

# 软件设计文档

SD

团队名称：起来嗨

注：此项目采用潘茂林老师【系统分析与设计】课程中的同名团队项目，成员信息一致。

| 小组成员 | 学号       |
|------|----------|
| 陈业成  | 12330046 |
| 邓立   | 12330065 |
| 邓祺晟  | 12330067 |
| 徐欣   | 12330349 |
| 张佳宁  | 12353257 |
| 陈浩川  | 12353008 |

## 文件修改记录

| 修改日期       | 版本    | 修改页码、章节、条款 | 修改描述 | 作者  |
|------------|-------|------------|------|-----|
| 2015-07-17 | V1.00 | 安卓方面       | 安卓   | 邓立  |
| 2015-07-18 | V2.00 | 后台方面       | 后台   | 张佳宁 |
|            |       |            |      |     |
|            |       |            |      |     |
|            |       |            |      |     |
|            |       |            |      |     |

# 目 录

|                 |    |
|-----------------|----|
| 一、 技术选型及理由..... | 3  |
| 二、 架构设计 .....   | 4  |
| 三、 模块划分 .....   | 7  |
| 四、 软件设计技术.....  | 15 |
| 五、UML 图.....    | 17 |
| 六、测试用例.....     | 19 |

## 一、 技术选型及理由

### 客户端：

我们的项目客户端选择运行在 android 4.1 系统上，因为随着 android 的逐步发展，android 2.3 使用量已经逐步压缩，android 已进入 4.X 版本的稳定时代，同时最近 5.X 版本亦已出现，但使用量并没明显增多，因此我们选择了大热的 4.1 版本，此版本亦可稳定运行在比 4.1 更高版本的系统上。

选择在安卓系统上开发，第一个是技术上，人员大都选修过安卓应用开发的课程，对 android 应用开发技术较为熟悉，同时开发经验亦相对丰富；第二点是成本上，由于 android 的开源以及互联网各大技术网站对 android 开发的支持都较为成熟，拥有大量现成的库以及学习 demo，遇到问题基本能得到解决，节省了大量的开发时间；第三点是用户方面，由于 android 的流行以及智能手机的普及，android 端可以说是一个不错的选择。

### 服务器：

软件服务端采用SSH 架构，数据库采用MySQL，服务器采用Linux（搭建在阿里云），Web 服务器采用Tomcat。

服务端的SSH 构架采用轻量级J2EE 应用标准的结构——领域模型层、数据访问层、业务逻辑层、控制器层、表现层。

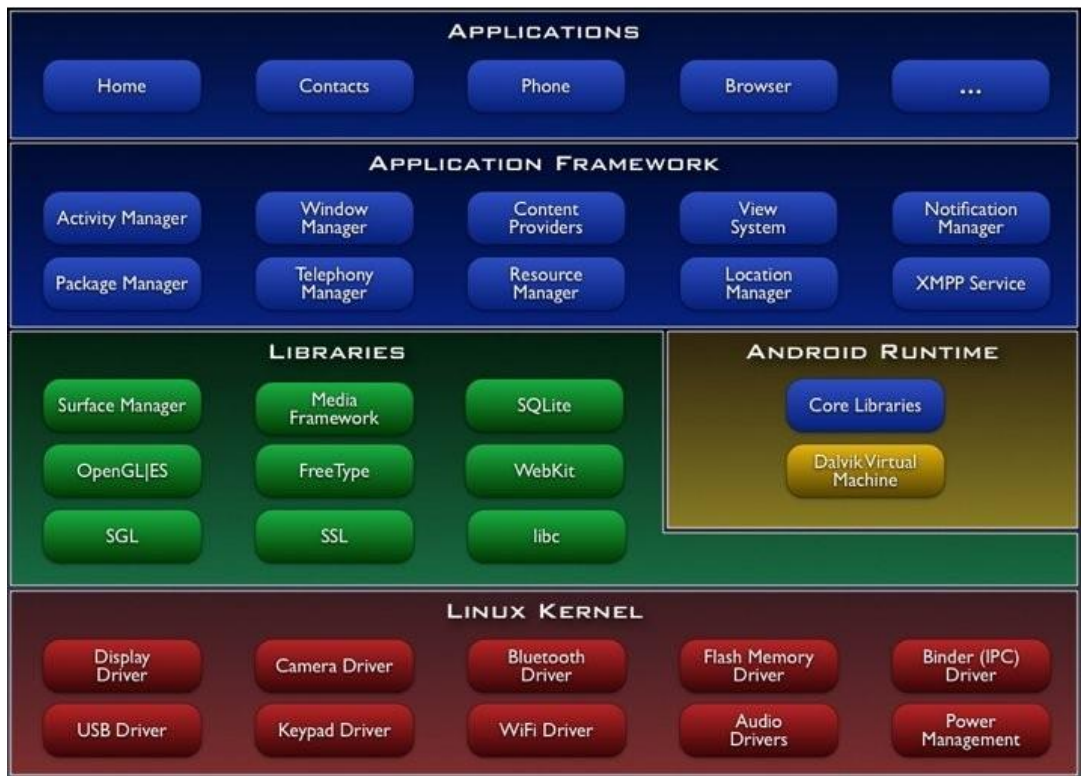
在完整的服务端程序之上，提供各种各样的服务，通过对这些服务的组合，完成产品的各项功能。

数据库采用MySQL 5.6（GPL）版本，原因是免费高效易于使用，开发者对此也相对熟悉，并且有大量文档可供参阅。

开发架构使用 Struts2+Spring4+Hibernate4，这一 SSH 架构。SSH 的架构特点是完全的面向对象，层次分明、完全解耦，简化了数据库的操作。对于维护和学习 OOA、OOD 具有较好的帮助。

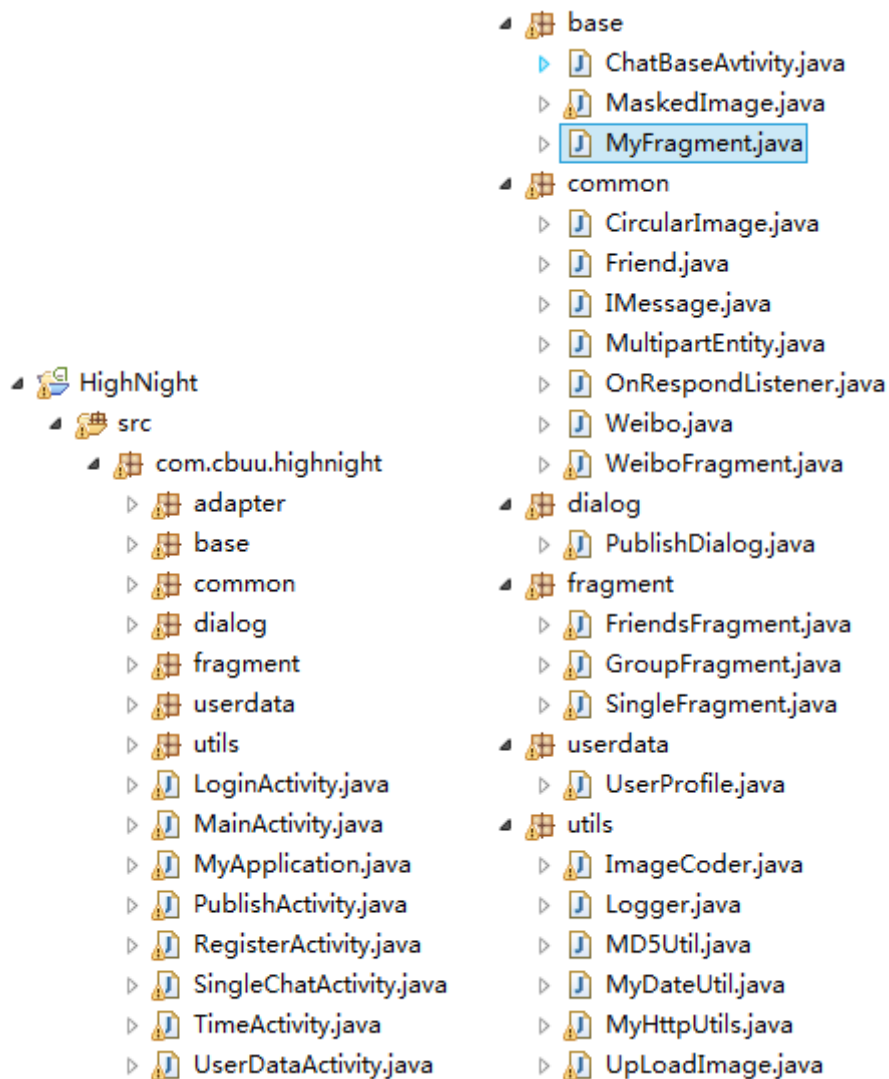
## 二、 架构设计

android 架构如下：



底层内核为 linux 内核，上面跑了一个 dalvik 虚拟机，并集成了各种类库，而我们的项目主要是在最上层的应用层中开发。

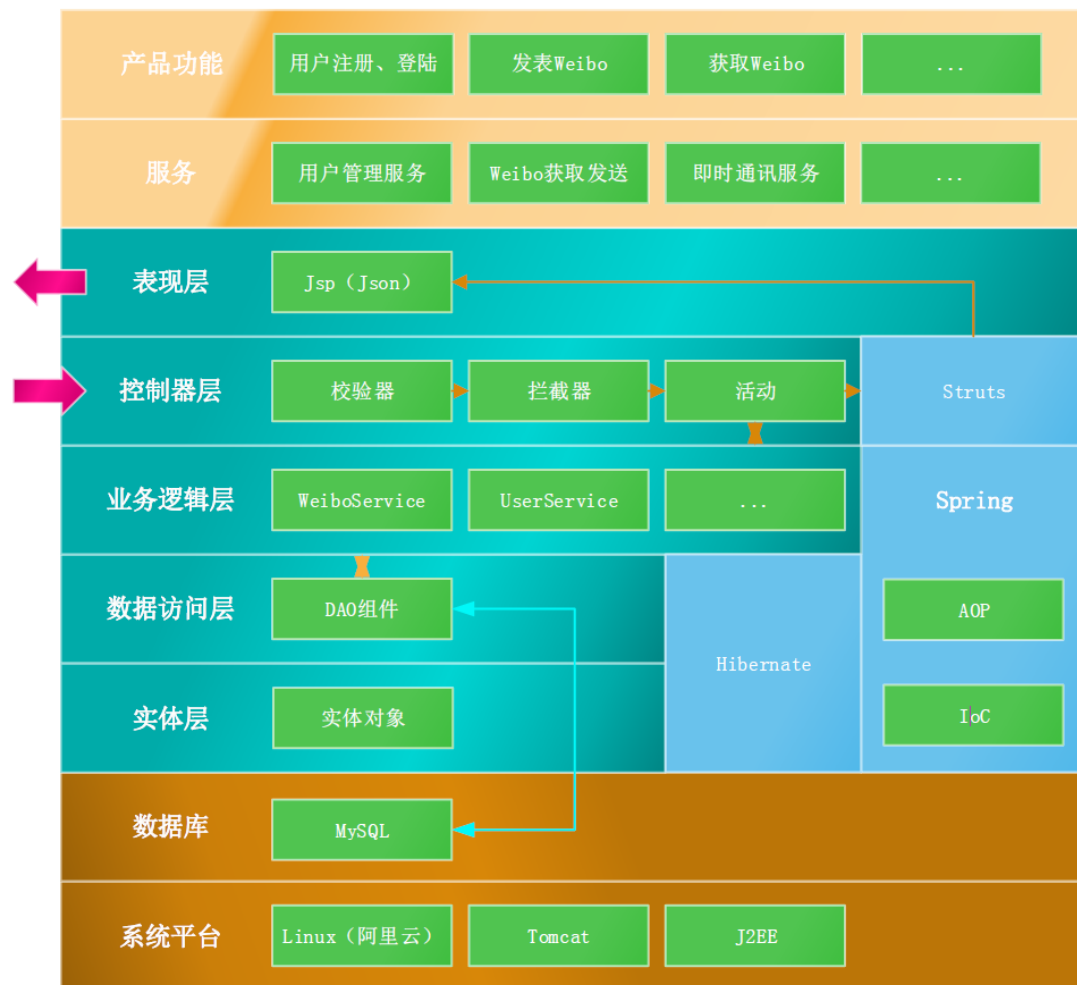
项目中的代码架构如下图：



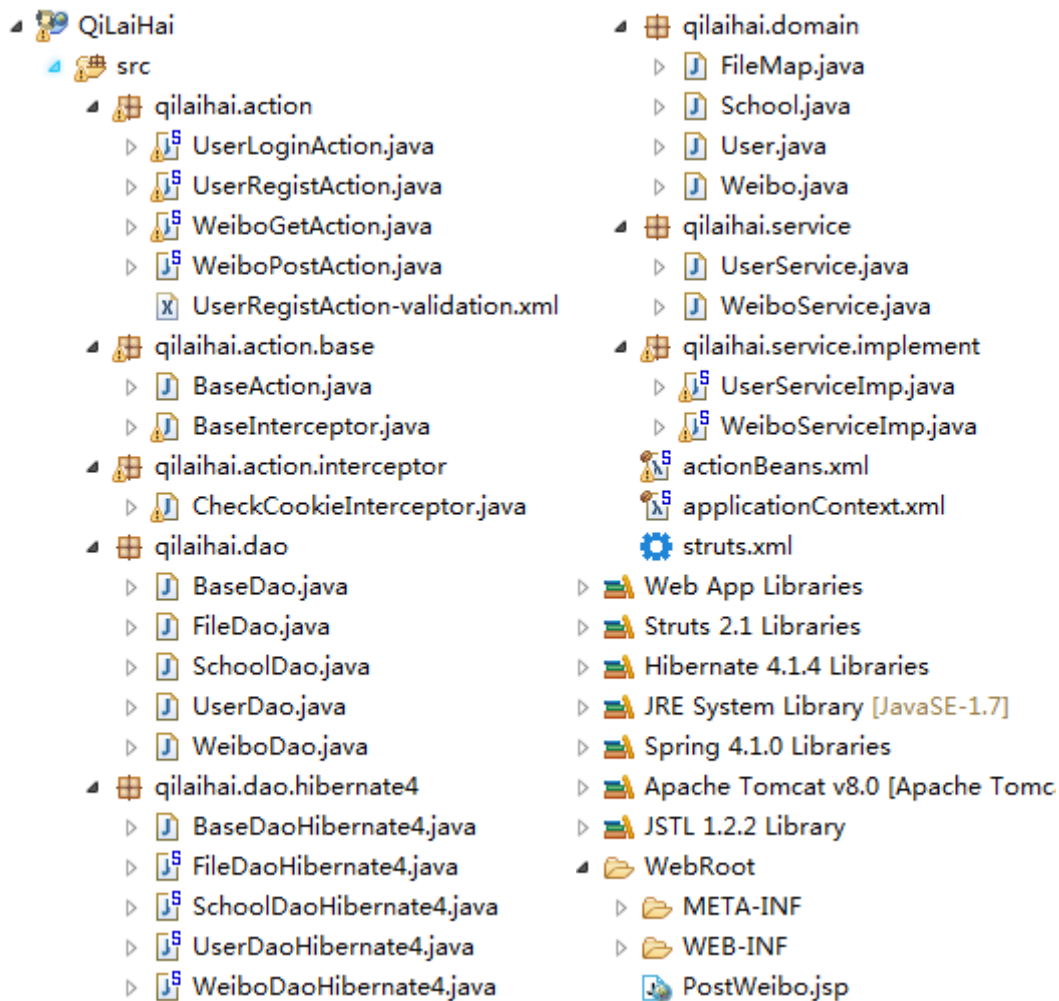
整个安卓项目我们根据功能去划分活动（界面），现阶段主要功能有注册，登录，浏览信息，发布信息，而浏览信息为主要界面（MainActivity）。

主要界面有三个子界面，然而我们并不为每个界面都实现一个 Activity，而是用现安卓新的 UI 技术碎片（Fragment）去实现三个子界面，这样既减少了应用切换界面的消耗，又方便了对界面跳转的管理。同时对每一条微博，也用了一个小碎片去呈现，这样所有的微博用的同一个 UI，只有数据上的不同，保证了良好的复用。

服务端：



项目中的代码架构如下图：



### 三、模块划分

Android 端:

#### 1、数据管理---UserProfile

由于经常需要使用到用户的数据以及客户端与服务端通讯所需要的信息，因此把所有的数据都抽出来独立作为一个模块，实现一个单例类，保存着登录用户的信息，这样每次打开应用，都会直接显示登录的帐号和密码，方便用户直接登录。同时也保存了客户端与服务端通信的 **Cookie**，用于客户端向服务端拉取信息时的身份验证。

#### 2、数据存储模块

如上述的数据单例类

```

public class UserProfile {

    private static UserProfile mUserProfile;
    private Context context;
    private SharedPreferences preferences;

    interface DataKey{
        String USERNAME = "username";
        String PASSWORD = "password";
    }

    private String cookies;

    public static UserProfile getInstance() {
        if (mUserProfile==null) {
            mUserProfile = new UserProfile();
        }
        return mUserProfile;
    }
}

```

用了安卓提供的 API SharedPreferences 实现了简单的键值对存储。

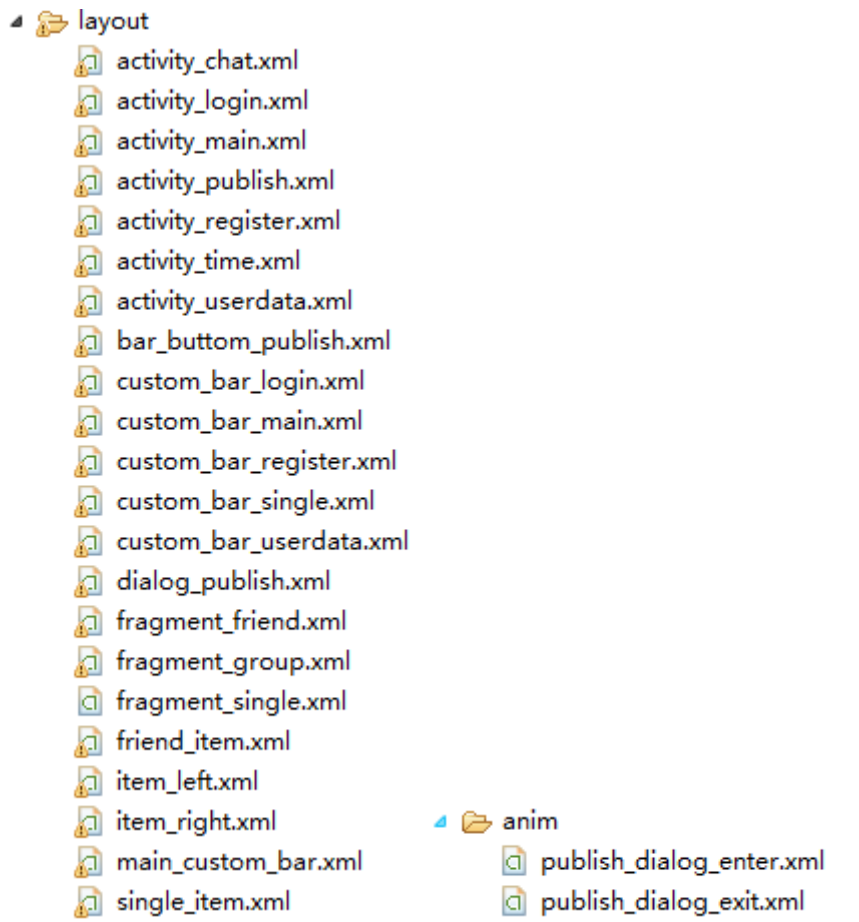
### 3、UI 模块

由于安卓系统本身已经把 UI 抽离出来,既可以用代码实现 UI,亦可以用 xml 去实现 UI,为了更好地管理,我们决定全部用 xml 实现


界面以及子界面的 xml

动画 xml





为何良好地复用各种 UI，我们把 UI 拆分成一个个的子 UI，例如

 **bar\_bottom\_publish.xml** 目前是在主界面的每个子界面都有，如果有一天某个子界面不需要这个底部发布按钮，而其他两个需要，那么并不需要重写三个子界面的 xml，而是直接把发布按钮 include 进需要的子界面。

实现 UI 的高级功能，从屏幕底部弹入弹出的动画，当用户点击发布按钮时，会有选择框从底部弹出。

对碎片的复用，由于主界面的三个子界面都有一个 tab，所以我们先实现了一个基类，

```
public class MyFragment extends Fragment {  
    private String tabName = null;  
  
    public MyFragment(String tabName) {  
        super();  
        this.tabName = tabName;  
    }  
    |  
    public String getTabName() {  
        return tabName;  
    }  
  
    public void setTabName(String tabName) {  
        this.tabName = tabName;  
    }  
}
```

拥有字段 tabName 去表示界面的 tab，然后再实现三



个子类（子界面）对基类的继承，同时这里也用到了多态，在主界面对子界面管理的时候，可以把全部子界面当作是 MyFragment 统一管理。

#### 4、网络模块

项目中的网络模块主要是使用了 Volley 库去实现所有的 http 通信。通信内容格式为 JSON 格式

```
JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(url, null,
    new Response.Listener<JSONObject>() {

        @Override
        public void onResponse(JSONObject response) {
            Logger.log("receive"+response.toString());

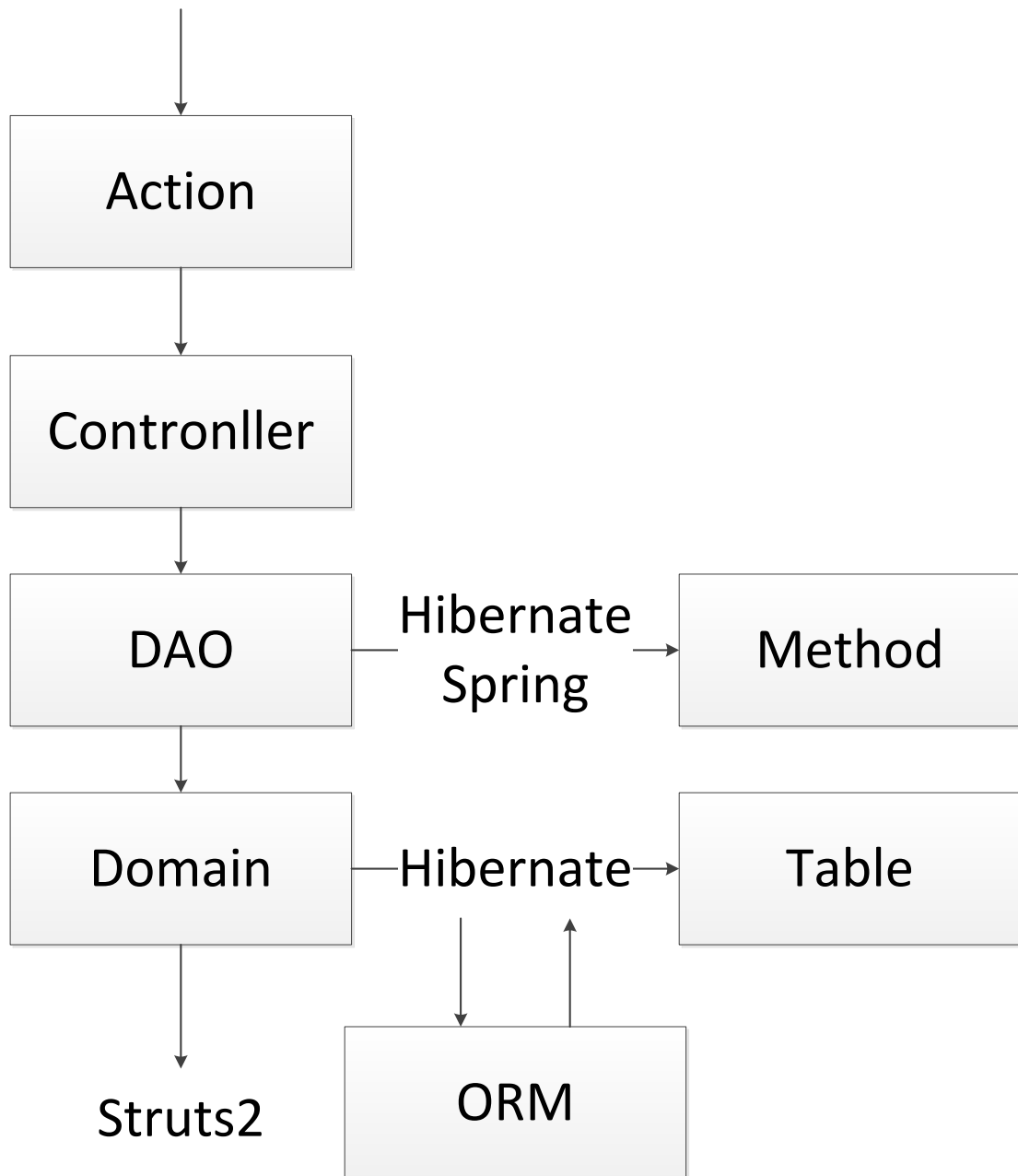
            String flag = null;
            try {
                flag = response.getString("status");
            } catch (JSONException e) {
                e.printStackTrace();
            }
            if (flag.equals("success")) {
                UserProfile.getInstance().setUserName(usernameEditText
                    .getText().toString());
                UserProfile.getInstance().setPassword(passwordEditText.getText()
                    .toString());

                startActivity(new Intent(LoginActivity.this, MainActivity.class));
                LoginActivity.this.finish();
            } else {
                Logger.log("Login failed");
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Logger.log("error:"+error.getMessage());
        }
    }) {
```

这里是用 Volley 库实现的登录功能，首先要创建一个 Request，然后把 request 放到队列中，系统会自动发送这个

request,我们只需要实现回调函数的功能，在回调中，我们解析了服务端回应的 json 对象，判断 status 是否为成功，若成功则跳转到主界面。

服务端：



## 1. 实体层

实体层包含数据库中需要保存的对象的属性，是纯粹的Java Bean。通过Hibernate注解（Annotation）和其他配置，将这些Java Bean及其关系，映射为数据库中表单。即通过J2EE中持久化规范的ORM（对象关系映射）将对象映射为关系，使得在程序中的数据库访问操作以面向对象的方式进行。

在代码模块domain中

- qilaihai.domain
  - FileMap.java
  - School.java
  - User.java
  - Weibo.java

例如：

```
@Entity
@Table(name="t_user")
@Cache(usage=CacheConcurrencyStrategy.READ_WRITE)
public class User implements Serializable {

    /**
     * Version 1.0
     */
    private static final long serialVersionUID = 1L;

    // ----- Field -----

    private Integer mId;
    private String mPassword;
    private String mNickName;
    private String mPhoneNumber;
    private Date mBirthday;
    private Integer mGender;
    private School mSchool;

    // -----













    public User() {
```

定义了user的实体。

## 2.数据访问层

数据访问层通过Hibernate与MySQL建立的会话，执行事务。在这一层中，直接对数据库进行访问操作，如增删改查。操作中保证存入数据库中的数据是合法的。

在代码中：

- ▲  qilaihai.dao
  - ▷  BaseDao.java
  - ▷  FileDao.java
  - ▷  SchoolDao.java
  - ▷  UserDao.java
  - ▷  WeiboDao.java
- ▲  qilaihai.dao.hibernate4
  - ▷  BaseDaoHibernate4.java
  - ▷  FileDaoHibernate4.java
  - ▷  SchoolDaoHibernate4.java
  - ▷  UserDaoHibernate4.java
  - ▷  WeiboDaoHibernate4.java

其中dao定义了映射：

```
2
3 import qilaihai.domain.School;
4
5 public interface SchoolDao extends BaseDao<School> {
6     School findByName(String name);
7 }
8
```

而hibernate4则用“HQL”语言对数据库进行操作：

```
public class SchoolDaoHibernate4 extends BaseDaoHibernate4<School> implements
    SchoolDao {

    @Transactional
    @Override
    public School findByName(String name) {
        School result = null;

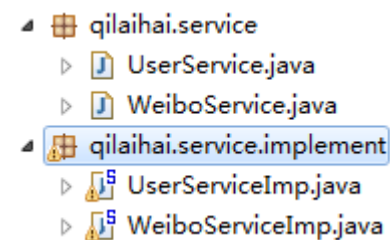
        List<School> schools = find("select s from School as s where s.name = ?0",
            name);

        if (schools != null && schools.size() > 0) {
            result = schools.get(0);
        }
    }
}
```

### 3.业务逻辑层

对软件的业务逻辑进行实际的操作。比如进行用户登录、调用数据访问层进行数据存取，对消息进行路由、传递等。

代码实现：



```
qilaihai.service
├── UserService.java
├── WeiboService.java
└── qilaihai.service.implement
    ├── UserServiceImp.java
    └── WeiboServiceImp.java
```

其中service.implementat是service中方法的实现

```
29 @Override
30 public User regist(User user) {
31     User repeatUser = mUserDao.findByPhoneNumber(user.getPhoneNumber())
32     if (repeatUser != null) {
33         return null;
34     }
35     mUserDao.save(user);
36     return mUserDao.findByPhoneNumber(user.getPhoneNumber());
37 }
38
```

### 4.控制器层

对来自客户端的输入进行校验、拦截，调用业务逻辑层执行业务逻辑，根据执行结果，

调度合适的视图进行渲染，返回给客户端。

代码位置：



```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configurat
<struts>

    <constant name="struts.devMode" value="true" />

    <package name="default" extends="struts-default" namespace="/">

        <interceptors>

            <interceptor name="checkCookie"
                class="qilaihai.action.interceptor.CheckCookieInterceptor" />

            <!-- 定义拦截器 -->
            <interceptor-stack name="cookieTestStack">
                <interceptor-ref name="checkCookie" />
                <interceptor-ref name="defaultStack" />
            </interceptor-stack>

        </interceptors>

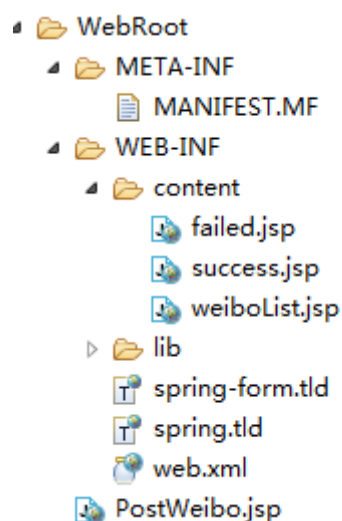
        <action name="userLogin" class="userLoginAction">
            <result name="success">/WEB-INF/content/success.jsp</result>
            <result name="failed">/WEB-INF/content/failed.jsp</result>
        </action>

        <action name="userRegist" class="userRegistAction">
```

## 5.表现层

表现层采用 Jsp 文件生成 Json 格式的数据返回给客户端。

代码位置：



```
WebRoot
├── META-INF
│   └── MANIFEST.MF
├── WEB-INF
│   ├── content
│   │   ├── failed.jsp
│   │   ├── success.jsp
│   │   └── weiboList.jsp
│   └── lib
│       ├── spring-form.tld
│       ├── spring.tld
│       └── web.xml
└── PostWeibo.jsp
```

```

<%@ page language= java contentType= text/html; charset=UTF-8
    pageEncoding="UTF-8"%>
<%@taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>发表微博</title>
</head>
<body>
    <form action="weiboPost" method="post" enctype="multipart/form-data">
        <input id="text" name="weibo.text" type="text" />
        <input id="imageName" name="imageName" type="text" hidden="hidden" ,
        <s:file id="image" name="image"
            onChange="if(this.value) insertImageName(this.value);"></s:file>
        <input type="submit" />
    </form>

    <script>
        function insertImageName(tValue) {
            var t1 = tValue.lastIndexOf("\\");
            if (t1 >= 0 && t1 < tValue.length) {
                document.getElementById("imageName").value = tValue.substr(
                    t1 + 1, tValue.length);
            }
        }
    </script>

```


## 四、 软件设计技术

1. Structure Programming 结构化编程，本程序结构划分良好。通过接口进行定义，实例的注入通过 Spring 的依赖注入完成，模块彼此之间没有耦合。在上述模块划分中即可看出。

2. 由于采用 java 和 Android，OOP 思想贯穿始终。

3. AOP 体现在 Spring 的面向切片编程，主要体现在 Spring 对事务的配置和优化。克服了单一继承的缺点，通过 Spring 将事物的处理分离出来。

代码举例：

 applicationContext.xml

中，

```

<!-- 配置事务异常封装 -->
<bean id="persistenceExceptionTranslationPostProcessor"
      class="org.springframework.dao.annotation.PersistenceExceptionTranslat

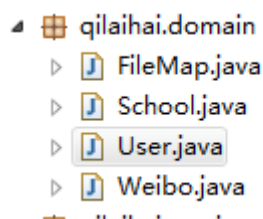
```

将事务异常封装出来。以达到改变这些行为的时候不影响业务逻辑。

4. Spring 也是 SOA 开发的首选 Java 轻量级框架，Spring 已经对 AOP 有了良好的支持，故没有用 SOA 进行优化。但实际上也足够达到效果。

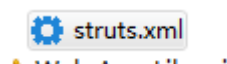
5. 设计模式：工厂、责任链（Struts 的拦截器逻辑，还有 Servlet 的 Filter 标准）等。

工厂模式：



中各个实体部分有所体现。

责任链：



```

<!-- 定义拦截器 -->
<interceptor-stack name="cookieTestStack">
    <interceptor-ref name="checkCookie" />
    <interceptor-ref name="defaultStack" />
</interceptor-stack>

</interceptors>

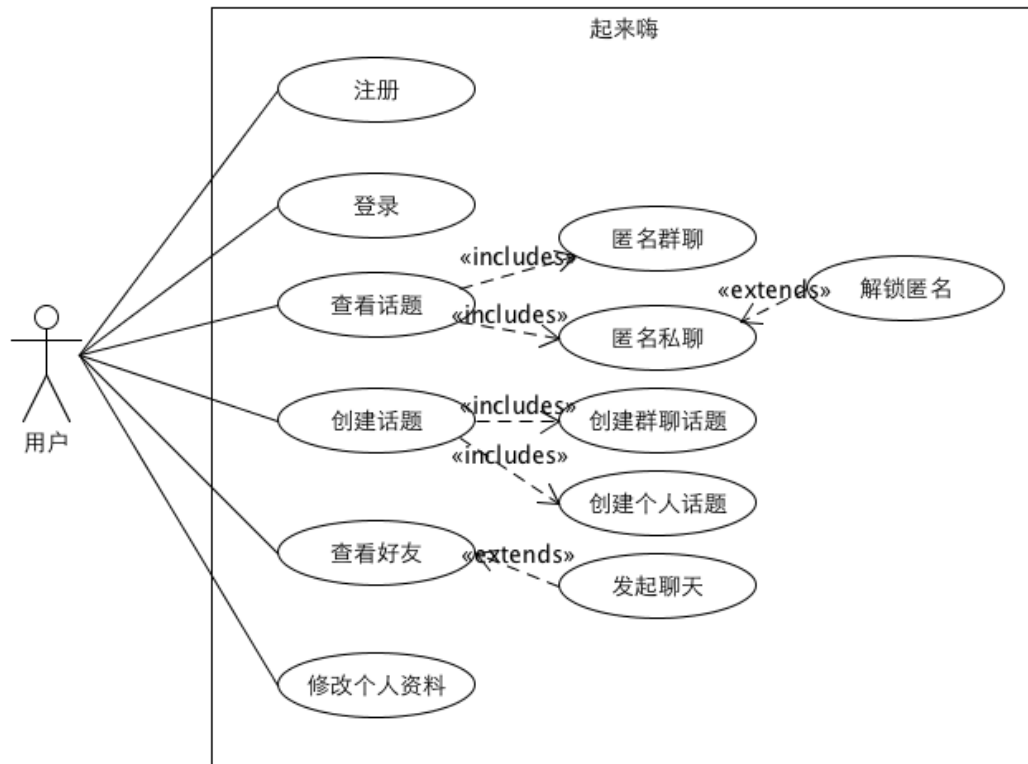
<action name="userLogin" class="userLoginAction">
    <result name="success">/WEB-INF/content/success.jsp</result>
    <result name="failed">/WEB-INF/content/failed.jsp</result>
</action>

```

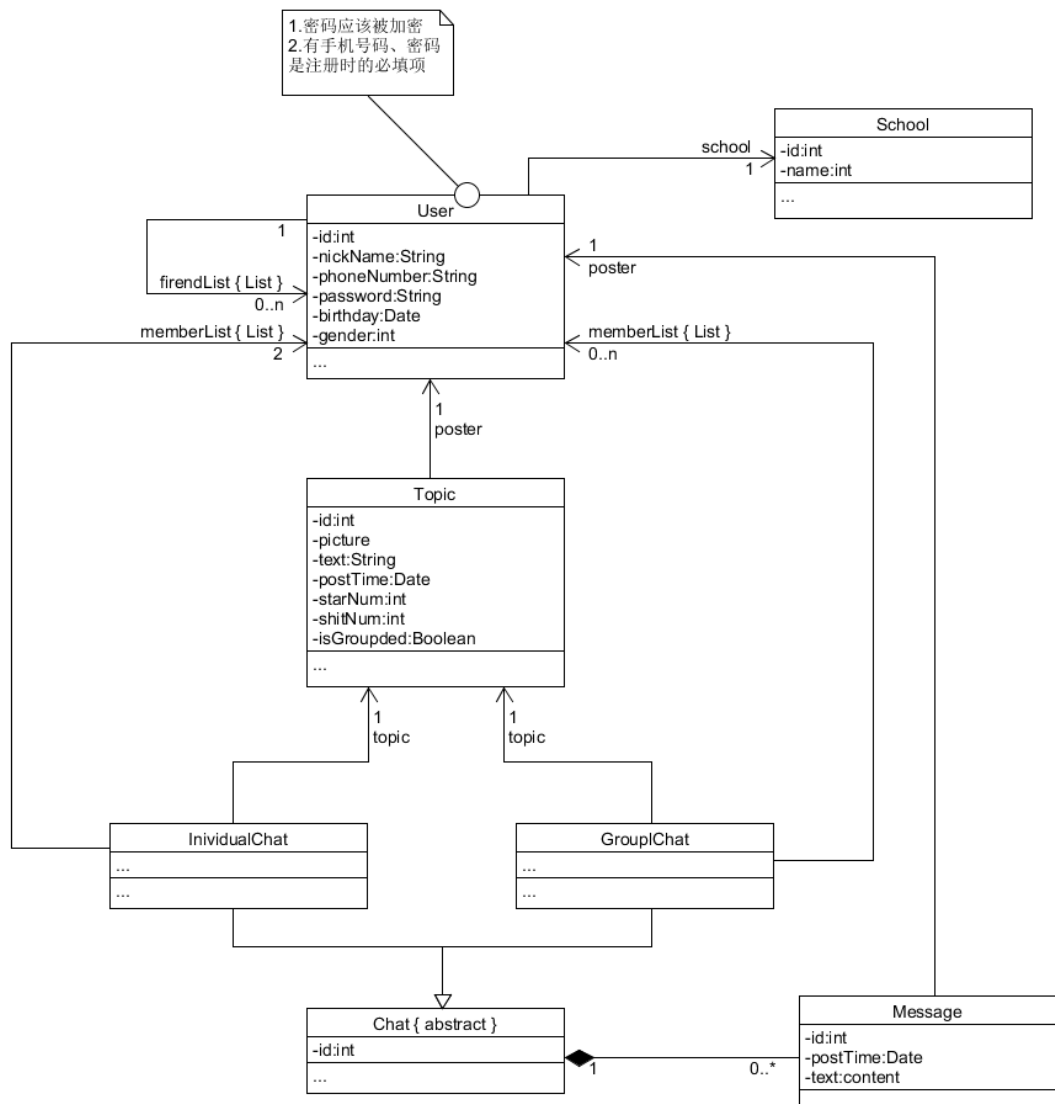


## 五、UML 图

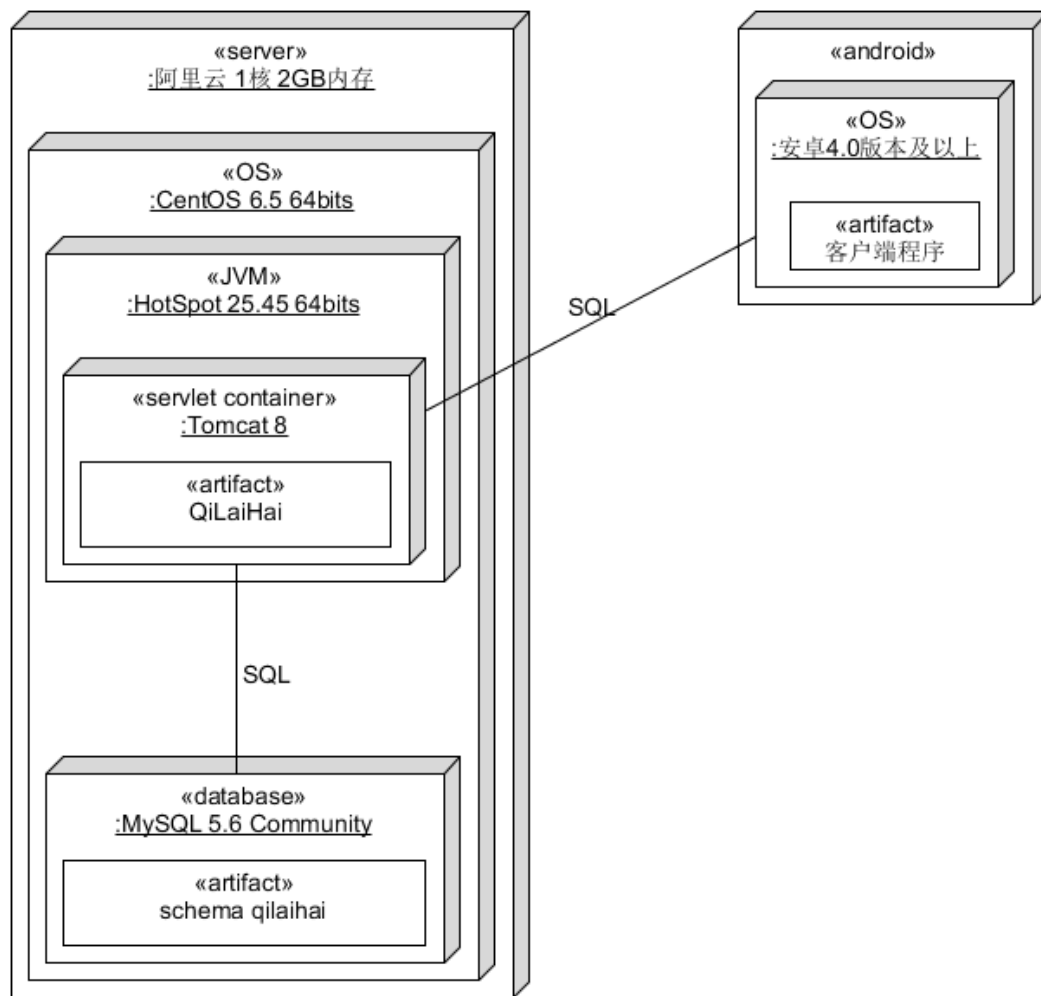
用例图：



领域模型：



部署图：



## 六、测试用例

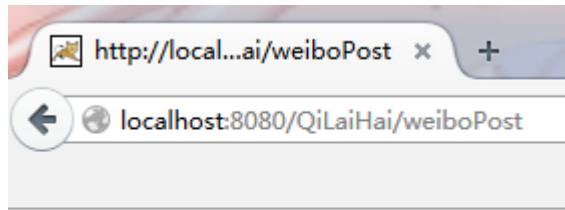
详见【使用说明书】。其中对每个按钮都进行了通过性测试。

失效性测试用例：

未进行登陆而直接进行发微博的操作：未进行登陆



发微博后：



`{"status": "failed"}`

fail 原因：未登陆。