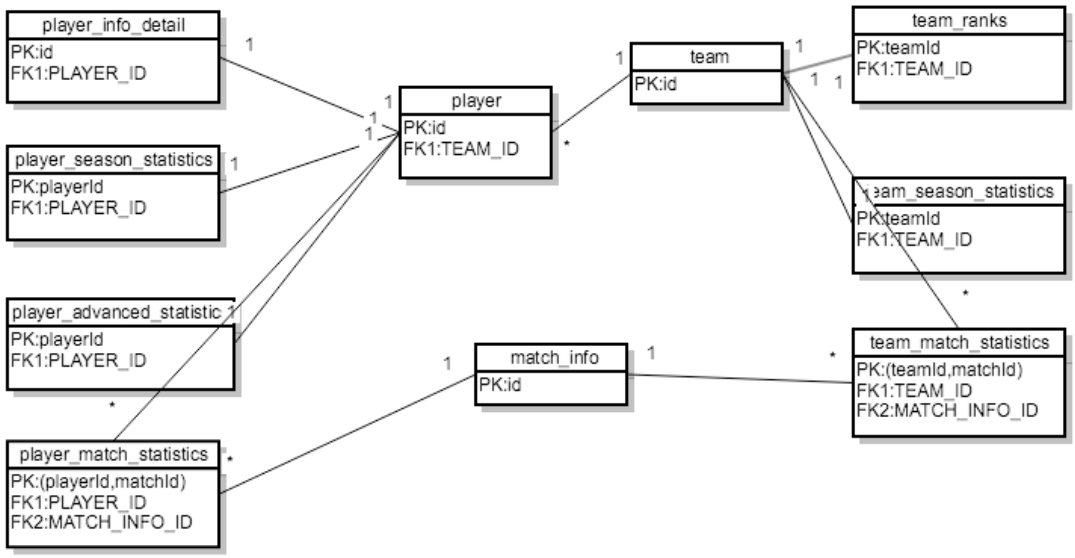

图 3.5 数据库 ER 模型图

NBA 数据分析系统数据库下包含 10 张表：player 表、player_info_detail 表、player_match_statistics 表、player_season_statistics 表、player_advanced_statistics 表、team 表、team_ranks 表、team_match_statistics 表、team_season_statistics 表、

match_info 表。

1：player 表存储联盟球员基本信息，包含球员 Id、球员号码、球员名、球员所处球队 Id、球员位置、球员身高、体重、年龄、出生日期与 NBA 球龄。

2：player_info_detail 表则存储球员详细信息，包括球员出生地、球员毕业学校、球员进入 NBA 年份、球员选秀情况、球员赛季最高分以及生涯最高分。player_info_detail 通过 id 和 player 表建立一对一外键关联映射。

3：player_match_statistics 表存储球员在一场比赛中的基础数据统计，包括球员是否首发、出场时间、总得分、两份出手次数与命中数、三分出手次数与命中数、罚球出手次数与命中数、前场篮板个数、后场篮板个数、篮板总数以及比赛中的助攻、抢断、盖帽、失误、犯规个数，player_match_statistics 通过 playerId 和 player 表建立起多对一的外键关联映射，同时通过 matchId 和 match_info 表建立起多对一外键关联映射。

4：player_season_statistics 表则通过对 player_match_statistics 表中球员比赛数据进行汇总统计之后得出的球员赛季各项基础数据统计，通过 playerId 和 player 表建立一对一的外键关联映射。

5：player_advanced_statistics 表是针对一些高阶数据需求通过对比赛数据的汇总建立的球员赛季高阶数据统计表，包括球员赛季两分命中率、三分命中率、罚球命中率、真实命中率、赛季效率值以及赛季的两双和三双次数。同样，player_advanced_statistics 表通过 playerId 和 player 表建立一对一的外键关联映射。6：team 表存储联盟三十支球队的基本信息，包括球队 Id、球队名、球队所在城市、球队所处联盟（东部或西部）、球队所在分区（东南区、中部区、大西洋区、太平洋区、西南区、西北区）、球队球馆、球队进入 NBA 年份以及球队队史获得的总冠军个数。

7：team_ranks 表存储赛季联盟排名，包含球队胜场数、负场数、球队胜率、球队落后的胜场差以及球队的场均得分和场均失分。team_ranks 表通过 teamId 和 team 表建立起一对一的外键关联映射。

8：team_match_statistics 表存储球队在一场比赛中的数据汇总统计，包括球队主客场情况、球队总得分、两份出手总数与命中总数、三分出手总数与命中总数、罚球出手总数与命中总数、前场篮板总数、后场篮板总数、篮板总数以及比赛中的助攻、抢断、盖帽、失误、犯规总数，team_match_statistics 表通过 teamId 和 team 表建立起多对一的外键关联映射，通过 matchId 和 match_info 表建立起多对一的外键关联映射。9：team_season_statistics 表则通过对 team_match_statistics 表中球队比赛数据进行汇总统计之后得出的球队赛季各项基础数据统计，通过 teamId 和 team 表建立一对一的外键关联映射。

10: match_info 表则存储赛季比赛数据，包含对阵球队名与球队 Id、对阵球

队得分信息、对阵球队主客场信息以及比赛类型（季前赛、季后赛、常规赛、全明星赛）和比赛时间。

3.4 本章小结

本章通过用例图和用例描述对于系统的需求进行了详细的描述，然后阐述了NBA 数据分析系统服务器模块的逻辑设计，和与 Android 客户端交互的过程，最后对于数据库设计进行了一定的叙述。

第四章 NBA 数据分析系统服务器模块的详细设计与实现

NBA 数据分析系统服务器模块整体划分为四个模块：球员模块、球队模块、比赛模块以及负责定时更新数据的数据维护模块。本章主要阐述这四大模块的详细设计与实现。

4.1 球员模块

球员模块主要负责向 Android 客户端提供与球员相关的数据接口，包括球员详细信息、球员赛季数据统计、球员赛季数据排行与每日数据排行、球员进阶数据统计与球员赛季效率值序列。

4.1.1 球员模块对外 API

球员模块为 Android 客户端提供的 API 如下表所示：

表 4.1 球员模块数据接口表

返回值说明	接口方法名称	参数说明
PlayerInfoDetail	getPlayerInfoDetail（获取球员详细信息）	playerId(Int)球员Id
List<PlayerDataRank>	getPlayerRanks(获取本赛季球员数据排行)	
List<PlayerDataRank>	getPlayerRanks(获取某日球员数据排行)	date(String)日期（格式为XXXX-XX-XX）
PlayerSeasonStatistics	getPlayerSeasonStatistics（获取球员赛季数据统计）	playerId(Int)球员Id
List<PlayerSeasonStatistics>	getTotalPlayerSeasonStatistics（获取联盟所有球员赛季数据统计）	
PlayerAdvancedStatistics	getPlayerSeasonAdvancedStatistics（获取球员赛季进阶数据统计）	playerId(Int)球员Id
List<PlayerAdvancedStatistics>	getTotalPlayerSeasonAdvancedStatistics（获取联盟所有球员赛季进阶数据统计）	

List<Integer>	getPlayerSeasonPERValues(获取球员赛季各场比赛效率值)	playerId(Int)球员Id

4.1.2 球员模块返回数据说明

PlayerInfoDetail 类：球员详细信息持久化类，映射数据表 player_info_detail

表 4.2 球员 PlayerInfoDetail 数据说明表

字段	字段类型	字段说明
id	int	球员id
name	String	球员姓名
birthday	String	生日
age	String	球员年龄
birthplace	String	球员出生地
college	String	球员大学
height	String	球员身高
weight	String	球员体重
startInNBA	String	球员进入NBA年份
yearInNBA	String	NBA球龄
draftStatus	String	球员选秀情况
sHS	String	赛季最高分
cHS	String	生涯最高分

PlayerDataRank类：球员数据排名,用于展示球员赛季数据排行与每日数据排行的实体类。Android客户端通过getPlayerRanks请求获取球员数据排行时，如果提供日期参数date，则服务器将对应日期下球员在得分篮板助攻抢断中前五名的数据以data字段封装传递给客户端；如果客户端没有提供date参数，则服务器将得分篮板助攻抢断中前五名的赛季场均数据以seasonData字段封装传递给客户端。

表 4.3 PlayeDataRank 类数据说明表

字段	字段类型	字段说明
id	int	球员id
name	String	球员姓名
teamName	String	球员所在球队名
data	int	球员某日数据
seasonData	double	球员赛季数据
type	int	数据类型（0:得分；1:篮板；2:助攻；3:抢断）

PlayerSeasonStatistics 类：球员赛季数据统计的持久化类，映射数据表

player_season_statistics

表 4.4 PlayerSeasonStatistics 类数据说明表

字段	字段类别	字段说明
playerId	int	球员id
playerName	String	球员名
season	String	赛季描述
teamId	int	球队id
teamName	String	球队名
isFirst	int	首发次数
totMatches	int	出场次数
time	double	场均出场时间
twoHit	double	场均两分命中次数
twoShot	double	场均两份出手次数
threeHit	double	场均三分命中次数
threeShot	double	场均三分出手次数
FreeThrowHit	double	场均罚球命中次数
FreeThrowShot	double	场均罚球出手次数
offReb	double	场均前场篮板个数
defReb	double	场均后场篮板个数
totReb	double	场均篮板总数
ass	double	场均助攻数
steal	double	场均抢断数
blockShot	double	场均盖帽数
turnOver	double	场均失误次数
foul	double	场均犯规次数
score	double	场均得分

PlayerAdvancedStatistics 类：球员赛季进阶数据统计的持久化类，映射数据表 player_advanced_statistics

表 4.5 PlayerAdvancedStatistics 类数据说明表

字段	字段类型	字段说明
playerId	int	球员Id
playerName	String	球员名
season	String	赛季描述
teamId	int	球队id
teamName	String	球队名
isFirst	int	首发次数
totMatches	int	出场次数
time	double	场均出场时间
twoRate	double	赛季两分命中率（百分制，保留两位小数）
threeRate	double	赛季三分命中率（百分

		制，保留两位小数）
freeThrowRate	double	赛季罚球命中率（百分制，保留两位小数）
trueRate	double	赛季真实命中率（百分制，保留两位小数）
PER	double	场均效率值
doubleDouble	int	赛季两双次数
tripleDouble	int	赛季三双次数

4.1.3 球员模块类设计与实现

球员模块被设计成 MVC 的三层架构实现方式，PlayerController 负责球员数据请求的转发，PlayerService 层负责主要的业务逻辑处理，BaseDao 层则负责持久化数据的存取。在接收到 Android 客户端的请求时，PlayerController 将请求映射至对应的方法，在方法下调用注入的 playerService 进行逻辑处理，playerService 调用 BaseDao 下的 get 与 find 方法获取持久化数据，处理之后返回给 playerController。

球员模块的类设计图如下所示：

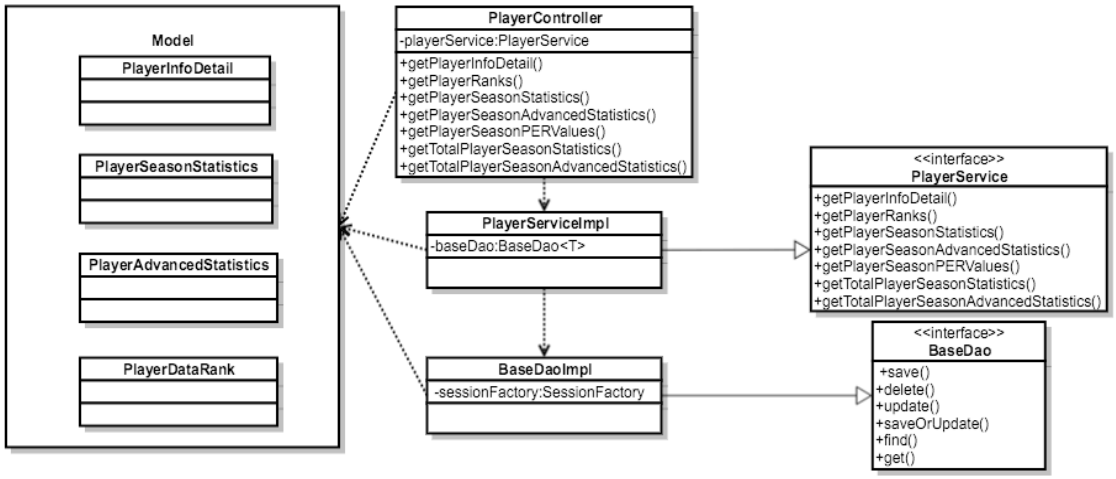

图 4.1 球员模块类设计图

4.2 球队模块

球队模块主要负责向 Android 客户端提供与球队相关的数据接口，包括联盟

球队列表、球队基本信息、球队球员列表、球队赛季排行、球队球员赛季数据统计、球队球员赛季进阶数据统计、球队赛季数据统计、球队与其他球队交锋数据统计和球队与其他球队交锋比赛列表。

4.2.1 球队模块对外 API

球队模块为 Android 客户端提供的 API 如下表所示：

表 4.6 球队模块数据接口表

返回值说明	接口方法名称	参数说明
List<Team>	getTeams（获取联盟球员列表）	
Team	getTeamInfos（获取某支球队基本信息）	teamId(int)球队Id
List<Player>	getTeamPlayerList（获取某支球队球员列表）	teamId(int)球队Id
List<TeamSeasonRank>	getTeamSeasonRanks（获取球队赛季排行列表）	
List<PlayerSeasonStatistics>	getTeamPlayerStatistics（获取球队球员赛季数据统计）	teamId(int)球队Id
List<PlayerAdvancedStatistics>	getTeamPlayerAdvancedStatistics（获取球队球员赛季进阶数据统计）	teamId(int)球队Id
TeamSeasonStatistics	getTeamSeasonStatistics（获取球队赛季比赛数据统计）	teamId(int)球队Id
List<TeamSeasonStatistics>	getTeamVsStatistics（获取两支球队赛季交锋数据统计）	teamId(int)球队Id;vsTeamId(int)交锋球队Id
List<Match>	getTeamVsMatchList（获取两支球队赛季交锋比赛列表）	teamId(int)球队Id;vsTeamId(int)交锋球队Id

4.2.2 球队模块返回数据说明

Player 类：球员基本信息持久化类，映射数据表 player

表 4.7 Player 类数据说明表

字段	字段类型	字段说明
id	int	球员id
num	int	球员球衣号码

teamId	int	球员所属球队id
name	String	球员名
pos	String	球员位置
height	double	球员身高
weight	double	球员体重
age	int	球员年龄
birthday	String	球员出手日期
yearInNBA	int	球员NBA球龄

Team 类：球队基本信息持久化类，映射数据表 team

表 4.8 Team 类数据说明表

字段	字段类型	字段说明
id	int	球队id
name	String	球队名
city	String	球队所在城市
league	String	球队所在联盟（东部或西部）
conference	String	球队所处分区
court	String	球队球馆
startYearInNBA	int	球队进入NBA年份
numOfChampions	int	球队历史总冠军个数

Match 类：比赛基本信息持久化类，映射数据表 match_info

表 4.9 Match 类数据说明表

字段	字段类型	字段说明
id	int	比赛Id
vId	int	客队Id
visitintTeam	String	客队名
hId	int	主队Id
homeTeam	String	主队名
visitingScore	int	客队得分
homeScore	int	主队得分
type	int	比赛类型（0:常规赛；1:季后赛；2:季前赛；3:全明星赛）
season	String	赛季描述
date	String	比赛日期（格式：XX月XX日）
year	int	比赛年份
time	String	比赛时间（格式：XX:XX）

TeamSeasonStatistics 类：球队赛季数据统计持久化类，映射数据表 team_season_statistics

表 4.10 TeamSeasonStatistics 类数据说明表

字段	字段类型	字段说明
teamId	int	球队 Id
teamName	String	球队名
season	String	赛季描述
totalMatches	int	球队赛季比赛场次
twoHit	double	球队场均两分命中次数
twoShot	double	球队场均两份出手次数
threeHit	double	球队场均三分命中次数
threeShot	double	球队场均三分出手次数
FreeThrowHit	double	球队场均罚球命中次数
FreeThrowShot	double	球队场均罚球出手次数
offReb	double	球队场均前场篮板个数
defReb	double	球队场均后场篮板个数
totReb	double	球队场均篮板总数
ass	double	球队场均助攻数
steal	double	球队场均抢断数
blockShot	double	球队场均盖帽数
turnOver	double	球队场均失误次数
foul	double	球队场均犯规次数
score	double	球队场均得分

TeamSeasonRank 类：球队赛季排行持久化类，映射数据表 team_ranks

表 4.11 TeamSeasonRank 类数据说明表

字段	字段类型	字段说明
teamId	int	球队 Id
rank	int	球队排名
name	String	球队名
league	int	球队所处联盟（0:东部；1:西部）
wins	int	赛季胜场
loses	int	赛季负场
winRate	double	赛季胜率
gamesBehind	double	胜场差
pspg	double	赛季场均得分
papg	double	赛季场均失分

PlayerSeasonStatistics 类和 PlayerAdvancedStatistics 类的数据说明参考球员模块下的数据说明。

4.2.3 球队模块类设计与实现

球队模块被设计成 MVC 的三层架构实现方式，TeamController 负责球队数据请求的转发，TeamService 层负责主要的业务逻辑处理，BaseDao 层则负责持久化数据的存取。在接收到 Android 客户端的请求时，TeamController 将请求映射至对应的方法，在方法下调用注入的 teamService 进行逻辑处理， teamService 调用 BaseDao 下的 get 与 find 方法获取持久化数据，处理之后返回给 teamController。

球队模块的类设计图如下所示：

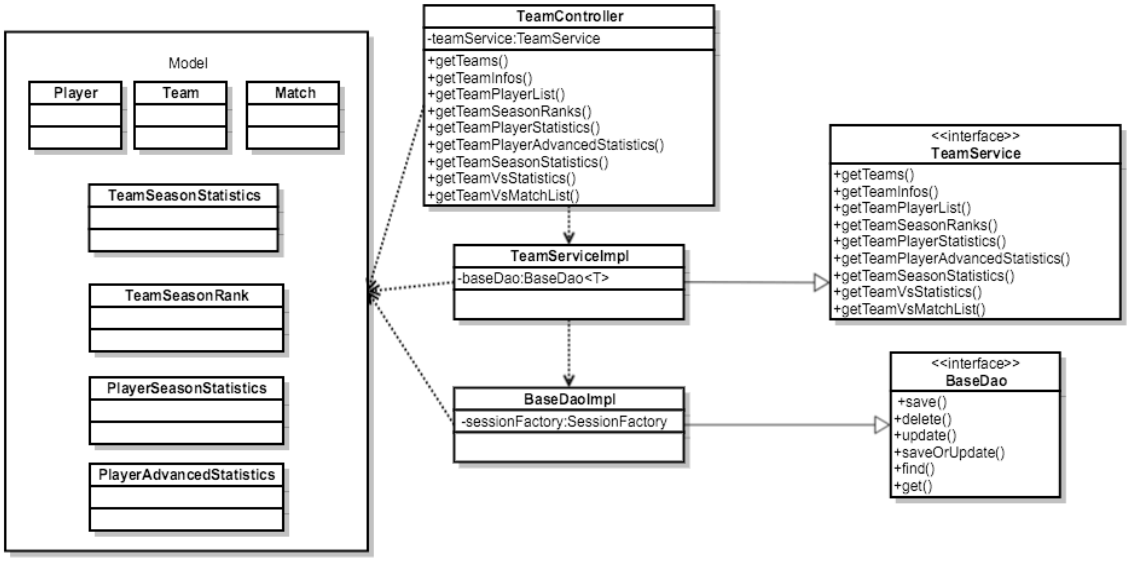

图 4.2 球队模块类设计图

4.3 比赛模块

比赛模块主要负责向 Android 客户端提供与比赛相关的数据接口，包括最近比赛列表、某一日的比赛列表、某支球队当前赛季比赛列表和某场比赛下球员数据统计与球队数据统计。

4.3.1 比赛模块对外 API

比赛模块为 Android 客户端提供的 API 如下表所示

表 4.12 比赛模块数据接口表

返回值说明	接口方法名称	参数说明
List<Match>	getLatestMatchList（获取最近 50 场比赛列表）	
List<Match>	getMatchListOfDay(获取某一日比赛列表）	date(String) 日期（格式为 XXXX-XX-XX）
List<Match>	getTeamMatchList（获取球队赛季比赛列表）	teamId(int)球队 Id
List<PlayerMatchStatistics>	getPlayerMatchStatistics（获取比赛球员数据统计）	matchId(int)比赛 Id
List<TeamMatchStatistics>	getTeamMatchStatistics（获取比赛球队数据统计）	matchId(int)比赛 Id

4.3.2 比赛模块返回数据说明

Match 类的数据说明参考球队模块下对 Match 的数据说明。

PlayerMatchStatistics 类：球员比赛数据统计的持久化类，映射数据表 player_match_statistics

表 4.13 PlayerMatchStatistics 类数据说明表

字段	字段类别	字段说明
matchId	int	比赛Id
playerId	int	球员id
playerName	String	球员名
teamId	int	球队id
isFirst	int	是否首发（0:首发；1:替补）
time	int	出场时间
twoHit	int	两分命中次数
twoShot	int	两份出手次数
threeHit	int	三分命中次数
threeShot	int	三分出手次数
FreeThrowHit	int	罚球命中次数
FreeThrowShot	int	罚球出手次数
offReb	int	前场篮板个数
defReb	int	后场篮板个数
totReb	int	篮板总数
ass	int	助攻数
steal	int	抢断数
blockShot	int	盖帽数
turnOver	int	失误次数

foul	int	犯规次数
score	int	得分

TeamMatchStatistics 类：球队比赛数据统计的持久化类，映射数据表 team_match_statistics

表 4.14 TeamMatchStatistics 类数据说明表

字段	字段类型	字段说明
matchId	int	比赛 Id
teamId	int	球队 Id
ifHome	int	是否主场（0:客场；1:主场）
time	int	球队上场总时间
twoHit	int	球队两分命中总数
twoShot	int	球队两分出手总数
threeHit	int	球队三分命中总数
threeShot	int	球队三分出手总数
FreeThrowHit	int	球队罚球命中总数
FreeThrowShot	int	球队罚球出手总数
offReb	int	球队前场篮板总数
defReb	int	球队后场篮板总数
totReb	int	球队篮板总数
ass	int	球队助攻总数
steal	int	球队抢断总数
blockShot	int	球队盖帽总数
turnOver	int	球队失误总数
foul	int	球队犯规总数
score	int	球队总得分

4.3.2 比赛模块类设计图与实现

比赛模块被设计成 MVC 的三层架构实现方式，MatchController 负责比赛数据请求的转发，MatchService 层负责主要的业务逻辑处理，BaseDao 层则负责持久化数据的存取。在接收到 Android 客户端的请求时，MatchController 将请求映射至对应的方法，在方法下调用注入的 matchService 进行逻辑处理，matchService 调用 BaseDao 下的 get 与 find 方法获取持久化数据，处理之后返回给 matchController。

比赛模块的类设计图如下所示：

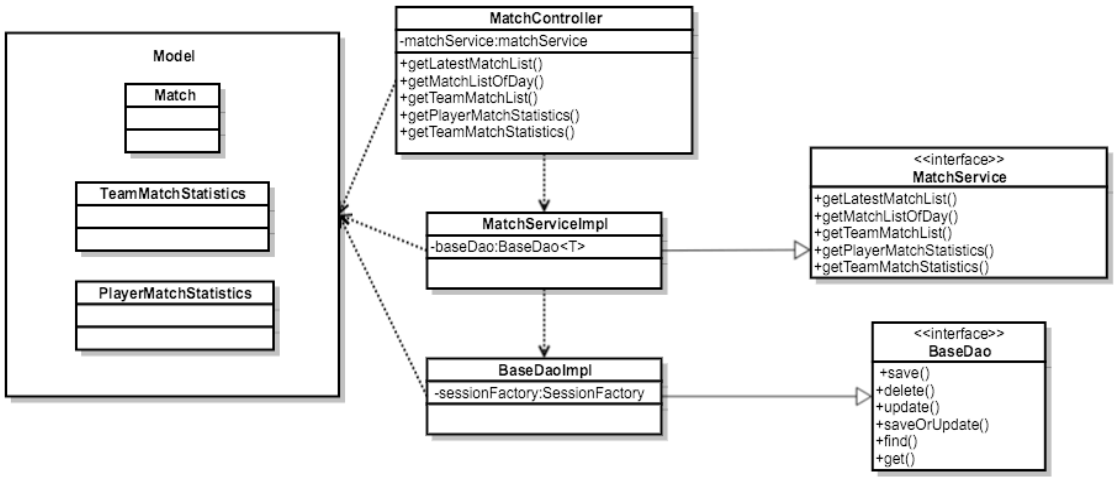

图 4.3 比赛模块类设计图

4.4 数据维护模块

NBA 数据分析系统需要即时地获取 NBA 最新的球队、球员与比赛动态。为此需要设置定时任务定时从网页获取最新的比赛数据（比赛列表、球队比赛数据统计以及球员比赛数据统计）并对球队的赛季排行、球队阵容信息和球员个人信息作出定期的更新，在对比赛数据统计作出数据更新之后，还需要对其进行处理更新球队与球员的赛季数据统计。定时任务的设计与网站数据的爬取是本系统详细设计的一大重点。

4.4.1 数据维护模块类设计与实现

在数据维护模块下，Task 负责定时执行数据更新任务，DataSpider 负责从网站爬取 NBA 数据（本系统选择的是新浪 NBA 数据库），BaseDao 负责持久化数据的存储与更新。在定时任务启动之后，Task 调用 DataSpider 从网站获取数据并封装到 model 中，然后再调用 BaseDao 完成数据的存储或更新。以球员数据的维护为例，数据维护模块下类的设计图如下所示：

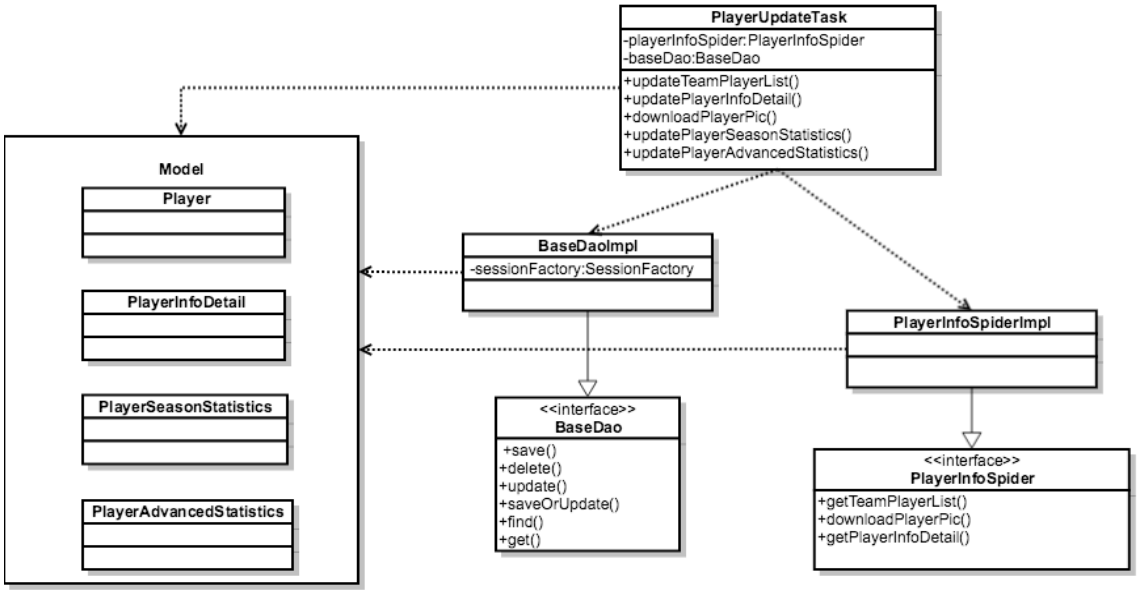

图 4.4 球员数据维护模块类设计图

4.4.2 数据爬取

数据爬取采用方法是将网页源码逐行读入至 StringBuffer 中（去掉换行符避免换行符对正则表达式的影响），然后根据正则表达式去匹配获取网页中所需的数据项。

```
        String urlStr = DataSourceUrl.getMatchListURL(year, month);
        StringBuffer                webPageBuffer                =
WebPageReader.readWebPage(urlStr);
        pattern = Pattern
                .compile("(.*)(<!-- 赛程内容开始 -->)(.*?)(<!-- 赛
程内容end-->)(.*)");
        matcher = pattern.matcher(webPageBuffer);
        if (matcher.matches()) {
            String str1 = matcher.group(3);
            pattern = Pattern.compile("(<tr>)(.*?)(</tr>)");
            matcher = pattern.matcher(str1);
            String date = "";
            while (matcher.find()) {
            String str2 = matcher.group(2);
```

图 4.5 数据爬取实例代码图

以上这段代码是爬取某月赛程列表的部分代码，webPageBuffer 存储某月比

赛列表的静态页面，在页面中，赛程列表的内容包含在两句注释：<!— 赛程内容开始 -->与<!-- 赛程内容 end—>之间，对静态页面进行相应的匹配，就能够通过 matcher.group(3)获取注释之间的比赛列表的内容。而由于在页面中，比赛列表数据是以表单的形式呈现的，所以在 str1 中循环匹配(<tr>)(.*?)(</tr>)，就能依次获取各场比赛的记录。需要注意的是，*限定符是贪婪的，它会尽可能多的匹配文字，所以要在其后加上？来实现非贪婪或最小匹配。

4.4.3 定时任务设置

由于在数据之间存在的层级关系，比赛数据的更新会导致球队比赛数据与球员比赛数据的更新需求，进而会导致球队赛季数据统计与球员赛季数据统计的更新需求，所以在设置定时任务时，比赛相关数据的维护是以比赛表为基础的。在比赛列表更新之后，负责更新球队比赛数据、球员比赛数据的定时任务会通过数据对比识别出最新的比赛信息，再依据比赛信息相应的获取比赛中球员与球队的统计数据。

更新球队比赛数据统计定时任务代码如下所示：

```
    /**
     * 定时更新球队比赛信息
     * 更新时间：在赛季期间（每年的10月至12月，一月至6月），每天0时至14时（由于在北京时间14点之后极少有比赛进行），从5分开始每20分钟更新一次
     */
    @Scheduled(cron = "0 5/20 0-14 * 1-6,10-12 ?")
    public void updateTeamMatchList(){
        TeamMatchStatistics lastTeamMatchStatistics=teamMatchStatisticsDao.find("from TeamMatchStatistics where matchId = (select max(matchId) from TeamMatchStatistics)").get(0);
        int lastMatchId=lastTeamMatchStatistics.getMatchId();
        List<Match> matchList=matchDao.find("from Match");
        List<Integer> matchIdList=new ArrayList<Integer>();
        for (Match match : matchList) {
            if(match.getId()>lastMatchId){
                matchIdList.add(match.getId());
            }
        }
        List<TeamMatchStatistics> updateList=matchInfoSpider.getTeamMatchStatistics(matchIdList);
        if(updateList.size()>0){
            for (TeamMatchStatistics statistics : updateList) {
                teamMatchStatisticsDao.saveOrUpdate(statistics);
```

```
            }
            String info="Update TeamMatchStatictics Success! Add
"+updateList.size()+" Records.";
            MyLog.log(info);
        }
    }
```

图 4.6 更新球队比赛数据统计定时任务代码图

更新球员比赛数据统计的定时任务如下所示：

```
    /**
     * 定时更新球员比赛数据列表
     * 在赛季期间（每年的10月至12月，一月至6月），每天0时至14时（由于在北京时间
14点之后极少有比赛进行），从5分开始，每20分钟更新一次
     */
    @Scheduled(cron = "0 5/20 0-14 * 1-6,10-12 ?")
    public void updatePlayerMatchStatistics(){
        PlayerMatchStatistics
lastPlayerMatchStatistics=playerMatchStatisticsDao.find("from
PlayerMatchStatistics where matchId = (select max(matchId) from
PlayerMatchStatistics)").get(0);
        int lastMatchId=lastPlayerMatchStatistics.getMatchId();
        List<Match> matchList=matchDao.find("from Match");
        List<Integer> matchIdList=new ArrayList<Integer>();
        for (Match match : matchList) {
            if(match.getId()>lastMatchId){
                matchIdList.add(match.getId());
            }
        }
        List<PlayerMatchStatistics>
updateList=matchInfoSpider.getPlayerMatchStatistics(matchIdList);
        if(updateList.size()>0){
            for (PlayerMatchStatistics statistics : updateList) {
                playerMatchStatisticsDao.saveOrUpdate(statistics);
            }
            String info="Update PlayerMatchStatictics Success! Add
"+updateList.size()+" Records.";
            MyLog.log(info);
        }

    }
```

图 4.7 更新球员比赛数据统计定时任务代码图

PlayerSeasonStatistics 、PlayerAdvancedStatitics 与 TeamSeasonStatistics 是对 PlayerMatchStatistics 表和 TeamMatchStatistics 表中数据统计处理后得到的数据

表，因此在定时任务中直接从数据库获取比赛数据列表对这三张表进行更新。以球员赛季数据统计表（PlayerSeasonStatistics）为例，由于赛季数据统计需要获取球员赛季所有比赛数据统计，因此在 PlayerSeasonStatistics 类中设置构造方法，接收 List<PlayerMatchStatistics>进行数据统计：

```
   public        PlayerSeasonStatistics(Player        player,Team
team,List<PlayerMatchStatistics> dataList,String season){
       super();
       this.season=season;
       this.playerId=player.getId();
       this.playerName=player.getName();
       this.teamId=team.getId();
       this.teamName=team.getName();
       int totMatches=dataList.size();
       this.totalMatches=totMatches;
       int                        isFirstTot                        =
0,timeTot=0,twoHitTot=0,twoShotTot=0,threeHitTot=0,threeShotTot=0
;
       int
freeThrowHitTot=0,freeThrowShotTot=0,offRebTot=0,defRebTot=0,totR
ebTot=0;
       int
assTot=0,stealTot=0,blockShotTot=0,turnOverTot=0,foulTot=0,scoreT
ot=0;
       for (PlayerMatchStatistics s : dataList) {
          if(s.getIsFirst()==0){//首发次数
             isFirstTot++;
          }
          /*
           * 统计赛季各项数据总值
           */
       }
       /*
        * 根据赛季数据总值和赛季上场次数统计赛季场均数据
        */

   }
```

图 4.8 PlayerSeasonStatistics 构造函数代码图

更新球员赛季数据统计的定时任务如下所示：

```
   /**
    * 定时更新球员赛季数据统计
    * 在赛季期间（每年的10月至12月，一月至6月），每天0时至14时（由于在北京时间
```

```
14点之后极少有比赛进行），从10分开始，每20分钟更新一次
     */
    @Scheduled(cron = "0 10/20 0-14 * 1-6,10-12 ?")
    public void updatePlayerSeasonStatistics(){
        List<PlayerSeasonStatistics>                          list=new
ArrayList<PlayerSeasonStatistics>();
        List<Player> players=playerDao.find("from Player");
        for (Player player : players) {//更新每位球员赛季数据统计
            /*
             * 获得球员赛季常规赛比赛统计数据列表
             */
            Team team=teamDao.get(Team.class, player.getTeamId());
            List<PlayerMatchStatistics>
dataList=playerService.getPlayerMatchStatistics(player.getId(),
CommonDataManager.SEASON);
            int totMatches=dataList.size();
            if(totMatches==0){
                continue;//跳过未上场球员
            }
            PlayerSeasonStatistics                        statistics=new
PlayerSeasonStatistics(player,team,dataList,CommonDataManager.SEA
SON);
            list.add(statistics);
        }
        for (PlayerSeasonStatistics pss : list) {
            playerSeasonStatisticsDao.saveOrUpdate(pss);
        }
        String info="Update PlayerSeasonStatictics Success! ";
        MyLog.log(info);
    }
```

图 4.9 更新球员赛季数据统计定时任务代码图

4.5 本章小结

本章主要阐述了 NBA 数据分析系统的详细设计与实现。对于球员、球队和比赛模块，主要阐述了模块对 Android 客户端提供的 API 与返回数据，并辅助以模块的类设计图；对于数据维护模块，则从模块的类设计图、数据爬取与定时任务的设计角度对其进行了阐释。

第五章 总结与展望

5.1 总结

本文通过项目的需求分析与概要设计、项目主要模块的详细设计与实现的阐述，介绍了 NBA 数据分析系统的实现，并对项目中所使用的关键技术进行了简要的介绍。

需求分析通过用例图和用例描述两个方面，分析了 NBA 数据分析系统的用户需求；概要设计主要阐述了服务器模块的逻辑设计和与 Android 客户端交互的过程，并对数据库设计进行了一定的叙述；详细设计则对球员、球队和比赛模块对外 API 和返回数据进行了详细的说明，并辅助以模块的类设计图来阐述模块的实现，并对数据维护模块下定时任务的设计与数据爬取的实现进行了一定的叙述。

5.2 展望

随着大数据时代的降临，NBA 应用对于数据的分析会越来越详尽和具体，更多富有意义的统计指标会帮助球迷、球探和教练更多的满足自身的需求，越来越多数据因子的加入能够帮助我们在观看比赛的过程中更好地把握比赛的动态和前景。因此，可拓展性对于 NBA 数据分析系统显得极为重要，同时为了应对更多的数据访问需求，对服务器进行更好的负载均衡优化与查询优化也是 NBA 数据分析系统在以后所需要做的。

参考文献

[1] NBA 百度百科介绍，http://baike.baidu.com/link?url=Zd_MCcJkf2BPYHoyjslxFNciTjNWLqvQV-uswM1BiOTZcGBp7MiNBKQQ--VT7Ld2tP1KlkmlwK5B6pq0uTa5jK

[2] Spring MVC 简介，http://www.cnblogs.com/wawlian/archive/2012/11/17/2775435.html

[3] 基于注解的 Spring MVC 介绍，http://haohaoxuexi.iteye.com/blog/1343761/

[4] Spring 依赖注入：注解注入总结，http://outofmemory.cn/code-snippet/3670/spring-inject-by-annotation

[5] Hibernate 百度百科，http://baike.baidu.com/link?url=q3F5dN34qm2v6zGsjjBovtawGkqPoMI9XZ4nz3-urpj9-UVn0KvNczC454yHN82Qz_YeXdzFKXgHCXjRV-rb5_zt8ltAFDwhSYdjarHcTfi

[6] Hibernate 核心组件，http://blog.csdn.net/tkj510510/article/details/9077149

[7] Hibernate 复合主键，http://blog.163.com/ljxe_mail/blog/static/157253232010102463327856/

[8] Hibernate 注解关系映射，http://blog.163.com/wzx_dd/blog/static/194285072201282095713850/

[9] Java 正则表达式，http://www.runoob.com/java/java-regular-expressions.html

[10]cron 表达式，http://www.cnblogs.com/linjiqin/archive/2013/07/08/3178452.html

致谢

本篇论文的工作是在刘钦老师的指导下完成的，刘钦老师在每周会定时对我们的项目进展进行询问并指出项目当前阶段存在的不足之处以及下个阶段所需解决的问题。刘老师严谨的治学态度和科学的工作方法，对于我们有着极大的帮助，在此对刘老师几个月以来的指导表示衷心的感谢。

其次要感谢 NBA 数据分析系统项目组所有的组员，大家在项目的开发过程中一起对系统实现的许多细节进行了商讨，在他们的帮助下，我顺利地完成了自己的任务和工作。

最后感谢的我的家人和朋友，在你们的理解与支持下，我才顺利完成了项目的开发与论文的撰写。