后台设计文档

技术选型：

NodeJS(服务器端) + Express(服务器开发框架) + MongoDB(数据库)

技术选型理由：

1. NodeJS 是运行在后台的 javascript，采用事件驱动机制和高性能异步 IO 模型，充分利用系统资源。
2. Express 是一个简洁而灵活的 nodejs Web 应用框架，Express 不对 nodejs 已有的特征进行二次抽象，只是在它之上扩展了 Web 应用所需的功能，提供丰富的 HTTP 工具以及来自 Connect 框架的中间件。可以帮助开发者快速搭建后台。
3. MongoDB 是一个高性能、开源、无模式的文档型数据库，在许多场景下可以替代传统的关系型数据库或键/值存储方式。MongoDB 非常适合实时的插入、更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。MongoDB 的 BSON 数据格式非常适合文档化的存储与查询。

架构设计：

MVC 架构

Model:

userModel (routes\model\userModel.js)

messageModel (routes\model\messageModel.js)

View:

Web-front-end

Controller:

Web 端：

userController (routes\controller\user.js)

messageController (routes\controller\message.js)

Android 端：

mobileUserController (routes\mobile-controller\user.js)

mobileMessageController (routes\mobile-controller\message.js)

模块划分：

User 模块：

Web 端：

user.logout //用户登出

user.login //用户登录

user.register //用户注册

Android 端：

user.logout //用户登出

user.login //用户登录

user.register //用户注册

Message 模块：

Web 端：

message.home //主页面显示

message.me //管理个人发布的信息

message.queryMessageById //查询单条信息
message.createMessage //创建信息，向前端 render 创建信息的页面
message.postMessage //发布信息
message.editMessage //编辑信息
message.deleteMessage //删除信息
message.queryMessage //查询满足条件的信息

Android 端：

message.postMessage //发布信息
message.editMessage //编辑信息
message.deleteMessage //删除信息
message.queryMessage //查询满足条件的信息
message.queryMyMessage //查询当前用户发布的信息

前端设计文档

采用 multi-page app 模式（页间有刷新）+局部更新，局部更新方式：客户端发起 ajax 请求，服务端渲染 html 片段，然后客户端通过 DOM 操作完成页面局部更新

组件封装：每个组件封装为相应的 DOM Elements 和供其他组件调用的接口，组件之间的通信通过方法调用来完成

采用模板引擎 Jade，好处是最大程度减小冗余的 html 代码，增强可读性

采用 css 预编译器 less 代替原生 css，主要用到的特性是变量声明和嵌套语法，使得 css 具有更好的模块语义

Android 端设计文档

模块划分
登陆注册模块
个人信息管理模块
信息发布模块
信息查询模块
地图支持模块

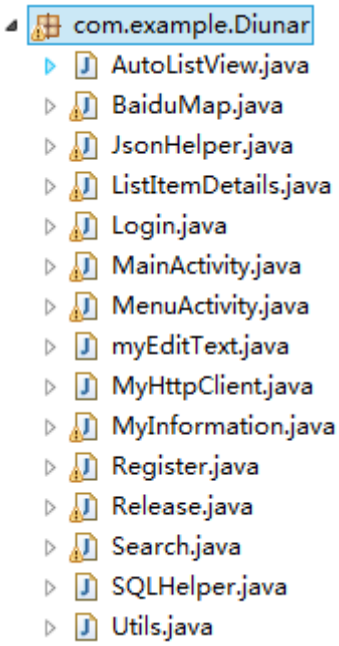
设计技术
选型理由：
1. 采用面向对象思想设计的结构，可读性高，由于继承的存在，即使改变需求，那么维护也只是在局部模块，所以维护起来是非常方便的。
2.AndroidSDK 能使开发人员更容易动态创建用户界面，使 Android 开发更为方便。
3. SQLite 作为轻型数据库，可以在本地存储轻量级的数据，避免多余的服务器请求。
4. 采用 MVC 模型进行设计，将用户之间的操作通过 Controller 进行管理，符合现代软件设计的习惯，方便管理服务器。

Object-Oriented Programming

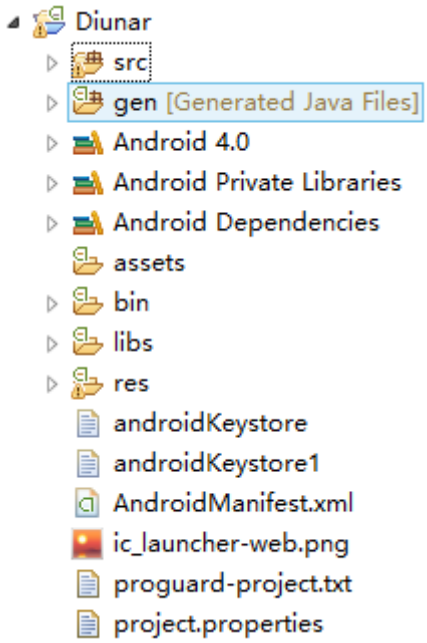
```
public class AutoListView extends ListView implements OnScrollListener {

    // 区分当前操作是刷新还是加载
    public static final int REFRESH = 0;
    public static final int LOAD = 1;

    // 区分PULL和RELEASE的距离的大小
    private static final int SPACE = 20;

    // 定义header的四种状态和当前状态
    private static final int NONE = 0;
    private static final int PULL = 1;
    private static final int RELEASE = 2;
    private static final int REFRESHING = 3;
    private int state;

    private LayoutInflater inflater;
    private View header;
    private View footer;
    private TextView tip;
    private TextView lastUpdate;
```

```
public class myEditText extends EditText{

  private int lineColor;//横线颜色
  private float lineWidth;//横线宽度

  public myEditText(Context context) {
    super(context);

    //设置默认颜色和横线宽度
    lineColor = Color.BLACK;
    lineWidth = 0.8f;//默认宽度为0.5
  }

  public myEditText(Context context,int color,float width) {
    super(context);
```

MVC



Android SDK



SQLite

```
public class SQLHelper extends SQLiteOpenHelper {

    private static final String DB_NAME = "mydatas.db";
    private static final int version = 1;

    public SQLHelper(Context context) {
        super(context, DB_NAME, null, version);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // TODO Auto-generated method stub
        String sql = "create table Item(title varchar(100) not null , "
                    + "tag varchar(100) not null ,"
                    + "type varchar(30) ,"
                    + "time varchar(30) ,"
                    + "place varchar(100) ,"
                    + "x varchar(30) ,"
```