寻宝猫软件架构文档（SAD）

1. 引言

1.1 目的及范围

该架构文档对寻宝猫 app 整体结构进行描述，本应用程序旨在建成一个山东大学 校内二手交易平台，用于校内师生进行空闲物品的交易。登陆方式采用山东大学统一认证方式登陆，保证交易人群均为校内人群。程序前端打算采用 ReactNative 实现，后端采用 NestJS 实现，易于维护，并且由较好的可扩展性。

1.2 文档结构

文档的组织结构如下：

第一部分 引言

本部分主要概述了文档内容组织结构，使读者能够对文档内容进行整体了解，并快速找到自己感兴趣的内容。同时，也向读者提供了架构交流所采用的视图信息。

第二部分 架构背景

本部分主要介绍了软件架构的背景，向读者提供系统概览，建立开发的相关上下文和目标。分析架构所考虑的约束和影响，并介绍了架构中所使用的主要设计方法，包括架构评估和验证等。

第三、四部分 视图及其之间的关系

视图描述了架构元素及其之间的关系，表达了视图的关注点、一种或多种结构。

第五部分 需求与架构之间的映射

描述系统功能和质量属性需求与架构之间的映射关系。

第六部分 附录

提供了架构元素的索引，同时包括了术语表、缩略语表。

1.3 视图编档说明

所有的架构视图都按照标准视图模板中的同一种结构进行编档。

2. 架构背景

2.1 系统概述

C/S 架构（客户机/服务器模式）是一种比较早的软件架构，架构主要分为两层，第一层是在客户机系统上结合了表示与业务逻辑，第二层是通过网络结合了数据库服务器。客户端服务器的点对点直接连接，使得业务更加安全，同时具有相应速度快、减少通信流量等优点。

2.2 架构需求

2.2.1 技术环境需求

基于 C/S 架构的工程最终实现一个 App，因此需要我们掌握编写前端和后端的能力，开发组决定前端使用 ReactNative 实现，而后端则采用 NestJS 实现。

2.2.2 功能需求

寻宝猫 App 最主要的功能就是用户浏览商品、发布商品、搜索商品以及进行聊天沟通。考虑到用户所发布商品需要健康合法，因此需要寻宝猫项目组能够加强对交易环境的检查。并且登陆方式采用山东大学统一认证方式登陆，保证交易人群为在校师生。

2.2.3 质量属性需求

(1) 寻宝猫 App 应当支持 100 人以上同时访问服务器，服务器响应时间短于 1 秒。

(2) 针对系统错误、恶意攻击等行为，采用设计故障转移方式，以及人工处理订单异常等方式进行避免。

(3) 将内部软件代码进行分层，各层之间低耦合，保证系统灵活性以及可扩展性。

(4) 尽可能引入少的新概念，采用向前兼容的方式，保证概念一致性。

(5) 推动数据文档格式统一标准的定义与实现，保证互操作性。

(6) 尽可能代码复用，保证可重用性，使系统整合更为方便。

(7) 根据 SOCK 模型思想，保证软件的可测试性。

2.3 主要设计决策及原理

寻宝猫系统前端采用 ReactNative 框架，整体上分为三大块，Native、JavaScript 与 Bridge：Native 管理 UI 更新及交互，JavaScript 调用 Native 能力实现业务功能，Bridge 在二者之间传递消息。即：最上层提供类 React 支持，运行在 JavaScriptCore 提供的 JavaScript 运行时环境中，Bridge 层将 JavaScript 与 Native 世界连接起来。具体的，Shadow Tree 用来定义 UI 效果及交互功能，Native Modules 提供 Native 功能（比如蓝牙），二者之间通过 JSON 消息相互通信。Bridge 层是 React Native 技术的关键，设计上具有 3 个特点：异步（asynchronous）：不依赖于同步通信。可序列化（serializable）：保证一切 UI 操作都能序列化成 JSON 并转换回来。批处理（batched）：对 Native 调用进行排队，批量处理。

后端采用采用 Nest 框架，是一个渐进式的 Node 框架。用于构建高效、可扩展的 Node 服务器端应用程序的框架。使用 TypeScript 构建，保留与纯 JS 的兼容性，集 OOP（面向对象编程），FP（函数式编程），FPP（响应式编程）一身。服务引擎盖默认使用它 Express，但是也提供与各种其他库的兼容性，例如 Fastify，允许轻松使用可用的无数第三方插件。

3. 视图

3.1 逻辑视图

3.1.1 顶层逻辑视图

3.1.1.1 主表示

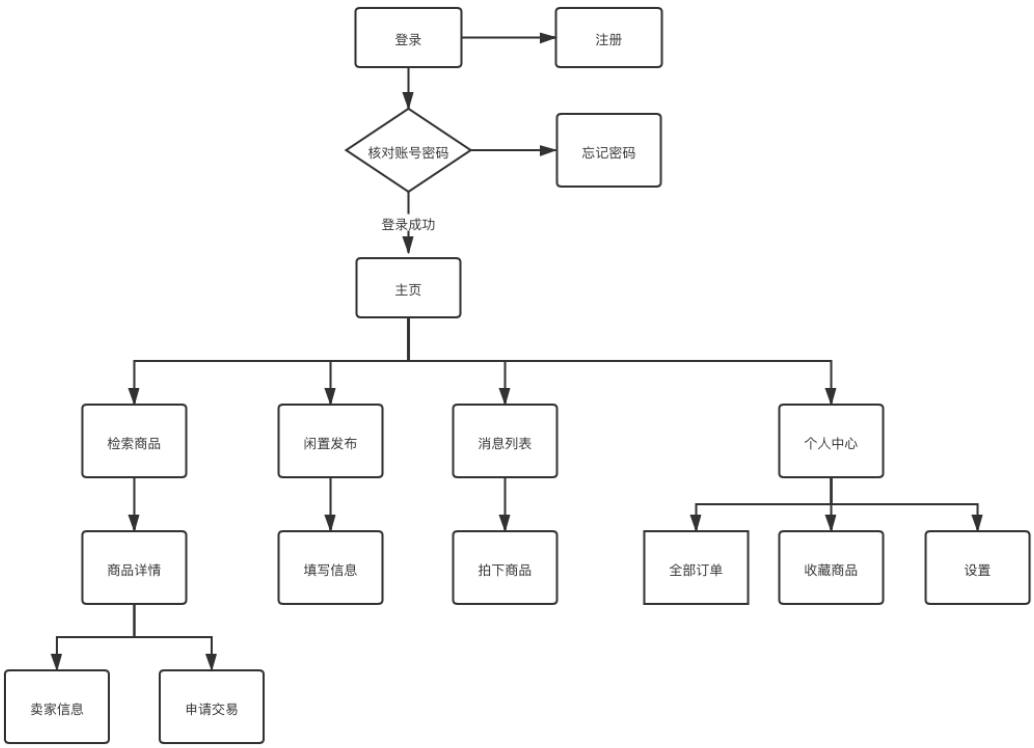


图 1　系统总逻辑视图

3.1.1.2 构件目录

A.构件及其特性

构件	*描述*
用户	需要包含用户 ID，学生姓名，学号，邮箱等注册信息。
搜索	用户可以根据商品名称、商品类别进行搜索。在搜索过程中，还可以根据商品热度（访问量、收藏量）等进行排序。

商品	需要包含商品 ID、商品名称、商品定价、商品类别、商品名称以及至少一张商品图片。另外可以选择性补充商品购买价格、商品新旧程度、商品购买渠道以及商品库存等信息。
卖家空间	陈列卖家的相关信息，包括关注人数、商品成交量、以及目前在售的全部商品。同时提供发起聊天、空间内搜索、分享卖家等功能操作。
个人中心	用户管理界面，提供全部订单、商品收藏、在售商品管理、个人资料编辑等功能入口。
全部订单	陈列该用户作为买家和卖家两种身份参与的所有订单，支持按交易状态（已拍下、待确认、交易成功）查找。
商品收藏	陈列该用户收藏的所有商品。

3.1.1.3 上下文图

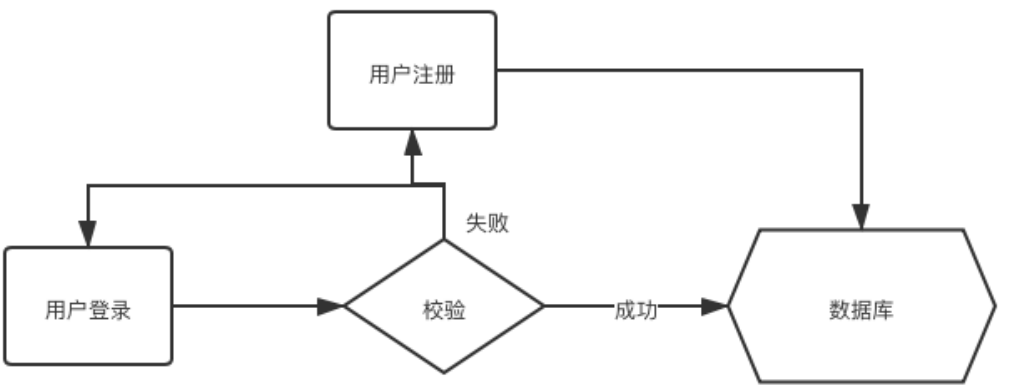


图 2　注册/登录系统逻辑视图

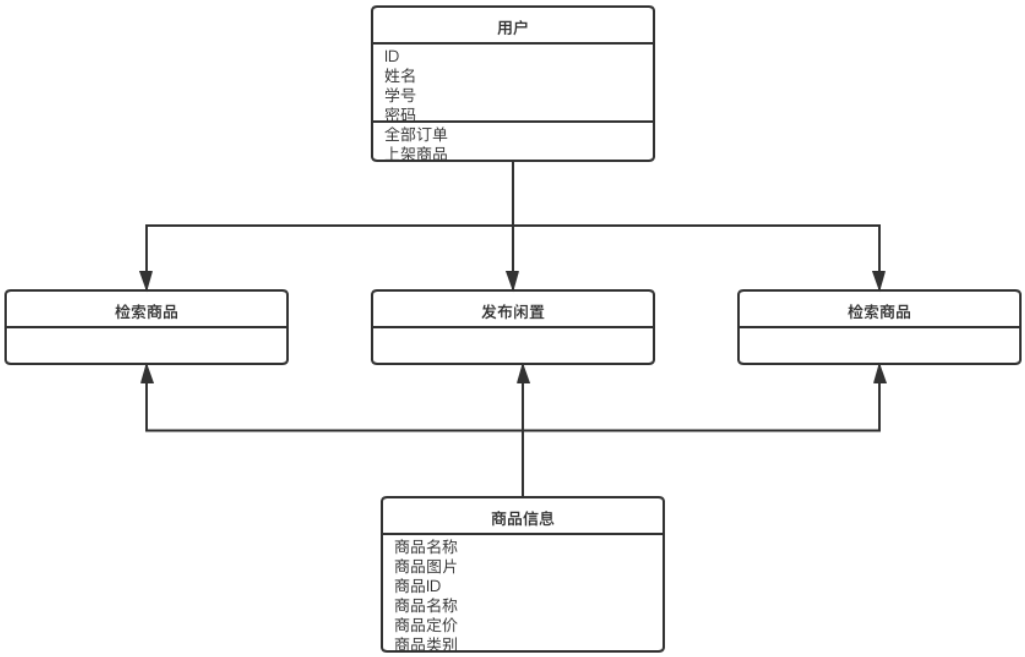


图 3　商品系统逻辑视图

3.1.2 可变性

我们采用面向对象的编程方式，代码复用性高，可变性强，后期可通过新增类来进行功能的增加。

3.1.3 原理

通过合理的交易模式，使得交易更加规范。本系统添加了“买家拍下、卖家二次确认”的交易流程，每次都必须完成交易流程，才能达成交易。保证了买卖双方的交易过程的规范，交易时间的可控。

通过高效的商品查询，使得交易更加便捷。本系统通过分类的方式，将商品分为多类，用户可以很方便的根据分类挑选自己喜欢的商品。同时，用户也可以通过直接搜索的方式，搜索商品的名称，找到自己心仪的商品。

通过科学的买卖机制，使得交易更加安全。本系统通过“二次确认”的模式，确保买卖双方已经达成购买意向，并设置了待收货的模块，确保买卖双方最终完成交易。并且，买卖双方均可举报交易中的不正当行为，保护自己的合法权益不被侵犯。

通过简洁的商品页面，使得交易更加直观。本系统 ui 界面美观大方，商品逻辑井井有条，没有冗余的信息和多余的广告。用户只需点击“我想要”按钮，即可与卖家进行沟通，非常的直观，清晰可见。

3.1.4 相关视图

无

3.2 开发视图

3.2.1 顶层开发视图

3.2.1.1 主表示

针对开发的系统情况，确定了如下图所示的开发技术架构，下文详细说明了前后端的交互和服务关系：

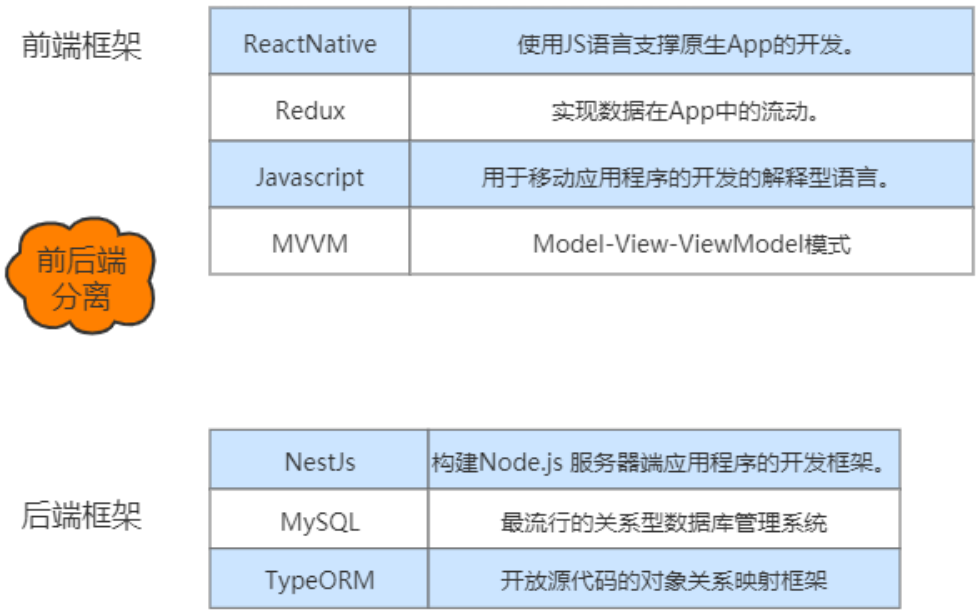


图 4　软件开发架构设计

3.1.1.2 构件目录

A. 前端构件

前端构件	*描述*
ReactNative	整体上分为三大块，Native、JavaScript 与 Bridge：Native 管理 UI 更新及交互，JavaScript 调用 Native 能力实现业务功能，Bridge 在二者之间传递消息。即：最上层提供类 React 支持，运行在 JavaScriptCore 提供的 JavaScript 运行时环境中，Bridge 层将 JavaScript 与 Native 世界连接起来。具体的，Shadow Tree 用来定义 UI 效果及交互功能，Native Modules 提供 Native 功能（比如蓝牙），二者之间通过 JSON 消息相互通

	信。Bridge 层是 React Native 技术的关键，设计上具有 3 个特点：异步（asynchronous）：不依赖于同步通信。可序列化（serializable）：保证一切 UI 操作都能序列化成 JSON 并转换回来。批处理（batched）：对 Native 调用进行排队，批量处理。
Javascript	JavaScript（通常缩写为 JS）是一种高级的、解释型的编程语言。JavaScript 是一门基于原型、头等函数的语言，是一门多范式的语言，它支持面向对象程序设计，命令式编程，以及函数式编程。它提供语法来操控文本、数组、日期以及正则表达式等，不支持 I/O，比如网络、存储和图形等，但这些都可以由它的宿主环境提供支持。它已经由 ECMA（欧洲电脑制造商协会）通过 ECMAScript 实现语言的标准化。它被世界上的绝大多数网站所使用，也被世界主流浏览器（Chrome、IE、Firefox、Safari、Opera）支持。
Redux	Redux 是一个用来管理管理数据状态和 UI 状态的 JavaScript 应用工具。随着 JavaScript 单页应用（SPA）开发日趋复杂，JavaScript 需要管理比任何时候都要多的 state（状态），Redux 就是降低管理难度的。（Redux 支持 React，Angular、jQuery 甚至纯 JavaScript）。
MVVM	MVVM（Model－view－viewmodel）是一种软件架构模式，是 MVC 的改进。 MVVM 有助于将图形用户界面的开发与业务逻辑或后端逻辑（数据模型）的开发分离开来。在前端页面中，把 Model 用纯 JavaScript 对象表示，View 负责显示，两者做到了最大限度的分离。ViewModel 负责把 Model 的数据同步到 View 显示出来，还负责把 View 的修改同步回 Model。

B.后端构件

控件类型	*后端控件*	*介绍*
数据库	MySQL	MySQL 原本是一个开放源码的关系数据库管理系统，原开发者为瑞典的 MySQL AB 公司， 2009 年，甲骨文公司（Oracle）收购昇阳微系统公司，MySQL 成为 Oracle 旗下产品。MySQL 在过去由于性能高、成本低、可靠性好，已经成为最流行的开源数据库，因此被广泛地应用在 Internet 上的中小型网站中。随着 MySQL 的不断成熟，它也逐渐用于更多大规模网站和应用，比如维基百科、Google 和 Facebook 等网站。
后端	TypeORM	TypeORM 是一个开放源代码的对象关系映射框架，对

服务框架		JDBC 进行了轻量级的对象封装,是一个全自动的 ORM 框架，TypeORM 可以自动生成 SQL 语句，自动执行。
	NestJs	Nest （NestJS） 是一个用于构建高效、可扩展的 Node.js 服务器端应用程序的开发框架。它利用 JavaScript 的渐进增强的能力，使用并完全支持 TypeScript （仍然允许开发者使用纯 JavaScript 进行开发），并结合了 OOP （面向对象编程）、FP （函数式编程）和 FRP （函数响应式编程）。

3.2.2 构件连接方法

A.前后端分离架构

在前后端分离的应用模式中，后端仅返回前端所需的数据，不再渲染 HTML 页面，不再控制前端的效果。至于前端用户看到什么效果，从后端请求的数据如何加载到前端中，都由前端自己决定，网页有网页的处理方式，App 有 App 的处理方式，但无论哪种前端，所需的数据基本相同，后端仅需开发一套逻辑对外提供数据即可。下图即前后端分离的基本模式：

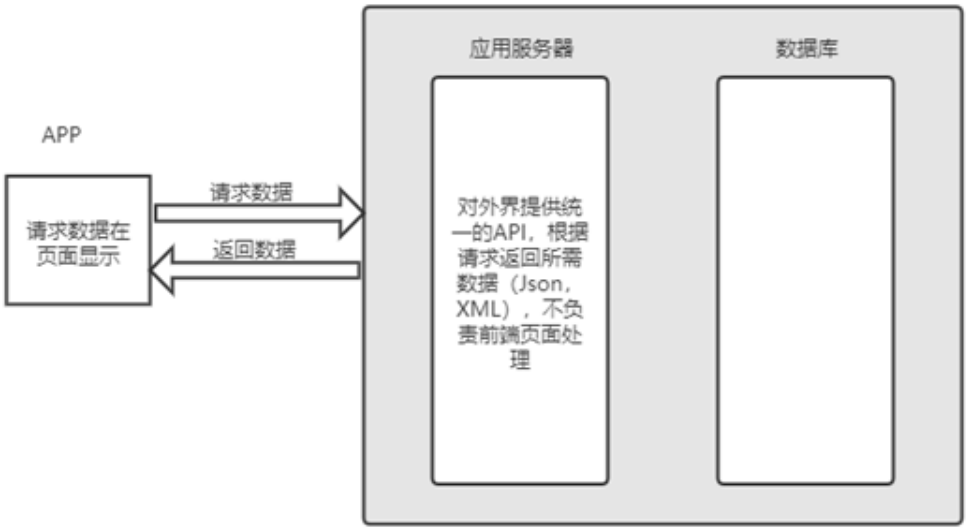


图 5　前后端分离架构设计

根据 MVVM 架构，能够清楚的展现出具体的前后端分离情况。

前端：负责 View 和 Controller 层。

后端：只负责 Model 层，业务 / 数据处理等。

B.服务器架构

传统的服务器架构基于 MVC 模式，通过 model，view 和 controller 之间的交互实现

数据流转。而 Nest 架构从客户端发送请求开始，通过 controller 进行处理，并注入依赖，发送到服务器，采用 DAO 类实现与数据库之间的数据访问和交换。

3.2.3 技术原理

3.2.3.1 前端原理

React Native 框架中，JSX 源码通过 React Native 框架编译后，通过对应平台的 Bridge 实现了与原生框架的通信。如果我们在程序中调用了 React Native 提供的 API，那么 React Native 框架就通过 Bridge 调用原生框架中的方法。 因为 React Native 的底层为 React 框架，所以如果是 UI 层的变更，那么就映射为虚拟 DOM 后进行 diff 算法，diff 算法计算出变动后的 JSON 映射文件，最终由 Native 层将此 JSON 文件映射渲染到原生 App 的页面元素上，最终实现了在项目中只需要控制 state 以及 props 的变更来引起 iOS 与 Android 平台的 UI 变更。

3.2.2.2 后端原理

网络服务器解析浏览器的请求后从数据库获取资源，将生成的文件封装至响应包中，返回至浏览器解析浏览器解析 HTTP 响应后，下载文件，继而根据文件内包含的外部引用文件、图片或者多媒体文件等逐步下载，最终将获取到的全部文件渲染成完整的页面。

3.2.4 相关视图

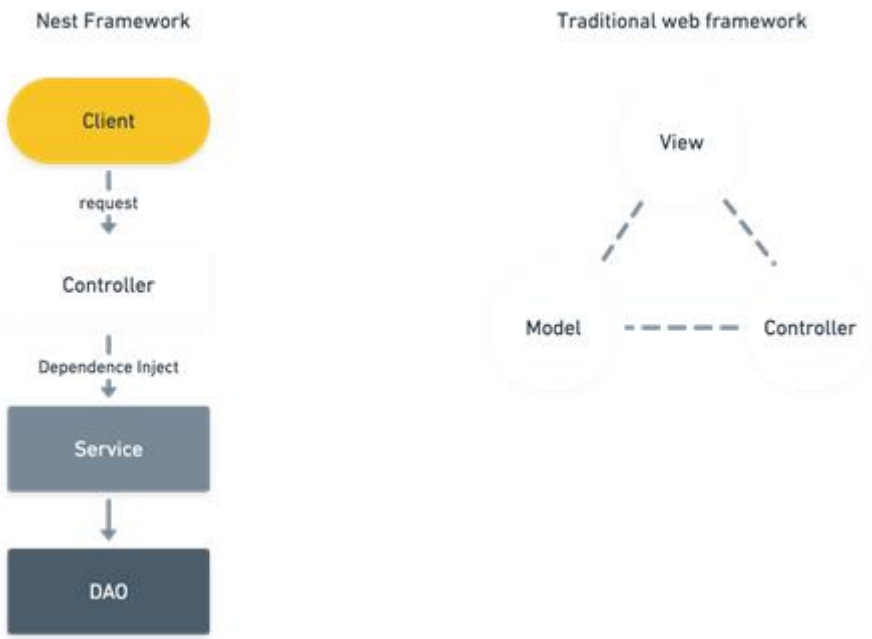


图 6　服务器架构视图

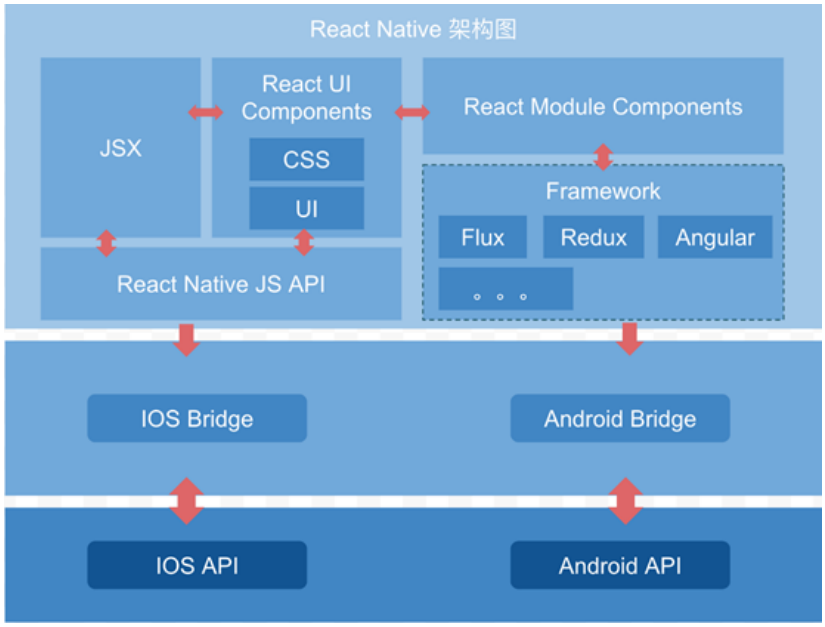


图 7　ReactNative 架构视图

3.3 运行视图

3.3.1 顶层运行视图

3.3.1.1 主表示

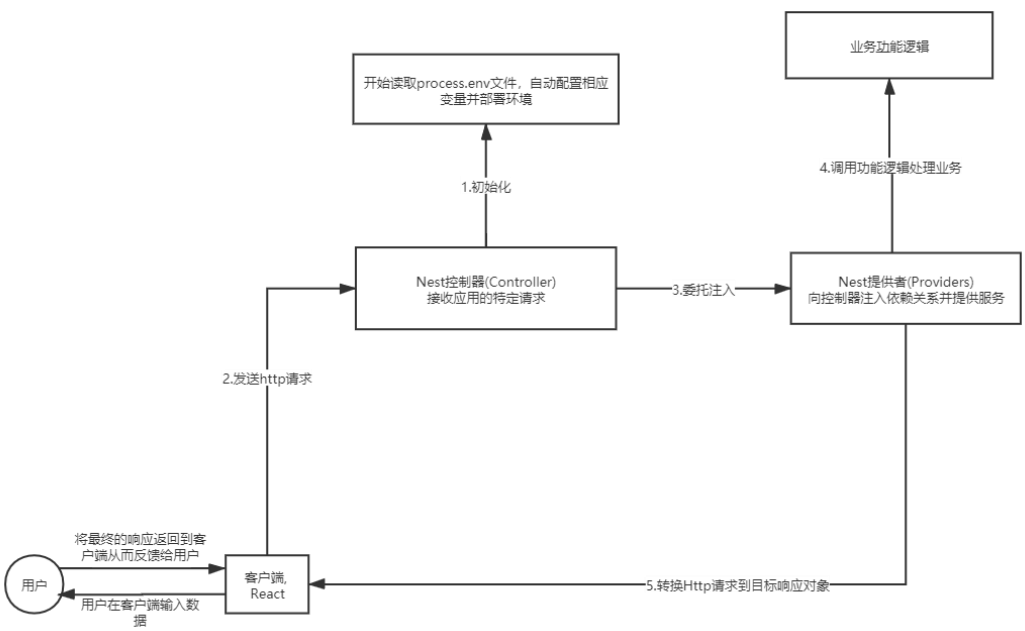


图 8　寻宝猫系统运行视图

3.3.1.2 构件目录

A.构件及其特性

构件	*描述*
NestJs	Nest 是一个用于构建高效，可扩展的 Node.js 服务器端应用程序的框架。它使用渐进式 JavaScript，内置并完全支持 Typescript（但仍然允许开发人员使用纯 JavaScript 编写代码）并结合了 OOP（面向对象编程），FP（函数式编程）和 FRP（函数式响应编程）的元素。
React-Native	总体上分为三大块，Native、JavaScript 与 Bridge：Native 管理 UI 更新及交互，JavaScript 调用 Native 能力实现业务功能，Bridge 在二者之间传递消息。
TypeORM	TypeORM 是一个开放源代码的对象关系映射框架，对 JDBC 进行了轻量级的对象封装，是一个全自动的 ORM 框架，TypeORM 可以自动生成 SQL 语句，自动执行。

B.关系及其特性

NestJs

1、使用者通过客户端发送 HTTP 协议的数据请求。

2、HITP 请求到达服务器后，经过控制器，路由机制控制哪个控制器接收哪些请求

3、控制器应处理 HTTP 请求并将更复杂的任务委托给 providers。Providers 是纯粹的 JavaScript 类。

4、Providers 处理具体业务请求，Providers 是 Nest 的一个基本概念，service，repository，factory，helper 等等都可能被视为 providers

5、将业务逻辑经**过转换形**成 JSP 业务进行处理，并将处理结果返回

6、用户在客户端或者浏览器上得到 HTTP 请求的响应。

Nest 有一个内置的控制反转（"IoC"）容器，可以解决 providers 之间的关系。 此功能是上述依赖注入功能的基础，但要比上面描述的要强大得多。@Injectable() 装饰器只是冰山一角，并不是定义 providers 的唯一方法。

TypeORM

TypeORM，负责跟数据库的交接。通过持久化数据对象，进行对象关系的映射，并以对象的角度来访问数据库。不同于现有的所有其他 JavaScript ORM 框架，TypeORM 支持 Active Record 和 Data Mapper 模式，这意味着你可以以最高效的方式编写高质量的、松耦合的、可扩展的、可维护的应用程序。

React-Native

JavaScriptCore + ReactJS + Bridges 就成了 React Native。**JavaScriptCore** 负责 JS 代码解释执行。ReactJS 负责描述和管理 VirtualDom,指挥原生组件进行绘制和更新,同时很多计算逻辑也在 js 里面进行。**ReactJS** 自身是不直接绘制 UI 的，UI 绘制是非常耗时的操作，原生组件最擅长这事情。**Bridges** 用来翻译 ReactJS 的绘制指令给原生组件进行绘制，同时把原生组件接收到的用户事件反馈给 ReactJS。

C.构件接口

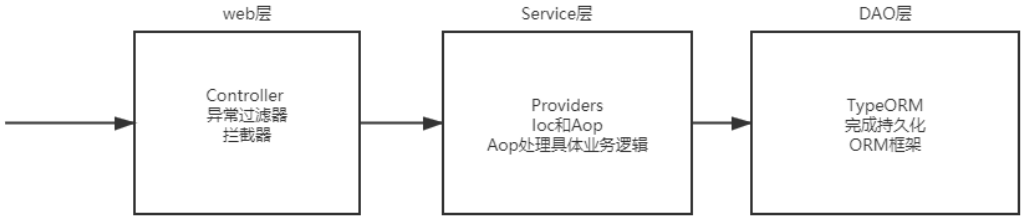


图 9　Nestjs 运行视图

D.构件行为

NestJs 行为

1.初始化一个指向 Servlet 容器（例如 Tomcat）的请求

2.这个请求经过一系列的过滤器（Filter）。

3.接着 Controller 被调用，Controller 将请求委托给 Providers，Providers 来决定这个请是否需要调用某个 Service 等。

4.如果 Providers 决定需要调用某个 Service，Providers 把请求的处理交给 Service

7. Providers 实例使用命名模式来调用，在调用 Service 的过程前后，涉及到相关拦截器（Intercepter）的调用。

8.返回响应的结果

TypeORM 行为

1.通过 config.buildSessionFactory()得到 sessionFactory。

2.sessionFactory.openSession()得到 session。

3.session.beginTransaction()开启事务。

4.persistent operate;

5.session.getTransaction().commit()提交事务

6.关闭 session;

7.关闭 sessionFactory;

React-Native 行为

1. ReactInstanceManager 创建时会配置应用所需的 java 模块与 js 模块，通过 ReactRootView 的 startReactApplication 启动 APP。

2. 在创建 ReactInstanceManager 同时会创建用于加载 JsBundle 的 JSBundlerLoader，并传递给 CatalystInstance。
3. CatalystInstance 会创建 Java 模块注册表及 Javascript 模块注册表，并遍历实例化模块。
4. CatalystInstance 通过 JSBundlerLoader 向 Node Server 请求 Js Bundle，并传递给 JSCJavaScriptExectutor，最后传递给 javascriptCore，再通过 ReactBridge 通知 ReactRootView 完成渲染
5. ReactRootView 创建负责 React Native 和 Native 通信的 RCTBridge 实例的初始化。
6. 初始化真正展示视图的 RCTRootContentView。
7. 通过 runApplication 方法把必要的参数（moduleName, params）传给 JS 侧的 AppRegistry 的 runApplication 方法，从而运行起了 React Native 的功能。

3.3.1.3 上下文图

无

3.3.2 可变性

无

3.3.3 原理

无

3.3.4 相关视图

无

3.4 部署视图

部署图描述的是系统运行时的结构，展示了系统的硬件的配置及其软件如何部署到网络结构中。

3.4.1 顶层部署视图

3.4.1.1 主表示

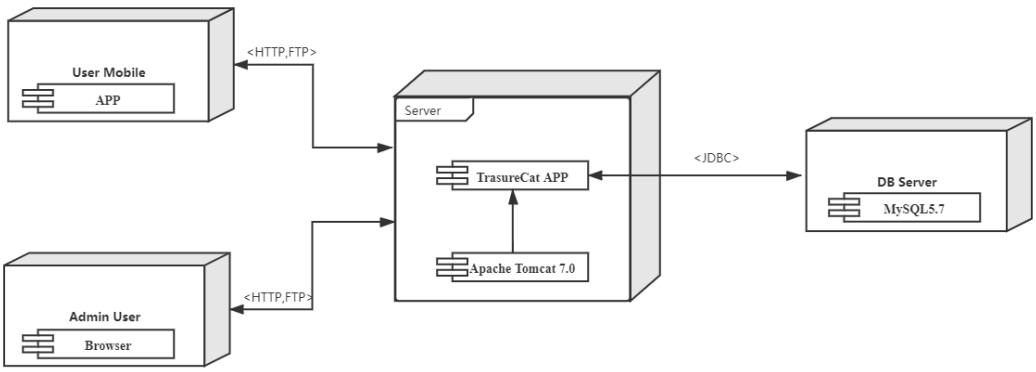


图 10　系统运行部署视图

3.4.1.2 构件目录

A.构件及其特性

构件	*描述*
User Mobile	User Mobile 代指使用寻宝猫 APP 的普通用户，拥有普通用户权限，包括：浏览商品信息，发布商品，购买商品，与其他用户聊天等权限。用户使用移动设备登陆并使用 APP，支持的设备包括：Andriod 手机及平板，ios 手机，ipad 等其他移动设备
Admin User	Admin User 为寻宝猫 APP 的系统管理员用户，通过浏览器访问 APP 后台，拥有管理员权限，包括：维护数据库，管理用户，发布系统公告等权限。支持的浏览器包括 IE、Fire Fox、Google Chrome 等。
Server	寻宝猫 APP 的服务器，用于运行寻宝猫 APP 系统。系统部署在 Tomcat7.0 容器上，它与数据服务器通过 jdbc 连接。
DB Server	数据库服务程序，用于处理对数据库进行的操作
TrasureCat APP	寻宝猫 APP 系统
Apache Tomcat 7.0	Tomcat 服务器是一个免费的开放源代码的 Web 应用服务器，最新的 Servlet 和 JSP 规范总是能在 Tomcat 中得到体现。
MySQL5.7	一个开放源码的小型关系式数据库管理系统，体积小、速度快、总体成本较低。

B.关系及其特性

用户通过打开移动设备上的 APP 软件对寻宝猫服务器进行访问，所以用户首先需要下载寻宝猫 APP 文件，之后还需要连接到互联网，才可以访问寻宝猫服务器，。服务器端还需要提供 XXX 数据库等。

用户与服务器之间的通讯主要体现在两方面：传输聊天文本和传输商品信息。其中，商品信息中包含图片数据，因此，在用户与服务器之间进行通讯的时候，需要用到 HTTP 协议和 FTP 协议。

系统部署在 Tomcat7.0 容器上，它与数据服务器通过 JDBC 连接。JDBC 是 JAVA 数据库连接，一套为数据库存取编写的 Java API。

C.构件接口

客户端与服务端的连接可以是局域网或互联网。

D.构件行为

User Moblie：指使用寻宝猫 APP 的普通用户，拥有普通用户可以搜索商品，浏览商品信息，收藏商品，发布商品，购买商品，与其他用户聊天，管理商品历史信息。并且也可以对个人信息进行修改。

Admin User：指寻宝猫 APP 的系统管理员用户，通过浏览器访问 APP 后台。该类用户主要负责对系统的维护和管理，包括管理已发布商品信息，管理用户信息，发布系统公告。

APP：指运行在普通用户移动设备上的应用程序用户界面。普通用户通过打开 APP 获取对应的用户权限，进入系统完成操作。

Server：指系统服务器。

TrasureCat APP：指寻宝猫 APP 应用程序。该应用程序主要有 3 部分的处理功能：1.处理用户登录/注册操作。2.接收用户提交的数据，根据用户传来的数据，对数据库执行相应的操作，包括：1.查询数据库，返回满足条件的信息数据。2.向数据库中添加用户传输的商品信息。3.查询指定商品信息，将商品信息附加标记，指示该商品已被用户收藏/拍下。3.接收用户发来的聊天内容，转发给另一个用户。

Apache Tomcat 7.0: 该组件表示 Web 应用服务器。系统部署在 Tomcat7.0 容器上，Tomcat 不仅仅是一个 Servlet 容器，它也具有传统的 Web 服务器的功能：处理 html 页面。

MySQL5.7: 该组件表示数据库管理系统。一个开放源码的小型关联式数据库管理系统。寻宝猫系统用来存储管理员和用户信息，保存商品信息；存储用户收藏，拍下商品信息；存储聊天记录信息等。

3.4.1.3 上下文图

无

3.4.2 可变性

1.客户端的可兼容性：在开发阶段，我们团队在前端使用 React-Native 框架，在 Windows10 平台上进行开发，并使用该框架生成原生 Andriod 和 ios 系统的代码，因此，寻宝猫 APP 可以支持多系统的兼容。

2.响应速度：在 APP 上线初期，只会对很少一部分学生进行开放，因此，在初期系统的响应速度较快。之后随着 APP 的服务量逐渐增加，平台对用户的响应速度会变慢，但由于寻宝猫 APP 面向的用户是山东大学的在校学生，受众人数相对较少，因此影响速度不会非常明显。但是，当发现 APP 的响应速度降低较多时，也需要对 APP 的框架和需求处理算法进行迭代，以便维持一个正常的响应速度。

3.服务质量：在 APP 上线初期，整个 APP 处于测试阶段，再加上此时服务的人数不多，上线的系统可能会是一个相对简陋的系统，对于安全性，操作的便捷性，数据的准确性等方面可能存在一些质量问题。但是在后期，随着 APP 的不断迭代，整个系统的处理流程会不断简化，对于数据通信的算法会不断打磨，安全性措施会不断进行完善，以上问题基本会得到不同程度的解决。

4.服务器容量：在 APP 上线初期，由于服务人数较少，服务器的容量是足够的，但在后期，当人数增加到一定程度的时候，可能会出现同时在线人数较多，服务器超载导致服务质量变差的问题。此时可以考虑扩容服务器。

5.平台灵活性：需要能够较好地支持如上所述的改进需求，保证技术团队在后续 的迭代过程中可以更加便捷地完成版本的兼容和技术的重构。

3.4.3 原理

本软件的部署视图是根据 UML 部署图所产生的。UML 部署图是描述的是系统运行时的结构，展示了硬件的配置及其软件如何部署到网络结构中。一个系统模型只有一个部署图，部署图通常用来帮助理解分布式系统。基于 UML 部署图，可以很好的了解到软件的物理网络配置，对软件的可靠性也能有清楚的反应。

整体上说，整个 APP 的模型是一个客户端-服务器模型，用户通过 APP 访问服务器，服务器通过查询服务器向用户返回结果并显示。

由于本 APP 面向的是山东大学的在校生，且软件的登陆系统是连接到山东大学的统一认证平台，因此，用户的受众层面比较单一，不需要对用户权限进行区分。

3.4.4 相关视图

参考的 UML 部署图：

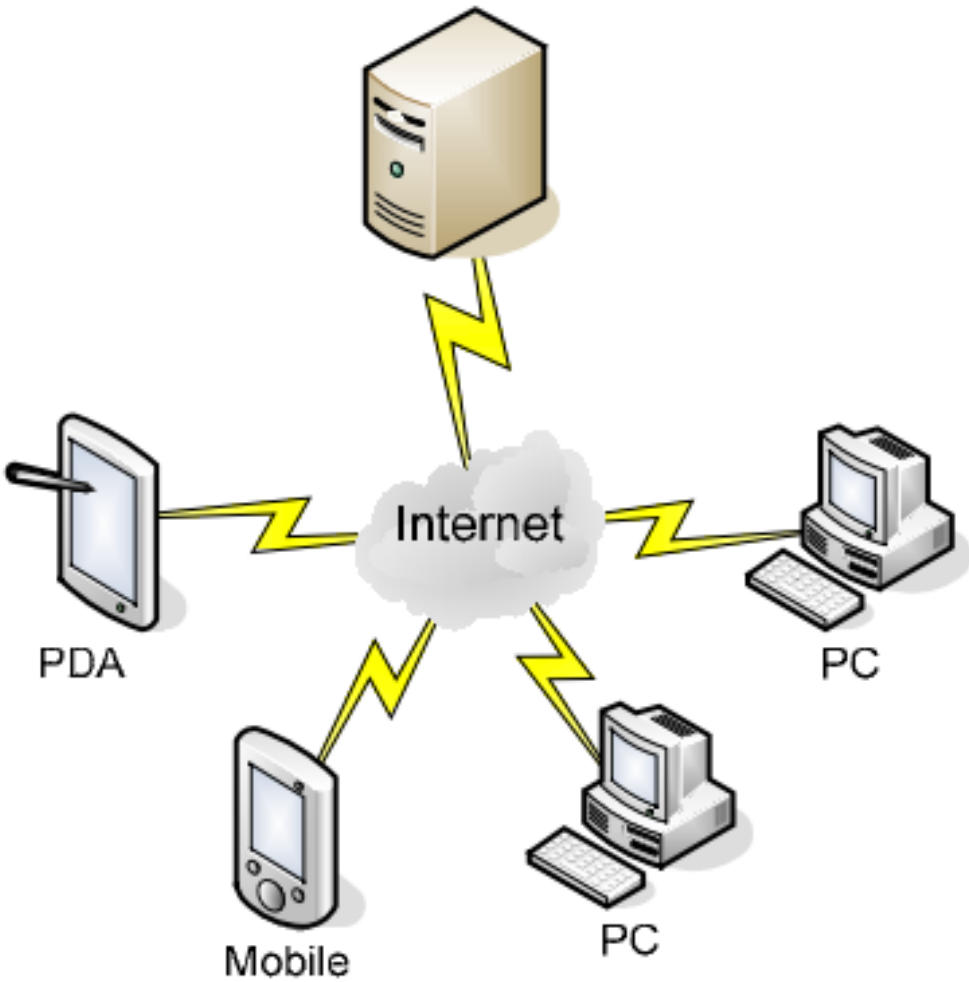


图 11　系统网络结构图

3.5 用例视图

3.5.1 顶层用例视图

3.5.1.1 主表示

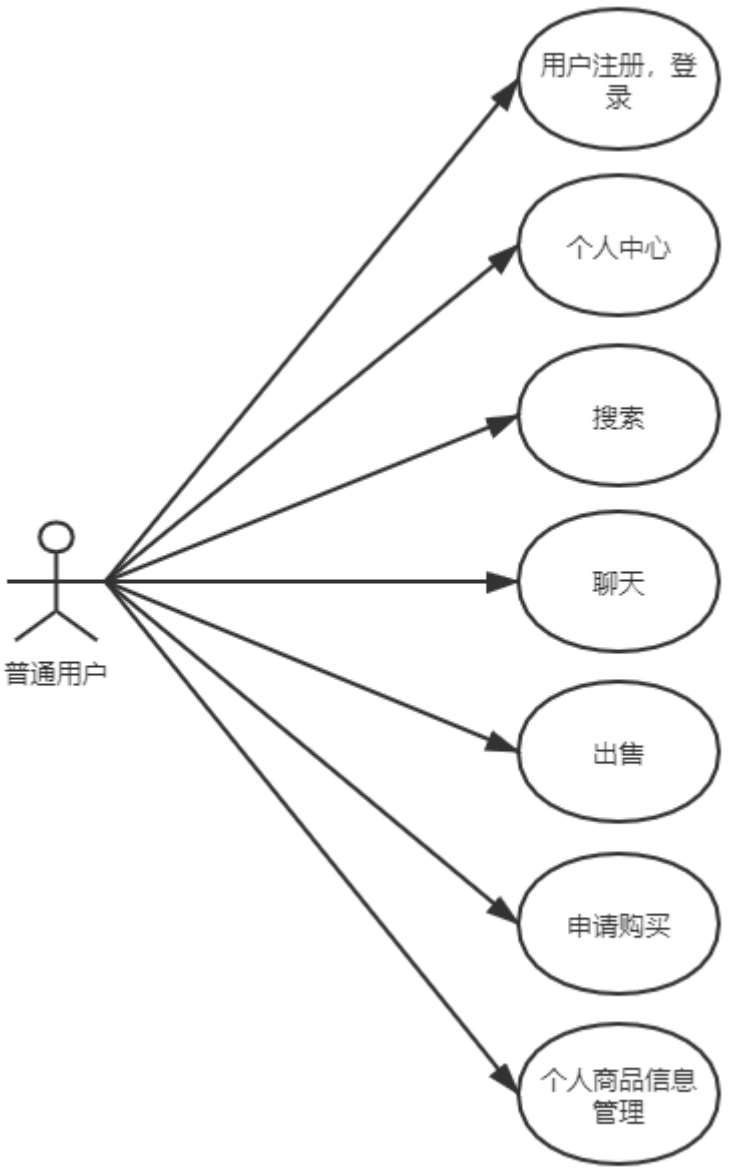


图 12　普通用户实例视图

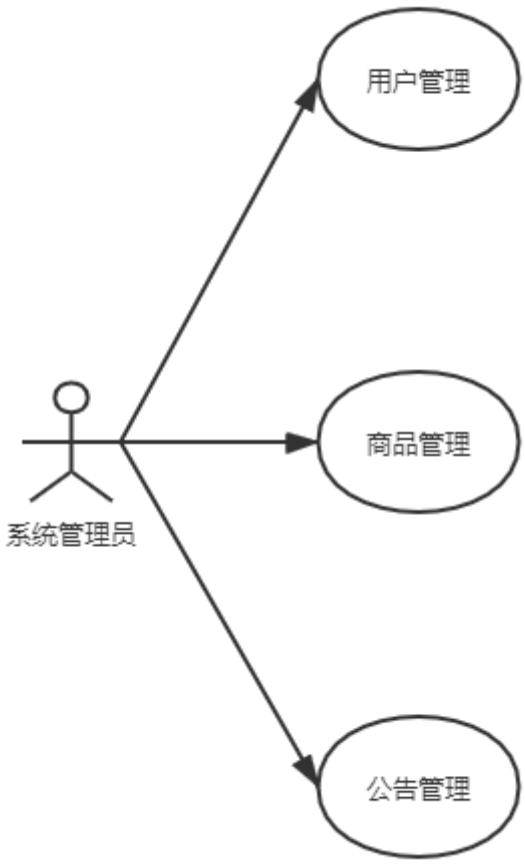


图 13　系统管理员用例视图

3.5.1.2 构件目录

A.构件及其特性

构件	*类型*	*描述*
用户注册，登录	用户注册	用户注册对接山东大学统一认证，通过统一认证进行第一次登陆后，在注册页面绑定手机号，接收验证码，完成注册。
	用户登录	用户登录对接山东大学同统一认证，用户通过统一认证平台登录。
个人中心	查看用户信息	在该页面，用户可以查看自己的账户的相关信

		息。
	修改信息	在该页面，用户可以修改自己账户的部分信息，包括，用户头像，个人资料，用户签名等。
	退出登录	在该页面，用户可以点击按钮退出登录状态。
搜索	搜索商品信息	用户可以输入关键字查看筛选出的符合条件的商品信息。同时，在搜索时支持模糊查询。
聊天	发送聊天信息	在聊天界面，用户可以选择其他用户进行聊天，发送聊天信息。
	接收聊天信息	在聊天界面，用户可以接收到其他用户发送给自己的聊天信息。
	确认撤销申请	在聊天界面，用户可以收到其他拍下自己商品的用户的撤销购买申请，用户可以在聊天页面询问具体原因和确认申请。
	查看系统公告	在聊天界面中默认置顶一个系统管理员“用户”，在该用户中会给使用系统的用户推送系统公告和用户出售/收藏/拍下的商品的状态信息。
出售	出售商品	用户通过填写商品的相关信息并提交，该商品的信息会上传到寻宝猫后端的数据库中，并且在用户的商品管理页面的已上架商品管理界面中显示。此时，该商品可以被其他用户搜索到并可以被其它用户拍下。
	确认出售	用户可以对其他用户对自己商品的拍下请求做出确认或者拒绝。在做出确认后，该商品会自动下架，其他用户对该商品的拍下请求会自动拒绝，该商品进入用户的待交易商品管理页面中和拍下用户的待购买页面中。如果拒绝，则撤销其他用户对商品的拍下操作。

申请购买	拍下商品	用户可以对商品进行拍下操作，在拍下该商品后，商品信息进入用户的已拍下商品管理界面。
	撤销拍下	用户在对商品进行拍下操作后可以撤销拍下的操作，此时商品信息从用户的已拍下商品管理界面中移除。
个人商品信息管理	查看已上架商品	在已上架界面可以查看用户已经上架的商品。
	查看待交易商品	当卖方对买方的拍下操作进行确认之后，商品信息从卖方的已上架界面删除，进入待交易界面。卖方用户可以通过此界面查看等待与买方交易的商品信息，也可以撤销该交易。此时需要在聊天界面向买方说明原因，等待买房确认。在交易成功后，卖方可以点击确认交易，商品信息会从卖方待交易和买方待购买商品界面删除，进入双方的历史界面。
	查看已拍下商品	买房对商品进行拍下操作后，商品信息会进入此页面。用户可以通过此页面查看自己发出拍下申请的的商品信息，也可以撤销申请。
	查看待购买商品	买方拍下成功后，商品信息会从已拍下页面删除，进入此页面，此时买方可以查看等待购买的商品信息，也可以撤销购买。此时需要在聊天界面中向卖方说明原因，等待卖方确认。买方可以点击确认交易，商品信息会从卖方待交易和买方待购买商品界面删除，进入双方的历史界面。
	查看历史信息	保存成功出售和购买的商品信息，供用户查看。
用户管理	注册用户	为系统添加新的注册用户信息。
	查看用户	查看已经注册的用户的相关信息。

	删除用户	删除一个已注册用户的信息和与其相关的商品信息。
	设置用户权限	给用户增添/删除若干用户权限。
商品管理	添加商品信息	在数据库中添加商品相关信息。
	删除商品信息	在数据库中删除不合法的商品相关信息。
	修改商品信息	修改商品的相关信息，并重新写入数据库。
	查看商品信息	查看指定商品的详细信息。
公告管理	添加公告	添加公告信息。
	删除公告	删除公告信息。
	修改公告	修改公告信息。
	查看公告	查看公告信息。

B.关系及其特性

用例视图中存在关联和包含关系。

关联关系：

用户与商品之间存在着关联关系，商品和用户之间依赖用户 ID 进行关联，是一个多对一的关系。

用户和聊天信息之间也存在着关联关系，聊天记录和用户之间依赖用户 ID 进行关联，是一个多对一的关系。

包含关系：

用户管理中包含了查看用户、删除用户、设置权限等用例的实现。

商品管理中包含了添加商品信息，删除商品信息，修改商品信息，查看商品信息等用例的实现。

公告管理包含了添加公告、删除公告、修改公告、查看公告等用例实现。

个人商品信息管理中包含了查看已上架商品，查看待交易商品，查看已拍下商品等用例的实现。

聊天中包含了发送聊天信息，接收聊天信息，确认撤销申请，查看系统公告等样例的实现。

C.构件接口

用户通过手机 APP 发送请求，服务端接受请求并进行数据的处理，在处理完数据后

服务器调用数据库服务程序，与数据库进行交互，或者转发数据到目标用户 APP 上。

D.构件行为

构建具体行为在构建及其特性的表格中已经有了具体，详细的描述，在此不再赘述。

3.5.1.3 上下文图

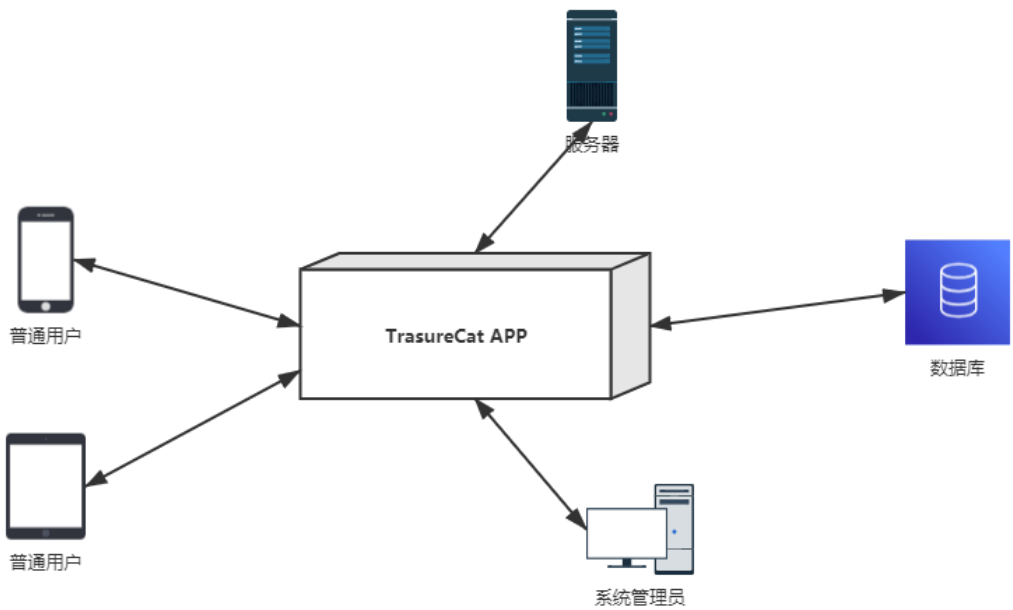


图 13 APP 用例上下文图

3.5.2 可变性指南

上述的功能只是寻宝猫系统的基本功能。目前为止还非常的不完善。由于现在的功能只是一个简陋的查询-显示结果-购买的过程，在使用的时候会显得非常的枯燥。因此，我们组目前的想法是在此基础上增加更多的社交元素，以拓展寻宝猫 APP 的泛用性和实用性。具体来说有以下几点:

1. 增加广场模块。这个模块类似于一个小论坛，在这个模块中可以发布一些帖子，分享一些趣事，发一些寻物启事，组织一些活动，增加一个 APP 内部的校园社交平台，方便校园生活。

2. 加入委托模块。这个模块中可以接取一些同学发布的帮带物品的委托，在方便的情况下接受并完成委托。增加同学之间的互助性。

3. 加入每日推荐模块。在之后的版本中，会根据用户之前的搜索行为推荐一些商

品，提高用户的使用体验。

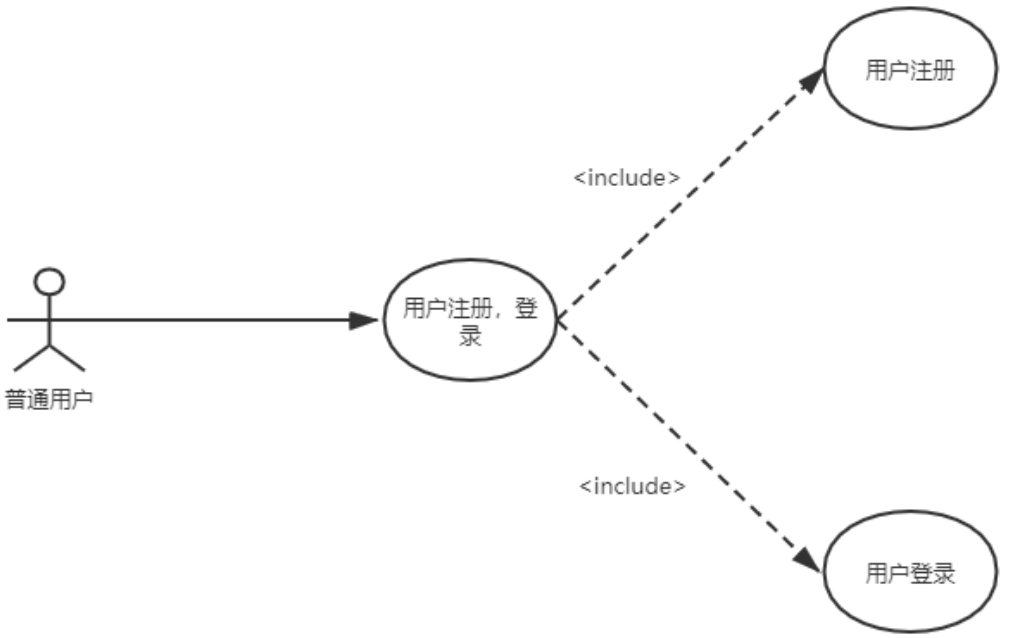
3.5.3 原理

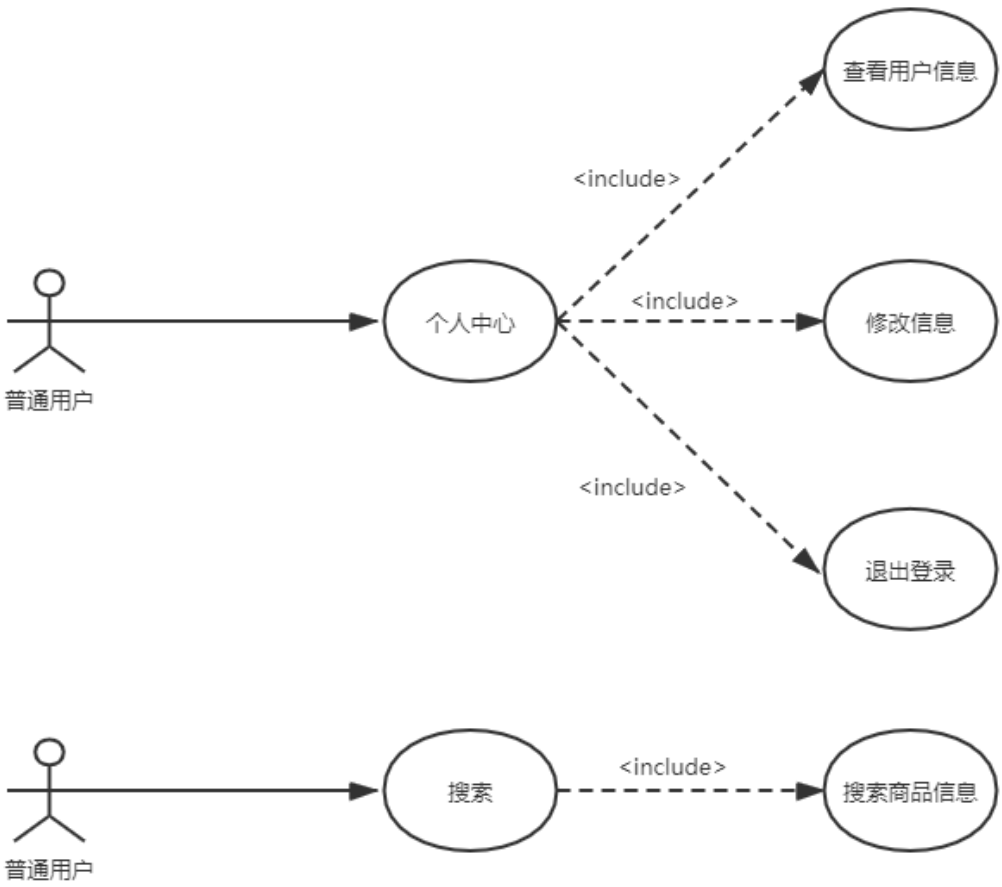
我们利用 UML 用例图来实现了论坛系统用例图。之所以采用用例视图来反映我们的体系结构，是因为从系统用户的角度考虑问题，设计出的架构能够满足业务逻辑的需求。

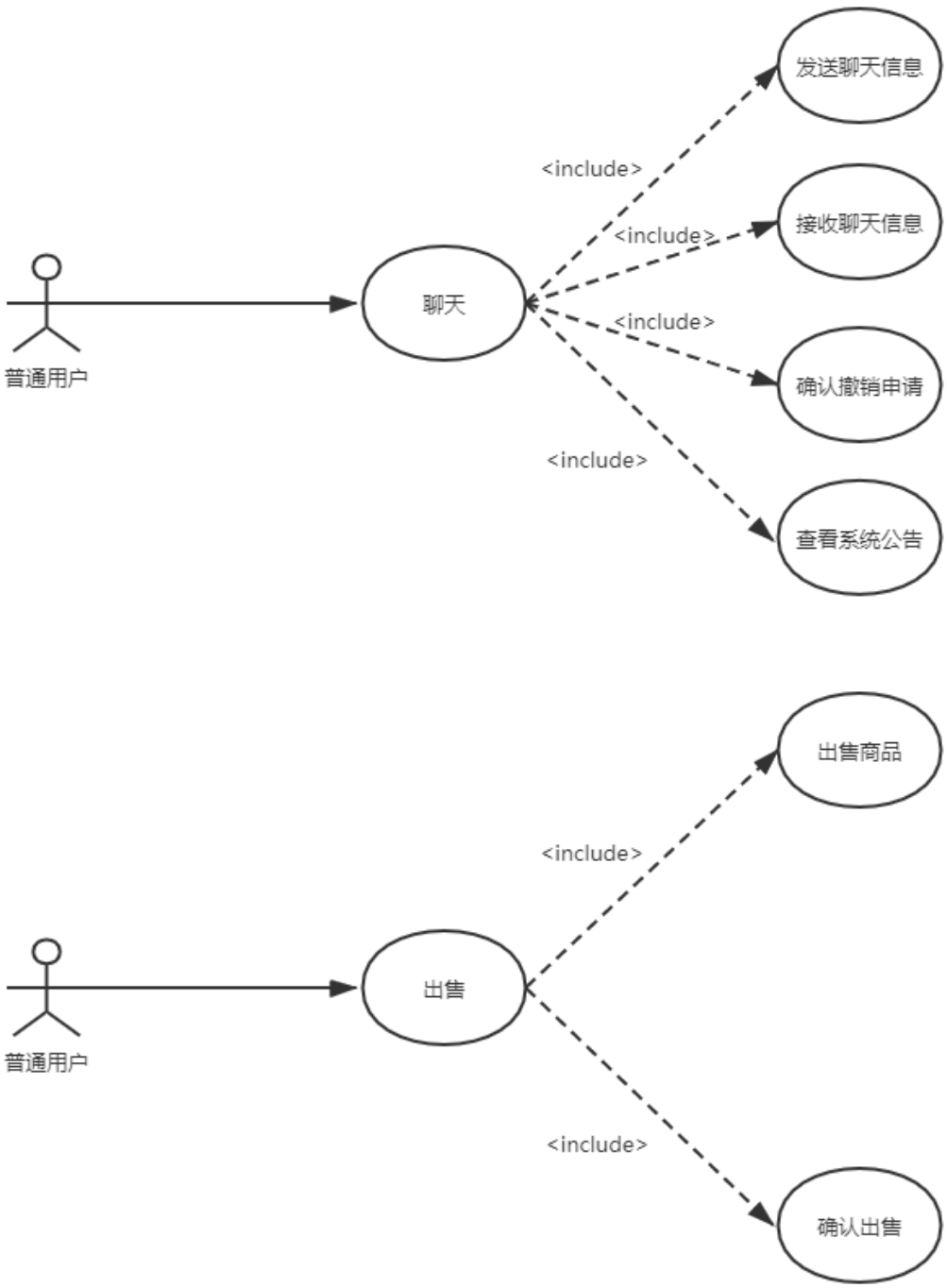
3.5.4 相关视图

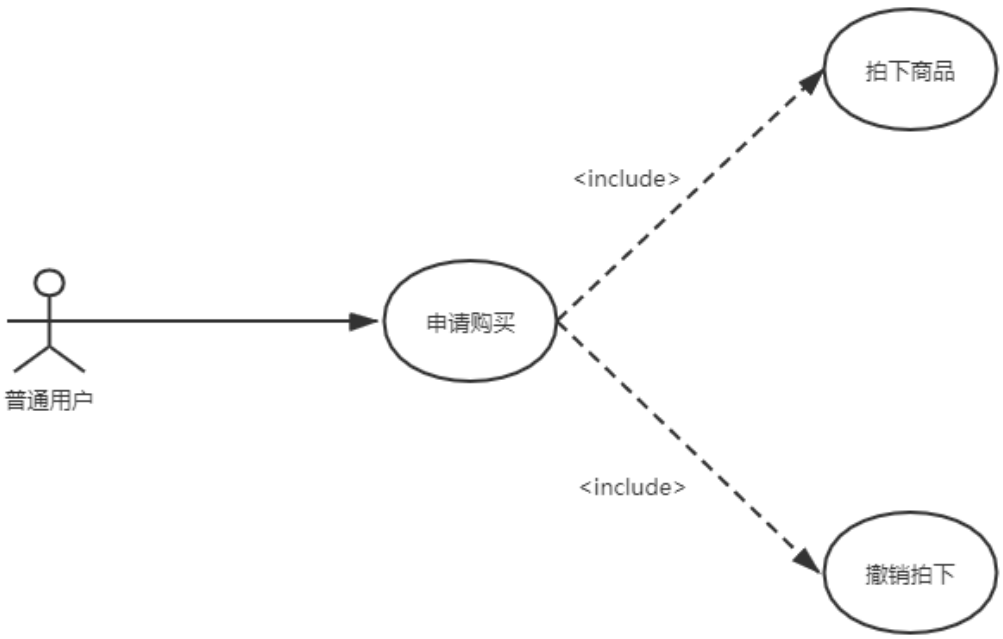
顶层用例视图的细分：

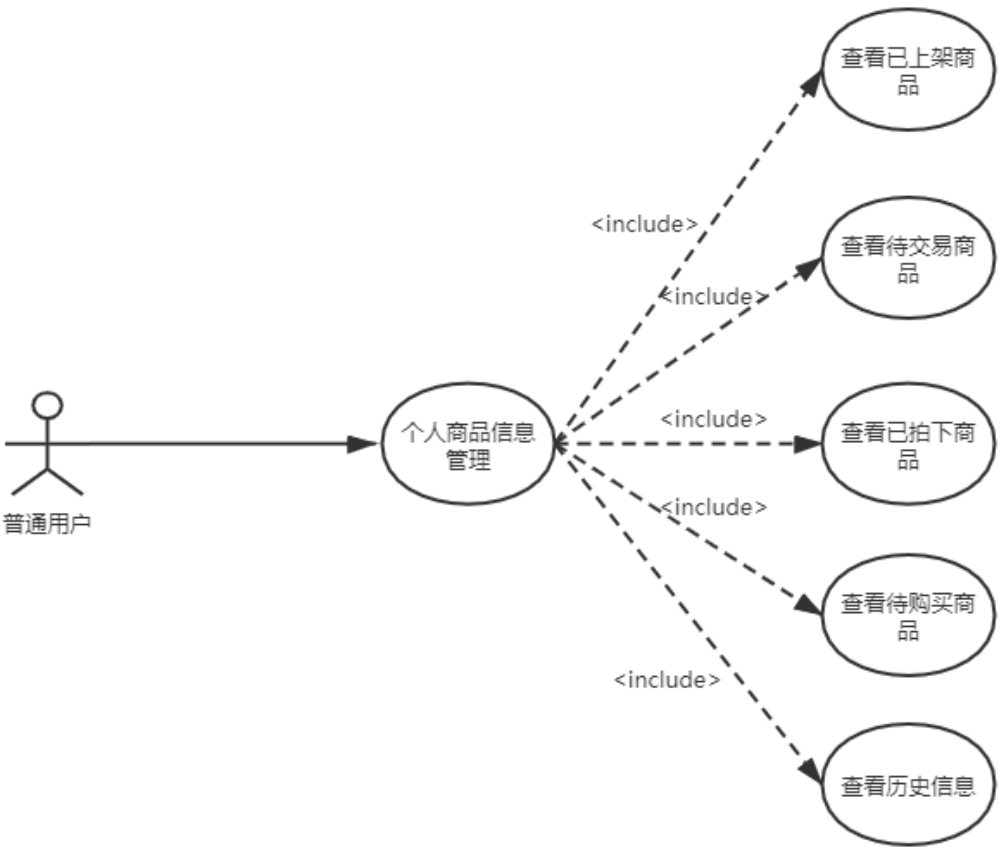
普通用户：



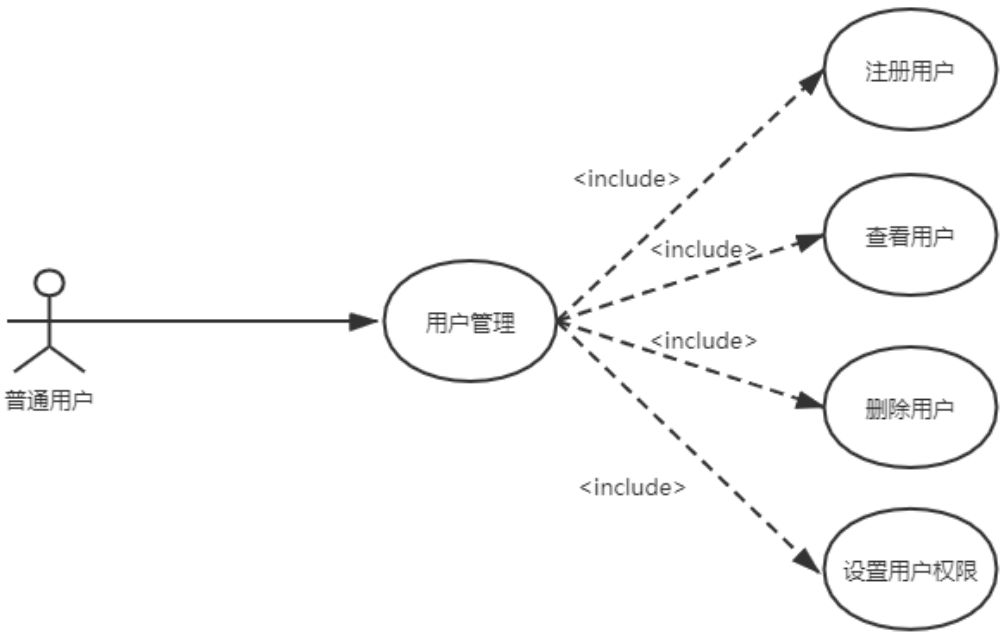

查看用户信息
<include>
个人中心
<include>
修改信息
普通用户
<include>
退出登录
搜索
<include>
搜索商品信息
普通用户

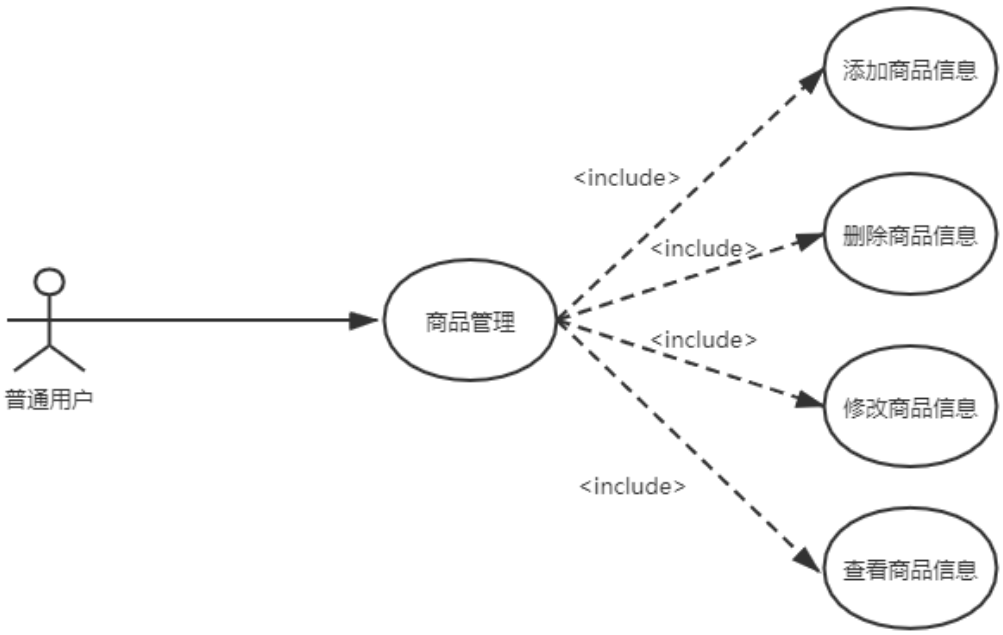

发送聊天信息
<include>
接收聊天信息
<include>
聊天
<include>
确认撤销申请
普通用户
<include>
查看系统公告
出售商品
<include>
出售
普通用户
<include>
确认出售

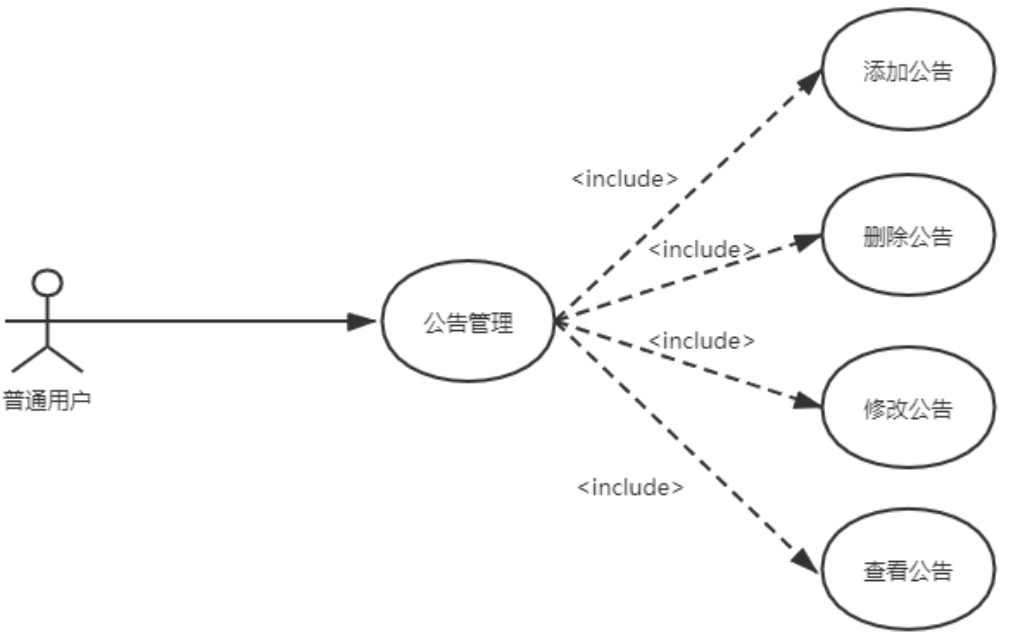

拍下商品
<include>
申请购买
普通用户
<include>
撤销拍下


查看已上架商品
<include>
查看待交易商品
<include>
<include>
查看已拍下商品
个人商品信息管理
<include>
查看待购买商品
<include>
查看历史信息
普通用户

系统管理员：


注册用户
<include>
查看用户
<include>
用户管理
<include>
删除用户
<include>
设置用户权限
普通用户

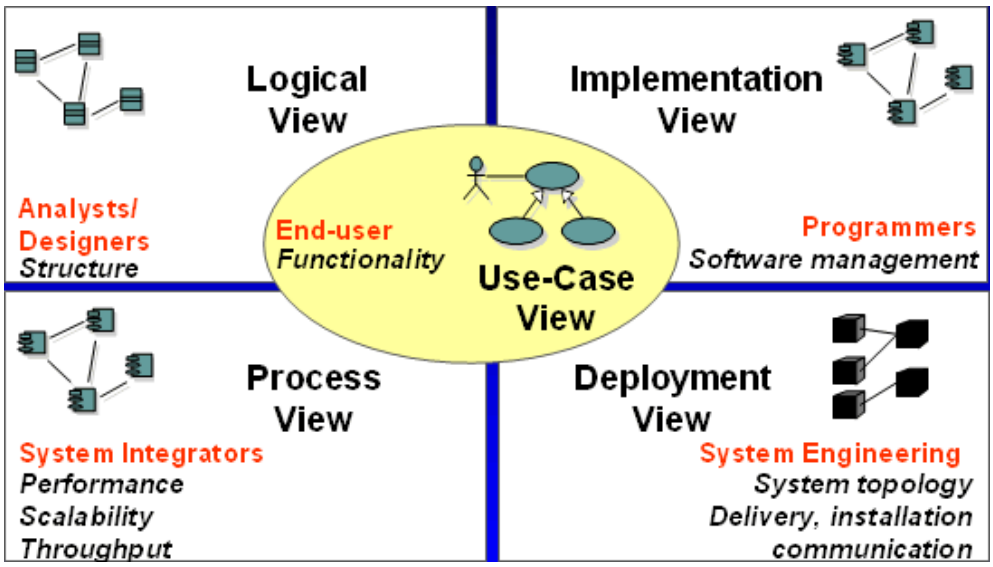

添加商品信息
<include>
删除商品信息
<include>
商品管理
<include>
修改商品信息
<include>
查看商品信息
普通用户



4. 视图之间关系

4.1 视图之间关系说明

上述 5 个视图是采用的 4+1 架构设计方法，其中的“1”就是指用例视图，其余 4 个视图围绕用例视图作为核心实现。



4.2 视图-视图关系

用例视图作为核心，确定了以下信息，而其他 4 个视图围绕以下信息进行设计：

系统边界：有了边界，才能够确定系统的设计范围；同时，通过边界能够识别出系统需要与用户或其它系统进行交互；

系统用户：明确的用户定义是系统需求分析的先决条件；

功能和场景：通过识别出系统与用户或其它系统的交互，可以分析出系统需要提供哪些功能，以及这些功能存在哪些应用场景。

（1）寻宝猫逻辑视图与开发视图关系

逻辑视图	*开发视图*
寻宝猫系统	C/S 架构
寻宝猫界面	ReactNative
寻宝猫买家卖家各种交互活动	
商品数据	MySQL 表 NestJS 后端服务框架
订单数据	
个人中心	

搜索功能以及其他系统服务	

（2）寻宝猫逻辑视图与部署视图关系

逻辑视图	*部署视图*
寻宝猫系统	Server
寻宝猫普通用户	User Mobile
寻宝猫管理员	Admin User
后台数据	MySQL5.7, DB Server
寻宝猫服务器	Apache Tomcat 7.0

5. 需求与架构之间的映射

略。

6. 附录

6.1 架构元素索引

略。

6.2 术语表

略。

6.3 缩略语

略。

6.4 组员学习笔记

6.4.1 张家豪学习笔记

软件体系结构学习笔记

在软件体系结构的学习方面，我比较关注的问题是软件体系结构的作用和当今比较常用的软件体系结构的特点。因此，我选择了《Software Architecture: a Roadmap》这

篇论文，并上网查阅相关资料来学习这两方面的内容。

在《Software Architecture: a Roadmap》这篇论文中，我首先对软件体系结构在软件开发中的地位有了一定程度的理解和认识。首先我了解到软件体系结构是一个描述软件系统总体结构的方法。它对整个软件系统的顶层设计进行了说明，是软件系统的需求和实现代码之间的一个桥梁。

并且，在文章中，作者还提到，一个好的软件体系结构可以至少在 6 个方面对软件开发发挥重要作用：首先是在理解软件系统上，好的软件体系结构可以使得软件系统的特点被很好的表现出，进而便于理解软件系统的高层设计。其次是在重用性方面，好的软件体系结构可以使得软件的组件在不同的软件开发时重用，提高开发效率。之后是在结构方面，好的体系结构可以描述软件的不同组件之间的依赖关系，进而使得软件结构更加清晰，有助于开发。之后是在软件的演进层面，好的体系结构可以使得组件内部结构的修改不会影响到组件之间的交互，从而便于软件的不断演进。再次是在分析方面，好的体系结构有助于对软件的分析。最后是在管理方面，好的体系结构对于软件开发的管理者而言可以更加方便的进行评估和风险分析，进而便于管理。

总之，通过对《Software Architecture: a Roadmap》这篇论文的学习，我了解到了软件体系结构在软件开发中的重要地位和作用。

另外，我也了解了一些常用的软件体系结构风格及其特点：

1.管道-过滤器风格

这种风格是根据数据流构建软件体系结构。在这种结构中，软件的每一个组件都被抽象成一个过滤器，过滤器之间通过管道相连，管道就是软件中数据流的抽象。在这种风格中，要求每一个过滤器（软件组件）都有一组输入到该过滤器的管道和从过滤器中输出的管道，过滤器本身的功能实现只依赖于过滤器本身，因此，该风格的软件部件之间的独立性较好。部件之间的耦合度较低。但是缺点也非常明显：交互性处理能力不足：一旦软件接收到外部输入，在最后的结果得出之前基本无法与用户进行交互操作。

2.调用-返回风格

这种风格是根据面向对象的思想而产生的。在这种风格中，数据的表示方法和它们的相应操作封装在一个抽象数据类型中。对象是这个抽象数据类型的一个实例。在软件中，每一个构建都被抽象成一个对象，对象之间的交互通过函数调用进行交互。由于对象之间的接口只有函数调用，因此，当一个对象内部结构改变时，只要其函数接口不变，对整个系统是没有影响的。当然，由于不同对象之间调用时需要通过一个对象的标识号唯一标识一个对象，当只要一个对象的标识改变了，就必须修改所有其他明确调用它的对象。这对整个系统来说是非常麻烦的。

3.层次结构风格：

层次结构风格与在计算机网络课程中所学的网络七层结构在思想上非常相似。层次结构风格是将软件整个结构分成不同的层级，其中与用户进行交互的部分为最高层，软件的核心部分为最底层，每一层为上层服务，并作为下层客户。并且只有相邻的层级之间才有函数调用接口，这样将一个复杂问题分解成一个增量步骤序列的实现。由于每一层

最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，为软件重用提供了强大的支持。同时，这种只支持相邻的上下层交互的设计方法使得单层结构改变时，其影响的范围只限于与其相邻的层，层与层之间的独立性较好。当然，层次结构风格比较理想化，在实际应用中，分成合适的层次结构是一个非常困难的事情。

4.客户端/服务器风格（C/S 风格）

C/S 风格是将应用功能分成表示层、功能层和数据层三部分。C/S 风格对这三层进行明确分割，并在逻辑上使其独立。

表示层是应用的用户接口部分，它担负着用户与应用间的对话功能。它用于检查用户从键盘等输入的数据，显示应用输出的数据。一般以图形界面显示。表示层只负责数据的读写功能，并不包含程序有关业务本身的处理逻辑。

功能层相当于应用的本体，它是将具体的业务处理逻辑地编入程序中。表示层与功能层之间有数据的直接交互，表示层接收到的数据会直接提交给功能层，功能层中得出的结果数据也直接提交给表示层。

数据层负责对软件数据库的读写操作。用于处理功能层向其传输的对数据库进行的操作。

C/S 风格的三层之间的关系是非常密切的：表示层接收数据，并将数据传给功能层，功能层根据数据向数据层发出对应数据库操作的 SQL 指令。数据层根据指令对数据库进行操作，并返回操作结果给功能层，功能层最后将结果传给表示层，表示层对接收到的结果进行处理后显示。

通过学习，我发现 C/S 风格的三层之间的独立性非常好，层与层之间只有数据之间的交互，因此在开发时三层可以独立进行开发。并且，按照层次划分使得程序的逻辑也相应的简单了。但是，C/S 风格维护成本很高，既要对服务器维护管理，又要对客户端维护和管理，这需要高昂的投资和复杂的技术支持，维护成本很高，维护任务量大。

5.仓库风格：

这种风格包含一个数据仓库和若干其他的部件。数据仓库位于整个结构的中心部分，用于存储数据。其他的部件通过访问数据仓库对其中的数据进行增删改等操作。仓库风格的部件之间是独立的，修改一个部件的内部结构对整个结构的影响很小。因此具有很高的可修改性和可维护性。但是，由于部件需要频繁的访问仓库，整个系统的执行效率较低。

以上就是我对软件体系结构的作用和当今比较常用的软件体系结构的一些认识和理解。

6.4.2 田正龙学习笔记

6.4.2.1 阅读章节

《Software Architecture in Practice》--CHARPATER17：Designing an Architecture

6.4.2.2 阅读原因

在软件架构文档的编写过程中，负责了部分和质量属性相关的内容，因此后来也看了这本书的第 17 章节，也就是和 ADD 构架设计方法相关内容。

6.4.2.3 ADD 介绍与过程

ADD（属性驱动的构架设计）是一种以质量属性实现为主要推动力的构架设计方法，它将功能实现放到了次要的位置上。其将一组质量属性场景作为输入，并使用对质量属性实现和构架之间的关系的了解，对构架进行设计。ADD 将分解过程建立在软件必须满足的质量属性之上。它是一个递归的分解过程，其中在每个阶段都选择战术和构架模式来满足一组质量属性场景，然后对功能进行分配，以实例化由该模式所提供的模块类型。ADD 的总体过程为，首先选择要处理的对象，一般指系统、子系统或者模块，确定这个对象需要实现的功能和需要满足的质量属性，然后选择能够实现这些质量属性的构架战术，根据战术设计框架，得到子模块，然后为子模块分配功能和应当承担的总体战术中的部分，最后检查功能与质量属性是否都被满足。

在书中，ADD 的过程细分成了 5 步：

1.选择要设计的系统元素。开始的部分就是整个系统，对于已经部分完成的设计（通过外部约束或者通过 ADD 进行的先前迭代的部分），这部分是尚未设计的元素。使用深度优先或者广度优先以及混合的方式可以获得下一个元素。

2.确定所选元素的 ASR（architecturally significant requirement）。

3.使用诸如现有系统、框架、模式和策略之类的设计资料，以及第 5-11 章节中的设计清单，为所选元素生成设计解决方案。

4.验证和完善需求，为下一次迭代生成输入：第 4 步是一个测试步骤，如果第 3 步的设计满足所有选定的 ASR，则结束，否则有重要元素不满足，并且无法通过进一步的完善来满足需求，则需要回溯。

5.重复步骤 1-4，直到满足所有 ASR 要求，或者对体系结构进行了充分的细化以使实现者可以使用它为止。

6.4.2.4 ADD 的缺点

在网上查询资料得知，ADD 对于将质量属性链接到设计选择很有用，但是它有几个需要解决的缺点：

（1）ADD 指导架构师使用并组合策略和模式，以实现质量属性场景的满意度。然而，模式和策略都是抽象的，方法没有解释如何将这些抽象映射到具体的实现技术。

（2）ADD2.0 版本是在敏捷方法被广泛采用之前发明的，因此，它没有为敏捷环境中的架构设计提供指导。

（3）ADD 未提供有关如何开始设计过程的指导。虽然这种省略增强了它的可归纳性，但它给新手设计师带来了困难，他们通常不知道从哪里开始。ADD 并没有明确地促进（重用）参考体系结构，这是许多架构师理想的起点。
（4）ADD 没有明确考虑不同的设计目的。例如，可以将设计作为售前流程的一部分，或者作为构建的“标准”设计的一部分。不同的目的将导致添加的不同用途。
（5）ADD 没有考虑到设计需要解决一些架构问题（即内部需求），不管它们是否在“传统”驱动因素（需求和约束）列表中表示。很少有用户会要求系统“可测试”或要求系统提供特殊的测试接口，但明智的架构师可能会选择包括这样的基础设施，特别是当系统很复杂并且在难以控制和复制的环境中使用时。
（6）ADD2.0 包括了初始文档和分析，但它们不是设计过程的明确步骤。
ADD3.0 版本对以上的问题做了改进。这种改进是演变而不是革命。

6.4.2.5 总结

ADD 是以驱动程序为输入并生成体系结构的迭代体系结构设计方法，这种方法可以有效提高软件系统的成功率。因此，在 ADD 使用范围较窄的情况下，仍然有很多优点值得我们借鉴。

6.4.3 李世昱学习笔记

6.4.3.1 阅读内容

Software Architecture : a Roadmap

6.4.3.2 阅读原因

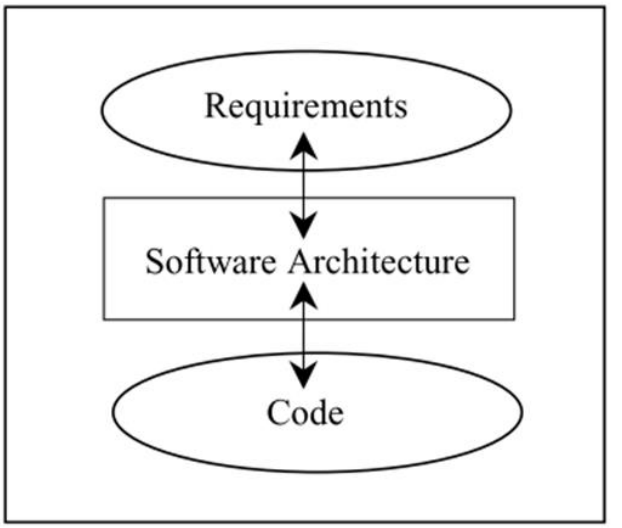
对于软件工程来说，软件架构是承上启下的桥梁；而对于想要从事软甲开发行业的我来说，软件架构就是比较神秘的存在，因此，我选择阅读此篇论文，想要学习软件架构的昨天，今天，明天。

6.4.3.3 简介

完整的软件架构是交互组件的集合。一个好的软件架构能够确保一个系统能够满足软件的核心要求：性能、可靠性、可移植性、可伸缩性和互操作性。

软件架构已经如火如荼的发展了很多年，有非常多的书籍，但是，却依旧存在着很多不足，本文将探索软件架构在搜索和训练方面的趋势.

6.4.3.4 软件架构的作用



如上图所示，软件架构是需求和开发之间的桥梁。通过提供一些抽象的定义，软件架构能够表述出软件的特性，提炼精髓。通过这样的提炼，软件架构能够指导整个的开发系统，告诉软件开发者应该用什么样的方式搭建软件，告诉软件设计者（产品经理）应该用什么样的方式来满足需求。软件架构有以下 6 个重要的作用：

1. 可理解性：软件架构简化了整个系统，提供了一些抽象的定义。因此，较为复杂的高级设计能够被很好的理解。同时，一些特定的架构选择也能够被理解。
2. 可重复性：软件架构能够支撑在很多层次重复使用。组件可以被重复使用，架构设计模式可以被重复使用，组件的整合方式也可以被重复使用。
3. 开发的便利：软件架构告诉软件开发者应该用什么样的方式搭建软件。
4. 可迭代性：软件体系结构可以预期系统发展的维度，这样就方便软件的迭代。
5. 可分析性：软件体系结构为分析提供了新的机会，包括系统一致性检查、对体系结构风格施加的约束的一致性、对质量属性的一致性、依赖性分析，以及对特定风格构建的体系结构的特定领域分析。
6. 可管理性：对架构的开发，能够评估整个软件的成功与否。

6.4.3.5 软件架构的昨天

以前的软件开发，基本上就是第二节软件架构的作用的反例。不好理解，不好重复，不好分析等等。但是，人们越来越意识到软件架构的重要性，这种趋势主要体现在两个方面：

1. 对构建复杂软件系统的方法、技术、模式和习惯用法的共享指令的认识的提高。
2. 更加关注利用特定领域中的共性，为产品族提供可重用的框架。

6.4.3.6 软件架构的今天

软件架构在今天被更加重视，以可视化的方式描述软件架构几乎成了标配。

6.4.3.6.1 软件描述语言和工具

软件描述语言和工具就是为了更好的表述软件架构开发的语言和工具。比如 Adage 等。但是，根据软件体系结构明确设计的语言并不是唯一的方法。人们对在建筑建模中使用通用对象设计表示法有相当大的兴趣。此外，最近有许多提议试图说明如何将 ADL 中的概念直接映射到像 UML 这样的面向对象的符号表示。

UML 是一种开放的方法，用于说明、可视化、构建和编写一个正在开发的、面向对象的、软件密集系统的制品的开放方法。UML 展现了一系列最佳工程实践，这些最佳实践在对大规模，复杂系统进行建模方面，特别是在软件架构层次已经被验证有效。

6.4.3.6.2 产品线和产品标准

产品线：一组共享一组公共管理的特征的产品，这些特征满足一个选定市场的特定需要。一个产品线的定义是基于市场策略的，而不是基于它的成员产品之间的技术相似性。为一个产品定义的特征，可能需要完全不同于其它成员产品的解决方案。一个产品线可能是与一个产品族一起提供出来，但是它也可能需要不止一个产品族。

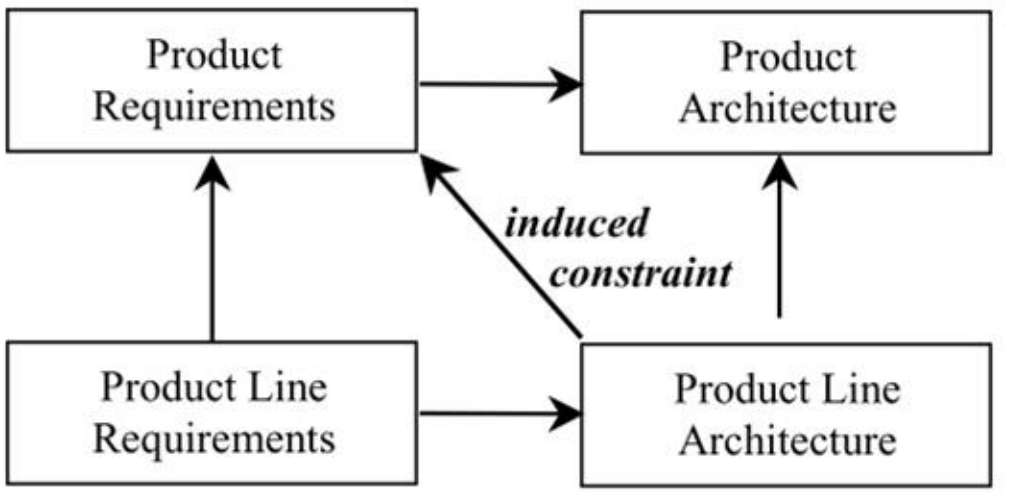


Figure 3: Product Line Architectures

产品标准：这个比较简单。就是公司或组织为软件或一种产品定义的接口标准、技术标准，规格标准等一系列标准。

6.4.3.7 软件架构的明天

6.4.3.7.1 改变购买软件和搭建软件之间的平衡

下列两表很好的度量了建造软件和购买软件的优劣。

建造软件	
优点	缺点
您拥有该代码，这意味着您甚至可以营销和出售它以获取利润。	构建软件需要大量的时间，资源，计划，专业知识和精力，并且要有一个成熟的开发团队。除开发外，您还负责维护代码并年复一年地为用户提供支持。
您可以构建它以使其与堆栈中的所有其他软件无缝协作。	很有可能其他人已经开发了与您计划的软件非常相似的软件。如果真是这样，您将需要花很多时间和金钱来复制他们的工作。
如果发现它缺少您的业务所需的功能，则可以优先添加该功能。	甚至最佳计划的项目也容易受到范围扩大，预算超支和截止日期的影响。

购买软件	
优点	缺点
购买时，实施时间可能要花几个月或更短的时间，而建造时可能要花几年或更长时间。	购买软件时，您将完全依赖于开发软件的供应商。如果您发现它不能与关键工具很好地集成，或者缺少所需的功能，那么您所能做的就是提出请求，并希望开发人员可以与您合作。
购买软件时，前期成本通常要低得多。	尽管构建自己的软件的前期成本可能要高得多，但从长远来看，它的成本也可能大大降低。相反，购买软件，尤其是 SaaS 产品。每月或每年的订阅费用每年总计可达数十万甚至数十万美元，尤其是在您需要扩展容量和功能时。
购买时，SaaS 提供商可以处理从培训到托管再到维护的所有事务，使您可以专注于自己的业务。	购买现成的软件时，您将使用与许多竞争对手相同的工具和功能，从而减少了获得竞争优势的机会。

而因为软件架构的不断进步，现在构建软件能够更加方便，也不容易出错，因此，构

建软件会在未来变得更加普遍。

6.4.3.7.2 以网络为中心的计算

未来的趋势是从以 pc 为中心的计算模型到以网络为中心的计算。

然而，以前的软件架构不够开放，几乎是完全静态的，这是远远不能满足以网络为中心计算，开发软件的要求。挑战主要有以下四点：

1. 需要能够扩展到 Internet 的大小和可变性的架构
2. 需要支持计算，动态的，特定任务的，分布式的，自我资源管理。
3. 需要找到灵活适应商业应用程序服务提供者的体系结构。
4. 需要开发允许最终用户进行自己的系统组合的体系结构

6.4.3.7.3 更普遍的计算

未来是万物互联的时代，是 5G 的时代，因此，CPU 和计算将会出现在所有的电器上，这会出现一系列的挑战。

1. 需要适合于系统资源使用分配的体系结构。
2. 这些系统的架构必须比现在更加灵活。
3. 需要更好地处理用户移动性的架构。

6.4.3.8 读后收获

通过这次阅读，不仅学到了上述所有的知识，了解了软件架构的由来，现状和未来趋势，更理解了软件架构的作用，软件架构是如何被描绘的，软件架构究竟是什么东西。这些让我收获颇丰。

6.4.4 赵雨晗读书笔记

我选择的是 Software Architecture in Practice ，3rd Edition 这本书的 Capter 13：Architectural Tactics and Patterns，原因是因为我个人比较喜欢游戏开发，包括 gameplay 与客户端架构方面的开发，而游戏作为软件开发中的一个分类，当然需要许多设计模式与架构的支撑。关于设计模式我读过 game programming patterns 和 GoF 的《设计模式：可复用性面向对象软件的基础》两本书，其中由于时间原因以及许多设计模式应用不到，GoF 并没有读完，对设计模式有比较深的理解，所以选择了这本书的第十三章进行阅读。

6.4.4.1 谈谈架构与设计模式

书中首先对设计模式做了一个定义性的概括：设计模式是可以对 a context， a problem， a solution 建立起关系的一种描述。一种设计模式可以被一些数据类型的集合、一些交互机制或连接方法的集合，一些元素的拓扑表示和一些包括拓扑、元素行为、交互机制的限制的集合所描述。在 GoF 的第一章也有类似的描述，用自己的话来说，设计模式是一系列适用于多次出现的问题的架构设计决策，可根据问题出现的上下文进行参数化设置。

与 GoF 不同，这本书主要从对架构的设计切入来讲对架构的设计模式，在《设计模式》中，作者将设计模式分为三个大类：创建型模式、结构型模式和行为型模式，分别主要关注在对象的初始化方式，类的结构组织，以及类的行为和对象之间的交互方法三个方面，而本书中将架构的设计模式分为：Module pattern, Component-connector pattern 和 Allocation pattern 三个大类，以我的理解大概是这样：

- Module pattern 关注组件内部结构
- Component-connector pattern 关注系统级 不同系统间的相互关系
- allocation pattern 关注系统与外部环境

可见，设计模式更注重对于类与接口的规范与解耦，而本书中讨论的架构的设计更倾向于对于系统做出整体的设计，两者有交集但不完全相同，比如 MVC 模式在两本书中都有讨论到。

6.4.4.2 对于各种架构设计模式的笔记与整理

1.分层模式 Layer pattern

分层模式定义了层与层之间的单向允许使用关系，通常使用代表　　层的方框的叠加来表示层的依赖关系。层：提供一组内聚服务的模块组。换句话说：层内是内聚服务，层间演个单向调用

优点：把复杂的问题分解为多个组件，提高可扩展性和可复用性；

缺点：层的增加会带来系统的成本和复杂度的增加；给性能带来负面影响；设计不好的层会妨碍到抽象；过多的层间关系不利于可移植性和可修改性

2.代理模式

代理模式：broker 是运行时组件，协调 client 和 server 之间的交流。

优点：高可扩展性，高可用性；server 可以被动态修改

缺点：延迟，通信瓶颈：增加了一层，会延迟 client 和 server 间通信，可能会成为通信瓶颈；broker 会增加系统复杂度；broker 可能出现单点失效；broker 可能成为被攻击的目标；broker 难于被测试

3.MVC 模式

把系统功能分为 3 个组件：MVC，Controller 是中介，通知关系连接三部分的实例，并通知相关元素的状态变化。

缺点：用户接口简单的情况下，带来的复杂度是不值得的，不一定能很好地使用一些用户接口工具

4.piper and filter 模式

数据从系统外部输入到输出的过程中，经过一系列由管道连接的过滤器的变化处理

过滤器：一种转化数据的组件，过滤器间可以并行处理

管道：将数据从过滤器的输出端口传递给另一个过滤器的输入端口的连接件，不会修改数据

缺点：不适合交互式系统；有大量独立的过滤器会增大大量的计算开销；不适合长时间的计算任务

5.CS 模式

客户端发起与服务器的交互，根据需要 调用服务 等待返回请求结果

客户端：调用服务器服务的组件，直到它所需要的服务的端口

服务器：给客户端提供服务的组件，有提供服务的端口。

请求/响应连接件：使用请求响应协议实现的连接件，客户端使用它来调用服务器的服务。

缺点：服务器会成为性能瓶颈；服务器淡定失效；在哪里放置功能的决定是复杂的，切系统构建后很难改变

6.peer to peer 模式

计算由多个节点合作完成，各个节点之间要通过网络相互提供服务和相互调用

点：网络节点上独立的组件

请求响应连接件：用于连接网络上的点，查找其他点，调用服务

缺点：安全性，数据一致性，数据/服务的可用性，备援和恢复都变的复杂；小的点对点系统可能无法实现质量目标，如性能和可用性

7.Service-oriented 模式

计算由一组相互合作的组件提供，他们在网络上提供服务，包括服务提供者，服务消费者，SOAP 连接件、REST 连接件

缺点：构建复杂；中间件带来性能开销；服务成为性能瓶颈，无法提供性能保证

8.publish-subscriber 模式

组件发布和订阅时间，发布事件后，基础连接件将时间分发到所有注册的订阅者。

缺点：增加通信延迟，对伸缩性降低，消息传递时间的预测降低；缺乏对消息顺序的控制，消息是不受保护的

9.shared-data 模式

数据存取器通过共享的数据商店通信。数据商店负责实现数据持久化。

缺点：数据商店性能瓶颈；单点失效；数据的生产者和消费者可能耦合紧密

10.Map-reduce 模式

提供了一个分析大规模分布式数据集合的框架，在一组处理器上并行处理。Map 提取和转化；reduce 转载结果

优点：并行，因而低延迟，高可用

缺点：数据集小时，开销不合理；数据不能分为小的子集时，并行就没有优势；需要多 reduce 的操作是复杂的

11.multi-tier based 模式

计算结构由多个逻辑划分的组件团体组成。每一个团体就是一个层(tier)。

缺点：前期成本高，复杂度高

Layer 是实际存在的清晰的，有层次关系的组织

Tier 不是在物理上实际存在的，是一种逻辑上的，概念上的组合，没有明确的层次关系

6.4.4.3 谈谈 pattern 和 tactics

本书中花了很大的篇幅来讲策略，从本章开始才开始着重讲架构设计模式，所以本章中

有一些和关于策略与模式的对比，经过阅读后在我的理解中是这样的：

1. tactics 更加简单，使用单一结构或机制来达成单一架构目标
2. patterns 是多种相关设计决策进行组合的结果
3. tactics 是构建 patterns 的组成模块
4. 大部分 patterns 由多种不同的 tactics 组成
5. 二者共同构成软件架构的主要工具

6.4.4.4 总结

以上就是我阅读 Capter 13：Architectural Tactics and Patterns 后的感想与心得，软件工程是一门很复杂的学科，不仅有许多计算机的相关知识，很多知识更是包括产品与管理学等各个学科综合的知识，是一门很深的学问与哲学，希望在软件工程这门课中我可以收获到所想要的知识，总结出一套自己的方法论，并在未来实践于项目与研发中。

6.4.5 袁凤池学习笔记

我阅读的是 An Introduction to Software Architecture。感受颇多，简述如下。

首先从全文的角度来看，全文首先对软件体系结构领域进行了介绍。软件体系结构这一新兴领域。文章首先阐释了当前系统所基于的一些常见的体系结构风格，并展示了如何在一个设计中组合不同的风格。在第三部分，文章列举了六个案例来说明体系结构表示如何提高我们对复杂软件系统的理解。最后，文章总结了该领域中存在的一些突出问题，并展望了一些有前途的研究方向。

文章需要重点研读的在第二部分，八种常见的体系结构风格。

6.4.5.1 管道与过滤器风格

首先是管道与过滤器风格。在管道/过滤器风格的软件体系结构中，每个构件都有一组输入和输出，构件读输入的数据流，经过内部处理，然后产生输出数据流。这个过程通常通过对输入流的变换及增量计算来完成，所以在输入被完全消费之前，输出便产生了。因此，这里的构件被称为过滤器，这种风格的连接件就象是数据流传输的管道，将一个过滤器的输出传到另一过滤器的输入。此风格特别重要的过滤器必须是独立的实体，它不能与其它的过滤器共享数据，而且一个过滤器不知道它上游和下游的标识。一个管道/过滤器网络输出的正确性并不依赖于过滤器进行增量计算过程的顺序。

改种风格具有许多很好的特点：

使得软构件具有良好的隐蔽性和高内聚、低耦合的特点；允许设计者将整个系统的输入/输出行为看成是多个过滤器的行为的简单合成；支持软件重用。重要提供适合在两个过滤器之间传送的数据，任何两个过滤器都可被连接起来；系统维护和增强系统性能简单。新的过滤器可以添加到现有系统中来；旧的可以被改进的过滤器替换掉；允许对一些如吞吐量、死锁等属性的分析；支持并行执行。每个过滤器是作为一个单独的任务完成，因此可与其它任务并行执行。

但是，这样的系统也存在着若干不利因素。

通常导致进程成为批处理的结构。这是因为虽然过滤器可增量式地处理数据，但它们是独立的，所以设计者必须将每个过滤器看成一个完整的从输入到输出的转换；不适合处理交互的应用。当需要增量地显示改变时，这个问题尤为严重；因为在数据传输上没有通用的标准，每个过滤器都增加了解析和合成数据的工作，这样就导致了系统性能下降，并增加了编写过滤器的复杂性。

6.4.5.2 数据抽象和面向对象

该种体系结构具有如下特点：对象实体是具有自身属性和行为能力的独立个体。 对象实体是主动的管理者，包括待处理的对象数据和所有参与过程的实体；任何事务处理都是对象相互发送“消息”作用的结果。发送消息是“请求”，接受消息是对请求的“响应”；响应“消息”的动作是接受请求对象自身的行为能力。并且它也可以 向其他对象发出请求。

6.4.5.3 事件驱动和隐式调用

该体系结构的特点是事件不直接被各响应处理接收，而是通过隐藏在中间的层次间接地被接收和处理中间的层次，完成消息形式的统一处理及调整和调度，这样，可以对接受事件进行必要操控请求与响应成分之间构成松散耦合，为灵活设计创造了条件对于资源不对等的异步并发系统是一种极好的控制方式。

6.4.5.4 分层系统

分层系统是按层次结构组织的，每一层向上面的层提供服务，并作为下面的层的客户端。在一些分层系统中，内层是隐藏的，除了邻近的外层，除了一些精心选择的用于导出的功能。因此，在这些系统中，组件在层次结构的某个层上实现一个虚拟机。(在其他分层系统中，层可能只有部分不透明。)连接器由决定层如何交互的协议定义。拓扑约束包括限制相邻层之间的相互作用。

6.4.5.5 黑板知识库

该风格每个知识源的动作依据是黑板中的信息，包括需要的交互及协同系统中的执行者和决策者享有同等的地位。使系统中多对多的关系，以及执行者和决策者的复杂关系变得简单而清晰，对于协同求解等专家系统是极好的控制方式。

6.4.5.6 表驱动解释器

在解释器组织中，虚拟机是在软件中产生的。解释器包括被解释的伪程序和解释引擎本身。伪程序包括程序本身和解释器对其执行状态(激活记录)的模拟。解释引擎包括解释器的定义及其执行的当前状态。因此解释器通常有四个组成部分:执行工作的解释引擎，包含被解释的伪代码的内存，解释引擎控制状态的表示，以及被模拟程序的当前状态的表示。

6.4.5.7 其他几个常见的体系结构

还有许多其他的建筑风格和模式。一些是广泛的，其他是特定的领域。

•分布式:分布式系统已经为多流程系统开发了一些常见的组织。有些可以主要通过它

们的拓扑特征来表征，如环和星组织。其他的更好地描述了用于通信的进程间协议的类型(例如，心跳算法)。

分布式系统体系结构的一种常见形式是“客户机-服务器”组织。在这些系统中，服务器表示向其他进程(客户端)提供服务的进程。通常，服务器不预先知道将在运行时访问它的身份或客户端数量。另一方面，客户机知道服务器的身份(或者可以通过其他服务器找到它)，并通过远程过程调用访问它。

•主程序/子程序组织:许多系统的主要组织反映了系统所使用的编程语言。对于不支持模块化的语言，这通常导致一个围绕主程序和一组子程序组织的系统。主程序充当子例程的驱动程序，通常提供一个控制循环，以某种顺序对子例程进行排序。

•特定领域的软件架构:最近人们对为特定领域开发“参考”架构有相当大的兴趣。这些体系结构提供了一个为一系列应用程序量身定制的组织结构，如航空电子设备、指挥和控制或车辆管理系统。通过将体系结构专门化到领域，就有可能增加结构的描述能力。实际上，在许多情况下，体系结构受到足够的约束，可以自动或半自动地从体系结构描述本身生成可执行系统。

•状态转换系统:许多反应系统的常见组织是状态转换系统。这些系统被定义为一组状态和一组将系统从一种状态移动到另一种状态的命名转换。

•过程控制系统:旨在提供物理环境动态控制的系统通常被组织为过程控制系统。这些系统大致被描述为一个反馈回路，过程控制系统使用来自传感器的输入来确定一组输出，这些输出将产生一个新的环境状态。

6.4.5.8 总结

通过仔细阅读，了解了其中一些代表性的、广泛使用的架构风格。架构选择的丰富丰富性，以及每一种架构的独特点。除此之外，文章通过六个案例使我直观感受到了体系结构表示能够大大提高我们对复杂软件系统的理解。最后，文章总结了该领域中存在的一些突出问题，并展望了一些有前途的研究方向，给予了我非常多的思考。

6.5. 提高质量的因素以及添加策略

项目将从可维护性，可复用性，性能，安全性四个方面来分析项目的质量属性，以及从项目现阶段的质量属性出发，针对提高项目的每一条质量属提做出相应的设计。

6.5.1 可维护性

可维护性有两个不同的角度，一个是指从软件用户和运维人员的角度，另一个是从软件开发人员的角度。

从用户和运维人员的角度，软件的可维护性是指软件是不是容易安装，升级，打补丁，有了问题是不是容易修复，能不能很容易的获得支持。

从开发人员的角度，软件的可维护性是指软件的架构是不是清楚简单，代码是不是容易阅读，有了问题是不是容易定位错误的原因，有没有可以提供帮助的文档，等等。

6.5.1.1 可维护性及策略分析

A. 使用多种设计模式降低耦合度、

在后端使用了多种设计模式，例如在控制器使用了依赖注入，在后端多处用到了单例与原型工厂模式，为了降低耦合，还是用了消息队列模式，多处通过注册回调函数的方法降低耦合。

前端 React-Native 使用了 MVVM 的设计模式，View 可以独立于 Model 变化和修改，一个 ViewModel 可以绑定到不同的 View 上，当 View 变化的时候 Model 可以不变，当 Model 变化的时候 View 也可以不变。

视图负责界面和显示。它通过 DataContext(数据上下文)和 ViewModel 进行数据绑定，不直接与 Model 交互，视图的责任便是定义用户在屏幕上能看到的一切的结构以及外观。理想的视图背后的代码只包含调用 InitializeComponent 方法的构造函数。视图通常扮演以下关键角色：

- 视图是可视化元素，例如窗口，页面，用户控件或者数据模版
- 视图定义了包含在视图里的控件以及可视化层以及样式
- 视图通过 DataContext 属性应用视图模型
- 绑定了控件以及数据的属性以及命令被视图模型暴露出来
- 视图可以定制化视图与视图模型间数据绑定行为
- 视图定义以及处理 UI 可视化行为例如动画
- 视图背后的代码实现了用 XAML 很难表达的可视化行为
- 视图与视图模型之间可以通过绑定 Behavior/Comand 来对 ViewModel 的方法进行调用，Command 是 View 到 ViewModel 的单向通行，通过实现 Silverlight 提供的 IComand 接口来实现绑定，让 View 触发事件，ViewModel 来处理事件，以解决事件绑定功能。

视图模型

视图模型在 MVVM 模式中为视图封装了展示逻辑，它并不是直接引用视图或者任何其他关于视图特定的实现或者类型。视图模型实现了属性以及命令使得视图进行数据绑定，并通过改变事件通知来提醒视图状态已经改变了。视图模型提供的属性和命令定义了提供给 UI 的功能。但是视图定义了如何渲染的功能。

视图模型负责协调视图与任何需要的模型类的交互。很典型的，视图模型与视图类有着一堆多的关系。视图模型可以选择直接将模型类暴露给视图，因此视图的控件能够直击进行数据绑定。视图模型可以转换或者操纵模型数据所以能够很容易被视图使用。很典型的，视图模型会定义能被展现在 UI 上并被用户调用的的命令或者行为。一个通用的例子就是当视图模型需要提交命令时会允许用户提交数据到网络服务或者数据库。视图可以选择用一个按钮来展示所以用户能够点击该按钮提交数据。典型地，当命令编程不可用的，它相关的 UI 展示也变得不可用。视图模型通常扮演下面这些关键角色：

- 模型视图是非可视化类，它封装了展现逻辑
- 视图模型是可以独立于视图与模型调试的
- 视图模型很典型地是不直接引用视图的

- 视图模型实现了视图用来数据绑定的属性与命令
- 视图模型通过改变提醒事件通知视图状态的改变：INotifyPropertyChanged 与 INotifyCollectionChanged
- 视图模型协调视图与模型的交互
- 视图模型可以定义视图展现给用户的逻辑状态

视图模型是 View 和 Model 的桥梁，是对 Model 的抽象，比如：Model 中数据格式是“年月日”，可以在 ViewModel 中转换 Model 的数据为“日月年”供 View 显示。

model 在 MVVM 模式中封装了业务逻辑以及数据，业务逻辑定义了像所有检索和程序数据管理相关的程序逻辑一样，用来确保所有的保证数据持久与有效的业务规则被应用。最大化代码重用，模型不能包含任何特定的情况，特定的用户任务以及程序逻辑。

典型的有模型为程序展现了客户端域模型，模型也可能包含支持数据访问与缓存的代码，即使有一个分离的数据库或者服务被使用。模型通常扮演如下的关键角色：

- 模型类是不可视类，它封装了程序数据
- 模型类不直接应用视图或视图模型类
- 模型类不依赖于它们是如何实现的
- 模型类是典型地通过 INotifyPropertyChanged/INotifyCollectionChanged 接口提供属性与集合更改事件的.
- 模型类很典型地继承自 ObservableCollection<T> 类
- 模型类是很典型地通过 IDataErrorInfo/INotifyDataErrorInfo.提供数据验证与错误报告
- 模型类典型地与封装了数据访问的服务一起使用。

Model 具有对数据直接访问的权利，例如对数据库的访问，Model 不依赖于 View 和 ViewModel，也就是说，模型不关心会被如何显示或是如何被操作，模型也不能包含任何用户使用的与界面相关的逻辑。Model 在实际开发中根据实际情况可以进行细分。

B.版本升级与快速迭代管理

项目开发使用 npm 管理依赖包，项目部署使用 maven 构建环境并导出资源包。

maven 主要是用来解决导入 java 类依赖的 jar,编译 java 项目主要问题。(最早手动导入 jar，使用 Ant 之类的编译 java 项目)

以 pom.xml 文件中 dependency 属性管理依赖的 jar 包，而 jar 包包含 class 文件和一些必要的资源文件。当然它可以构建项目，管理依赖，生成一些简单的单元测试报告。

Maven 是基于项目对象模型，可以通过一小段描述信息来管理项目的构建，报告和文档的软件项目管理工具。Maven 能够很方便的帮助管理项目报告，生成站点，管理 jar 文件，等等。例如：项目开发中第三方 jar 引用的问题，开发过程中合作成员引用的 jar 版本可能不同，还有可能重复引用相同 jar 的不同版本，使用 maven 关联 jar 就可以配置引用 jar 的版本，避免冲突。

6.5.2 性能

性能是软件开发中最被重视的质量属性，通常以系统执行某操作所需要的响应时间

或者在某单位时间所能完成的任务的数量来定义性能指标。

性能和其它的质量属性的相关性很高，有一些会对性能产生正面影响，有一些则是负面的。一般而言，计算机提供了许多的资源，包括 CPU，内存，硬盘等等，提高性能的核心就是充分利用这些资源。要保证对资源的使用是正确和有效的。

6.5.2.1 性能及策略分析

A.项目缓存设计

项目使用了 MySQL 数据库，运用缓存机制来适应高并发与高性能的需要。

MySQL 缓存机制就是缓存 sql 文本及缓存结果，用 KV 形式保存再服务器内存中，如果运行相同的 sql,服务器直接从缓存中去获取结果,不需要在再去解析、优化、执行 sql。如果这个表修改了，那么使用这个表中的所有缓存将不再有效，查询缓存值得相关条目将被清空。表中得任何改变是值表中任何数据或者是结构的改变，包括 insert, update, delete, truncate,alter table,drop table 或者是 drop database 包括那些映射到改变了的表的使用 merge 表的查询，显然，者对于频繁更新的表，查询缓存不合适，对于一些不变的数据且有大量相同 sql 查询的表，查询缓存会节省很大的性能。

SQLData Cache 是存储数据页（Data Page）的缓冲区，当 MySQL 需要读取数据文件（File）中的数据页（Data Page）时，MySQL 会把整个 Page 都调入内存（内存中的一个 Page 叫做 buffer），Page 是数据访问的最小单元。

当用户修改了某个 Page 上的数据时，MySQL 会先在内存中修改 Buffer，但是不会立即将这个数据叶写回硬盘，而是等到 CheckPoint 或 lazy Writer 进程运行时集中处理。当用户读取某个 Page 后，如果 MySQL 没有内存压力，它不会在内存中删除这个 Page，因为内存中的数据页始终存放着数据的最新状态，如果有其他用户使用这个 Page，MySQL 不需要从硬盘中读取一次，节省语句执行的时间。理想情况是 MySQL 将用户需要访问的所有数据都缓存在内存中，MySQL 永远不需要去硬盘读取数据，只需要在 CheckPoint 或 lazy Write 运行时把修改过的页面写回硬盘即可。

B. 项目资源共享设计

项目使用 MySQL5.7 版本，支持线程池来进行多线程的访问。

线程池是 Mysql5.6 的一个核心功能，对于服务器应用而言，无论是 web 应用服务还是 DB 服务，高并发请求始终是一个绕不开的话题。当有大量请求并发访问时，一定伴随着资源的不断创建和释放，导致资源利用率低，降低了服务质量。线程池是一种通用的技术，通过预先创建一定数量的线程，当有请求达到时，线程池分配一个线程提供服务，请求结束后，该线程又去服务其他请求。 通过这种方式，避免了线程和内存对象的频繁创建和释放，降低了服务端的并发度，减少了上下文切换和资源的竞争，提高资源利用效率。所有服务的线程池本质都是位了提高资源利用效率，并且实现方式也大体相同。

在 Mysql5.6 出现以前，Mysql 处理连接的方式是 One-Connection-Per-Thread,即对于每一个数据库连接，Mysql-Server 都会创建一个独立的线程服务，请求结束后，销毁线程。再来一个连接请求，则再创建一个连接，结束后再进行销毁。这种方式在高并发情况下，会导致线程的频繁创建和释放。当然，通过 thread-cache，我们可以将线程缓存

起来，以供下次使用，避免频繁创建和释放的问题，但是无法解决高连接数的问题。One-Connection-Per-Thread 方式随着连接数暴增，导致需要创建同样多的服务线程，高并发线程意味着高的内存消耗，更多的上下文切换(cpu cache 命中率降低)以及更多的资源竞争，导致服务出现抖动。相对于 One-Thread-Per-Connection 方式，一个线程对应一个连接，Thread-Pool 实现方式中，线程处理的最小单位是 statement(语句)，一个线程可以处理多个连接的请求。这样，在保证充分利用硬件资源情况下(合理设置线程池大小)，可以避免瞬间连接数暴增导致的服务器抖动。

C.异步数据获取

后端使用的 NestJs 框架使用 TypeScript 并支持原生 JaveScript，支持数据的异步获取，支持 async/await 操作。

如果在函数定义之前使用了 async 关键字，就可以在函数内使用 await。当 await 某个 Promise 时，函数暂停执行，直至该 Promise 产生结果，并且暂停并不会阻塞主线程。 如果 Promise 执行，则会返回值。 如果 Promise 拒绝，则会抛出拒绝的值。

使用异步编程，适应高并发的场景，体高性能。

6.5.3 可复用性

6.5.3.1 可复用性分析

可复用性，是指软件中可以重复使用的程度。良好的软件可复用性可以提高软件的生产效率，增加软件的质量并且也便于软件的维护。

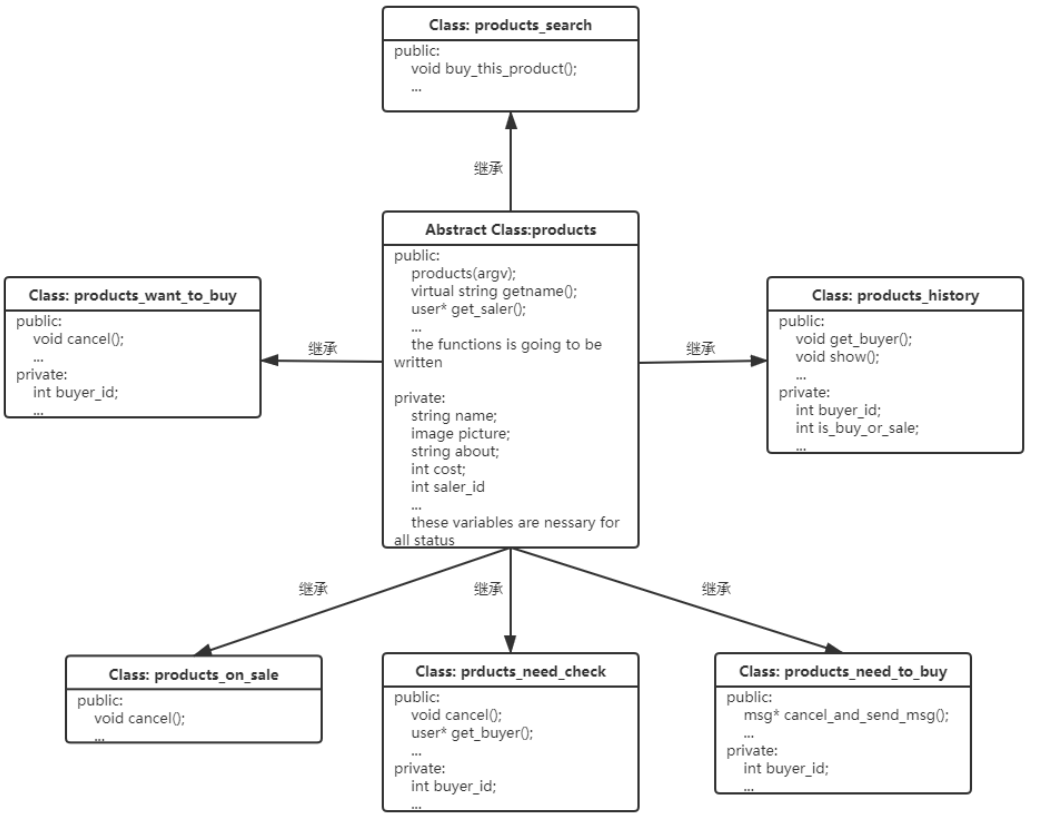
对于寻宝猫 APP 来说，我们组主要从两方面对软件的可复用性进行了分析：后端的可复用性和前端的可复用性。

对于后端来说，可复用性主要集中在数据结构复用上。在寻宝猫 APP 中，由于商品的状态不同导致了商品在不同状态时需要保存的额外信息不尽相同，此时如果为每一个状态单独设计一个类来保存是非常麻烦且冗余的，并且会使代码的程序量大大增加，同时，这样也对后期软件的维护产生了非常大的麻烦。但是，仔细分析可以发现，虽然商品有不同的状态，但是，在不同状态下存储的大部分信息都是相同的，不同的只在于商品在这个状态先需要存储一些额外的信息和不同状态下对商品操作的不同。基于以上分析，我们组认为可以考虑在数据结构上进行复用。

对于前端，可复用性主要集中在代码复用上。在寻宝猫 APP 中，为了使 APP 内整体具有一致性和美观性，商品显示页面之间，用户管理页面之间，聊天页面之间……这些功能类似的页面的布局应尽量相同，而这一点，决定了在前端设计时，页面的设计可以在代码层面进行复用，以减少程序量和额外的工作量。

6.5.3.2 针对可复用性添加的策略

对于后端来说，可以通过设定抽象类的方法来实现数据结构的复用，具体来说，可以将商品在不同状态中的共性的数据挑出，将这些信息定义在一个抽象类中，而将不同的商品状态定义为抽象类的继承类，如下图所示：



同时，如果使用抽象类，由于继承类之间还有访问关系，因此，还要求继承类之间需要有低的耦合度。具体来说，在理想状态下，继承类与继承类之间需要尽可能的保持在数据耦合的状态，因此，需要对继承类与继承类之间建议数据的接口函数。具体来说，不同继承类之间的关系如下：

1.处于 product_search 类中的商品可以被用户拍下进入 product_want_to_buy 类下，但此时该商品并不会从该类中删除。

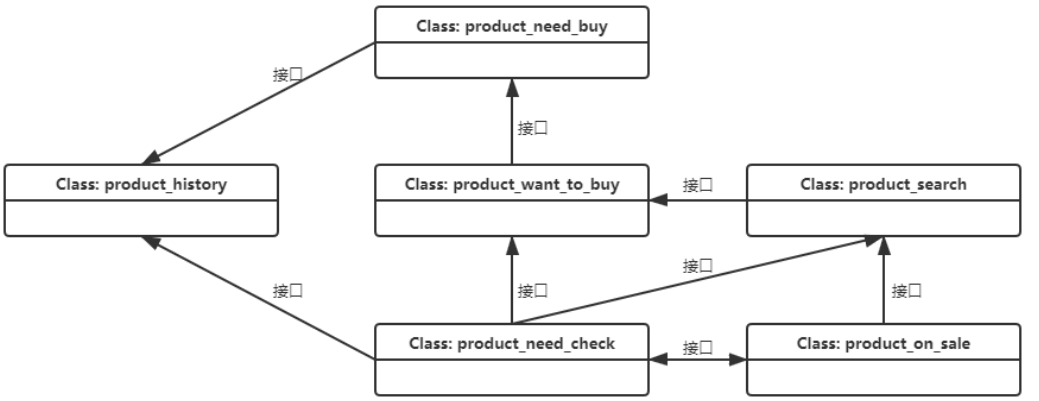
2.处于 product_want_to_buy 类下的商品接收到买方在 product_need_cheak 类中的确定后会进入到 product_need_to_buy 类中，自身被删除。

3.处于product_need_to_buy类下的商品在被买方和卖方任意一方确认后会进入到买方的 product_history 中。

4.处于 product_on_sale 类下的商品在被卖方确认之后会进入到 product_need_cheak 类中，并从该类中删除，同时也需要从 product_search 中删除。

5.处于 product_need_cheak 类下的商品在被卖方或买方任意一方确认后会进入卖方的方的 product_history 中，并从该类中删除。被取消后会回到 product_on_sale 类和 product_search 类中。

具体关系图如下：



这样，不同类之间的数据通过类之间的接口进行传输，使得类与类之间的耦合只存在于数据耦合的层面，使得抽象类的实现成为可能。同时，这样布局也有利于软件开发完成后的维护工作。

对于前端来说，可以将页面的格式部分单独写成一个标识格式的文件，在开发不同的页面时，根据不同的页面类型，可以调用不同的标识格式的文件，这样即使新开发的页面与格式之间有些许的差距，在开发时也不需要从头写起，而是在原来的页面基础上进行修改即可。这样，也便于在软件开发完成后新功能的开发，使得在软件开发完成和的软件更新和软件维护变得更加便捷。

6.5.4 安全性

安全性指系统向合法用户提供服务的同时，阻止非授权使用的能力。

6.5.4.1 安全性分析

对寻宝猫 app 来说，安全性主要分为两个方面：数据访问控制安全、页面访问控制安全。数据访问控制安全指的是寻宝猫普通用户只能够对自己的商品、订单、账户等信息进行操作，而管理员用户则可以删除违规商品，对全体用户发送公告等。而页面访问控制安全指的是只有用户名和密码符合要求才能够访问寻宝猫 app，阻止黑客等进行非法侵入访问。

6.5.4.2 针对安全性添加的策略

针对数据访问控制安全，我们可以在软件开发和设计中增加一些必要的安全防护措施，比如权限管理模块、数据加密模块、传输加密模块等，这样就可以像一般程序测试一样，进行安全功能测试。

针对页面访问控制安全，安全性战术一般包括抵抗攻击、检测攻击以及从攻击中恢复。

1. 抵抗攻击

我们把认可、机密性、完整性和确定为目标。组合使用以下策略实现：

（1）对用户身份认证。寻宝猫 app 采用山东大学统一认证的方式进行登陆，使用户

均为在校师生。

（2）对用户进行授权。授权能够保证经过身份验证对用户有权访问和修改数据与服务，通常作为管理员来实现。

（3）维护数据的机密性。对数据进行保护，以防止未经授权对访问。一般通过对数据和通讯链路进行某种形式对加密来实现机密性。另一方面，通信链路一般不具有授权控制，对于通过公共可访问对通信链路传输数据来说，加密是唯一的保护措施。

（4）限制访问。防火钱根据消息源或目的地端口来限制访问。来自未知源的消息可能是某种形式的攻击。限制对已知源对访问并不总是可行对。

2. 检测攻击

检测攻击通常通过“入侵检测”系统进行。

3. 从攻击中恢复

可以把从攻击中恢复的策略分为恢复状态相关策略和与识别攻击者相关的策略。在将系统或数据恢复到正确状态时使用的策略与用于可用性的策略发生了重叠，特别要注意维护系统管理数据的冗余副本。

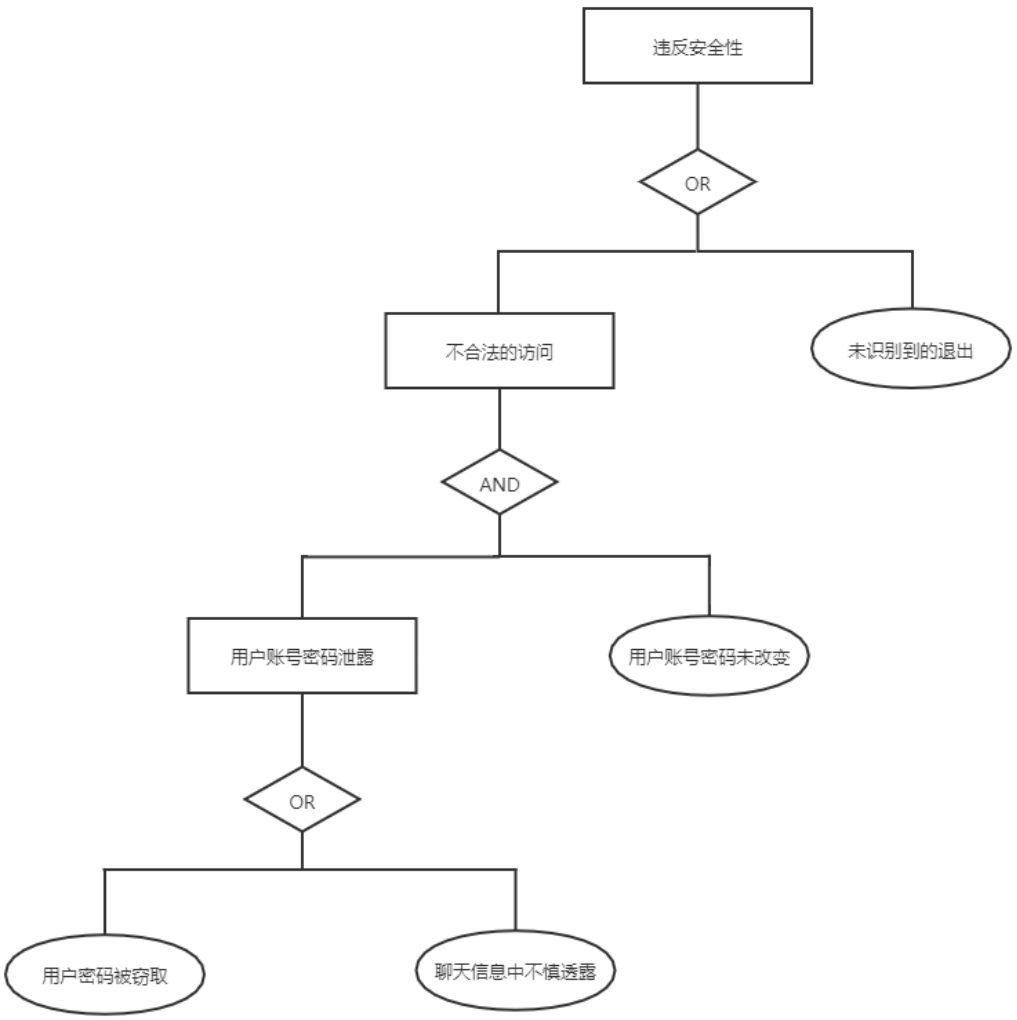
永固识别攻击者的策略是“维持审计追踪“。审计追踪就是应用到系统中的数据的所有事务和识别信息的一个副本。可以使用审计信息开追踪攻击者的操作。支持认可并支持系统恢复。

6.6. 软件故障树分析及割集树转化

在使用故障树对软件进行分析上，我们组选择了安全性，一致性，和通讯这 3 个方面，对软件可能的故障进行了分析。

6.6.1 安全性

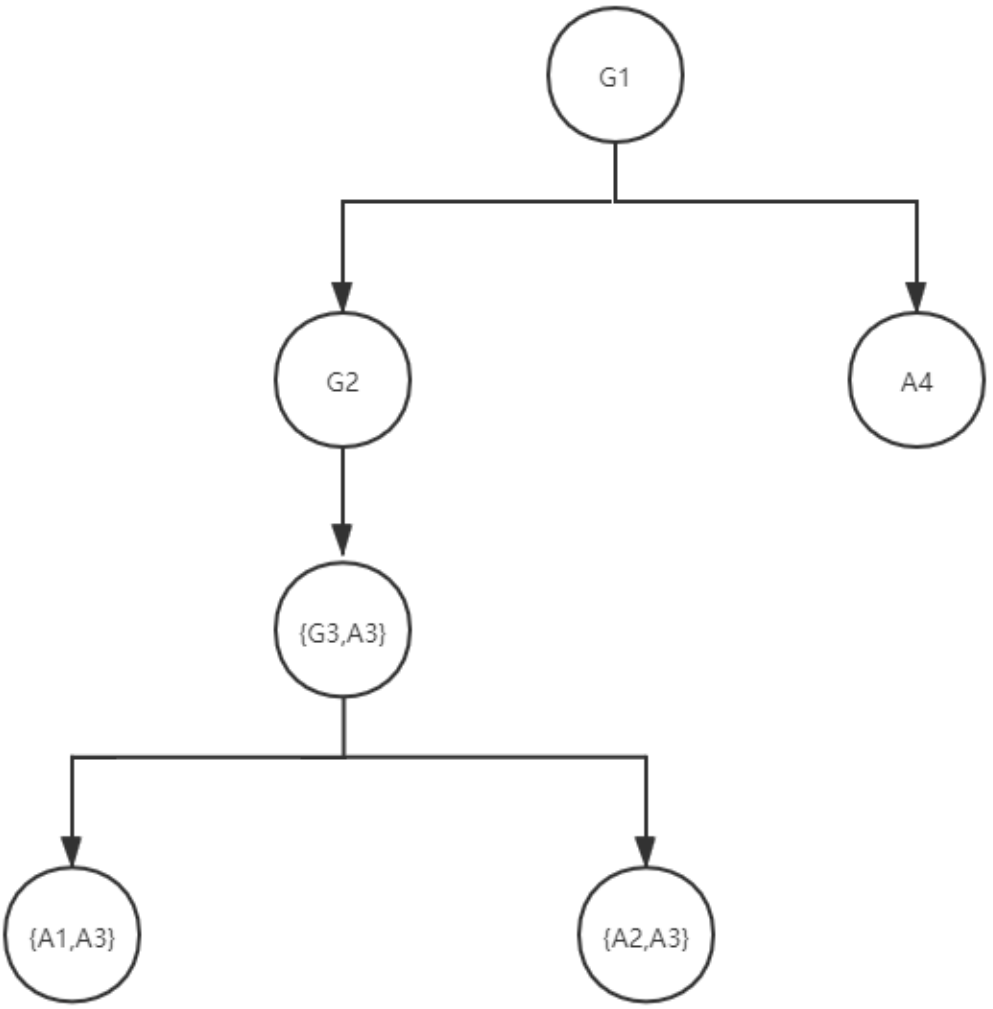
首先在安全性方面，关于这方面的故障主要是集中在用户的账户信息安全方面，在本系统中，关于用户安全性的操作集中在用户登录的时刻。当出现用户异常登录时就存在了安全性方面的问题。经过分析，得到的故障树如下图所示:



进一步，将该故障树转化为割集树，为了表示方便，现定义事件及对应简称如下：

简称	事件名称
Gi	逻辑门，只包含与门，或门
A1	用户密码被窃取
A2	聊天信息中不慎透露
A3	用户账号密码未改变
A4	未识别的退出

由以上定义，可做出对应的割集树如下图所示：



6.6.2 一致性

我们认为，另一个可能出现故障的时候是用户在使用软件进行相关操作的时候，具体来说可能会出现以下几种故障:

1.用户要求上架/下架商品，但是最终商品上架/下架操作未成功/上架/下架了错误的商品。

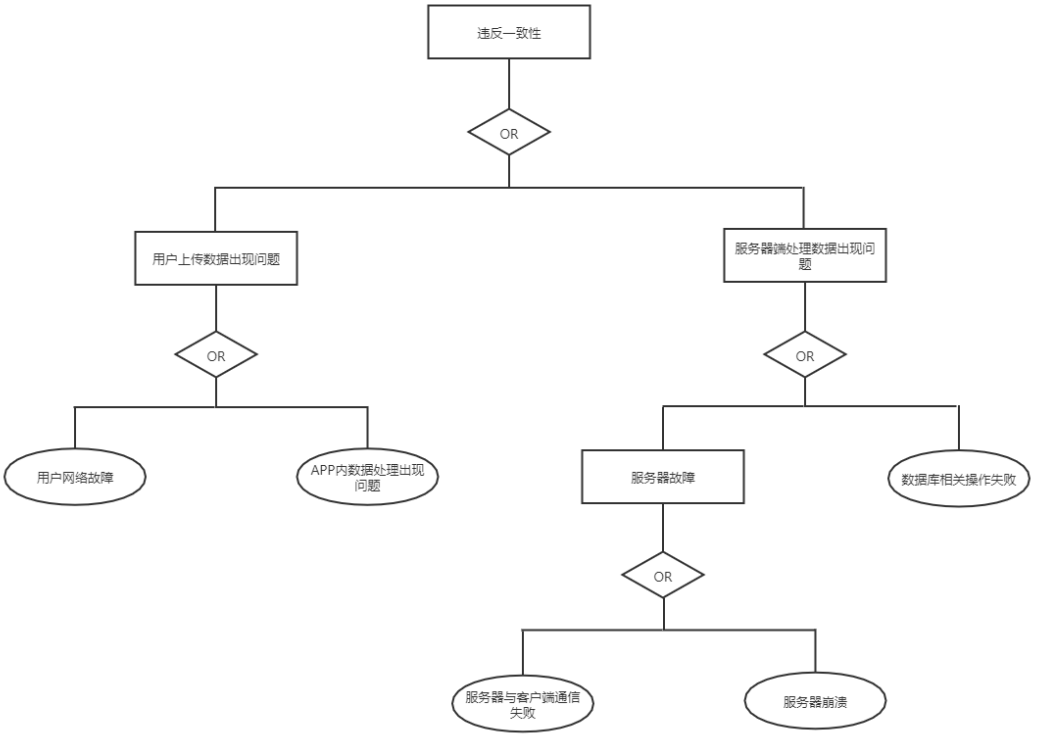
2.用户请求拍下/收藏商品，但最终对商品的操作未成功。

3.用户请求确认交易，但最终请求失败。

对这些故障来说，造成它们发生的原因有很大的共性，且最终导致的结果就是用户的请求的结果与期望不一致。因此，我们组这些问题统称为一致性问题，统一进行分析。

在一致性方面，我们组认为故障主要集中在用户在与服务器进行通信，要求对数据

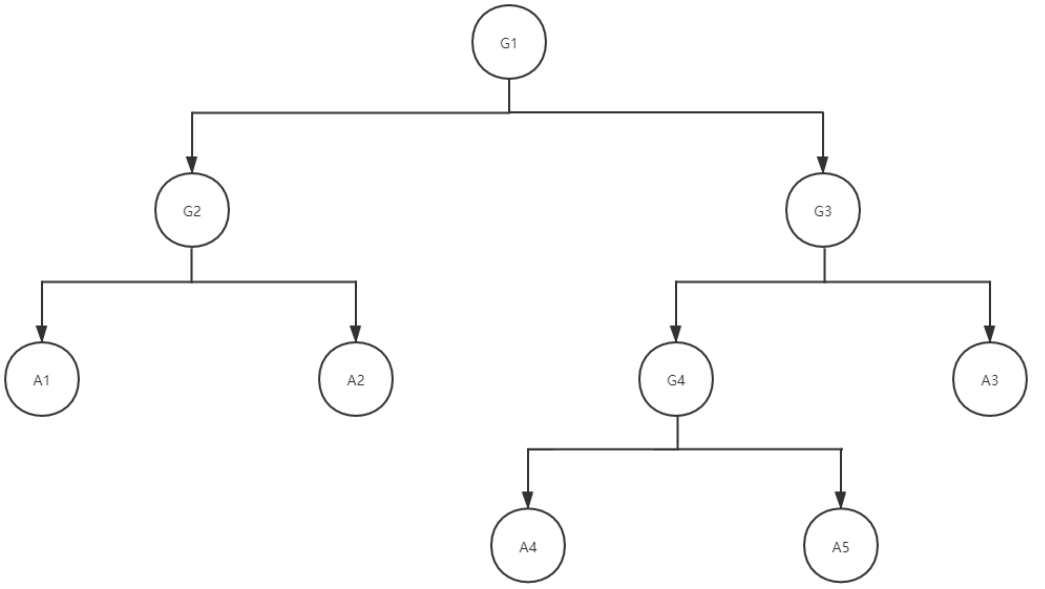
库进行修改时产生的问题，导致处理结果与预期结果产生的不一致的问题，经过分析，该问题的故障树如下图所示：



进一步，将该故障树转化为割集树，为了表示方便，现定义事件及对应简称如下：

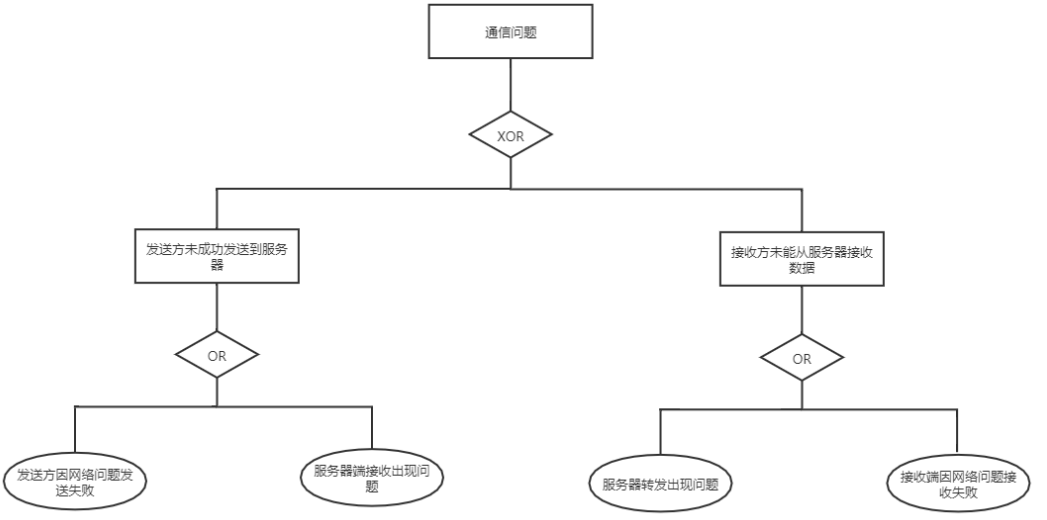
简称	事件名称
Gi	逻辑门，只包含与门，或门
A1	用户网络故障
A2	APP 内数据处理出现问题
A3	数据库相关操作失败
A4	服务器与客户端通信失败
A5	服务器崩溃

由以上定义，可做出对应的割集树如下图所示：



6.6.3 通信

在通信方面，即卖方和卖方在进行聊天时可能会出现聊天内容发送失败的问题。我们组将该故障定义为通信问题。经过分析，该故障的故障树如下图所示:



进一步，将该故障树转化为割集树，为了表示方便，同时由于异或门的关系，使用逻辑代数简化后需要引入对立事件，因此现定义事件和其对立事件的对应简称如下:

简称	事件名称
Gi	逻辑门，只包含与门，或门

A1	发送方因网络问题发送失败
A2	服务器接收出现问题
A3	服务器转发出现问题
A4	发送方因网络问题接收失败
A5	发送方发送成功，即 A1+A5=U
A6	服务器接收成功，即 A2+A6=U
A7	服务器转发成功，即 A3+A7=U
A8	接收方接收成功，即 A4+A8=U

由以上定义，可做出对应的割集树如下图所示：