线上打车软件系统

体系结构设计文档

Ver 2.0

2019.12.9

刘子豪

2017211971

软件工程导论

2019秋季学期

# 版本历史

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **描述** | **作者** | **备注** |
| 2019.12.8 | Ver 1.0 | 刘子豪 | 体系结构框架设计 |
| 2019.12.11 | Ver 2.0 | 刘子豪 | 体系结构细化与文档编写 |
|  |  |  |  |
|  |  |  |  |

# 文档许可

以下体系结构设计报告已经被以下机构人员批准并认可：

|  |  |  |  |
| --- | --- | --- | --- |
| **签名** | **姓名** | **标题** | **日期** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[版本历史 2](#_Toc27055234)

[文档许可 2](#_Toc27055235)

[1. 引言 5](#_Toc27055236)

[1.1. 编写目的 5](#_Toc27055237)

[1.2. 范围 5](#_Toc27055238)

[1.3. 定义、术语和缩写语 6](#_Toc27055239)

[1.4. 参考文献 6](#_Toc27055240)

[2. 体系结构概述 7](#_Toc27055241)

[2.1. 项目概述 7](#_Toc27055242)

[2.2. 用户角色 7](#_Toc27055243)

[2.3. 系统功能 8](#_Toc27055244)

[2.4. 架构视角 9](#_Toc27055245)

[3. 体系结构目标和约束 10](#_Toc27055246)

[3.1. 体系结构目标 10](#_Toc27055247)

[3.2. 体系结构约束 10](#_Toc27055248)

[3.2.1. 基本约束 10](#_Toc27055249)

[3.2.2. 系统约束 10](#_Toc27055250)

[3.2.3. 工程设计约束 11](#_Toc27055251)

[4. 用例视图 11](#_Toc27055252)

[4.1. 用例图——乘车者 12](#_Toc27055253)

[4.2. 用例图——司机 13](#_Toc27055254)

[4.3. 用例图——系统管理员 13](#_Toc27055255)

[4.4. 系统整体用例图 14](#_Toc27055256)

[4.5. 重要用例简介 14](#_Toc27055257)

[5. 逻辑视图 16](#_Toc27055258)

[5.1. 相关体系结构模式——三层B/S体系结构模式 16](#_Toc27055259)

[5.2. 本项目体系结构概述——包和子系统的分层架构 17](#_Toc27055260)

[5.3. 本项目体系结构概述——类图以及关系图 18](#_Toc27055261)

[5.4. 本项目体系结构概述——详细的逻辑视图 21](#_Toc27055262)

[6. 进程视图 22](#_Toc27055263)

[6.1. 进程 22](#_Toc27055264)

[6.1.1. 身份验证进程 23](#_Toc27055265)

[6.1.2. 身份验证控制器 23](#_Toc27055266)

[6.1.3. 乘车者应用进程 23](#_Toc27055267)

[6.1.4. 乘车者应用控制器 24](#_Toc27055268)

[6.1.5. 司机应用进程 24](#_Toc27055269)

[6.1.6. 司机应用控制器 24](#_Toc27055270)

[6.1.7. 系统管理员应用进程 24](#_Toc27055271)

[6.1.8. 系统管理员应用控制器 24](#_Toc27055272)

[6.1.9. 信息管理进程 25](#_Toc27055273)

[6.1.10. 信息管理控制器 25](#_Toc27055274)

[6.1.11. 打车进程 25](#_Toc27055275)

[6.1.12. 打车进程控制器 25](#_Toc27055276)

[6.1.13. 地图导航进程、地图系统、导航系统 25](#_Toc27055277)

[6.1.14. 费用支付进程、支付系统 26](#_Toc27055278)

[6.2. 进程设计元素 26](#_Toc27055279)

[6.2.1. 个人信息缓存 26](#_Toc27055280)

[6.2.2. 订单信息缓存 27](#_Toc27055281)

[6.3. 进程模型依赖设计 27](#_Toc27055282)

[6.4. 进程实现 28](#_Toc27055283)

[6.4.1. Socket类 28](#_Toc27055284)

[6.4.2. Remote接口 28](#_Toc27055285)

[6.4.3. Runnable接口 28](#_Toc27055286)

[6.4.4. 实现细节 29](#_Toc27055287)

[7. 部署视图 29](#_Toc27055288)

[7.1. 乘车者手机终端 29](#_Toc27055289)

[7.2. 司机手机终端 30](#_Toc27055290)

[7.3. 系统管理员终端 30](#_Toc27055291)

[7.4. 系统应用服务器 30](#_Toc27055292)

[7.5. 系统数据库 30](#_Toc27055293)

[7.6. 第三方软件系统 31](#_Toc27055294)

[8. 规模与性能 31](#_Toc27055295)

[9. 系统的质量要求 31](#_Toc27055296)

[9.1. 可靠性 31](#_Toc27055297)

[9.2. 易用性 32](#_Toc27055298)

[9.3. 密安性 32](#_Toc27055299)

[9.4. 可维护性 32](#_Toc27055300)

[9.5. 可移植性 33](#_Toc27055301)

[9.6. 可扩展性 33](#_Toc27055302)

[10. 质量分析和评价 33](#_Toc27055303)

[10.1. 场景分析 33](#_Toc27055304)

[10.1.1. 用例场景 33](#_Toc27055305)

[10.1.2. 增长性场景 34](#_Toc27055306)

[10.1.3. 探索性场景 35](#_Toc27055307)

[10.2. 原型分析 35](#_Toc27055308)

# 引言

## 编写目的

本文档是在对线上打车软件系统的功能需求和非功能需求进行了全面的分析之后，对软件的体系结构进行多方面的设计之后编写的文档。该文档的内容全面，对用例视图、逻辑视图、进程视图和部署视图进行了详细的设计。旨在为读者展示该系统的总体结构，包括逻辑设计、物理结构，分析系统的体系结构需求，包括约束条件、系统规模、性能指标、质量需求等，并使用多种视图给出体系结构设计的解决方案并分析建模，最后进行体系结构的质量分析。

该文档作为产品立项和产品开发的参考文档，给出各用户详细的功能要求，系统功能块组成及联系，进程部署和硬件要求等，有益于提高软件开发过程中的能见度，便于软件开发过程中的控制与管理。此体系结构设计文档是进行软件项目设计开发的基础，也是编写测试用例和进行系统测试的主要依据，它对开发的后续阶段性工作起着指导作用。同时此文档也可作为软件用户、软件客户、开发人员等各方进行软件项目沟通的基础。

本文档的预期读者对象为：

1. 开发人员：根据本文档了解预期项目的功能，并据此进行系统设计与开发。
2. 项目经理：根据体系结构定义的构件结构制定项目的开发计划。
3. 测试人员：根据体系结构设计系统的总体测试框架。
4. 维护人员：根据本文档中确定的体系结构进行软件系统维护。
5. 用户：了解预期项目的功能和性能与整体结构。
6. 其他相关人员：如用户文档编写者、项目管理人员等。

## 范围

* 产品名称：线上打车管理系统（OTMS）。
* 产品目标用户：打车者和出租车司机。
* 产品设计背景：在一些大型城市中，人们的打车需求越来越大，而在大多数情况下，乘客在路口很难找到一辆空载的出租车。与此同时，出租车也很难能够找到需要乘车的乘客，因此同时出现了乘客打车难，司机空车率高的问题。
* 文档内容：本体系结构设计文档概括地描述了线上打车软件系统的主要功能，阐述了系统的总体结构，说明了系统的总体设计策略，给出了体系结构设计的解决方案并分析建模，最后进行体系结构的质量分析和评估。
* 应用范围：本软件体系结构设计文档适合于城市共享停车管理系统的总体应用结构，目的是满足系统的质量和可信赖性要求，以及线上打车软件系统未来的维护、运行和升级改造等要求。

## 定义、术语和缩写语

|  |  |
| --- | --- |
| 术语/缩写语 | 定义描述 |
| OTMS | 线上打车软件系统 |
| 乘车者 | 系统的最终用户，通过使用该系统来完成叫车等操作 |
| 司机 | 系统的最终用户，通过该系统完成寻找乘车人等工作 |
| 系统管理员 | 系统的最终用户，进行系统的用户管理，以及业务管理等工作 |
| 用户 | 包括乘车者和司机在内，表示软件的最终使用人员 |
| B/S结构 | B/S结构（Browser/Server，浏览器/服务器模式），是WEB兴起后的一种网络结构模式，WEB浏览器是客户端最主要的应用软件。这种模式统一了客户端，将系统功能实现的核心部分集中到服务器上，简化了系统的开发、维护和使用。 |
| 三层B/S结构 | 三层B/S体系结构是在B/S结构的基础上，在数据管理层(Server)和用户界面层(Client)增加了一层结构，称为中间件(Middleware)，使整个体系结构成为三层。 |

表1 定义、术语和缩写语

## 参考文献

[1] 王安生.《软件工程化》[M]. 北京：清华大学出版社，2014

# 体系结构概述

## 项目概述

该系统为线上打车管理系统，该系统用于打车事件的管理。

该项目的动机为：在一些大型城市中，人们的打车需求越来越大，而在大多数情况下，乘客在路口很难找到一辆空载的出租车。与此同时，出租车也很难能够找到需要乘车的乘客，因此同时出现了乘客打车难，司机空车率高的问题。移动互联网和智能终端的高速发展为利用打车软件解决该问题提供了机遇。

通过使用该系统，乘车者在线上即时找到空载的司机，并及时送达至目的地。司机使用该系统，能够找到附近所有需要乘车的顾客，并提供准确的路径导航功能。该系统能够同时解决乘车者打车难，司机空车率高的问题，让打车这件事情不再成为难题。

## 用户角色

在OSTM中，有三类用户角色：乘车者、司机以及系统管理员。这三类用户角色有不同的系统使用权限，不同用户角色对系统使用的需求不同，具体的需求如下所示：

* 对于乘车者：
  + 乘车者能够查看并修改自己的个人信息。
  + 打车时，系统能够显示乘车者周边的地图信息，并显示乘车者的当前位置以及周边的司机位置信息。
  + 乘车者能够选择上车位置，以及服务类型，目的地信息等。
  + 输入行程信息后，系统能够显示预期需要交付的行程费数额。
  + 当有司机抢到单时，系统能够显示司机的基本信息以及位置信息。
  + 司机到达之后，乘车者能够看到车辆导航信息。
  + 乘车者有任何疑难问题等，可以向客服询问。
* 对于司机：
  + 司机能够在系统中查看并修改自己的个人信息
  + 系统能够显示司机周边的地图信息。
  + 系统能够显示附近的订单信息，司机可以使用该系统选择想要接的订单，并进行抢单操作。
  + 接到乘车者之后，系统能够为司机提供导航信息，提供路径规划服务。
  + 将乘车者送到目的地之后，系统能够计算得到本次订单产生的金额，并要求用户在24小时内完成支付。
  + 司机有任何疑难问题等，可以向客服询问。
* 对于系统管理员：
  + 当一个订单存在交易错误时，能够查看该订单并进行管理。
  + 对用户的认证信息进行管理。

## 系统功能

在上一部分中，对不同最终用户所需要实现的系统功能进行了详细描述，因此在本部分给出系统的功能结构图：

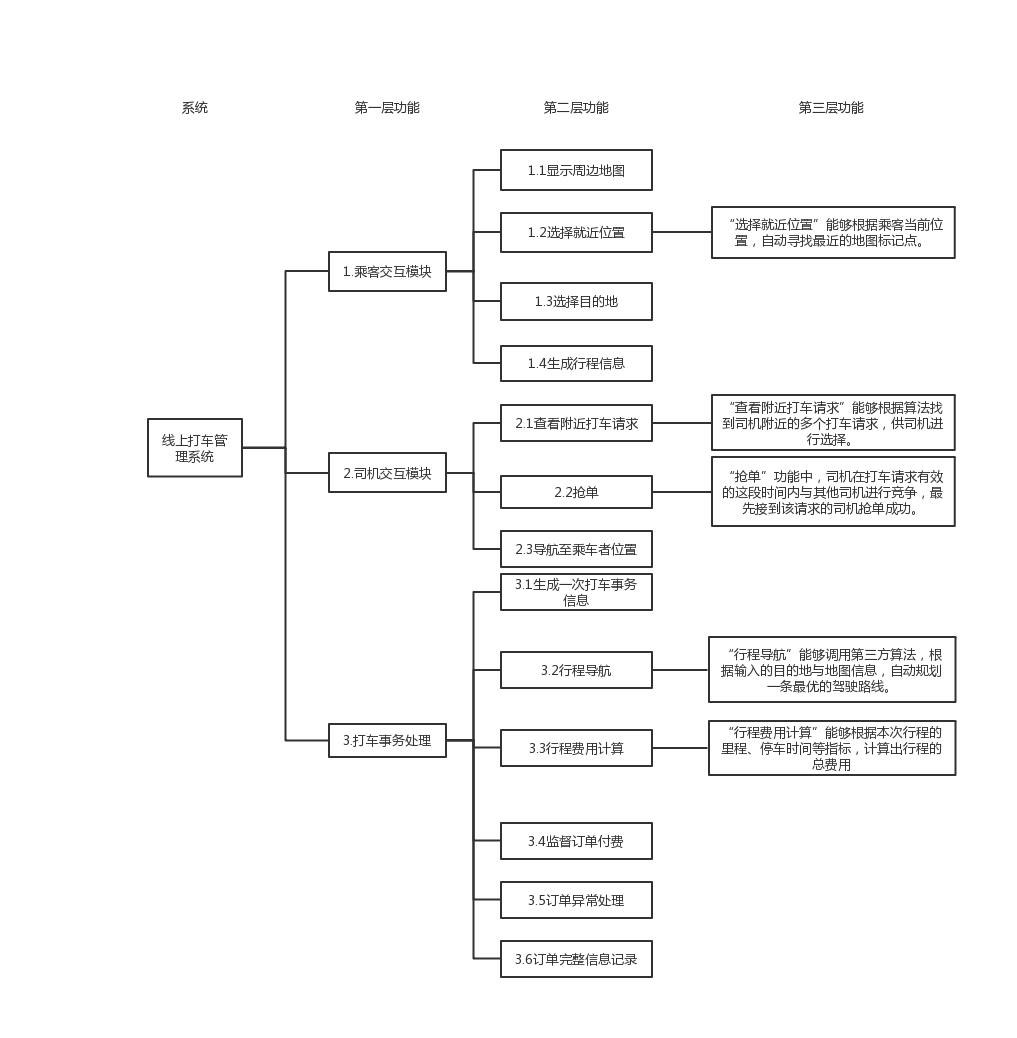


图2-1 系统功能结构图

## 架构视角

该系统的体系结构是从多个视角（用例视角，逻辑视角，进程视角，部署视角）进行展示的，这样能够从不同方面指出相关利益方关心的事情，外加从使用者的角度对用例进行观察，分析其影响系统的上下文和商业目标情况。图2-1中，比较形象地描述了几个视角的关系。本文档中没有对每个视角进行单独的实现，只是通过UML建模语言给出相应的模型描述。

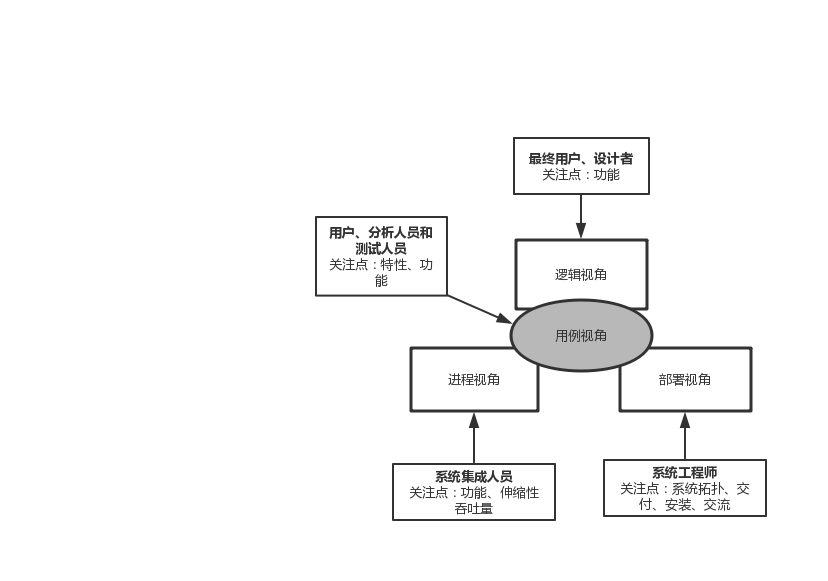


图2-2 系统体系结构概述

在该模型中，相关利益人员各自的要求如下所示：

系统的最终用户和设计者要求系统可以实现需求分析阶段的基本功能，对于OSTM，要求系统能够满足需求分析阶段为乘车者、司机、系统管理员等用户定义的所有需求，包括它们的功能需求和非功能需求。

用户、分析人员和测试人员要求系统满足需求分析阶段设想的系统特性，要求系统能正确可靠的运行，数据完整，系统可操作性强且用户体验良好。

系统集成人员要求系统满足需求分析阶段提出的性能指标，如系统时间性能和空间性能上的要求。要求系统各个模块和层次之间尽量松耦合以方便各个部件之间的集成。

程序员要求系统软件各部分结构合理，方便进行修改、升级、测试和维护，要求系统接口设计合理，便于进行某些模块功能代码的重用，方便被当作模版框架来进行其他系统的二次开发。

系统工程师要求系统在规定时间内完成并交付，并在系统交付时对项目负责人进行相应的培训以便其正确使用系统。

# 体系结构目标和约束

## 体系结构目标

通过设计在本文档中描述的软件体系结构，在系统实现阶段对其进行实现后，系统能够实现在需求分析阶段定义的所有需求，并且满足非功能需求的要求，比如可靠性、可移植性、性能等各个方面。

## 体系结构约束

### 基本约束

|  |  |
| --- | --- |
| 基本要素 | 主要约束 |
| 法律和政策约束 | 软件开发过程中，必须严格遵守法律规范。 |
| 设备约束 | 软件必须能够在安卓系统和IOS系统上可运行 |
| 项目基本运行范围 | 完成项目的开发与测试 |
| 项目开发时间 | 6个月 |
| 项目开发成本 | 10000~20000元 |

表2 基本约束

### 系统约束

在体系结构设计过程中，有以下几个关键的系统约束需要得到满足：

1. 所有的用户都需要进行相应的身份验证，身份验证成功后才可以使用本系统。
2. 对于不同的功能，不同的用户有不同的使用权限，当用户使用某个功能时首先需要确认其身份。
3. 本系统必须能够保护每一个用户的所有数据，保证不会因为外部攻击或者未授权的访问而泄露。
4. 系统中的支付模块必须通过接口与第三方支付软件进行连接，以支持系统的付款功能，这个接口将会在之后的接口规格说明中进行详细定义。
5. 系统中的地图导航模块必须能够通过接口与第三方地图导航软件进行连接，以支持在系统中访问用户当前位置的地图，并在打车的过程中进行路径导航，这个接口将会在之后的接口规格说明中进行详细定义。
6. 所有的用户都可以通过网络（4G网络，Wifi等）连接到本系统中，从而使用系统并与系统进行交互。
7. 该系统使用客户端-服务端结构进行体系结构的设计，客户端系统可以部署到每个用户的手机终端中，服务器端必须部署到内部专用的UNIX服务器中。
8. 在需求分析规格说明文档中定义的所有系统需求，在本文档中都需要考虑在内并在体系结构设计中得到实现。

### 工程设计约束

|  |  |  |
| --- | --- | --- |
| 设计要素 | | 主要约束 |
| 运行环境 | 操作系统 | Android系统/IOS系统/Linux系统 |
| 数据库 | MySQL 14.0及以上 |
| 微信小程序 | Android系统/IOS系统  系统中装有微信软件 |
| 用户端软件 | 操作系统 | Android系统/IOS系统 |
| 开发环境支持 | 操作系统 | Linux Ubuntu 16.04及以上 |
| 开发工具 | Eclipse等可用Java进行开发的IDE |
| Web框架 | Spring MVC |
| 内存 | 16GB |

表3 工程设计约束

# 用例视图

该部分对软件体系结构的用例视图进行详细描述。用例视图对于选择每次迭代中关键的用例场景是关键的输入，它描述了用例场景的集合以及代表一些重要的，核心的功能的用例。该系统中的用户角色包括乘车者、司机和系统管理员，针对这些用户角色，给出他们相应的用例视图。

## 用例图——乘车者

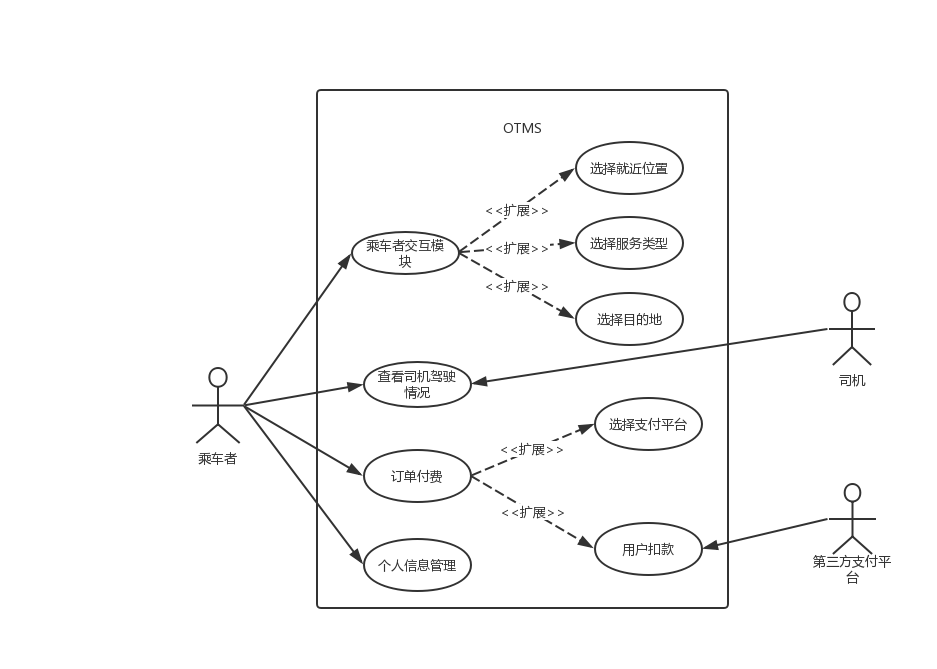


图4-1 用例图——乘车者

在用例图中，乘车者的主要功能为：

1. 与系统交互模块，包括选择就近位置，服务类型，目的地
2. 订单付费
3. 查看司机驾驶情况
4. 个人信息管理

## 用例图——司机

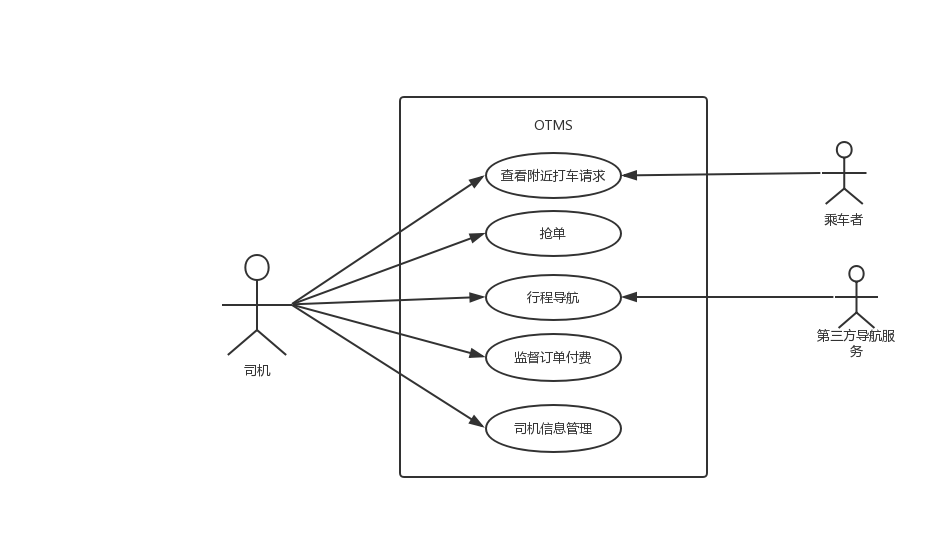


图4-2 用例图——司机

可见，在该用例图中，司机的主要功能为：

1. 查看附近打车请求
2. 抢单
3. 行程导航
4. 监督乘车者的订单付费
5. 司机信息管理

## 用例图——系统管理员

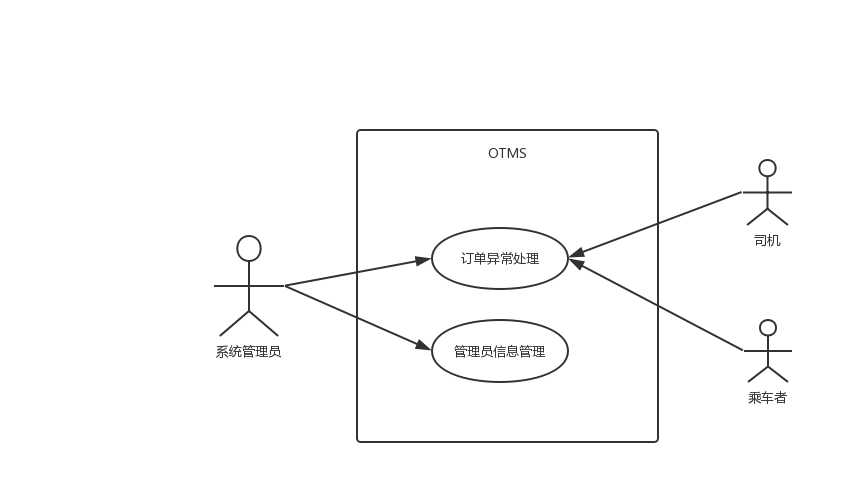


图4-3 用例图——系统管理员

在该系统中，系统管理员主要负责的是订单异常情况下的处理。

## 系统整体用例图

下图中，对关键的用例进行整合，并得到系统的整体用例图：

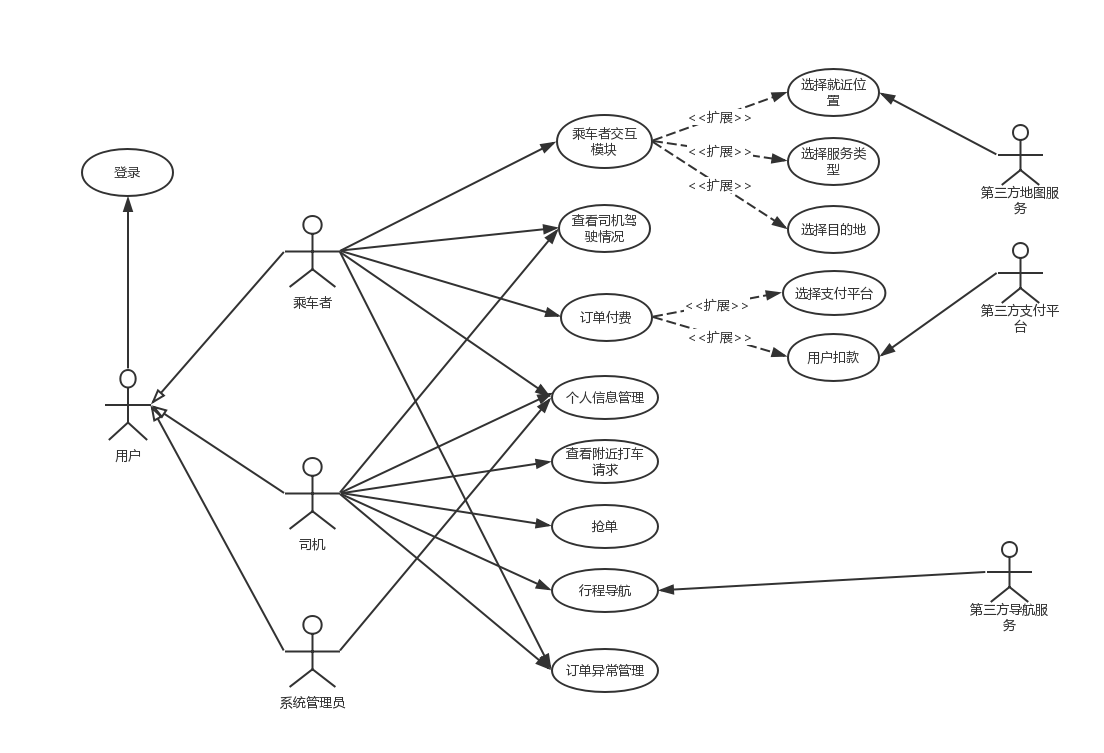


图4-4 整体用例图

## 重要用例简介

该部分中，对一些重要的用例进行说明：

* + 1. **选择就近位置**
       1. 介绍

根据用户当前的地理位置，得到几个就近的可乘车位置，以便让用户选择。

* + - 1. 输入

系统通过调用第三方服务，获得用户的地理位置作为输入。

* + - 1. 过程

系统通过调用高德地图的服务来获得多个满足要求的上车点，将这些上车点与乘客位置比较，找到与乘客距离最近的几个乘车点，并将其显示在终端上。

* + - 1. 输出

多个乘车点的地图坐标以及详细地址（字符串形式）。

* + 1. **查看附近打车需求**
       1. 介绍

根据司机当前的地理位置，得到多个附近的乘车请求，可让司机选择。

* + - 1. 输入

司机的地理位置（通过调用第三方服务获得），乘客的行程信息（在前面的乘客交互模块中生成）。

* + - 1. 过程

系统对未处理的乘客行程信息进行筛选，根据行程信息中的乘车位置，找到与司机距离较近的几个乘车请求，将其显示在终端上。

* + - 1. 输出

多个乘客乘车请求以及这些请求对应的详细行程信息。

* + 1. **抢单**
       1. 介绍

司机想要接受某一个乘客的乘车请求时，使用该功能与其他司机进行竞争，如果竞争成功，司机即可接受该请求，并开始为该乘客服务。

* + - 1. 输入

司机想要接受的乘车请求。

* + - 1. 过程

乘客发出行程信息之后，系统会增加一个计时器，用来判断规定时间内有哪位司机接到了这个请求。

司机想好了要接受乘客的请求之后，点击“抢单”按钮进行抢单，与其他司机竞争。

乘客端的系统一旦收到了一位司机的抢单信息，便会终止此次请求，防止其他司机接到这个单。

* + - 1. 输出

如果司机抢单成功，系统会弹出“成功”的窗口，并进入为乘客服务的界面；如果司机抢单失败或者该单已经超时，系统则会弹出“失败”的窗口，并在界面上将该请求删除。

* + 1. **行程导航**
       1. 介绍

系统可为司机提供本次行程的导航，提供最优的驾驶路线。

* + - 1. 输入

本次行程的出发地和目的地。

* + - 1. 过程

系统调用高德地图等第三方服务，得到多条路径规划方案，司机根据不同的需求（高速优先，时间最短等等），选择最佳的路径方案之后，系统就会为司机提供路线上细致的导航。

* + - 1. 输出

多个路径规划方案。

* + 1. **监督订单付费**
       1. 介绍

系统根据本次行程信息自动计费，订单产生后，乘客需要在24小时之内完成支付。

* + - 1. 计费标准

本次行程的部分信息，包括总里程，停车时间等。

* + - 1. 用户缴费

系统提供多种第三方支付平台，用户可自由选择不同的支付平台进行支付。

* + - 1. 未付费时的系统处理

如果乘客未付费，系统会禁止乘客使用系统的基本功能，直到用户付钱。

如果乘客在24小时之内仍未付款，系统会给予用户警告，警告大于2次时会冻结该用户，直到该用户完成所有的订单。

# 逻辑视图

该部分对体系结构的逻辑视图进行描述，描述内容包括最重要的类，它们在服务包和子系统中的组织，这些子系统在分层结构中的组织，还包括了最重要的用例实现。使用类图来说明具有体系结构意义的类、子系统、包和层之间的关系。

## 相关体系结构模式——三层B/S体系结构模式

B/S结构（Browser/Server，[浏览器](https://baike.baidu.com/item/%E6%B5%8F%E8%A7%88%E5%99%A8)/[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8)模式），是[WEB](https://baike.baidu.com/item/WEB)兴起后的一种网络结构模式，WEB浏览器是[客户端](https://baike.baidu.com/item/%E5%AE%A2%E6%88%B7%E7%AB%AF)最主要的[应用软件](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E8%BD%AF%E4%BB%B6)。这种模式统一了[客户端](https://baike.baidu.com/item/%E5%AE%A2%E6%88%B7%E7%AB%AF)，将系统功能实现的核心部分集中到[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8)上，简化了系统的开发、维护和使用。客户机上只要安装一个[浏览器](https://baike.baidu.com/item/%E6%B5%8F%E8%A7%88%E5%99%A8)，如[Netscape Navigator](https://baike.baidu.com/item/Netscape%20Navigator)或[Internet Explorer](https://baike.baidu.com/item/Internet%20Explorer)，[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8)安装[SQL Server](https://baike.baidu.com/item/SQL%20Server)、[Oracle](https://baike.baidu.com/item/Oracle)、MYSQL等数据库。[浏览器](https://baike.baidu.com/item/%E6%B5%8F%E8%A7%88%E5%99%A8)通过Web Server 同数据库进行数据交互。

而三层B/S体系结构是在B/S结构的基础上，在数据管理层(Server)和用户界面层(Client)增加了一层结构，称为中间件(Middleware)，使整个体系结构成为三层。

三层结构是伴随着中间件技术的成熟而兴起的，核心概念是利用中间件将应用分为表示层、业务逻辑层和数据存储层三个不同的处理层次，三个层次的划分是从逻辑上分的，具体的物理分法可以有多种组合。

## 本项目体系结构概述——包和子系统的分层架构

以上文提到的三层B/S体系结构为基础，根据本系统中所涉及到的需求与用例实现，对本系统的体系结构逻辑视图进行设计。

基于三层B/S体系结构，本系统的逻辑视图由三个包/层组成，分别为用户表示层、业务服务层、业务对象层，具体的逻辑包图如下所示：

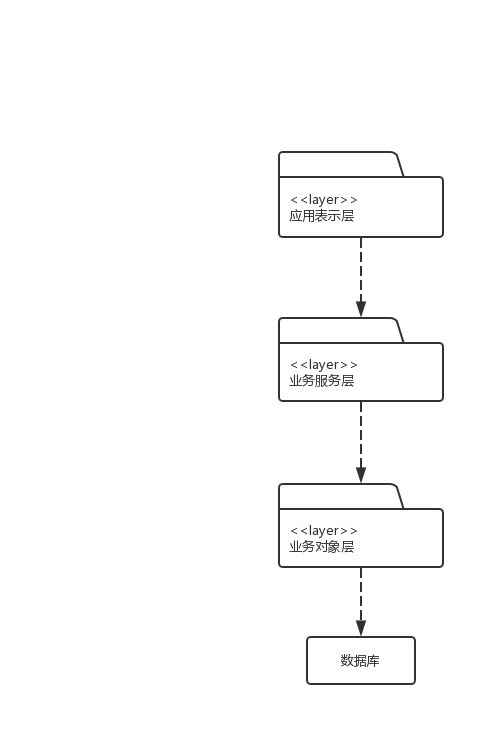


图5-1 逻辑包图

* + 1. **应用表示层**

在该体系结构中，应用表示层包含了所有与客户端UI界面有关的类，这些类负责将系统的信息以友好、美观的方式展示给用户，同时能够与用户进行直接交互通信。

这些类为用户提供输入和输出，用户对系统进行操作之后，这些类能够响应用户的操作，并将用户请求提交给业务服务层，当业务服务层返回结果时，这些类能够将这个结果反馈给用户。通过该层，将业务逻辑与用户界面设计分离开，从而简化了客户端的操作。

* + 1. **业务服务层**

业务服务层包含了系统实现业务逻辑与数据操作的核心控制类，它的任务是接收从业务服务层发来的请求，首先执行扩展的应用程序并与数据库进行链接，通过SQL方式向数据库服务器提出数据处理申请，然后等到数据库服务器将数据处理的结果提交给Web服务器，再由Web服务器将结果传回给客户端。它提供所有的业务逻辑处理功能，整个系统中对数据库的操作都在这一层中完成。

* + 1. **业务对象层**

业务对象层中的类将数据库中的数据进行封装，方便业务服务层的类进行调用。该层中的类直接与系统的数据库服务器进行连接，负责进行数据的直接访问与存储。它们的任务是接受中间层对数据库操作的请求，实现对数据库的查询、修改、更新等功能，并将运行结果返回给业务服务层。

## 本项目体系结构概述——类图以及关系图

在本部分中，以该系统的需求分析规格说明为基础，为上文中提到的包所包含的类进行设计。

由于用户表示层中的类与GUI设计有关，与系统的体系结构设计关系不大，因此在本文档中对用户表示层中的类不进行详细说明。

* + 1. **业务对象层的类图**

业务对象层中的对象主要是将数据库中的数据记录以类的形式进行封装，因此该层中主要包括四个实体类：乘车者类、司机类、系统管理员类和订单类。

* + - 1. **类图——乘车者类**



图5-2 类图——乘车者类

乘车者类中有下面5个属性：

* 用户ID：系统分配给每个用户的ID，用来唯一标识用户。
* 用户名：在软件中与其他用户交互时显示的名字。
* 密码：用户登录需要的密码。
* 注册时间：用户注册时间。
* 支付方式：乘车者选择的默认支付方式（微信、支付宝等）。
  + - 1. **类图——司机类**

****

图5-3 类图——司机类

司机类中有下面6个属性：

* 用户ID：系统分配给每个用户的ID，用来唯一标识用户。
* 用户名：在软件中与其他用户交互时显示的名字。
* 密码：用户登录需要的密码。
* 注册时间：用户注册时间。
* 车牌号：司机所使用车辆的车牌号。
* 车辆型号：司机所使用车辆的型号信息。
  + - 1. **类图——系统管理员类**

****

图5-4 类图——系统管理员类

系统管理员类中，主要包含ID和密码两个属性，具体内容在上面已经进行了描述。

* + - 1. **类图——订单类**



图5-5类图——订单类

将订单信息单独封装成一个类，类中的属性包含了一个打车订单的所有详细信息，包括：

* 始发地：本次行程的始发地，类型为Address。
* 目的地：本次行程的终点，类型为Address。
* 总里程：本次行程的总行驶距离，类型为Distance，描述一段路程的距离大小。
* 驾驶时间：本次行程总共行驶的时间，类型为Time。
* 总金额：计算得到的本次行程总金额，类型为Money。
  + - 1. **实体-关系图（ER图）**

通过使用ER图来说明这些实体类之间的关系和多重性：

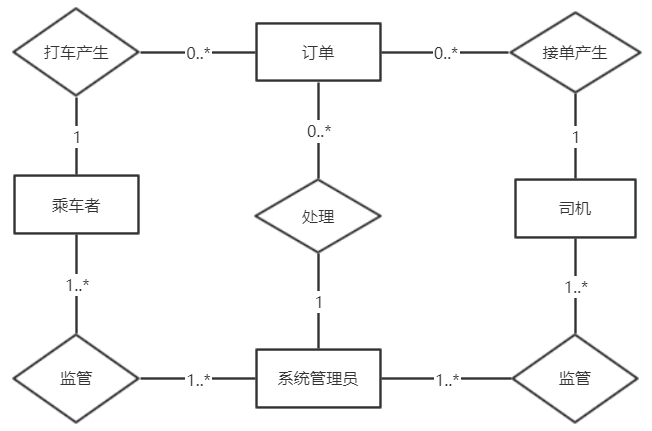


图5-6 ER图

* + 1. **业务服务层的类图**

业务服务层中的类对与系统的业务逻辑与数据操作进行控制和实现，在业务服务层中，主要包含以下几个类：

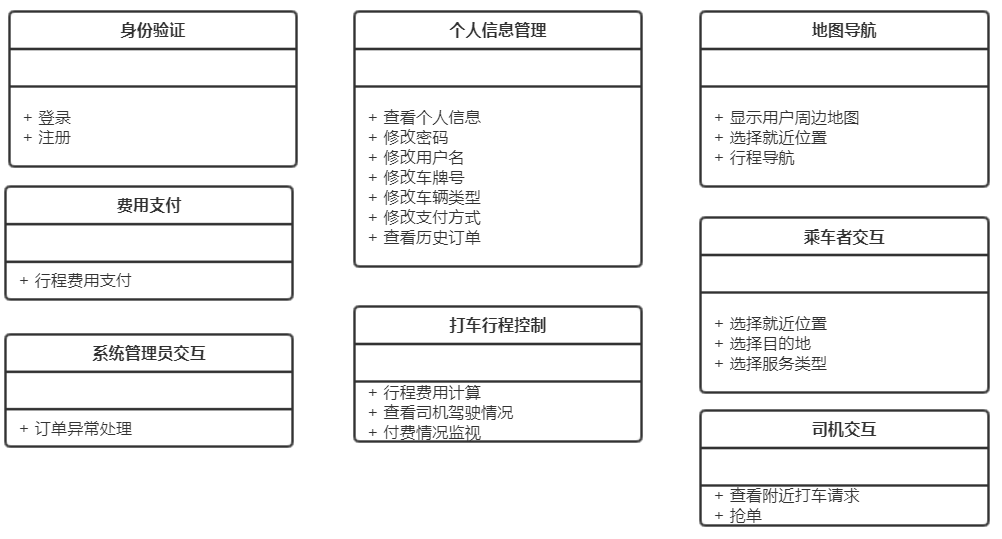


图5-7 业务服务层的所有类图

在该层中，每个控制类都包含相应的方法，表示对业务逻辑进行处理。通过以上的控制类，系统能够将所有的业务逻辑和数据操作都交给这些类来完成，提高了系统的性能和效率。

## 本项目体系结构概述——详细的逻辑视图

在对所有的类进行了详细地设计之后，逻辑视图的内容也就相应完整了，下图展示了详细的逻辑视图，包含三个主要的包以及关键的几个数据类/控制类。通过这样设计，系统的业务逻辑和数据操作均能够有序地进行处理。

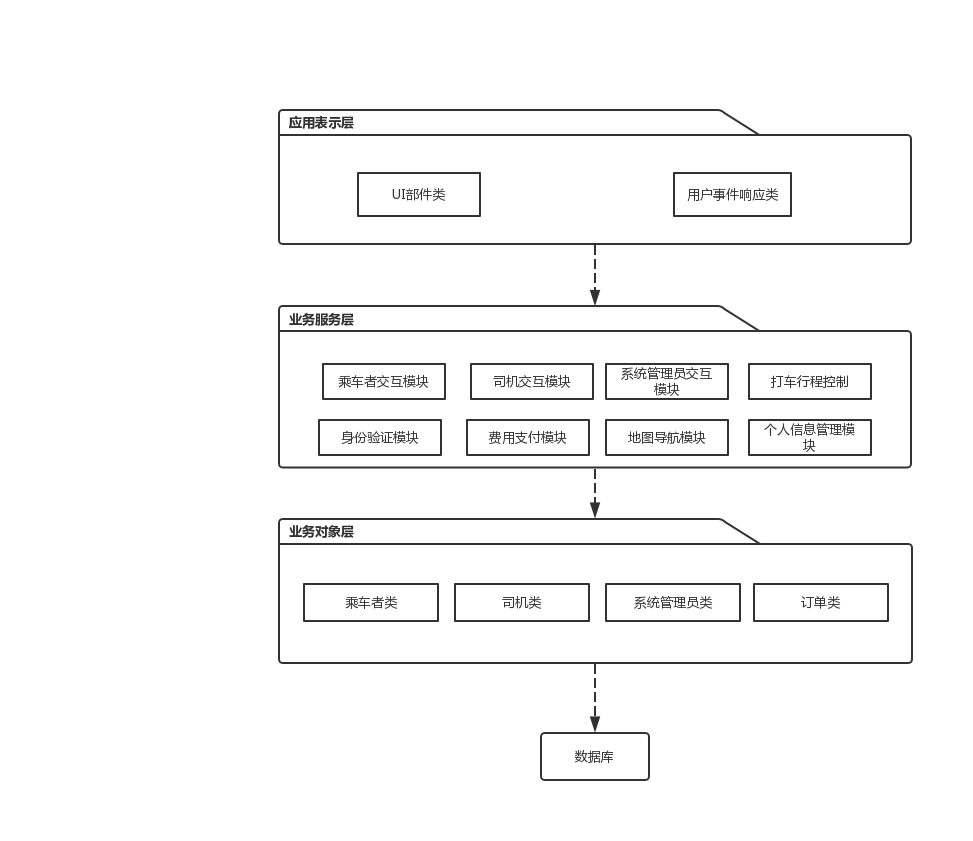


图5-8详细的逻辑视图

# 进程视图

在本部分中，对体系结构的进程视图进行描述，描述内容主要包括系统执行过程中所涉及到的进程和线程，它们之间的交互以及配置等等。并且描述了对象和类的分配。

进程模型将先前定义的类组织为可执行的进程，通过进程的执行，能够使得系统对功能的用例进行实现。

## 进程

下图给出了该系统体系结构的进程图，图中展示了系统的所有进程以及与这些进程相关联的控制器对象。

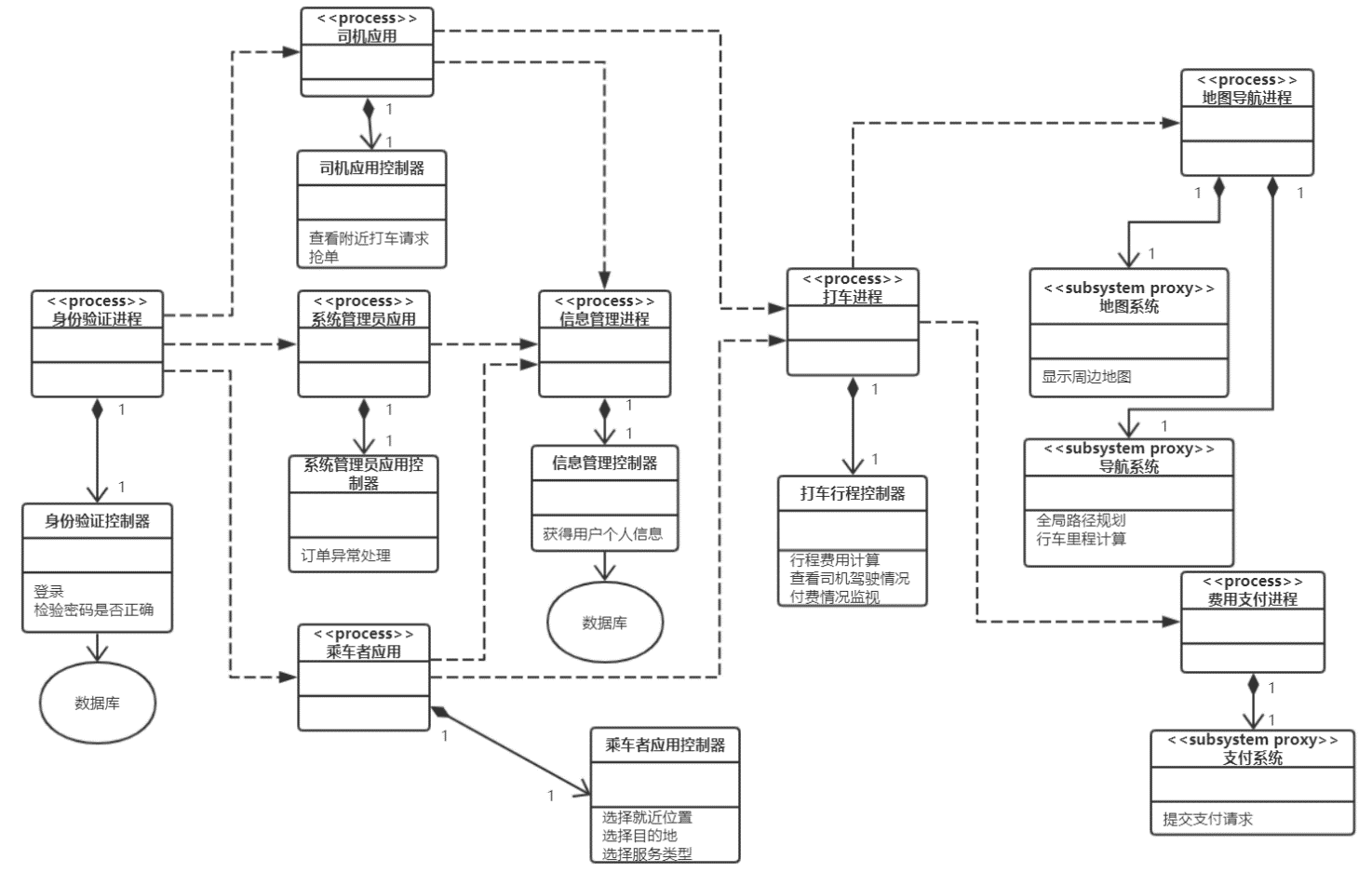


图6-1 体系结构进程图

下面详细说明图中的进程与控制器对象：

### 身份验证进程

该进程实例的任务是对每名访问该系统的用户进行身份验证，对于不同身份的用户，通过调用身份控制器进行身份验证，如果身份验证成功，则根据用户身份的不同，启动不同的应用进程，比如乘车者应用进程、司机应用进程和系统管理员应用进程。

每个进程实例都有一个对应的身份验证控制器，它们之间的关系为一对一。

### 身份验证控制器

该控制器对象对“身份验证”这一用例进行了实现。该对象通过访问实体类来间接访问数据库，并获得相应身份的用户数据库的信息。

### 乘车者应用进程

该进程的任务是对乘车者终端系统的功能进行管理，包括与乘车者进行交互的过程处理以及相关业务逻辑的过程处理。

该进程通过启动乘车应用控制器进行功能的实现，每个乘车者的应用中都会有一个该进程实例。

### 乘车者应用控制器

该控制器用来控制乘车者应用的接口，并实现乘车者应用进程中的功能，包括选择就近位置，选择目的地，选择服务类型等。每个乘车者应用进程都会启动一个唯一的控制器对象，它们之间的关系为一对一。

### 司机应用进程

该进程的任务是对司机终端系统的功能进行管理，包括与司机进行交互的过程处理以及相关业务逻辑的过程处理。

该进程通过启动司机应用控制器进行功能的实现，每个司机的应用中都会有一个该进程实例。

### 司机应用控制器

该控制器用来控制司机应用的接口，并实现司机应用进程中的功能，包括查看附近打车请求，抢单等。每个司机应用进程都会启动一个唯一的控制器对象，它们之间的关系为一对一。

### 系统管理员应用进程

该进程的任务是对系统管理员终端系统的功能进行管理，包括与系统管理员进行交互的过程处理以及相关业务逻辑的过程处理。

该进程通过启动系统管理员应用控制器进行功能的实现，每个系统管理员的应用中都会有一个该进程实例。

### 系统管理员应用控制器

该控制器用来控制系统管理员应用的接口，并实现系统管理员应用进程中的功能，即异常订单处理。每个系统管理员应用进程都会启动一个唯一的控制器对象，它们之间的关系为一对一。

### 信息管理进程

该进程对用户的个人信息进行管理，通过调用信息管理控制器完成“个人信息管理”相关的用例，比如显示个人信息、对个人信息中的某一项进行修改等。每个用户的终端系统中都会创建一个该进程，用来管理个人的基本信息。

### 信息管理控制器

该控制器对象被信息管理进程调用，具体的任务是对用户的个人信息进行管理。该对象通过对数据库进行访问，从而能够完成用户个人信息的查看，修改等操作。

由于每个用户都需要进行个人信息的管理，而且用户的数据对于其他的用户是不可见的，因此该控制器对象应当与进程一一对应。因此进程和控制器的关系为一对一。

### 打车进程

该进程的主要任务是对乘车者的一次行程进行控制和管理，它通过调用打车行程控制器来实现乘车者和司机在一次行程中需要使用的功能。乘车者和司机在一次行程中因为都需要获得该行程信息，因此打车进程被乘车者和司机的终端系统共享。该进程需要依赖于地图导航进程和费用支付进程，来共同实现系统的功能。

### 打车进程控制器

该控制器对象被打车进程调用，实现了一次行程中需要完成的功能，包括形成费用的计算，司机驾驶情况的查看，对乘车者付费情况进行监视。

每个行程都需要进行单独的控制，因此每个打车进程实例都需要唯一使用的一个控制器对象，它们之间的关系为一对一。

### 地图导航进程、地图系统、导航系统

地图导航进程的任务是为用户的终端应用提供当前位置的地图信息，并在一次行程中完成路径导航功能。该进程通过调用外部第三方的地图系统接口和导航系统接口来实现该功能。

地图系统和导航系统的作用是适配器接口，通过提供统一的接口方法定义，使得系统能够访问第三方的服务。同时第三方地图导航系统通过该接口与打车系统相连，从而实现了地图导航的功能。

地图导航进程与地图系统、导航系统的关系都是一对一。

### 费用支付进程、支付系统

费用支付进程的功能是将用户的支付请求传递给第三方支付平台，第三方平台处理支付请求之后，返回用户的支付请求处理成功的消息。该进程通过调用外部第三方的支付系统接口来实现该功能。

支付系统的作用是适配器接口，接口中定义了一系列方法，使得本系统能够通过调用这些方法来使用第三方系统，同时第三方支付系统通过该接口与打车系统相连，从而实现了用户支付的功能。

费用支付进程与支付系统的关系是一对一的。

## 进程设计元素

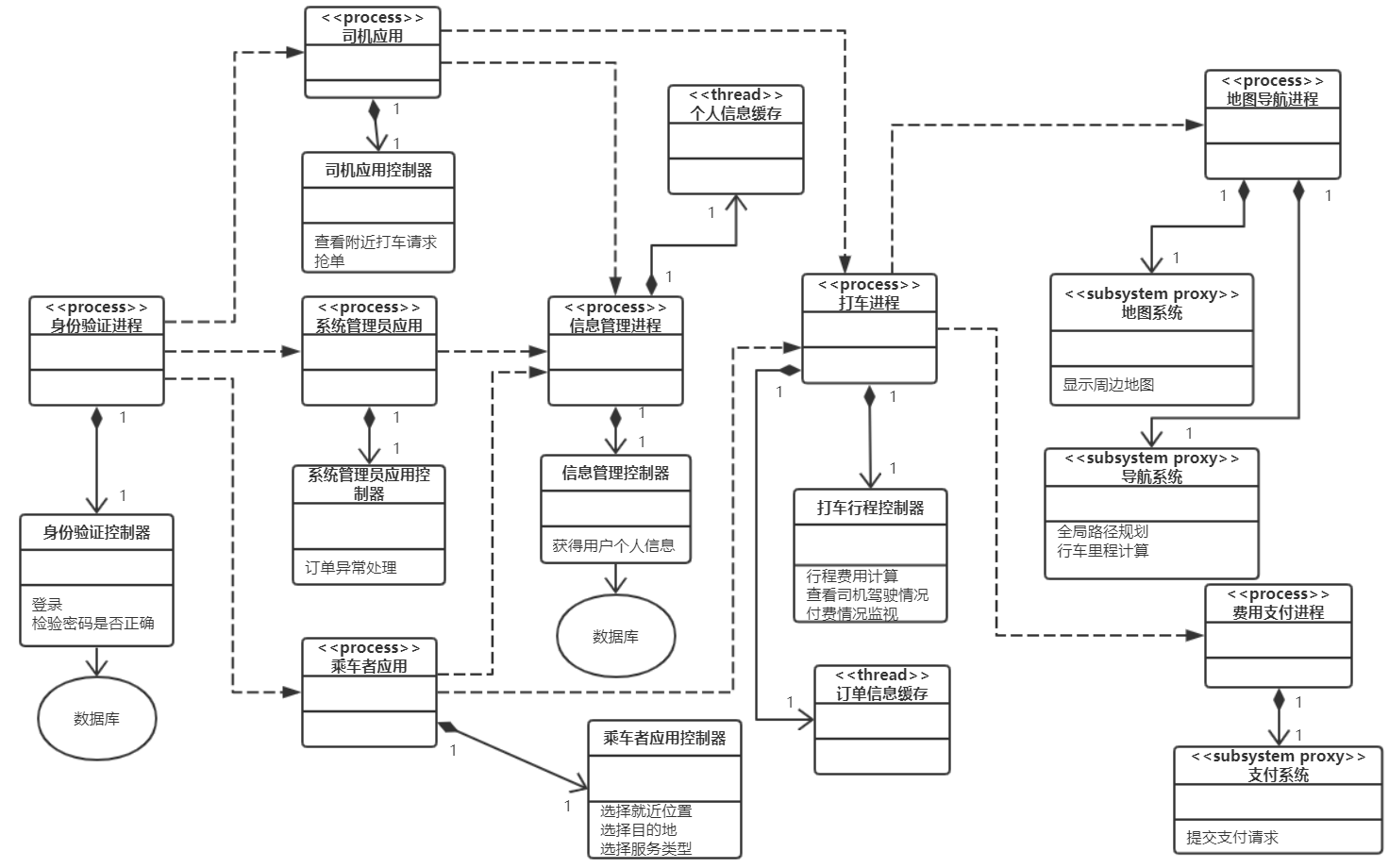
本部分中，对进程图进行设计上的细化，得到下面的进程设计元素图：

图6-2 进程设计元素图

### 个人信息缓存

个人信息缓存线程由信息管理进程创建，主要任务是异步地将数据库中的用户信息缓存下来，存储在缓存中。

### 订单信息缓存

订单信息缓存线程由订单管理进程创建，主要任务是异步地将一次行程产生的订单信息缓存下来，以方便之后将历史订单信息写入数据库。

## 进程模型依赖设计

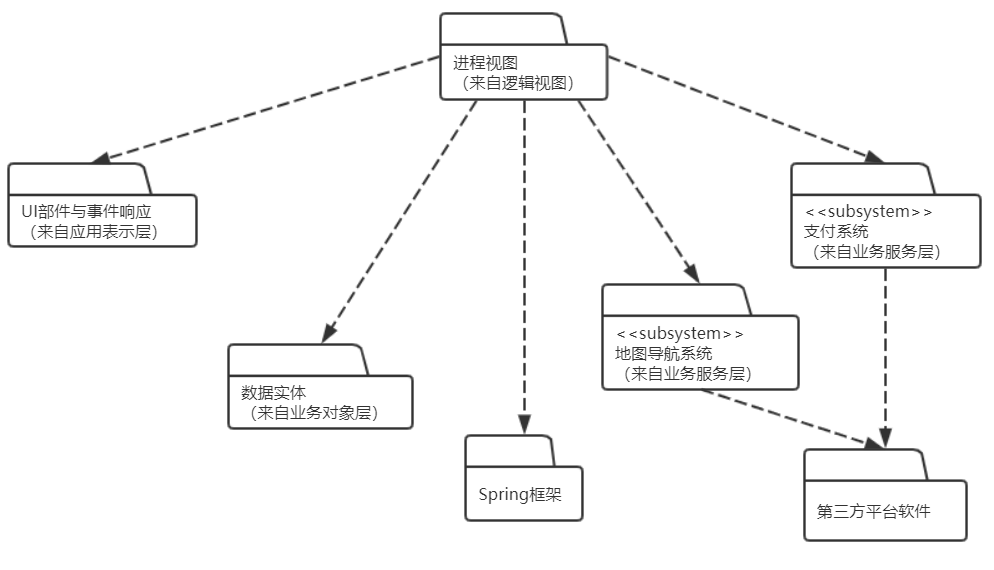


图6-3 进程模型依赖图

通过设计进程模型依赖图，能够保证进程模型元素之间与设计模型元素之间的依赖关系不会产生访问冲突。

## 进程实现

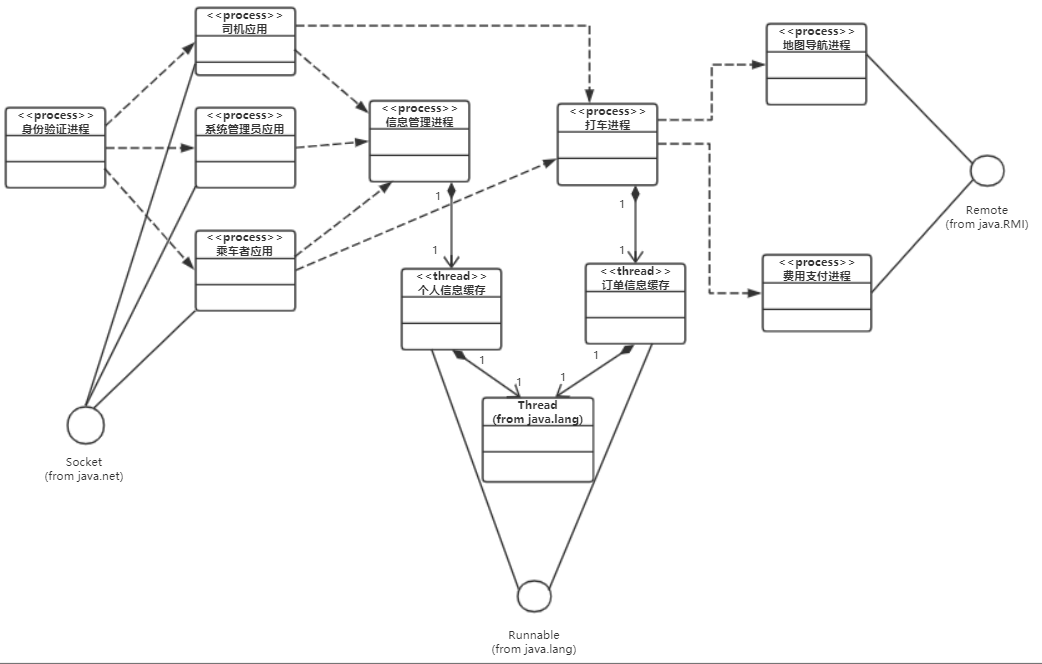


图6-4 进程实现图

在进程实现图中，给出了一部分进程在系统开发过程中的实现方法。

### Socket类

socket是基于应用服务与TCP/IP通信之间的一个抽象，他将TCP/IP协议里面复杂的通信逻辑进行封装，对用户来说，只要通过一组简单的API就可以实现网络的连接。

### Remote接口

Remote接口能够识别所有的远程对象。任何远程对象都需要间接或直接地实现该接口中的方法，只有这些接口中的方法才能够实现远程访问调用。

实现该接口的类能够实现任意数量的远程接口，并能够集成任何实现该接口的远程类。

### Runnable接口

Runnable接口能够被任意将会被一个线程执行的类所实现，该类必须定义一个无参方法run()。

该接口为对象提供了一个一般化的协议，这些对象期望在它们处于活跃状态时执行其中的代码。

### 实现细节

在用户的终端系统中，当需要与系统服务器进行网络通信时，就需要创建一个Socket对象，通过该对象与服务器建立TCP/IP连接。

在进程实现图中，个人信息缓存和订单信息缓存两个线程使用Runnable接口，或者通过继承Thread类来创建。而地图导航进程和费用支付进程通过使用Remote接口来访问位于第三方软件系统的函数，从而实现了调用第三方平台的功能。

# 部署视图

从系统软硬件物理配置的角度，描述系统的网络逻辑拓扑结构。模型包括各个物理节点的硬件与软件配置，网络的逻辑拓扑结构，节点间的交互和讯关系等。同时还表达了进程视图中的各个进程具体分配到物理节点的映射关系。由于本系统采用的是 B/S架构，结合现实的网络拓扑结构，本文设计部署视角如下图：

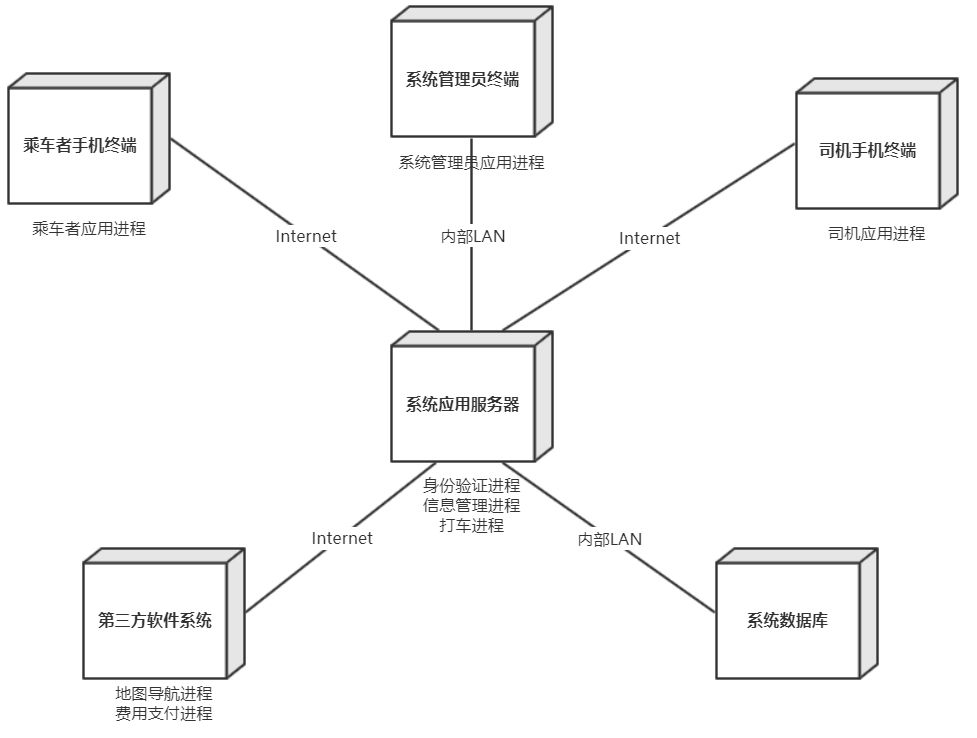


图7 体系结构部署视图

## 乘车者手机终端

乘车者手机终端通过Wifi，移动数据网络等手段连接到Internet，从而连接到系统应用服务器。该结点中完成了乘车者应用进程中所实现的功能，包括就近位置选择，选择服务类型，选择目的地等与用户直接交互的功能。同时，乘车者终端通过Internet，向系统服务器发送其他请求，并接收请求处理信息。

## 司机手机终端

司机手机终端通过Wifi，移动数据网络等手段连接到Internet，从而连接到系统应用服务器。该结点中完成了司机应用进程中所实现的功能，包括查看附近打车请求、抢单等与用户直接交互的功能。同时，司机终端通过Internet，向系统服务器发送其他请求，并接收请求处理信息。

## 系统管理员终端

系统管理员终端是内部设备，直接通过内部的LAN与系统应用服务器进行连接。该结点中完成了系统管理员应用进程中所实现的功能，即对异常订单的处理。同时，该终端也可以直接通过LAN向系统服务器发送其他请求，并接收请求处理信息。

## 系统应用服务器

系统应用服务器是部署视图中的核心结点，该结点直接与系统数据库相连接，通过Internet与其他所有的用户手机终端和第三方软件系统相连接，并通过本地内部LAN与系统管理员的终端进行连接。

该结点中部署的进程主要包括身份验证进程、打车进程和信息管理进程。系统服务器通过执行这些进程之后，将处理结果通过Internet返回给用户手机终端。对于身份验证和信息管理，系统服务器需要向系统数据库发送数据库请求，并等待数据库发送处理完成信息，从而完成这两个过程的处理。

## 系统数据库

系统数据库中存储了所有的用户数据，它直接与系统服务器连接。该结点的主要任务是处理来自系统应用服务器的数据库请求，将请求处理完之后，数据库需要向系统服务器发送处理完成信息，从而完成了整个过程。

## 第三方软件系统

第三方软件系统包括了该系统所使用的所有第三方服务系统，包括地图导航服务系统和支付系统。这些第三方软件系统通过一些专门的网络连接手段与系统应用服务器建立一个快速的传输连接。

该结点中部署的进程主要包括地图导航进程和费用支付进程，这些进程向第三方软件系统发送处理请求，这些第三方系统内部进行处理之后，只将最后的处理结果反馈给进程。然后这些进程将处理结果通过专用的网络连接发送到系统应用服务器。

# 规模与性能

在本部分中，对本系统应当达到的规模和性能要求进行定量的描述。

1. OTMS在正常情况下可以容纳20000个用户同时使用网络访问该系统。峰值情况下，该系统应能够容纳30000个用户并发访问。
2. 正常情况下，OTMS允许10000个用户并发访问第三方地图服务。
3. OTMS能够及时响应用户的查询，打车，登录请求，所有这些请求的响应时间不超过0.5秒。
4. 网络条件良好的情况下，当司机接单之后，乘客端的系统能够及时更新订单的状态，响应时间不超过0.5秒；同时，司机端系统也能够及时响应乘客端的订单状态改变，响应时间不超过0.5秒。
5. 系统出现错误之后，保证能够在2秒之内检测到错误，并制定相应的解决方案。
6. 系统被部署在用户手机终端和系统服务器中，位于用户终端的系统应当保证尽量小的内存消耗与尽量少的存储容量。

# 系统的质量要求

在本部分中，从可靠性、易用性、密安性、可维护性、可移植性和可扩展性这几个角度，对本系统应当达到的质量要求进行全面的阐述：

## 可靠性

可靠性是指产品在规定的条件下，规定的时间内，完成规定功能的能力。该系统的可靠性需求具体体现在系统能较长时间下稳定运行。同时，该系统需要具备一定的故障恢复能力，即有一定的容错能力。当用户的操作不当引起某些故障时，或者是由于操作系统或者网络发生故障时，系统需要具备一定的故障恢复能力。

1. OTMS能在较长的时间下稳定运行，同时保证每天对系统状态进行检查，如果出现异常情况，及时向用户报告；
2. OTMS应当具有较强的容错能力，能够尽量考虑到用户使用的各种情况，并制定相应的错误避免方案，从而在最大程度上减少系统发生错误的可能性。
3. 当涉及到金额的操作时，系统必须保证金额流转正常，不能出现金额流失或者出错的情况。
4. 系统应当具有完备的备份机制，当出现外界破坏时，数据可以及时通过备份恢复。

## 易用性

1. 系统应当具有简洁与人性化的GUI设计，并符合普通用户的使用习惯。
2. 系统每个操作应该简洁明了，保证普通的手机用户能够快速理解与掌握。
3. 系统应当为不同类型的用户提供完备的系统使用说明，方便用户学习使用。
4. 系统界面美观，吸引使用者。本系统的界面设计应该在保证基本功能的基础上做到布局美观友好，系统界面和市场面上大部分管理系统界面类似，用户的体验性在本系统的界面设计上因该充分提高。

## 密安性

1. 系统能够保护每一个用户的所有数据，保证不会因为外部攻击而泄露。
2. 当登录系统时，能够进行严格的身份验证，保证登录操作是本人进行。
3. 系统保证一周进行一次安全维护，对发现的安全漏洞进行排查与修补。
4. 本系统对于重要数据应该进行数据加密。初次登录系统时，应该输入相应的用户名和密码，对于密码的存储应该采取一定的加密措施，防止因为管理员不适当的信息泄露而产生不可挽回的错误。
5. 系统是能够被重建的，当系统损坏后，能够进行快速恢复。

## 可维护性

1. 进行系统开发时，要保证使用清晰的系统框架，防止架构混乱。
2. 进行系统开发时，要充分考虑业务流程中可能出现的对象，使得系统对技术和业务的变化具有足够的支持能力，避免代价过高的系统修改。
3. 进行系统开发时，需要提供完备的系统开发文档。
4. 软件维护人员应当在软件开发后期参与开发，并对系统架构，接口等进行初步理解。

## 可移植性

1. 由于该系统运行在智能手机平台上，因此该系统能够同时支持在安卓平台与IOS平台上运行。
2. 不与系统中的其他软件产生兼容性问题。
3. 本系统应该保证在市面上常用的系统环境下便于进行安装使用，不会因为环境的不同而导致系统的崩溃或者时系统功能的缺失。

## 可扩展性

1. 系统开发时，应当充分考虑到需求扩大的情况，并以此设计接口、结构，保证日后的升级与修改所消耗的时间成本最小。
2. 系统设计时应保证尽量小的耦合度。

# 质量分析和评价

在本部分中，使用ATAM（Architecture Tradeoff Analysis Method，构架权衡分析方法）方法对软件体系结构的质量进行综合全面的评价。这种方法不仅可以揭示出构架满足特定质量目标的情况，而且（因为它认识到了构架决策会影响多个质量属性）可以使我们更清楚地认识到质量目标之间的联系——即如何权衡诸多质量目标。

## 场景分析

通过场景可以很好地评估体系结构。SEI开发了通过手工评估和确认体系结构的方法，称为体系结构折中分析方法(ATAM --Architecture Tradeoff Analysis Method)。它是评价软件构架的一种综合全面的方法。这种方法不仅可以揭示出构架满足特定质量目标的情况，而且可以使我们更清楚地认识到质量目标之间的联系。本文档根据ATAM方法来对线上打车软件系统的体系架构进行分析和评估。

### 用例场景

用例场景是从使用者的角度出发，描述用户期望的与整个运行系统的交互。

|  |  |
| --- | --- |
| 场景编号 | 描述 |
| 场景1 | * 乘车者在乘车之前期望能够在系统中看到周边的地图信息，并且在找到司机之后会立即显示司机的基本信息，而不需要额外输入。 * 司机希望在查看附近打车请求时，自动显示所有的请求，并实时更新。 * 该场景代表了用户期望系统易于使用，即易用性。 |
| 场景2 | * 当系统发生故障时，缓存系统要能在1秒中从一个处理器切换到另一个处理器。 * 系统应当具有完备的备份机制，当出现外界破坏时，数据可以及时通过备份恢复。 * 该场景代表了系统可靠性需求。 |
| 场景3 | * 乘车者发送打车请求之后，附近的司机在0.5秒钟之内可以在终端中收到该乘客的打车请求。 * 用户在使用系统进行菜单切换时，该过程持续事件应当小于0.2秒。 * 网络条件良好的情况下，当司机接单之后，乘客端的系统能够及时更新订单的状态，响应时间不超过0.5秒. * 该场景代表了系统的性能要求。 |
| 场景4 | * 当发生数据异常时，系统应当立即通知所有用户，并在身份验证系统中要求用户对密码安全性进行增强。 * 当一次行程订单出现异常时，系统管理员应当在1天之内对异常进行排查，并进行解决。 * 该场景代表了用户期望的可靠性。 |

表4 用例场景

### 增长性场景

增长性场景是指预期未来系统修改时可能发生的场景，每个场景可能时相关的质量属性的衍生物。下表中对系统可能出现的增长性场景进行描述：

|  |  |
| --- | --- |
| 场景编号 | 描述 |
| 场景1 | 未来可能会增加新的数据服务器，降低远程用户的服务访问时间。 |
| 场景2 | 通过扩充现有数据库表的规模并改善数据库表的结构，降低数据库的检索时间。 |
| 场景3 | 通过向数据提供商提供更多的License，来提升系统数据库的并发度。 |

表5 增长性场景

### 探索性场景

探索性场景是推动系统封装和降低工作压力的场景。场景的目标是解释当前设计的边界条件的限制，揭露出可能隐含着的假设条件。探索性场景在某种程度上可以为未来线上打车系统的修改给出更实际的需求。

|  |  |
| --- | --- |
| 场景编号 | 描述 |
| 场景1 | 正常情况下，当一半服务器宕机时，不影响整个系统的可使用性。 |
| 场景2 | 改进系统的可使用性，使其从98%提升到99.9% |
| 场景3 | 通过对更多设备进行软件支持，用户可以使用种类更多的智能设备访问该系统。 |

表6 探索性场景

## 原型分析

在该部分中，使用效用树法建立一条效用树，用来进行原型分析。效用树（utility trees）提供自顶向下的机制，直接和有效地将系统的业务具体分解到质量的属性场景，效用树的输出是场景的优先权列表，可供评估人员在短时间内发现体系结构中存在的问题。该系统的效用树如下所示：

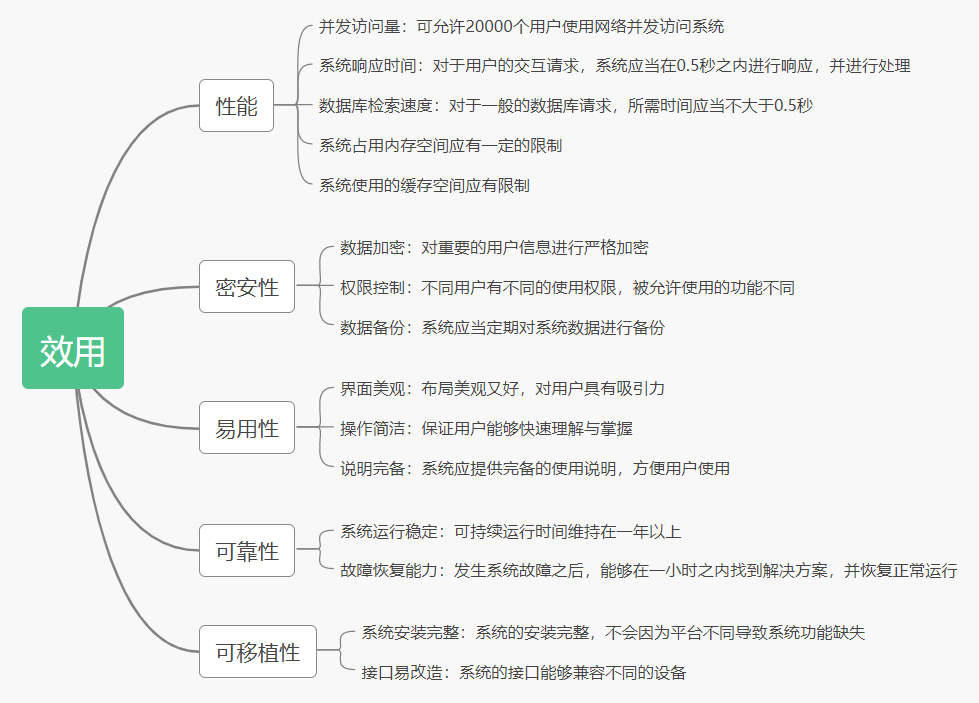


图10 OSTM系统效用树