基于JavaEE和Android的大学生成长档案的设计

摘要

# 第一章 绪论

## 1.1论文研究背景及意义

### 1.1.1 研究背景

自1999年中国高考扩招以来，我国高校不同程度地都在不断扩大招生和办学规模，所以伴随着的是大学生数量的增多，在高等教育大众化的趋势下，大学生群体带给高校对大学生思想政治教育工作以及就业工作的压力也逐步增大。而就高校大学生本身而言，他们在上大学会所经历的迷茫、无助也并非少数现象。

当前，我们已经进入移动互联的新时代，手机特别是APP的使用日益普遍，并且产生重要的社会影响。同时，大学生是APP的活跃用户，就APP的使用时间来说，据调查显示六成以上大学生日均使用APP在2小时以上，3小时以上受访者的占比近四成;而使用时长小于30分钟的不足5%(仅为3.57%)。总的来说，APP已成为大学生每天花费大量时间使用的新媒介技术应用。

因此，通过App为媒介，建立大学生的成长档案，

### 1.1.2 研究目的及意义

该APP开发面对的客户群体正是4018万的在校大学生，该APP给大学生们在心理健康状况、学习成绩、综合测评成绩、课余文化活动、科研创新成绩、社会实践、志愿服务等方面的大学生活的评估与排名也必将传导积极的正能量，评估报告中附带的霍兰德职业兴趣分析也能够在用户在就业选择方面给出有效意见。

## 1.2 论文主要工作

## 1.3 论文主要结构

# 第二章 系统总体设计

## 2.1 系统架构设计

本系统基于Android和JAVA EE的架构的大学生成长档案的设计，是一个完整的java EE 项目，符合java 服务端的设计规范，采用主流的分层方法。服务端分为视图层、数据传输层、业务逻辑层、数据访问层和数据持久化层。

1. 视图层由android端进行展现，android端不做复杂的业务逻辑处理，只是负责和服务端的数据交互并展现。Android端使用httpClient向服务端发送进行网络请求，返回Json数据，实现客户端和服务器端的网络通信。
2. 数据传输层即为控制层，为Spring MVC中的c（Controller）通过客户端发送请求，接受请求并决定调用哪个模型构建去处理请求，请求通过调用业务逻辑层的处理返回数据，最后控制器用相应的请求接口返回给android端进行展示，进行数据交互，呈现给用户。
3. 业务逻辑层即为Service层。主要功能为根据具体的业务需求，进行业务逻辑处理，调用dao对象的相关方法，将dao层的数据转为dto对象，返回给Controller层。
4. 数据访问层即为Dao层，主要功能为封装数据的增加、删除、更新、删除方法，负责和持久化对象交互。
5. 数据持久化层即为po层。应用Hibernate的O/R Mapping（实体关系映射）映射框架，将mysql关系数据库的数据映射为相应的对象，以面向对象的思想和方法，操作数据库。

## 2.2 数据库设计

本系统基于Android和JAVA EE的架构的大学生成长档案的设计，数据库采用关系型数据库Mysql。

## 2.3 系统开发环境

本系统基于Android和JAVA EE的架构的大学生成长档案的设计，分为客户端和服务端两部分。

客户端采用java原生开发，基于Android 6.0版本，所用的集成开发工具为Android studio，应用Gradle 进行项目构建。

服务端采用java语言，基于Spring + Spring MVC + Hibernate三大框架，所用的集成开发工具为IntelliJ IDEA，应用maven进行项目构建。

数据库为MySQL Server 5.7。

Jdk版本为jdk1.8.0\_20。

Web容器为apache-tomcat-8.0.39。

# 第三章 Android客户端设计

# 第四章 JAVA EE服务端设计

本文所述基于Java EE和Android的大学生成长档案的设计，服务端主要负责了全部的业务逻辑。服务器端设计符合MVC设计思想。系统架构为当下流行的SSH 的轻量级JAVA EE 架构，即为Spring ,Spring MVC , Hibernate，所使用的的数据库为Mysql，web容器为Tomact。

## 4.1 java EE 框架介绍

本平台开发采用的是SSH整合技术，SSH指Spring，Spring MVC ,Hibernate三大框架的整合。

### 4.1.1 Spring

Spring的基于控制反转（IOC）的核心机制，基于面向切面编程（AOP）实务管理，整合多种持久层技术（Hibernate），整合MVC框架（Spring MVC）等。Spring贯穿于项目中的持久层，业务层和表现层等。是java应用的整体解决方案。有以下优点：

1、用Spring的IOC容器，将对象之间的依赖关系交给Spring，降低组件之间的耦合性，让我们更专注于应用逻辑

2、对第三方主流持久层框架很好的集成支持，简化底层数据库的访问流程。如Hibernate ,JPA, Mybatis等

3、.Spring的高度可开放性，并不强制依赖于Spring，开发者可以自由选择Spring部分或全部

### 4.1.2 Spring MVC

Spring MVC 是当今最主流的web mvc 框架，提供了MVC(模型 - 视图 - 控制器)架构和用于开发灵活和松散耦合的Web应用程序的组件。 MVC模式导致应用程序的不同方面(输入逻辑，业务逻辑和UI逻辑)分离，同时提供这些元素之间的松散耦合。

模型（Model）：封装了应用程序数据，由POJO类组成。

视图(View)：负责渲染模型数据，由浏览器，手机端等展示。

控制器(**Controller**)：负责处理用户请求并构建适当的模型，并将其传递给视图进行渲染。

### 4.1.3 Hibernate

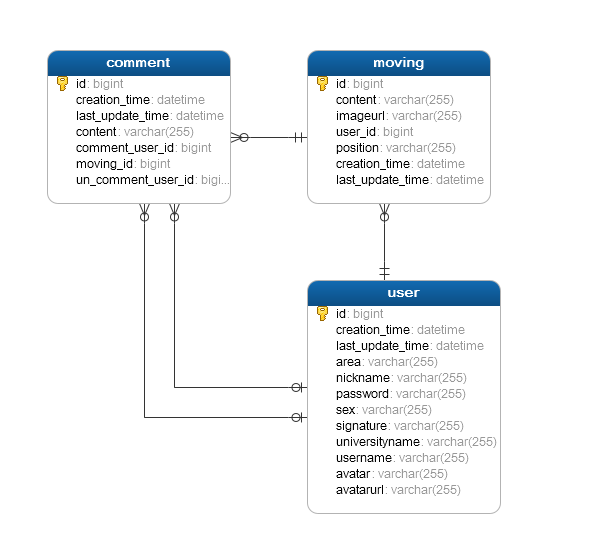
目前主流的数据库是关系型的数据库，比如：Mysql,Oracle。二维表结构易于理解；使用sql语言操作方便；丰富的完整性减小数据冗余，易于维护；支持表关联，实现复杂的查询。目前主流的编程语言是面向对象型的编程语言，比如：JAVA。面向对象建模和操作；继承和多态；相对面向过程容易理解，简单易用等。所以当我们使用面向对象的编程语言进行开发时，也是基于对象的分析，设计和开发，但是当实现数据持久化，进行数据库访问时，又不得不采用关系型数据库的访问方式，因此，编程语言和数据库之间具有设计模式上的区别和技术上的差异，发展不协调。ORM（**Object Relational Mapping**）框架应运而生，可以将关系型数据库转为面向对象模型，方便开发和维护。

Hibernate是典型的java EE持久层的解决方案，也是流行的ORM框架之一。Hibernate是对JDBC的进一步封装，从对象（Object）映射到关系(Relation)，再从关系映射到对象。这样在操作数据库的时候，不需要和复杂的SQL打交道，只需要操作po（persistant object），实现对数据表中的数据进行增，删，改，查等操作。

## 4.2 服务器端实现

### 4.2.1 数据持久层设计和实现

本应用总体的数据库结构设计如图：



在持久层，应用到了JPA（java持久化API）技术，数据库中的每张表对应一个entity（实体）。总的来说jpa包含以下三方面的技术：

1、ORM映射元数据，元数据描述对象和表之间的映射关系，框架将此实体对象持久化到数据库表中。

2、JPA的API，用来操作实体对象，执行CRUP（Create，Retrieve，Update，Delete）操作，框架在后台完成所有的事情，开发者从繁琐的JDBC和SQL代码中解脱出来。

3、查询语言，这是持久化操作中很重要的一个方面，通过面向对象而非面向数据库的查询语句查询数据，避免程序的SQL语句紧密耦合。

本文以动态表为例，来说明持久层实现步骤：



该例子只展现了属性值，省略了相应的getter/setter方法,由IDE自动生成。下面解释相应Annotation（注解)的含义。

1. @Entity标记该bean为实体bean
2. @Table(name = "MOVING")，映射到数据库表名为MOVING。
3. @Id标识主键，@GeneratedValue()标识主键的生成策略，默认为自动增长。
4. @Column，标识映射到数据库表中的字段名称。
5. @ManyToOne，描述多对一的关系，一个用户可以发表多条动态，所以多条动态可以对应一个用户。
6. @OneToMany，描述一对多的关系，一条动态对应了多条评论内容。

对于数据库表，比如，user表会记录用户注册时间，moving表记录动态发表时间，activity表记录活动创建时间。他们都有一个公共的属性createdTime。所以建立一个entity的基类BaseEntity，有两个属性creationTime和lastUpdateTime。所有实体都继承自BaseEntity。

如图所示：



### 4.2.2 数据访问层设计和实现

在Hibernate持久层的上层，一般应用DAO（Data Access Object）组件将一些数据库的操作进行进一步的封装，这就是JAVA EE中常用的DAO模式。

DAO 模式实现需要三部分组成：DAO 工厂类（应用工厂模式）、DAO 接口类及其对应的接口实现类。应用 DAO 模式则每个 DAO 组件可以完成对某个数据库表的增删改查操作，且包含数据库访问逻辑。

DAO组件提供各类持久化对象基本增删改查操作，有以下的基本的用法：persist（插入和更新），remove（删除）, findById（根据id单个查找），createQuery（jpql语句查询）。通过实现EntityManager接口，在业务逻辑层调用相应的CRUP

。方法实现对数据库的操作。

同样的，对于一些共用的方法我们抽离出来一个接口类，如下：



再写一个BaseDao的类实现IBaseDao，实现具体的逻辑。以persist和findById方法为例。



对于复杂的查询，使用jpql语句，进行查询。比如查询所有动态：

 数据访问层的实现，是通过结合BaseDao和jpql实现对数据的操作。

### 4.2.3 业务逻辑层设计和实现

业务逻辑层是数据访问层的门面，也就是业务逻辑层需要DAO层组件的支持。本层和DAO层的分离，保证了底层数据和业务逻辑相互透明，有利于复用性和可扩展性。业务逻辑层通过@Service注解，标注业务层组件。下面，以获取所有动态为例：



通过调用MovingDao类中的listMoving(int start,int size)方法，返回一个Moving实体的集合。再根据具体的业务需要，进行逻辑处理。

### 4.2.4 数据传输层设计和实现

在控制层暴露接口，供Android端调用，@Controller注解，标注为控制层组件。@ResponseBody注解，标注返回数据格式为json数据。@RequestMapping("/moving")注解，标注url路径。还是以获取所有动态为例：



通过url地址（ip 地址+ 端口号 + /moving/allMoving）,通过post请求，传入两个参数，offset和size，调用MovingService的方法，返回MovingDto的集合。在业务逻辑层将实体的集合转为DTO(Data Transfer Object)的集合，返回给Android端进行数据展示。

在数据传输层，通过DTO来进行服务器端和客户端的数据交互，通过比较moving的实体和moving的dto，不难发现，MovingDto展现的数据都是view层直接显示的数据格式，不需要在进行额外的处理。使用DTO的好处就是，按需组织DTO对象，view层需要的字段我才组织，不需要的我不组织，可以避免传输整个表的字段，一定程度上提高了安全性。下面是MovingDto的设计：



# 第五章 系统测试验证

## 5.1 环境测试

## 5.2功能测试

## 5.2测试总结

# 第六章 总结与展望

## 6.1论文总结

## 6.2未来工作展望