Bottle

软件设计文档（SD）

V1.0

1. 总体设计

一个基于地理位置的新式漂流瓶应用。

曾经风靡一时的漂流瓶应用，微信、qq、yy语音都推出过漂流瓶功能，但是他们却很少与地理位置画上关系，所以这就是我们产生这个想法的源点。

我们的应用是一个典型的C/S架构的系统，客户端与服务端通过http协议进行交流

2 技术选型及理由

前端技术选型

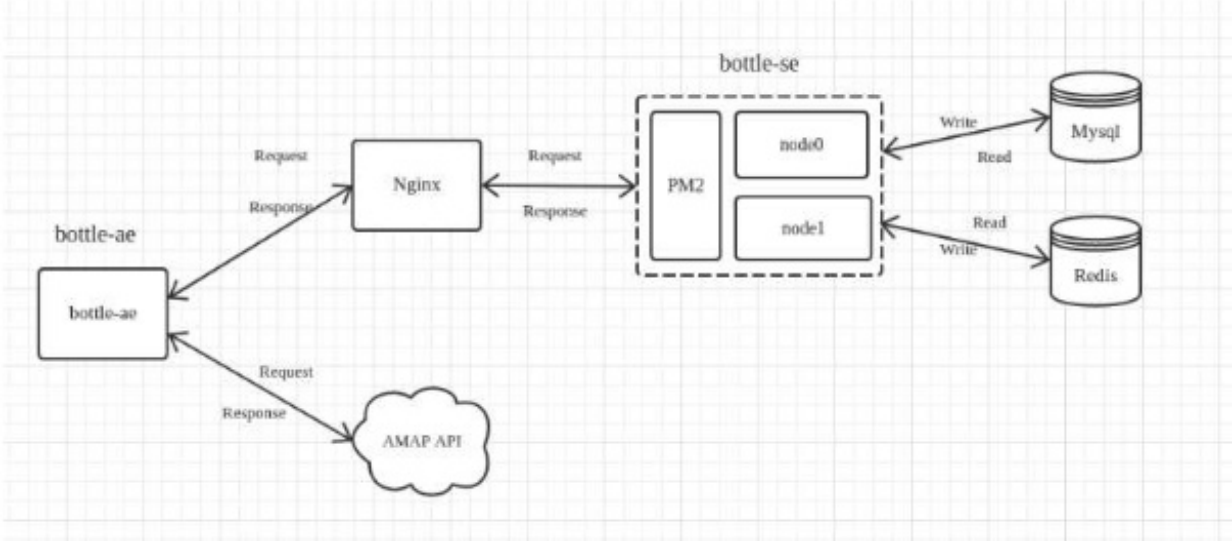
我们前端选择的是安卓客户端，理由如下：

- 安卓使用人群较多，开发安卓版本推广成本较低
- 安卓对于开发门槛较低，不像IOS版本对于开发机的要求比较严格，发布难度也比较大
- 安卓开发的生态比较好，全球有大量的自由开发者，遇到问题有比较多的解决方案

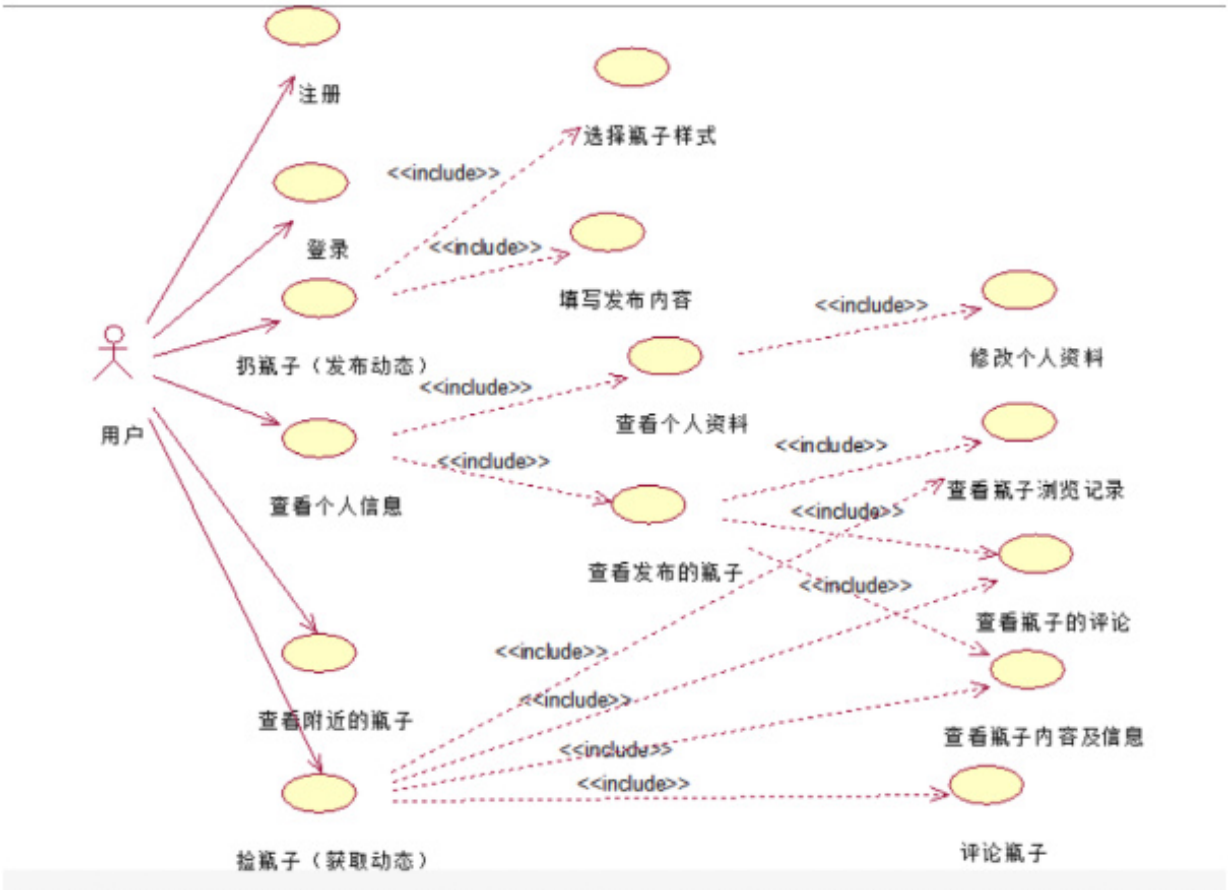
后端技术选型

- nodejs：
    - 便于编写异步高性能服务器
    - 简化对json数据格式的处理
    - 活跃的npm社区，成熟的web开发生态
- mysql：
    - 业界关系型数据库成熟解决方案
- redis：
    - 高速的内存型数据库，用于保存用户的session信息
    - 为每个api必经的session处理中间件提供低时延响应
    - 为多进程架构提供集中的session存储区

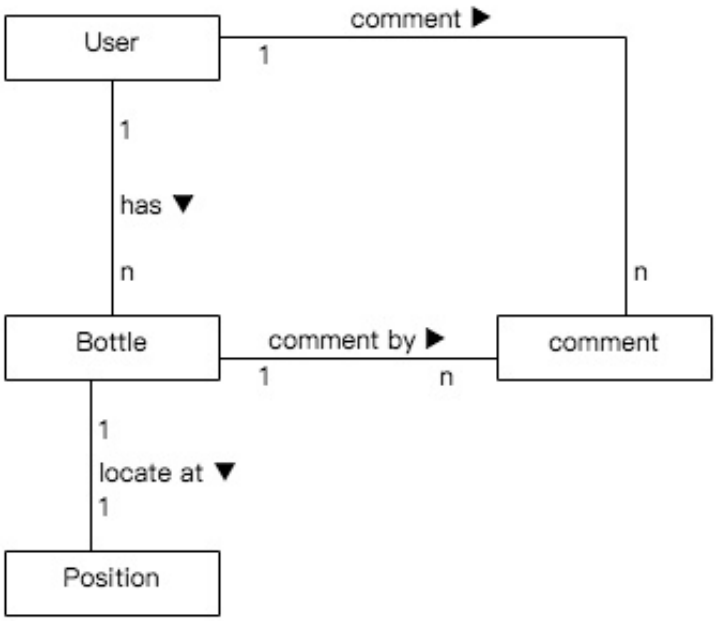
3 架构设计


bottle-se
Write
Read
Mysql
Request
Response
Nginx
Request
Response
PM2
node0
node1
Read
Write
Redis
bottle-ae
bottle-ae
Request
Response
AMAP API

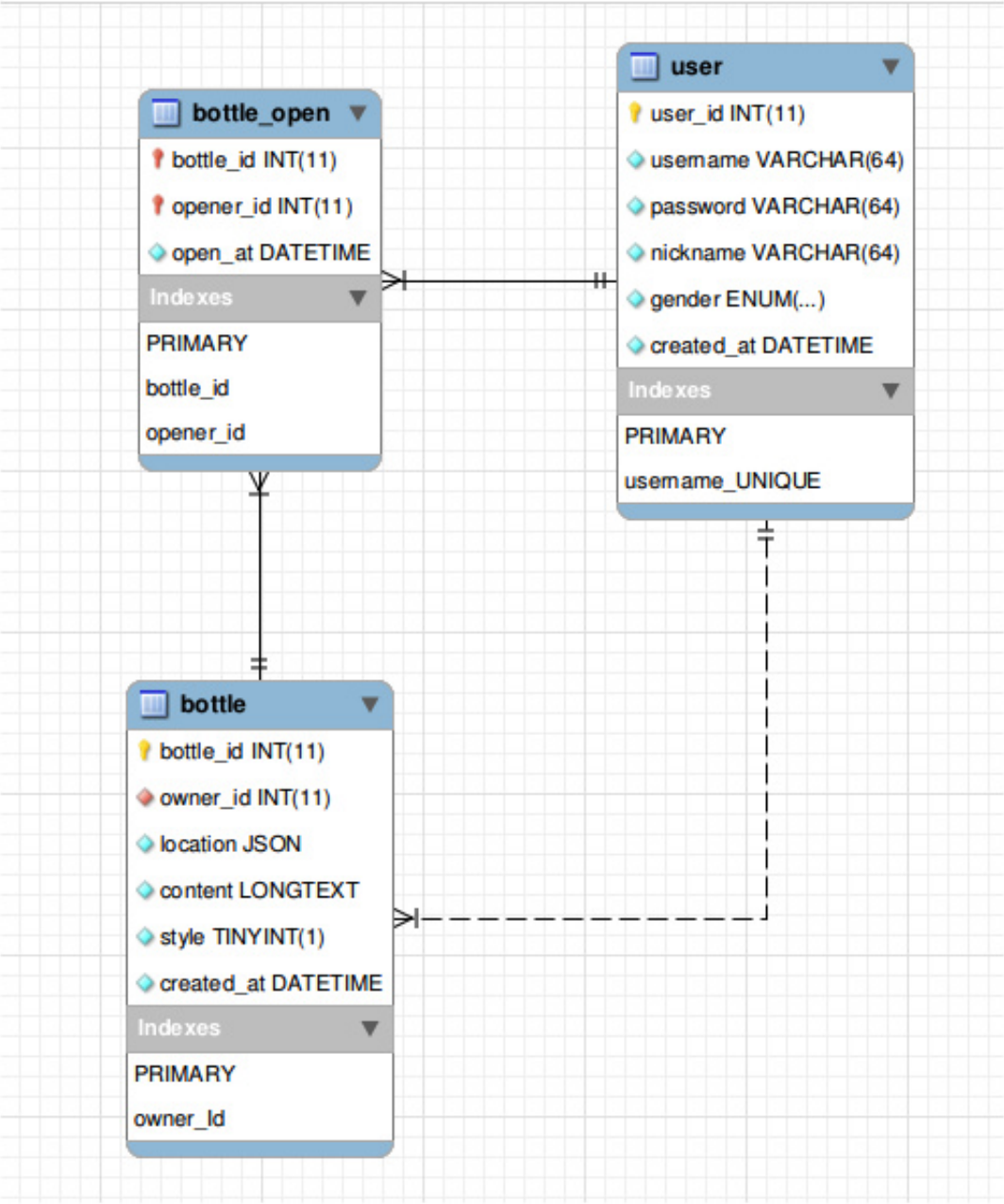
4 用例设计


注册
选择瓶子样式
<<include>>
登录
<<include>>
填写发布内容
<<include>>
修改个人资料
扔瓶子（发布动态）
<<include>>
查看个人资料
用户
<<include>>
<<include>>
查看个人信息
查看瓶子浏览记录
<<include>>
查看发布的瓶子
<<include>>
查看瓶子的评论
<<include>>
<<include>>
查看附近的瓶子
查看瓶子内容及信息
<<include>>
<<include>>
捡瓶子（获取动态）
评论瓶子

5 领域建模

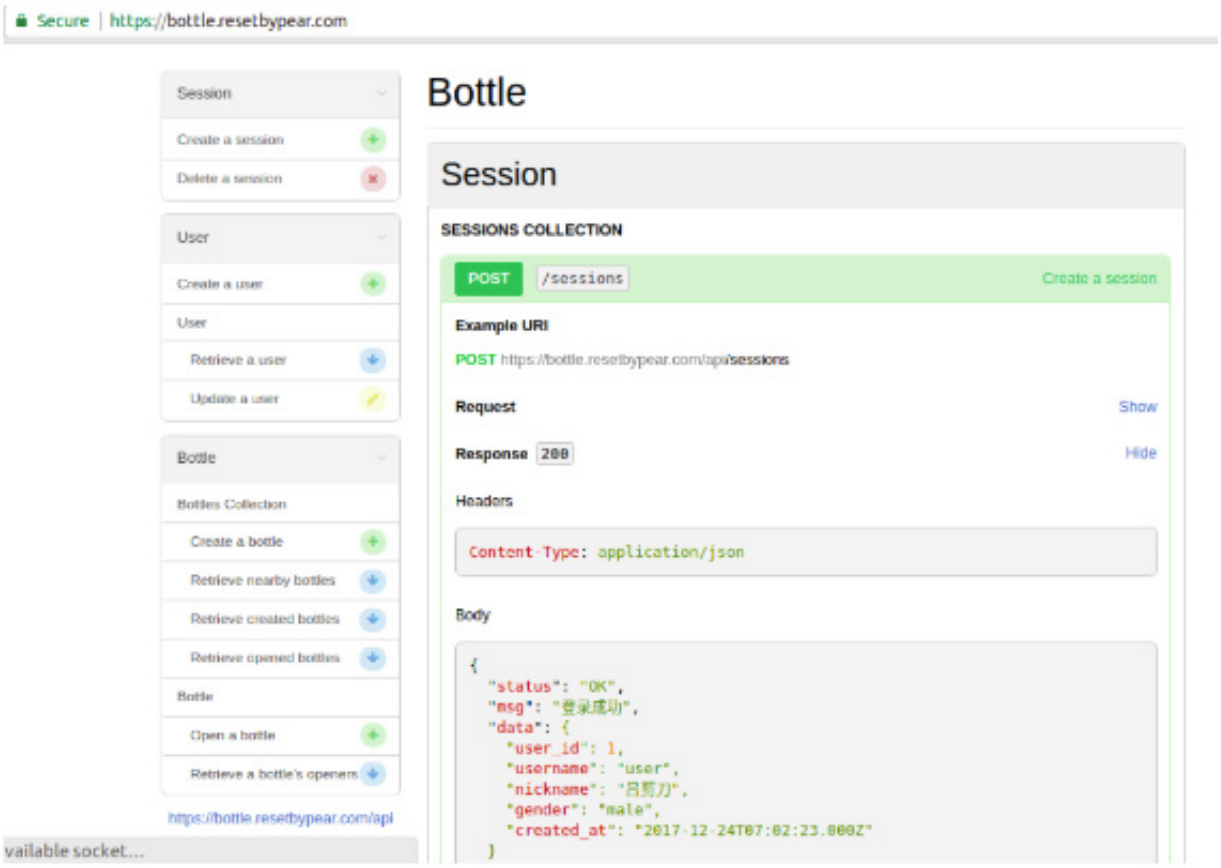

User
comment ▶
1
1
has ▼
n
n
Bottle
comment by ▶
1
n
comment
1
locate at ▼
1
Position

6 数据库设计



7 API设计

遵循RESTful规范进行API设计，使用blueprint编写API文档，利用aglio工具渲染出html，并通过nginx代理此静态文件，使得前端开发人员只需访问https://bottle.resetbypear.com/ 便可方便地浏览到最新的API文档。



8 前端模块划分

8.1 验证模块

该模块实现的功能为用户登录与用户注册，设计界面使用viewPager适配，以实现登录和注册功能的切换的滑动效果；并使用okhttp3+retrofit实现与后台数据库之间的交互

使用viewPager适配

```
class MySimpleAdapter extends PagerAdapter {
    private ArrayList<View> arrayList_view = new ArrayList<View>();
    MySimpleAdapter(ArrayList<View> list) {
        arrayList_view = list;
    }
    @Override
    public int getCount() {
        return arrayList_view.size();
    }

    @Override
    public boolean isViewFromObject(@NonNull View view, @NonNull Object object) {
        return view==object;
    }
    @Override
    public Object instantiateItem(ViewGroup container, int position) {
        container.addView(arrayList_view.get(position));//添加页卡
        return arrayList_view.get(position);
    }

    @Override
    public void destroyItem(ViewGroup container, int position, Object object) {
        container.removeView(arrayList_view.get(position));//删除页卡
    }

    @Override
    public CharSequence getPageTitle(int position) {
        return list_titles.get(position);//页卡标题
    }

}
```

定义Interface

```
public interface Services {
    @Headers({"Content-type:application/json","Accept: application/json"})
    @POST("users")
    Observable<ResponseUser> postUser(@Body RequestBody route);

    @Headers({"Content-type:application/json","Accept: application/json"})
    @POST("sessions")
    Observable<ResponseUser> loadUser(@Body RequestBody route);

    @GET("users/self")
    Observable<ResponseUser> get();

    /**
     * Added by Bowen Wu in 2018/01/04
     * Used when create bottle
     */
    @Headers({"Content-type:application/json","Accept: application/json"})
    @POST("bottles")
    Observable<ResponseBottle> postBottle(@Body RequestBody route);
```

请求时，将数据模型转位JSON并放入RequestBody中，或直接放入URL中：

```
public static RequestBody ObjToRequestBody(Object obj) {
    String data = new Gson().toJson(obj);
    Log.d("JSON", data);
    return RequestBody.create(okhttp3.MediaType.parse("application/json;charset=UTF-8"), data);
}
```

发起一个请求，定义完成和订阅的线程，创建一个订阅者实现请求完成后的逻辑：

```
Factory.getServices(MainActivity1.this).openBottle(bottle.bottle_id)
        .observeOn(AndroidSchedulers.mainThread())
        .subscribeOn(Schedulers.newThread())
        .subscribe(new Observer<ResponseBottle>() {
```

登录注册部分在页面设计上使用了TabLayout和自定义ViewPager的方式实现了页面切换的滑动效果，使用户体验更加友好。

```
<android.support.design.widget.TabLayout
    android:layout_marginLeft="100dp"
    android:layout_marginRight="100dp"
    app:tabIndicatorColor="@color/mainColor"
    app:tabMode="scrollable"
    app:tabTextColor="@color/textcolor"
    app:tabSelectedTextColor="@color/mainColor"
    android:id="@+id/tab"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
<android.support.v4.view.ViewPager
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/page"/>
```

登录和注册页面是分开两个xml设计，分别为load.xml和register.xml，然后将相应的xml放入对应的arraylist内以实现配置。配置自定义的ViewPager：

```
class MySimpleAdapter extends PagerAdapter {
    private ArrayList<View> arrayList_view = new ArrayList<View>();
    MySimpleAdapter(ArrayList<View> list) {
        arrayList_view = list;
    }
    @Override
    public int getCount() {
        return arrayList_view.size();
    }

    @Override
    public boolean isViewFromObject(@NonNull View view, @NonNull Object object) {
        return view==object;
    }
    @Override
    public Object instantiateItem(ViewGroup container, int position) {
        container.addView(arrayList_view.get(position));//添加页卡
        return arrayList_view.get(position);
    }

    @Override
    public void destroyItem(ViewGroup container, int position, Object object) {
        container.removeView(arrayList_view.get(position));//删除页卡
    }

    @Override
    public CharSequence getPageTitle(int position) {
        return list_titles.get(position);//页卡标题
    }

}
```

之后是配置相关按钮的监听器。这是登录按钮的监听器

```
private View.OnClickListener loadOnclickListener(final Services services) {
    return new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String name = load_name.getText().toString();
            final String pwd = load_pwd.getText().toString();
            if (name.isEmpty()) {

                load_name.setError("请输入用户名");
            } else if (pwd.isEmpty()) {
                load_pwd.setError("请输入密码");
            } else {
                final ProgressBar progressBar = findViewById(R.id.progressbar);
                tabLayout.setVisibility(View.GONE);
                pager.setVisibility(View.GONE);
                progressBar.setVisibility(View.VISIBLE);
                Map<String , String > map = new HashMap<>();
                map.put("username" , name);
                map.put("password" , pwd);
                Gson gson=new Gson();
                String strEntity = gson.toJson(map);
                RequestBody body = RequestBody.create(okhttp3.MediaType.parse("application/json;charset=UTF-8"),strEntity);
```

完善了登录功能的健壮性，确保用户能正确输入相关信息并登陆成功。实现了判断是否输入用户名或密码、是否密码错误等的信息提示。登陆成功后，发送相关网络请求，通过**retrofit**和**okhttp**的结合，以及**observable**观察订阅模式，实现对后台发送**GET**和**POST**请求，并**post session**，以及保存**cookie**，以实现用户登录的持久性。代码如下

```
RequestBody body = RequestBody.create(okhttp3.MediaType.parse("application/json;charset=UTF-8"),strEntity);
services.loadUser(body)
        .subscribeOn(Schedulers.newThread())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(new Subscriber<ResponseUser>() {
            @Override
            public void onCompleted() {
                finish();
            }

            @Override
            public void onError(Throwable e) {
                HttpException exception = (HttpException) e;
                if (exception.response().code() == 403) {
                    load_name.setError("用户名或密码错误");
                    Log.i("errorcode" , "403");
                }
            }

            @Override
            public void onNext(ResponseUser user) {
                startActivity(new Intent(MainActivity.this,MainActivity1.class));
            }
        });
```

注册页面的功能与登录功能类似，实现了相关错误信息输入的判断。代码如下：

```
private View.OnClickListener registerOnclickListener(final Services services) {
    return new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            int buttonId = radioGroup.getCheckedRadioButtonId();
            final String username = register_name.getText().toString()+"";
            String nickname = register_nickname.getText().toString();
            final String pwd = register_pwd.getText().toString();
            String con_pwd = register_con_pwd.getText().toString();
            if (username.isEmpty()) {
                register_name.setError("用户名不能为空");
            } else if (nickname.isEmpty()) {
                register_nickname.setError("昵称不能为空");
            } else if (pwd.isEmpty()) {
                register_pwd.setError("密码不能为空");
            } else if (con_pwd.isEmpty()) {
                register_con_pwd.setError("确认密码不能为空");
            } else if (buttonId == -1){
                TextView radioButton = view_register.findViewById(R.id.sex);
                radioButton.setError("请选择性别");
            } else {
                if (con_pwd.equals(pwd)) {
                    RadioButton radioButton = view_register.findViewById(radioGroup.getCheckedRadioButtonId());
                    String gender = radioButton.getText().toString();
                    Map<String , String > map = new HashMap<>();
                    map.put("username" , username);
                    map.put("nickname" , nickname);
                    map.put("password" , pwd);
                    RegisterUser registerUser = new RegisterUser();
                    registerUser.setNickname(nickname);
                    registerUser.setPwd(pwd);
                    registerUser.setUsername(username);
                    if (gender.equals("男")) {
                        registerUser.setGender("male");
                        map.put("gender" , "male");
                    } else {
                        registerUser.setGender("female");
                        map.put("gender" , "female");
                    }
                    Log.i("before" , "aaa");
                    Gson gson=new Gson();
```

其中在最后优化阶段完善的一点是：当注册成功的时候，也会直接跳转到主页面，而不是需要用户再次登录，这点改变了原有的用户体验不良好的效果。

```
String strEntity = gson.toJson(map);
RequestBody body = RequestBody.create(okhttp3.MediaType.parse("application/json;charset=UTF-8"),strEntity);
services.postUser(body)
        .subscribeOn(Schedulers.newThread())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(new Subscriber<ResponseUser>() {
            @Override
            public void onCompleted() {
                final ProgressBar progressBar = findViewById(R.id.progressbar);
                tabLayout.setVisibility(View.GONE);
                pager.setVisibility(View.GONE);
                progressBar.setVisibility(View.VISIBLE);
                Map<String , String > map = new HashMap<>();
                map.put("username" , username);
                map.put("password" , pwd);
                Gson gson=new Gson();
                String strEntity = gson.toJson(map);
                RequestBody body = RequestBody.create(okhttp3.MediaType.parse("application/json;charset=UTF-8"),strEntity);
                services.loadUser(body)
                        .subscribeOn(Schedulers.newThread())
                        .observeOn(AndroidSchedulers.mainThread())
                        .subscribe(new Subscriber<ResponseUser>() {
                            @Override
                            public void onCompleted() {
                                finish();
                            }

                            @Override
                            public void onError(Throwable e) {
                                HttpException exception = (HttpException) e;
                                if (exception.response().code() == 403) {
                                    load_name.setError("用户名或密码错误");
                                    Log.i("errorcode" , "403");
                                }
                            }

                            @Override
                            public void onNext(ResponseUser user) {
                                startActivity(new Intent(MainActivity.this,MainActivity1.class));
                            }
                        });
```

8.2 地图模块

此模块负责以下工作：

1. 显示地图
2. 获取附近的漂流瓶并显示在地图上
3. 打开地图上的漂流瓶
4. 使用一定的策略平衡体验和电量之间的矛盾

```
private void getNearbyBottles(double latitude, double longitude) {
    System.out.println("getNearbyBottles");
    Factory.getServices(MapActivity.this).getNearbyBottle(generateQueryMapForBottlesNearby(latitude, longitu
            .subscribeOn(Schedulers.newThread())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(new Observer<ResponseBottlesList>() {
                @Override
                public void onCompleted() {
                    System.out.println("onCompleted get nearby bottle completed");
                }

                @Override
                public void onError(Throwable e) {
                    System.out.println("ERROR " + e.getMessage());
                    ToastInfo(R.string.internal_error);
                }

                @Override
                public void onNext(ResponseBottlesList responseBottlesList) {
                    Gson gson = new Gson();
                    String data = gson.toJson(responseBottlesList);
                    System.out.println("onNext data : " + data);
                    clearBottleMarker();
                    MapActivity.this.responseBottlesList = responseBottlesList;
                    MapActivity.this.refreshMapMarker();
                }
            });
}
```

根据当前位置获取附近的漂流瓶，并使用高德地图sdk将瓶子标注在地图上

```
private void openBottle(Bottle bottle) {
    System.out.println(bottle.bottle_id);
    // 这是一个距离过远的瓶子，不能打开
    if (!bottle.whetherInArea(myLocation)) {
        ToastInfo(R.string.too_far_to_open_bottle);
        return;
    }
    ToastInfo(R.string.opening_bottle);
    Factory.getServices(MapActivity.this).openBottle(bottle.bottle_id)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribeOn(Schedulers.newThread())
            .subscribe(new Observer<ResponseBottle>() {
                @Override
                public void onCompleted() {

                }

                @Override
                public void onError(Throwable e) {
                    HttpException httpException = (HttpException)e;
                    System.out.println("open Bottle onError : " + httpException.response().code())
                    ToastInfo(R.string.internal_error);
                }

                @Override
                public void onNext(ResponseBottle bottle) {
                    // 显示对话框
                    LayoutInflater layoutInflater = LayoutInflater.from(MapActivity.this);
                    View dialogView = layoutInflater.inflate(R.layout.open_bottle_dialog, null);
                    TextView content = (TextView)dialogView.findViewById(R.id.content);
                    content.setText(bottle.data.content);
                    TextView formatted_address = (TextView)dialogView.findViewById(R.id.formatted_
                    formatted_address.setText(bottle.data.location.formatted_address);
                    AlertDialog.Builder builder = new AlertDialog.Builder(MapActivity.this);
                    builder.setView(dialogView);
                    builder.show();

                }
            });
```

监听用户点击地图上瓶子的动作，并判断能否打开（用户不能打开一个过远的瓶子），若点击的是一个可是打开的瓶子，则向服务器发出请求，表达自己已经打开了这个瓶子，服务端返回该瓶子的相关信息。

```
aMap = mapView.getMap();
myLocationStyle = new MyLocationStyle();
myLocationStyle.interval(1000 * 60); // 每分钟更新一次位置
myLocationStyle.myLocationType(MyLocationStyle.LOCATION_TYPE_FOLLOW_NO_CEN
aMap.setMyLocationStyle(myLocationStyle);
aMap.setMyLocationEnabled(true);
aMap.setOnMyLocationChangeListener(onMyLocationChangeListener);
aMap.setOnCameraChangeListener(onCameraChangeListener);
aMap.setOnMarkerClickListener(onMarkerClickListener);
uiSettings = aMap.getUiSettings();
uiSettings.setMyLocationButtonEnabled(true);
aMap.setMyLocationEnabled(true);
```

```
private AMap.OnMyLocationChangeListener onMyLocationChangeListener = new AMap.OnMyLocationChangeListener() {
    @Override
    public void onMyLocationChange(Location location) {
        queryGeoCode();
        myLocation = location;
        String positon = "";
        positon += new Double(location.getLongitude()).toString();
        positon += " ";
        positon += new Double(location.getLatitude()).toString();
        Log.d("Location", positon);
        System.out.println("onMyLocationChange : " + positon);
    }
};
```

8.3 基础模块构建

这里的基础模块主要指网络请求模块，因为我们定义使用json进行前后端交互，所以序列化/反序列化是一个经常被复用的模块，故我们构建了一个这样的模块作为基础模块。

```
public class PostBodyHelper {


    public static RequestBody mapToRequestBody(Map<String, String> map) {
        String strEntity = new Gson().toJson(map);
        return RequestBody.create(okhttp3.MediaType.parse("application/json;charset=UTF-8"),strEntity);
    }

    public static RequestBody RequestBottleModelToRequestBody(Bottle bottle) {
        String data = new Gson().toJson(bottle);
        Log.d("JSON", data);
        return RequestBody.create(okhttp3.MediaType.parse("application/json;charset=UTF-8"), data);
    }

    public static RequestBody ObjToRequestBody(Object obj) {
        String data = new Gson().toJson(obj);
        Log.d("JSON", data);
        return RequestBody.create(okhttp3.MediaType.parse("application/json;charset=UTF-8"), data);
    }

}
```

(bottle-ae/app/src/main/java/com/pear/bottle_ae/PostBodyHelper.java)

9 后端模块划分

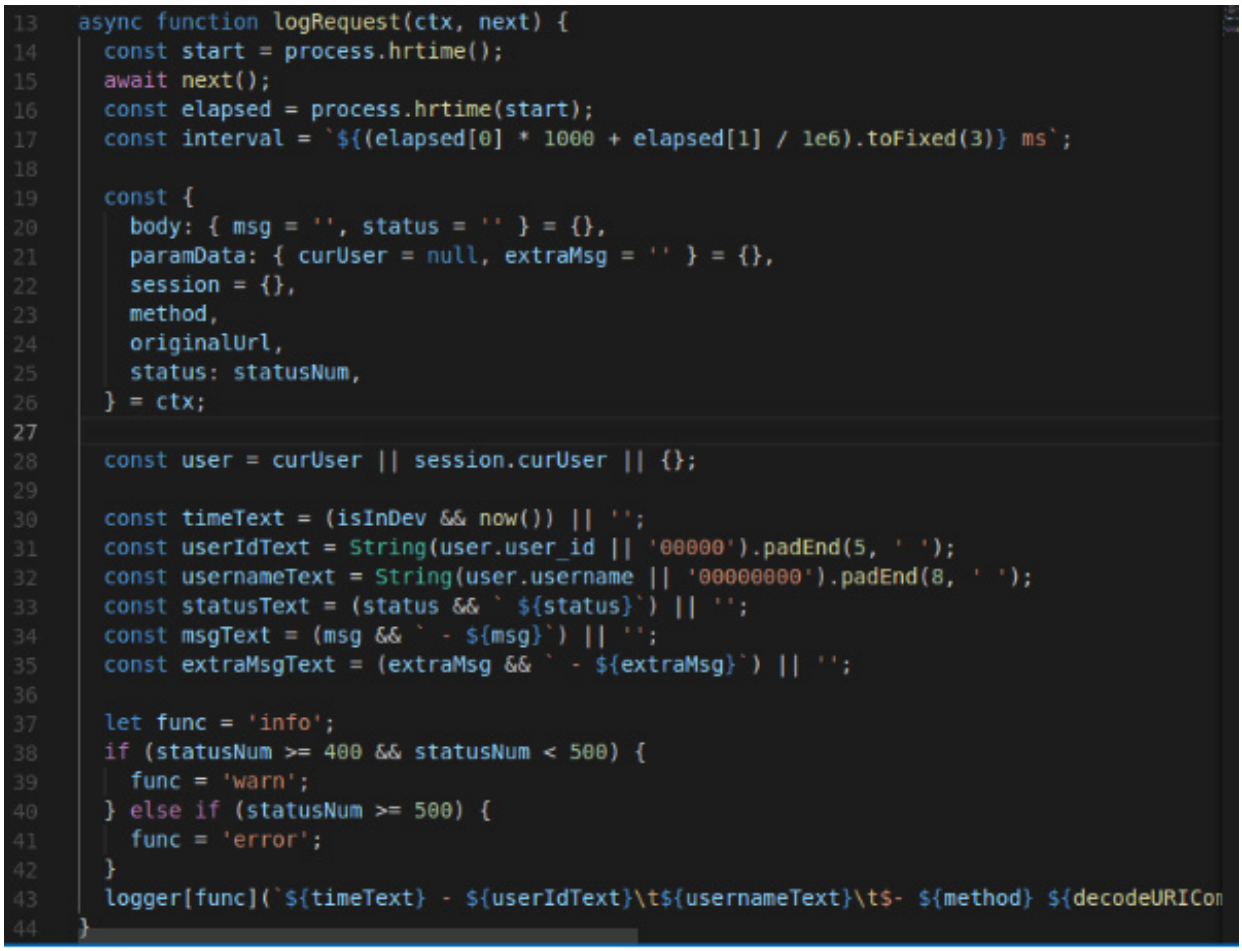
9.1 后端目录结构

```
├── README.md
├── database
|   └── bottle-ddl.sql
├── deploy
|   └── prod.json
├── docs
|   ├── api.html
|   └── api.md // RESTful API文档
├── package-lock.json
```

```
├── package.json // 依赖包
└── src
    ├── config
    |   ├── env // 环境配置
    |   |   ├── development.js // 开发环境配置文件
    |   |   └── production.example.js // 生产环境配置文件
    |   └── index.js
    ├── controllers // 控制器
    |   ├── bottle.js
    |   ├── session.js
    |   └── user.js
    ├── index.js
    ├── models // 数据持久化层
    |   ├── bottle.js
    |   └── user.js
    ├── routers // 路由层
    |   ├── bottle.js
    |   ├── index.js
    |   ├── session.js
    |   └── user.js
    ├── services // service
    |   ├── db // mysql
    |   |   └── index.js
    |   └── redis // cache
    |       ├── ClientManager.js
    |       ├── index.js
    |       ├── koa-redis-store.js
    |       └── session.js
    └── utils // 基础模块
        ├── AppError.js // 异常
        ├── index.js
        └── logger.js // logger
```

9.2 Logger

logger负责请求日志和出错日志的记录



```
async function logRequest(ctx, next) {
  const start = process.hrtime();
  await next();
  const elapsed = process.hrtime(start);
  const interval = `${(elapsed[0] * 1000 + elapsed[1] / 1e6).toFixed(3)} ms`;

  const {
    body: { msg = '', status = '' } = {},
    paramData: { curUser = null, extraMsg = '' } = {},
    session = {},
    method,
    originalUrl,
    status: statusNum,
  } = ctx;

  const user = curUser || session.curUser || {};

  const timeText = (isInDev && now()) || '';
  const userIdText = String(user.user_id || '00000').padEnd(5, ' ');
  const usernameText = String(user.username || '00000000').padEnd(8, ' ');
  const statusText = (status && ` ${status}`) || '';
  const msgText = (msg && ` - ${msg}`) || '';
  const extraMsgText = (extraMsg && ` - ${extraMsg}`) || '';

  let func = 'info';
  if (statusNum >= 400 && statusNum < 500) {
    func = 'warn';
  } else if (statusNum >= 500) {
    func = 'error';
  }
  logger[func](`${timeText} - ${userIdText}\t${usernameText}\t$- ${method} ${decodeURICon
}
```

(src/utils/logger.js)

9.3 AppError

AppError模块中定义SoftError和HardError，均继承于基类AppError，各自实现捕获时的处理逻辑。SoftError返回给用户提示信息，HardError在log中打印错误栈信息，而对用户隐藏出错详情。同时，利用koa的中间件机制在合适的位置处理AppError。最后，监听app的error事件，处理相应异常以防进程因出错的用户请求而退出。

```
class AppError extends Error {
  /**
   * 构造 App 异常
   * @param  {string}        status
   * @param  {string}        msg
   * @param  {Error|string}  [e]
   * @param  {number}        [code]
   * @param  {any}           [data]
   */
  constructor(status, msg, code = undefined, e = undefined, data = undefined) {
    let stack = null;
    const was = typeof e === 'object' ? 'error' : typeof e;
    if (e instanceof Error) {
      stack = e.stack.split('\n');
      stack[0] = '--------------------';
      stack = stack.join('\n');
    } else {
      e = e || msg;
    }
    super(e);
    Object.keys(e).forEach((key) => {
      this[key] = e[key];
    });
    this.was = was;
    this.name = Reflect.getPrototypeOf(this).constructor.name;
    this.info = { status, msg };
    if (code !== undefined) Object.assign(this.info, { code });
    if (data !== undefined) Object.assign(this.info, { data });
    this.stack += `\n${stack}\nInfo: ${JSON.stringify(this.info)}`;
    delete this.name;
  }
```

(src/utils/AppError.js)

```
  async function handleException(ctx, next) {
    try {
      await next();
    } catch (e) {
      if (e instanceof AppError.SoftError) return sendData(ctx, e);
      return handleError(ctx, e);
    }
  }
```

(src/utils/index.js)

9.4 ClientManager

ClientManager是一个工厂类，负责管理redis连接，可以用来使每次查询都使用同个redis连接，避免连接的频繁创建和销毁，减少资源占用

```
class ClientManager {
  constructor() {
    this.connections = [];
  }

  getClient(id, config = {}) {
    const self = this;
    if (self.connections[id]) {
      return self.connections[id];
    }
    const client = ClientManager.createClient(config);
    self.connections[id] = client;
    return client;
  }

  static createClient(config = {}) {
    config = assign(ClientManager.getDefaultConfig(), config);
    const client = redis.createClient(config);
    client.on('error', (e) => {
      logger.error('[redis climgr] Redis 查询出错', e.stack);
    });
    return client;
  }

  static getDefaultConfig() {
    return redisConfig;
  }
}
```

(src/services/redis/ClientManager.js)

9.5 koa-redis-store

koa-redis-store是中间件koa-redis的redis适配器

```
class RedisStore extends BaseRedisStore {
  /**
   * 构造 RedisStore
   * @param {object} options
   */
  constructor(options) {
    super(options);
    this.namespace = options.namespace;
  }

  genSid(sid) {
    const self = this;
    return `${self.namespace}:${sid}`;
  }

  get(sid, ...args) {
    const self = this;
    return super.get(self.genSid(sid), ...args);
  }

  set(sid, ...args) {
    const self = this;
    return super.set(self.genSid(sid), ...args);
  }

  destroy(sid, ...args) {
    const self = this;
    return super.destroy(self.genSid(sid), ...args);
  }
}
```

(src/services/redis/koa-redis-store.js)

9.6 DB模块

db模块负责mysql连接的创建和查询的执行，捕获mysql相关的错误并包装成HardError抛出。

```
/**
 * 在数据库连接池中申请连接
 * @return {Promise<Object>}       连接
 */
async function getConn() {
  try {
    const conn = await Pool.getConnectionAsync();
    return conn;
  } catch (e) {
    throw new AppError.HardError(AppError.INTERNAL_ERROR, '数据库连接出错', 500, e);
  }
}

/**
 * 查询数据库
 * @param  {string}              sql       数据库查询语句
 * @param  {any[]}               values    参数值
 * @param  {Object}              [conn]    数据库连接。传入时使用该连接，不传入的话直接用Pool
 * @return {Promise<any|any[]>}               查询结果
 */
async function queryDb(sql, values, conn) {
  // 若传入连接则使用该连接，否则默认使用连接池
  if (!conn) conn = Pool;
  try {
    return await conn.queryAsync(sql, values);
  } catch (e) {
    e.stack += `\n---------\n${e.sqlmessage}:\n${e.sql}\n---------`;
    throw new AppError.HardError(AppError.INTERNAL_ERROR, '数据库查询出错', 500, e);
  }
}
```

(src/services/db/index.js)

9.7 用户验证模块

使用session-cookie机制记录用户的登录状态。全局上，通过blockUnauthorized中间件拦截未登录用户对白名单之外的api的访问

```
async function blockUnauthorized(ctx, next) {
  if (ctx.isInWhiteList || ctx.session.curUser) {
    return next();
  }
  throw new AppError.SoftError(AppError.UNAUTHORIZED, '未登录');
}
```

（src/routers/index.js）

局部上，每个api会对用户权限进行必要的检查

```
exports.retrieveOpenersOfOneBottle = async (ctx) => {
  const { bottle: { bottle_id, owner: { user_id: owner_id } } } = ctx.paramData;
  const { curUser: { user_id } } = ctx.session;
  if (owner_id !== user_id) {
    throw new AppError.SoftError(AppError.FORBIDDEN, '您不是该瓶子的主人');
  }
```

(src/controllers/bottle.js)

9.8 瓶子管理模块

bottle表记录瓶子的信息；bottle_open表记录瓶子的打开记录。业务逻辑上，同一用户反复打开同一瓶子，bottle_open只会记录第一条记录。对此的实现上，利用关系型数据库主键不能重复的特性，将<bottle_id, opener_id>作为bottle_open的主键，并编写以下代码，捕获主键重复的错误，保持静默

```
exports.openOneBottle = async (ctx) => {
  const { bottle, bottle: { bottle_id, owner: { user_id: owner_id } } } = ctx.paramData;
  const { curUser: { user_id } } = ctx.session;
  if (owner_id === user_id) {
    return sendData(ctx, bottle, 'OK', '打开漂流瓶成功');
  }
  try {
    await Bottle.openOneBottle(bottle_id, user_id);
  } catch (err) {
    if (err.code !== 'ER_DUP_ENTRY') {
      throw err;
    }
  }
```

(src/controllers/bottle.js)

10 可拓展性

- 在我们这个系统中，我们构造了大量基础模块，我们构建这些模块时均没有考虑具体的业务，目标是构造出普适的模块，方便系统扩展
- 吸收了MVC的思想，每个模块仅仅负责管理单一的实体，做到了**低耦合，高内聚**