打车软件系统

软件体系结构设计文档

<3.0>

<2019.12.12>

<杨瀚林>

<2017211961>

<2017211504 班>

软件工程导论

2019 秋季学期

修订历史记录

日期	描述	作者	备注
2019/12/05	版本 1.0	杨瀚林	初步完成了软件体系结构设计文档的撰写和校对。
2019/12/08	版本 2.0	杨瀚林	比对新格式绘制逻辑视图、进程视图及部署视图。
2019/12/12	版本 3.0	杨瀚林	将新旧两个版本文档整合，重新进行排版并校对。

文档批准记录

以下需求分析报告已经被以下机构人员批准并认可：

签字	打印姓名	标题	日期

目录

图目录：

表目录：

1. 引言

在这一部分，主要介绍了本体系结构设计文档的编写目的、范围、概念定义、用到的参考文献与涉及到的术语以及本体系结构设计文档的组织结构。

1.1 编写目的

本体系结构设计文档使用许多不同的体系结构视图来描述打车软件系统的不同方面，从而提供了打车软件系统的全面体系结构概述。它旨在捕获和传达已在系统上做出的重要体系结构决策。

本体系结构设计文档是在对打车软件系统进行了全面细致的需求分析，明确了所要开发的系统应具有的功能、性能之后编写的文档，旨在阐述打车软件系统的总体结构，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、设计遵循的标准、非功能性需求；设计软件系统的总体设计策略以及所需要用到的技术；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和评估。

本文档可供任何与打车软件系统相关的用户以及开发人员使用。本系统应用的目的是解决城市打车难，打车慢，打车贵的问题，提高城市交通智能化程度。本体系结构设计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，同时方便软件用户、软件客户、开发人员等各方进行软件项目沟通。

本文档面向的读者群体为：

1) 开发人员：根据文档了解系统预期功能及相应的架构，并据此进行系统设计与开发。
2) 项目经理：根据文档预估系统开发的大致时间并根据定义的构件结构制定开发计划。
3) 测试人员：根据体系结构文档了解系统需要测试哪些部分并设计系统总体测试框架。
4) 维护人员：根据文档确定的体系结构了解如何对软件系统进行维护并进行系统维护。
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。

1.2 文档范围

（1） 软件系统名称：打车软件系统。

（2） 文档内容：本体系结构设计文档分析了打车软件系统的功能，该软件系统设计的总体结构，以及相应的设计策略。文档概括地描述了打车软件系统的主要功能与总体应用结构，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本软件体系结构设计文档能够满足系统的质量和可靠性要求，并且分析了未来打车软件系统的维护、运行、升级改造中可能遇到的问题及相应的解决方案。

（3） 应用范围：本软件体系结构设计文档适合于打车软件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，以及打车软件系统未来的维护、运行和升级改造等要求。打车软件系统通常是城市公共交通系统的一部分，负责为乘客提供最近的具有可靠运营资质的车辆，对于城市公共交通的正常运行起着重要的作用。系统对于不同的使用者提供不同的功能

- 司机：提供接收或拒绝订单、导航及路线规划、联系乘客、投诉乘客、资质审查等功能。
- 乘客：提供定位当前位置、修改上车地点、修改下车地点、选择乘车类型、预约乘车信息、查看前行路线、查看司机信息、查看车费细则、联系司机、评价服务、代叫服务、客服投诉、一键报警等功能。
- 系统监测人员：提供资质审查、行程监督、系统监测等功能。
- 系统管理人员：提供投诉处理、登陆异常、账号处理、功能修改等功能。

（4）系统适用于以下情况：

- 乘客需要在两分钟内叫到通过资质审核的车。
- 司机可以接到当前最容易到达位置的订单。
- 乘客可以通过第三方支付系统在 10 秒内完成支付。
- 系统保证密安性，不会泄露用户个人信息。
- 系统对用户友好，方便各年龄层用户使用。

（5）系统不适用于以下情况：

- 无法进行定位或无法接入网络的地区。
- 乘客没有第三方支付系统或者没有开通第三方支付功能。
- 司机没有通过系统监测人员的资格审查。

1.3 术语定义

名称	定义
用户角色	用户角色是指按照一定参考体系划分的用户类型，是能够代表某种用户特征、便于统一描述的众多用户个体的集合。
用户需求	关于系统服务和约束的自然语言加上方块图表述。为客户撰写。
系统需求	一个结构化的文档写出系统的服务。作为客户和承包商之间的合同内容。
功能需求	系统需要提供的服务的表述，系统应该如何响应特定输入，系统在特定的情形下应该如何动作。功能需求是对未来软件的服务功能进行的描述。功能需求更关注系统的输入、加工（业务流程）直到输出的情况。
非功能需求	非功能需求是指功能以外的需求描述，主要是系统的一些关键的特性，例如可靠性、响应时间、存储空间、I/O 设备的吞吐量、接口的数据格式等。系统提供的服务或功能上的约束，例如时间约束、开发过程约束、标准等。
用例图	用例图是指由参与者（Actor）、用例（Use Case）以及它们之间的关系 构成的用于描述系统功能的视图。用例图（User Case）是被称为参与者的外部用户所能观 察到的系统功能的模型图，呈现了一些参与者和一些用例，以及它们之间的关系，主要用 于对系统、子系统或类的功能行为进行建模。
打车软件系统	本文档负责的软件系统。
司机	使用该系统的驾驶员用户，拥有合法驾驶资质、合法车辆和合法运营资质。
乘客	使用该系统的普通用户，拥有可以定位且连接进入网络的安装了该系统的移动终端设备，且可以授权第三方支付系统完成车费支付。
系统监测人员	对司机进行资质审查且对行程进行监督的人员
系统管理人员	负责处理投诉和账户异常的系统管理员。
第三方支付系统	可以在乘客授权下完成车费支付的系统。

表 1 术语定义表

缩写	全称	名词解释
USPMS	Urban Shared-Parking Management System	城市共享停车管理系统
E-R 图	Entity Relationship Diagram	实体-联系图 ，提供了表示实体类型、属性和联系的方法，用来描述现实世界

		的概念模型。
UML	Unified Modeling Language	统一建模语言或标准建模语言，为软件开发的所有阶段提供模型化和可视化支持，由需求分析到规格，再到构造和配置。
SDL	Specification and Description Language	规格和描述性语言，描述一个现实世界中特定事件的发生过程的时序关系以及步骤。
DFD	Data Flow Diagrams	数据流图，数据流图从功能的角度对系统建模，追踪数据的处理有助于全面地理解系统，数据流图也可用于描述系统和外部系统之间的数据交换。
B/S 结构	Browser/Server	浏览器/服务器结构，Browser 指的是 Web 浏览器，极少数事务逻辑在前端实现，但主要事务逻辑在服务器端实现，Browser 客户端，WebApp 服务器端和 DB 端构成所谓的三层架构。B/S 架构的系统无须特别安装，只有 Web 浏览器即可。
C/S 结构	Client/Server	客户端服务器端架构，其客户端包含一个或多个在用户的电脑上运行的程序，而服务器端有两种，一种是数据库服务器端，客户端通过数据库连接访问服务器端的数据；另一种是 Socket 服务器端，服务器端的程序通过 Socket 与客户端的程序通信。
MVC	Model–view–controller	模型-视图-控制器，是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：模型、视图和控制器。目的是实现一种动态的程序设计，使后续对程序的修改和扩展简化，并且使程序某一部分的重复利用成为可能。
ATAM	Architecture Tradeoff Analysis Method	构架权衡分析方法，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（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。

表 2 缩略字及缩写表

1.4 参考文献

[1] 王安生. 软件工程化[M]. 北京：清华大学出版社,2014.

1.5 文档组织结构

在后续的章节中，本文档将以总—分的结构对打车软件系统的体系结构设计以及相关事项进行描述。

本文档以一系列视图的形式介绍了体系结构。用例视图，逻辑视图，流程视图和部署视图。 本文档中没有单独的实现视图。这些是使用 Rational Rose 开发的基础统一建模语言（UML）模型的视图。

本文档共分为五个章节，至此第一章节引言部分结束，介绍了本体系结构设计文档的编写目的、范围、用到的参考文献与涉及到的术语以及本体系结构设计文档的组织结构。第二章为项目相关信息，对于本文档面对的项目功能，用户角色等进行了简单的概述。第三章节为体系结构需求，描述了系统体系结构的相关信息。第四章为解决方案，提供了视图以及解决的问题。第五章为系统的质量分析和评价，提供了场景应用和原型分析，以及对系统潜在风险的分析。

2. 项目相关信息

在这一部分对打车软件系统的相关信息进行概括性的描述，包括项目概述、用户角色、项目功能。

2.1 项目概述

打车软件系统是为了方便乘客打车和司机接单而开发的基于移动终端的软件系统。通过打车软件系统，用户可以随时发送自己的所在地和目的地来快速得到附近的的士。本质上讲，打车软件系统是乘客感官的延展，使乘客能够实现更大范围的“招手”功能，而不局限于某个十字路口，从而优化资源配置，缩短叫车时间。

该系统主要是由系统管理人员进行管理，由系统监测人员进行实时监测，与乘客、司机、系统管理人员、系统监测人员实现即时交互。系统授予乘客、司机、系统管理人员、

系统监测人员四种用户角色不同的权限来为他们提供不同的功能服务，满足他们的不同需求。本系统旨在实现一个较为完备的、易于使用的且用户体验良好的打车软件系统。

2.2 用户角色

在城市共享停车管理系统中有四类用户：乘客、司机、系统管理人员和系统监测人员。这四类用户角色有不同的系统使用权限，可以对打车软件系统进行不同的操作。不同用户角色对系统使用的需求不同，具体如下：

（一）乘客交互模块：

乘客交互模块是针对乘客用户设计的。主要设计思想如下：

(1) 系统要求乘客用户在进入系统之前输入用户名和密码进行信息的验证。

(2) 系统要求当密码校验失败时，会提示错误，如果超过系统允许的最大连续出错值，那么会对该用户进行锁定。

(3) 系统应该能够实现让乘客用户可以根据自己的喜好来修改自己的用户名称和密码的功能。

(4) 系统应该能够实现让用户可以查看附近车辆信息。

(5) 乘客在叫车时确定乘车信息，包括乘车类型、乘车时间、乘车地点和 目的地，需要查看位置信息。

(6) 乘客在叫车成功后可以查看司机信息，包括司机的姓名、车 辆、评分、位置、接单数，需要查看位置信息。

(7) 在行程中，乘客可以查看行程信息，包括 当前位置、行程路线、当前用时等，需要查看位置信息和订单信息。

(8) 行程结束后乘客可以支付车费，乘客可以查看费用明细，为司机打分，这里需要查看订单信息。

(9) 当乘客发起投诉时，生成投诉信息，由系统管理人员处理与反馈投诉，乘客可以查看历史投诉信息。

（二）司机交互模块：

司机交互模块是针对司机用户设计的。主要设计思想如下：

(1) 系统要求司机用户在进入系统之前输入用户名和密码进行信息的验证。

(2) 系统要求当密码校验失败时，会提示错误，如果超过系统允许的最大连续出错值，那么会对该用户进行锁定。

(3) 系统应该能够实现让用户可以查看附近打车信息。

(4) 司机可以在规定时间内接受或拒绝订单，当司机超过规定时间则强制 接单。

(5) 接单后司机可以与乘客通过匿名电话和打车软件系统内消息功能联系。

(6) 行程开始后 司机可以查看导航和最优路线规划，需要查看位置信息和订单信息。

(7) 当司机发起投诉时， 生成投诉信息，由系统管理人员处理与反馈投诉，司机可以查看历史投诉信息。

(8) 司机可以 查看自己的资质审查详情和下次资质审查时间，由系统监测人员管理。

（三）系统管理人员交互模块：

系统管理人员交互模块是针对系统管理人员设计的。主要设计思想如下：

(1) 系统要求系统管理人员在进入系统之前输入用户名和密码进行信息的验证。

(2) 系统要求当密码校验失败时，会提示错误，如果超过系统允许的最大连续出错值，那么会对该用户进行锁定。

(3) 系统管理人员负责账号的管理，包括登陆异常处理、账号和 密码的找回、账号等级的判定和维护，需要查看账户信息。

(4) 系统管理人员还负责订单管 理，包括订单的存储、备份和错误恢复等，需要查看订单信息。

(5) 系统管理人员还负责投诉 处理，包括投诉的判定和反馈，需要查看投诉信息，订单信息。

（三）系统监测人员交互模块：

系统监测人员交互模块是针对系统监测人员设计的。主要设计思想如下：

(1) 系统要求系统监测人员在进入系统之前输入用户名和密码进行信息的验证。

(2) 系统要求当密码校验失败时，会提示错误，如果超过系统允许的最大连续出错值，那么会对该用户进行锁定。

(3) 系统监测人员负责资质审查，对到审查年限的司机和车辆进 行资质审查，需要查看司机信息。

(4) 。系统检测人员还负责异常检测，实时观测偏离规划路线 较大的行程，并与司机和乘客联系确保安全，需要查看订单信息、位置信息、乘客信息、 司机信息。

(5) 系统检测人员还负责实时监测打车软件系统的运行情况。

2.3 项目功能

通过打车软件系统，用户可以随时发送自己的所在地和目的地来快速得到附近的的士。本质上讲，打车软件系统是乘客感官的延展，使乘客能够实现更大范围的“招手”功能，而不局限于某个十字路口，从而优化资源配置，缩短叫车时间。

（一）乘客

●定位当前位置：打车软件可以定位乘客当前位置并在乘客叫车时根据位置为司机派单。

●修改上车地点：乘客可以不使用当前位置叫车而手动输入上车地点。

●修改下车地点：乘客可以在上车后修改下车地点。

●选择乘车类型：乘客可以选择乘坐出租车、快车、专车等不同车型。

●预约乘车信息：乘客可以提前预约乘车类型、上车地点和乘车时间。

●查看前行路线：乘客可以查看当前车辆的行进的路线。

●查看司机信息：乘客可以查看司机的评分、接单数和资质审查情况。

●查看车费细则：乘客可以在行程结束付费后查看费用详情。

●联系司机：乘客与司机可以通过电话（匿名）或打车软件系统内发送消息相互联系。

●评价服务：乘客可以对司机的服务做出评价或打赏。

●代叫服务：用户可以帮助其他用户叫车。

●客服投诉：乘客对司机或者车费有意见的时候可以联系客服投诉。

●一键报警：乘客可以在遇到危险或紧急情况时一键报警并联系紧急联系人。

（二）司机

●接收或拒绝订单：司机可以在规定时间内接受或拒绝打车软件系统为其分配的订单。

●导航及路线规划：打车软件系统为司机提供导航服务并规划至目的地的最优路线。

●联系乘客：司机和乘客可以通过电话（匿名）或打车软件系统内发送消息相互联系。

●投诉乘客：司机可以投诉对司机或车辆有损害行为的乘客。

●资质审查：司机查看当前审查情况和下一次审查时间。

（三）系统监测人员

●资质审查：系统监测人员可以对司机和车辆进行资质审查。

●行程监督：系统监测人员可以监督当前是否存在异常行程。

●系统监测：系统检测人员可以实时了解并维护打车软件系统的运行情况。

（四）系统管理人员

●投诉处理：可以解决处理乘客与司机之间的争端。

●登陆异常：可以帮助用户找回密码或者检查异常登陆。

●账号处理：注销异常账号并维护用户的账号等级。

●功能修改：可以上线或下线部分功能。

打车软件系统的功能结构图见图 1。

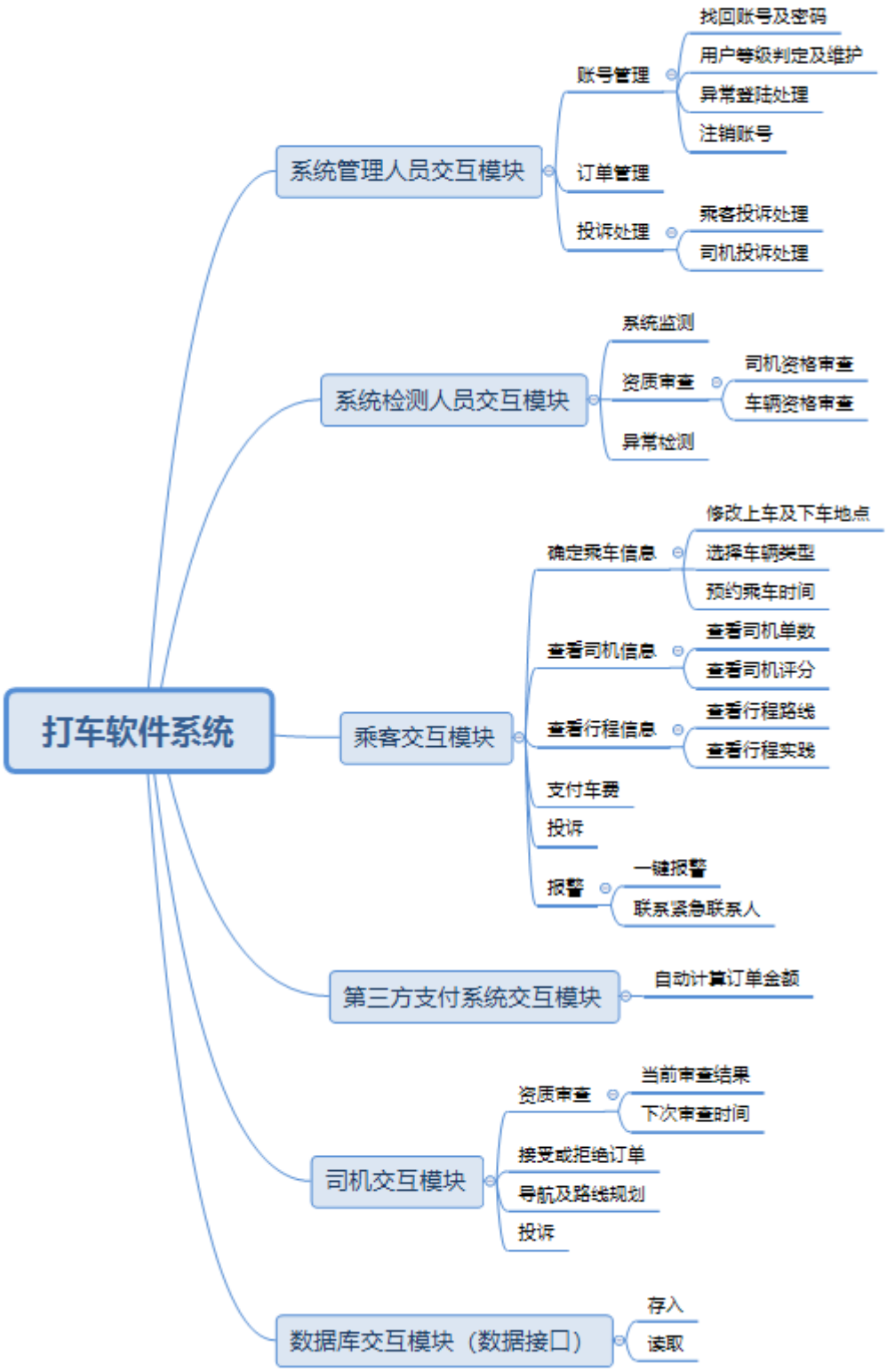

图 1 打车软件系统功能结构图

3. 体系结构需求

软件体系结构是具有一定形式的结构化元素，即构件的集合，包括处理构件、数据构件和连接构件。处理构件负责对数据进行加工，数据构件是被加工的信息，连接构件把体系结构的不同部分组组合连接起来。这一定义注重区分处理构件、数据构件和连接构件，这一方法在其他的定义和方法中基本上得到保持。由于软件系统具有的一些共通特性，这种模型可以在多个系统之间传递，特别是可以应用到具有相似质量属性和功能需求的系统中，并能够促进大规模软件的系统级复用。

3.1 关键指标

本系统主要面向乘客、司机、系统管理人员和系统监测人员，使用本系统，乘客可以很方便地查看附近的车辆信息，司机可以快速的得到打车信息，双方的信息能够进行实时交互，系统的实用性较高。考虑到此软件系统的应用场景为交通，软件系统必须是可靠的。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。软件系统必须定期备份数据以防止。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。选择数据库产品时，要考虑一定的数据负载能力。由于在处理员工信息、客户信息、账户信息等信息时，系统需要做大量的数据统计和处理，因此要具备相应的数据负载能力。 此外软件系统本身也需要提供一定的容错机制，比如必要的提示和确认机制以防止用户误操作。所以本系统在性能上还需要具备以下关键指标：

(1) 系统将信息更新至数据库时，对数据的处理量在 G 级别时，数据库单表操作时间不大于 0.5 秒，系统的响应时间应该在 5 秒之内。
(2) 在系统因并发访问人数过多而发生宕机时，系统应该在 6 分钟内能进行系统恢复。
(3) 当用户登陆时，系统必须在 0.5 秒内给出登陆成功或失败的提示。
(4) 乘客应当可以查看个人信息和司机信息，司机可以查看个人信息和资质审查详情，上述查询流程的响应速度最佳为 0.1 秒，平均为 0.5 秒，不超过 1 秒。系统应当可以接受 3 万人同时操作。

(5) 乘客和司机应可以查看历史订单详情和历史投诉详情，上述查询流程的响应速度最佳为 0.1 秒，平均为 0.5 秒，不超过 1 秒。系统应当可以接受 3 万人同时操作。

(6) 乘客和司机应可以修改个人信息，上述修改的响应速度最佳为 0.1 秒，平均为 0.5 秒，不超过 1 秒，且必须返回修改是否成功。系统应当可以接受 3 万人同时操作。

(7) 乘客支付车费的响应速度最佳为 0.1 秒，平均为 0.5 秒，不超过 1 秒，且必须返回支付是否成功。系统应当可以接受 50 万人同时操作。

(8) 乘客和司机查看行程信息时，定位更新响应速度最佳为 0.05 秒，平均为 0.1 秒，不超过 0.2 秒，系统需要实时刷新行程相关信息。系统应当可以接受 50 万人同时操作。

(9) 乘客一键报警响应速度最佳为 0.05 秒，平均为 0.1 秒，不超过 0.2 秒，系统及相关人员需要迅速响应。系统应当可以接受 1 万人同时操作。

(10) 系统月活用户数量为 1.5 亿，同时在线用户数的期望值为 100 万，但每一时刻打车的用户数目会随时间变化（比如工作日早晚高峰要多余非高峰时段和非工作日），因此其同时在线人数的最大值为 300 万 人，因此系统期望正常运行在 100 万人的并发访问数上，最多支持 300 万人的并发访问数。

3.2 体系结构用例

用例图是由参与者，功能用例以及它们之间的关系构成的图，其目的是描述系统功能，通过用例图呈现参与者和他们之间的关系构成的图，就可以更清晰地了解用户对系统、子系统或各项功能的使用和行为。本打车管理系统的用户角色包括乘客、司机、系统管理人员和系统监测人员。不同的用户对应着不同的功能，针对这些用户角色，可以画出如下用例图：

3.2.1 乘客用例

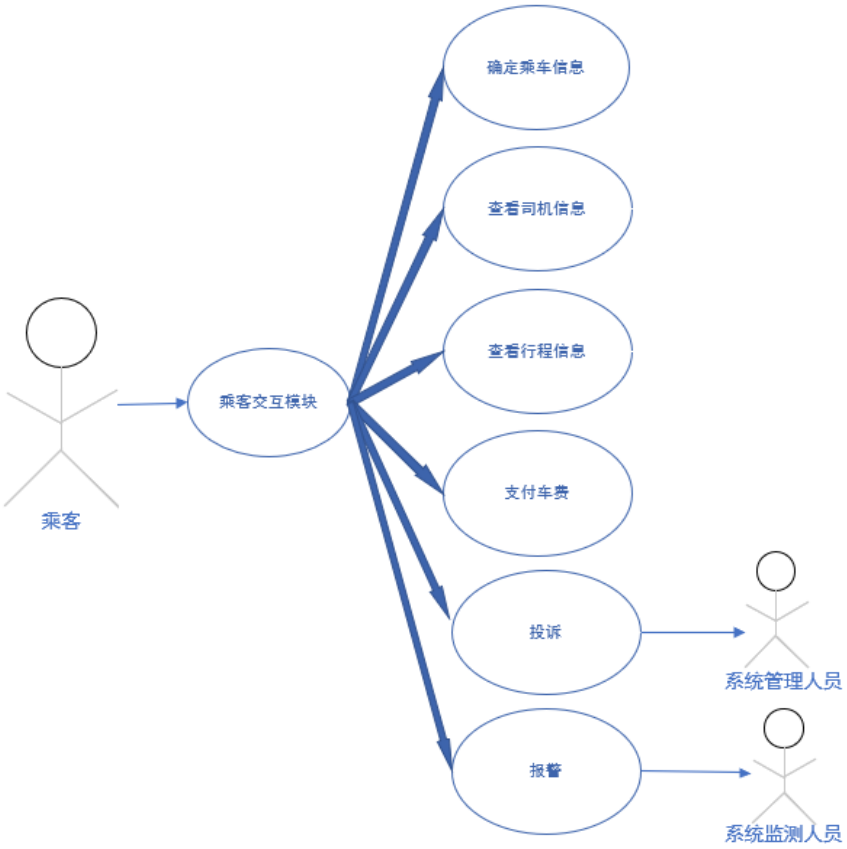

图 2 乘客用例图

3.2.1.1 介绍

主要处理乘客叫车功能的各个流程，包括确定乘车信息、查看司机信息、查看行程信息、支付车费、投诉、报警等。

3.2.1.2 输入

输入司机信息、订单信息、投诉信息、位置信息。

3.2.1.3 处理流程

1、乘客输入乘车类型、乘车时间、上车地点和目的地确定乘车信息。

2、接单成功后，订单形成。

3、乘客查看司机信息并与司机通过匿名电话或打车软件系统消息联系。

4、行程开始后乘客可以查看行进路线，当前用时等行程信息。

5、行程结束后乘客通过第三方支付系统支付车费并查看车费详情和评价司机。

6、当乘客对司机不满时可以发起投诉，由系统管理人员处理，可以查看历史投诉。

7、当乘客在行程中遇到危险可以一键报警并自动联系紧急联系人。

3.2.1.4 输出

上述流程中产生订单信息、和投诉信息。

3.2.1.5 错误处理

1、当用户无法在系统位置信息中找到上车地点或者目的地时可以将当前位置和地点名称提交并创建新地点。

2、当用户对车费有疑义时可以发起投诉。

3.2.2 司机用例

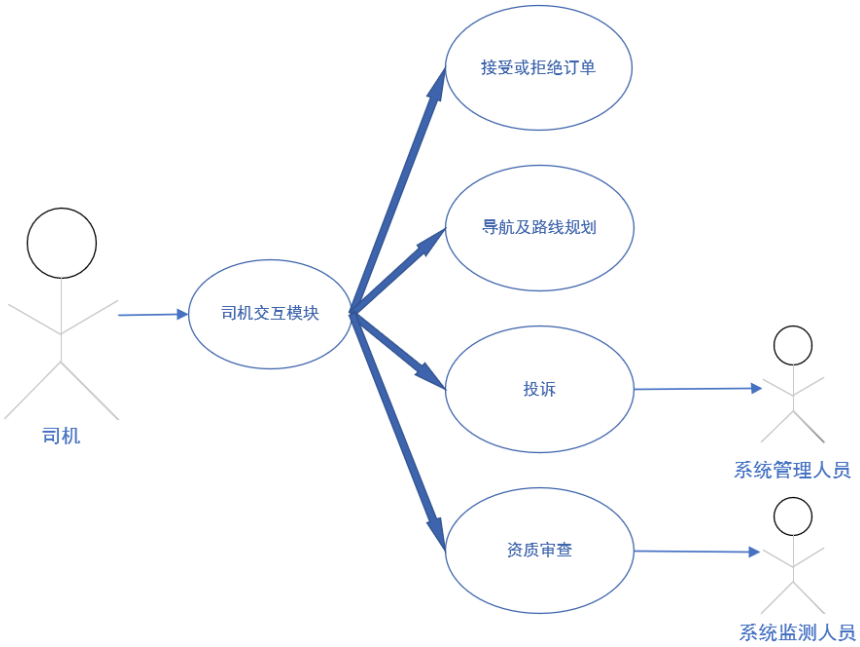

图 3 司机用例图

3.2.2.1 介绍

主要处理司机接单功能的各个流程，包括接受或拒绝订单、导航及路线规划、投诉、资质审查等。

3.2.2.2 输入

输入位置信息、订单信息、投诉信息。

3.2.2.3 处理流程

1、司机可以在规定时间内接受或拒绝订单，超过时间强制接单。

2、接单成功后司机得到乘客上车地点。

3、司机可以通过匿名电话或打车软件系统内部消息联系乘客。

4、司机运送乘客过程中可以得到导航及最优路线规划。

5、订单结束后司机可以对乘客发起投诉。

6、司机和车辆需要定期进行资质审查。

3.2.2.4 输出

输出订单信息，投诉信息。

3.2.2.5 错误处理

1、当乘客未在规定时间内到达上车地点则自动取消订单并降低用户等级。

2、乘客未经司机同意修改行程重点，司机可以在系统内予以否决。

3、当乘客针对司机的投诉成立后司机要缴纳罚金。

3.2.3 系统监测人员用例

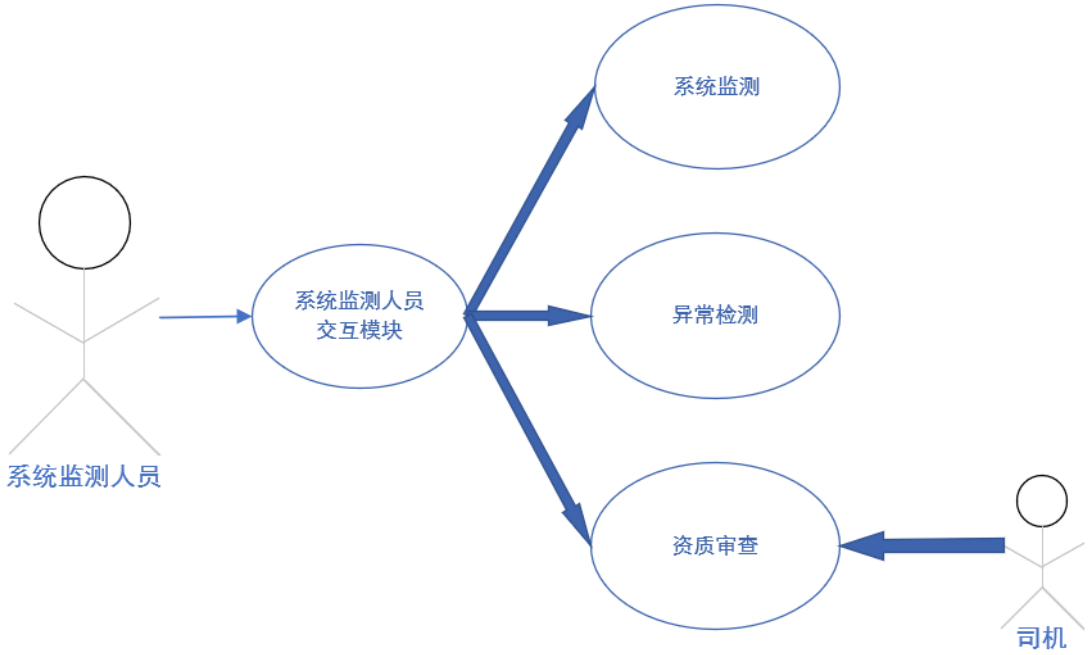

图 4 系统监测人员用例图

3.2.3.1 介绍

主要包括系统监测的各项功能，包括资质审查、异常检测、系统监测。

3.2.3.2 输入

输入司机信息、位置信息、订单信息。

3.2.3.3 处理流程

资质审查：寻找到达审查年限的司机和车辆并取得联系进行审查。

系统监测：对打车软件系统的运行进行实时监测，出现异常时及时联系维护人员。

异常检测：

1、监测行进路线严重偏离规划路线的订单。

2、与司机及乘客取得联系。

3、如果未能取得联系调取乘客和司机的个人信息。

4、联系警方提交相关信息。

3.2.3.4 输出

输出司机信息。

3.2.3.5 错误处理

1、未按规定时间完成资质审查的车辆将停止其账户接单功能。

3.2.4 系统管理员用例

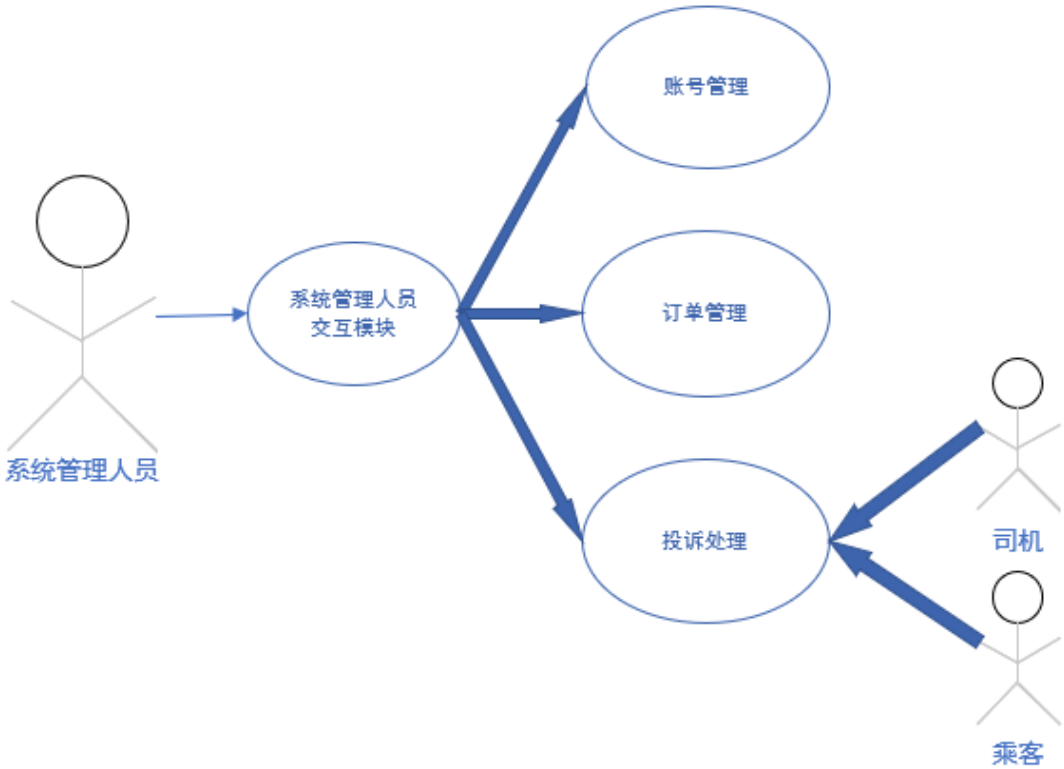

图 5 系统管理人员用例图

3.2.4.1 介绍

主要处理系统管理的相关功能，包括账号管理、订单管理、投诉处理。

3.2.4.2 输入

输入订单信息、账号信息、投诉信息。

3.2.4.3 处理流程

订单管理：对订单信息进行储存、更新、备份并支持错误恢复。

账号管理：

1、帮助用户找回账户或密码。

2、当用户的账号存在异常登陆情况时查明原因。

3、当账号存在违规或异常情况时注销账号。

4、根据用户的订单信息判定并维护用户等级。

投诉处理：

1、接受来自乘客或司机的投诉。

2、调取投诉涉及的订单信息。

3、结合投诉内容对投诉进行判定。

4、将判定及处理结果反馈给用户。

5、如果被投诉方存在过错，将投诉内容、判定结果及处理办法通知被投诉方。

6、及时更新投诉的处理情况。

3.2.4.4 输出

输出投诉信息、账号信息。

3.3 进程视图

对体系结构的过程视图的描述。描述系统执行中涉及的任务（进程和线程），它们的交互作用和配置。还描述了对象和类对任务的分配。

3.3.1 进程

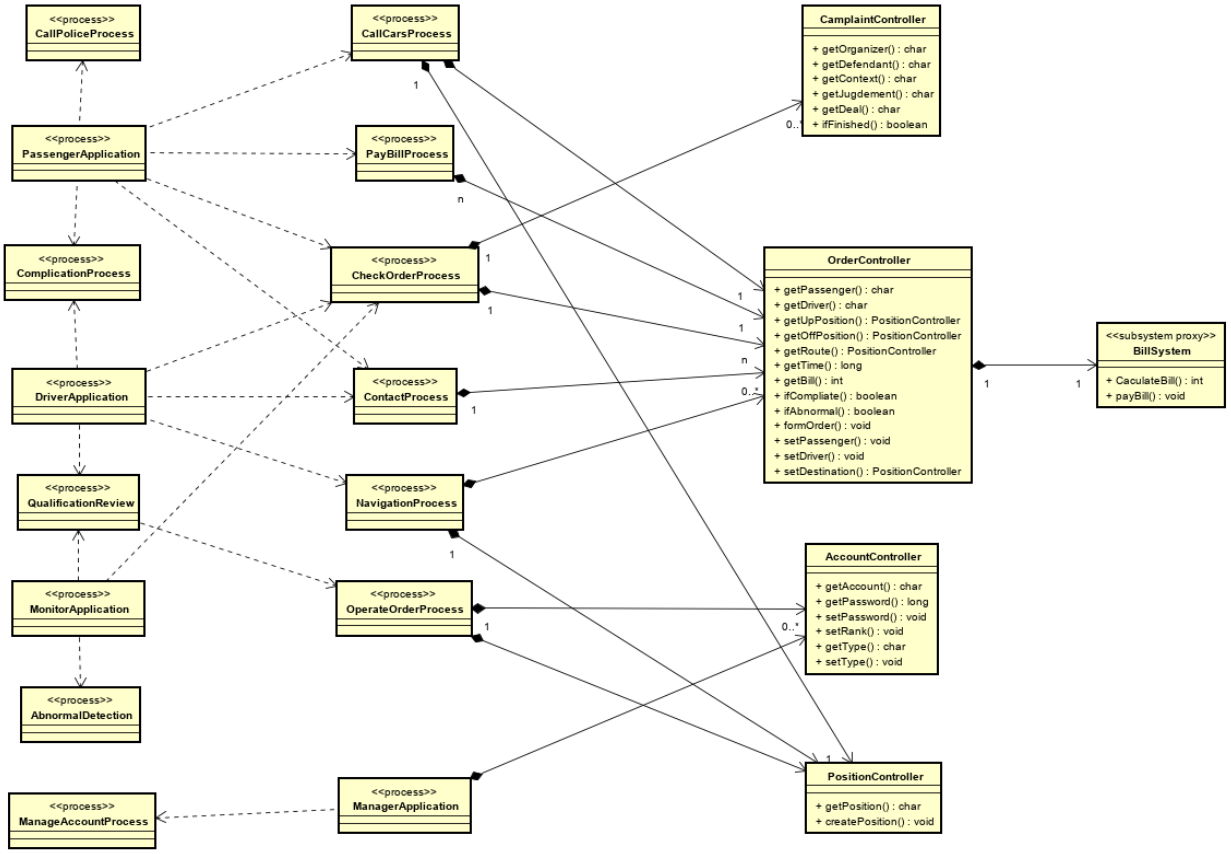

图 6 进程图

(1) 乘客交互模块： 乘客在叫车时确定乘车信息，包括乘车类型、乘车时间、乘车地点和目的地，需要查看位置信息。乘客在叫车成功后可以查看司机信息，包括司机的姓名、车辆、评分、位置、接单数，需要查看位置信息。在行程中，乘客可以查看行程信息，包括当前位置、行程路线、当前用时等，需要查看位置信息和订单信息。行程结束后乘客可以支付车费，乘客可以查看费用明细，为司机打分，这里需要查看订单信息。当乘客发起投诉时，生成投诉信息，由系统管理人员处理与反馈投诉，乘客可以查看历史投诉信息。

(2) 司机交互模块： 司机可以在规定时间内接受或拒绝订单，当司机超过规定时间则强制接单。接单后司机可以与乘客通过匿名电话和打车软件系统内消息功能联系。行程开始后司机可以查看导航和最优路线规划，需要查看位置信息和订单信息。当司机发起投诉时，生成投诉信息，由系统管理人员处理与反馈投诉，司机可以查看历史投诉

信息。司机可以查看自己的资质审查详情和下次资质审查时间，由系统监测人员管理。

(3) 系统监测人员交互模块：系统监测人员负责资质审查，对到审查年限的司机和车辆进行资质审查，需要查看司机信息。系统检测人员还负责异常检测，实时观测偏离规划路线较大的行程，并与司机和乘客联系确保安全，需要查看订单信息、位置信息、乘客信息、司机信息。系统检测人员还负责实时监测打车软件系统的运行情况。

(4) 系统管理人员交互模块：系统管理人员负责账号的管理，包括登陆异常处理、账号和密码的找回、账号等级的判定和维护，需要查看账户信息。系统管理人员还负责订单管理，包括订单的存储、备份和错误恢复等，需要查看订单信息。系统管理人员还负责投诉处理，包括投诉的判定和反馈，需要查看投诉信息，订单信息。

(5) 第三方支付系统交互模块：第三方支付系统可以根据用户订单详情自动计算车费金额，需要查看订单信息。

(6) 数据接口： 数据接口负责数据库与乘客信息、司机信息、账号信息、订单信息、投诉信息、位置信息的交互。数据库负责数据的更新、存储、备份、加密和管理，各类信息可以从数据库中提取和查看。

3.3.2 设计元素的过程

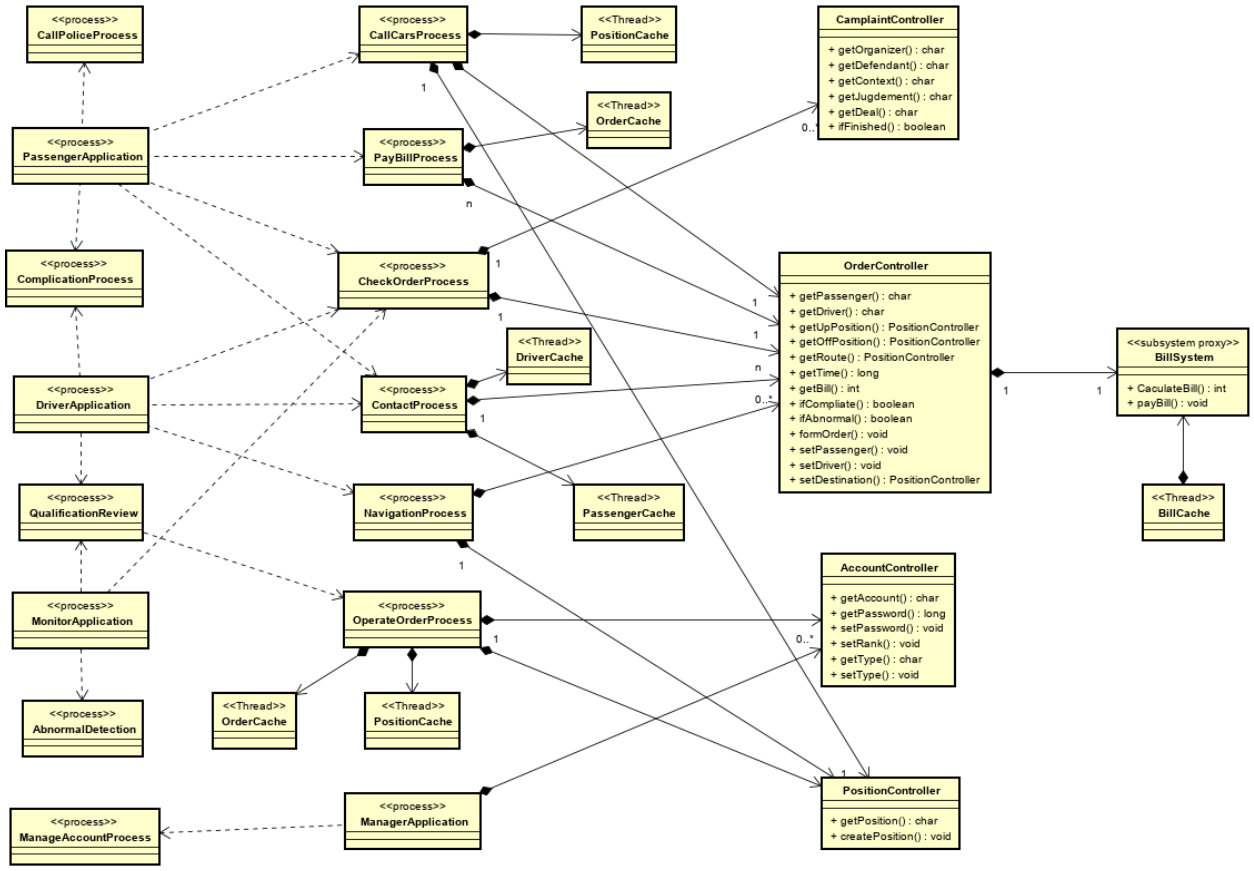

图 7 设计元素进程图

(1) OrderCache：Order Cache 用来同步当前订单的信息，在 PayBillProcess 中根据同步的订单的行程和时间计算价格，在 OperateOrderProcess 中为司机提供乘车人，出发地，目的地的信息以便司机判断是否接单。

(2) PositionCache：Position Cache 用来同步实时位置信息，在 PayBillProcess 提供准确的出发点和下车点，用于计算价格；在 OperateOrderProcess 中用于判断乘客和司机位置来决定是否给司机派单。

(3) PassengerCache：Passenger Cache 用来给司机提供当前乘客的信息，用来接客和沟通。

(4) DriverCache：Driver Cache 用来给乘客司机的具体信息，方便联系和保证行程安全。

3.3.3 过程模型到设计模型的依赖

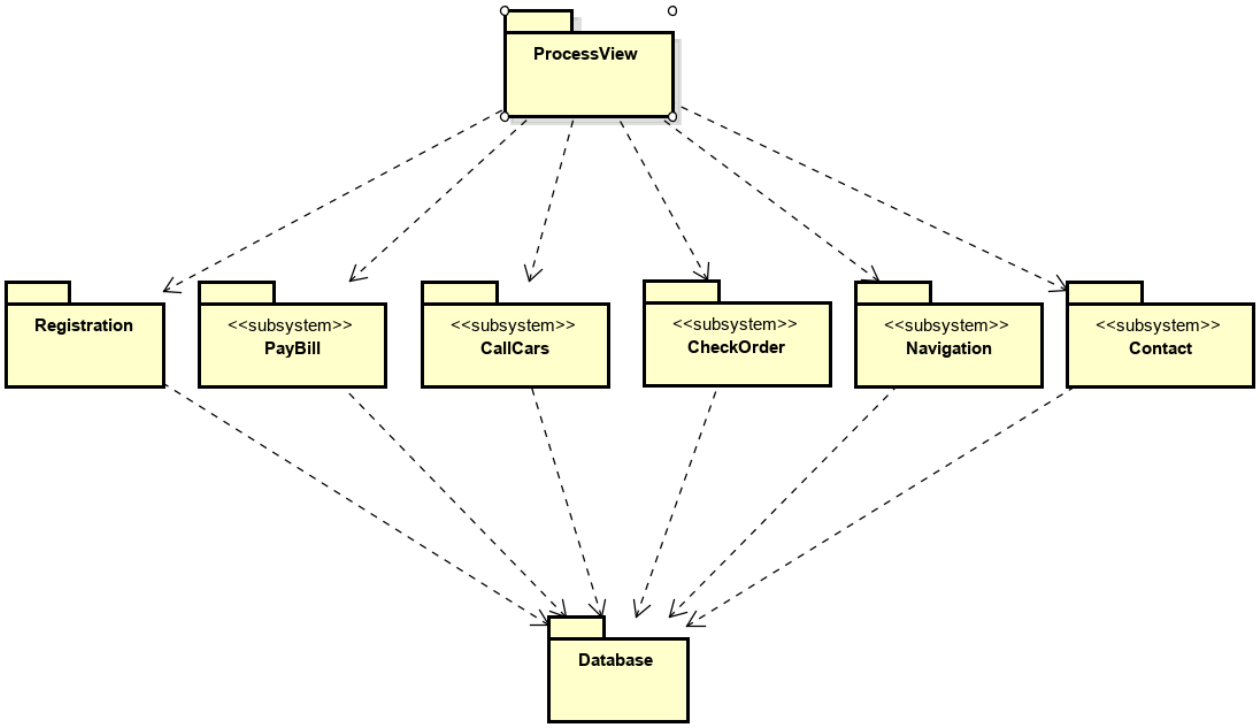

图 8 实现设计模型

3.3.4 实现进程

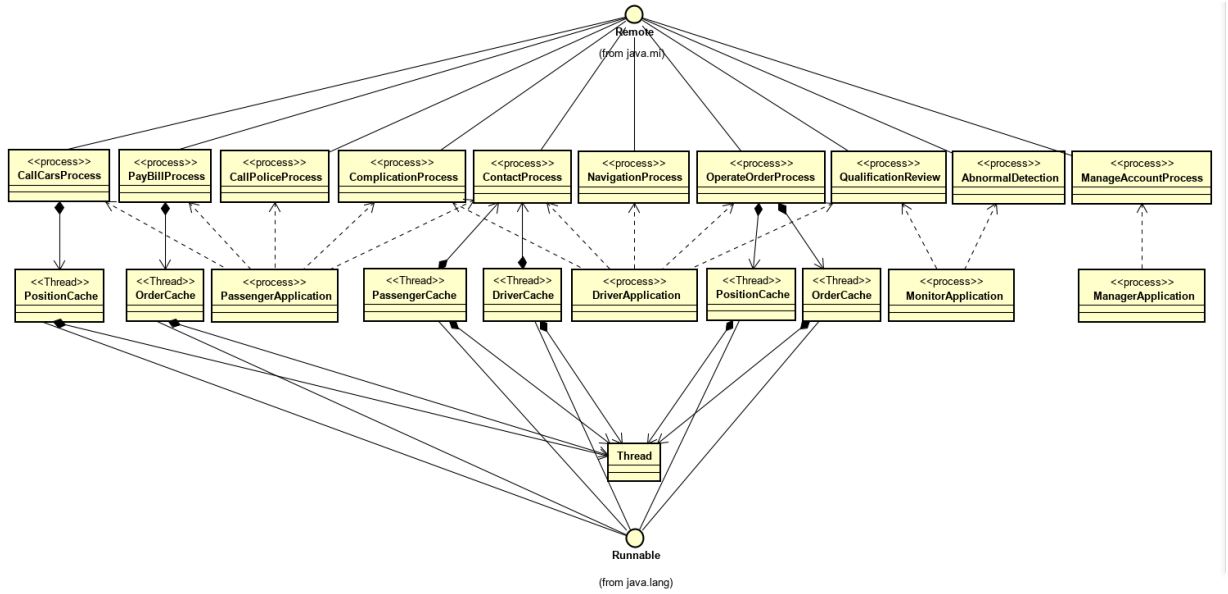

图 9 实现进程图

（一）远程

- 远程接口用于识别所有远程对象。任何作为远程对象的对象都必须直接或间接实现此接口。远程接口中仅指定了那些方法。
- 实现类可以实现任意数量的远程接口，并且可以扩展其他远程实现类。

（二） Runnable

- Runnable 接口应该由实例旨在由线程执行的任何类实现。该类必须定义一个没有参数的方法，称为 run。

- 此接口旨在为希望在活动状态下执行代码的对象提供通用协议。例如，Runnable 由 Thread 类实现。
- 处于活动状态仅表示线程已启动但尚未停止。

（三） 线程

- 线程是程序中的执行线程。Java 虚拟机允许应用程序具有多个并发运行的执行线程。
- 每个线程都有一个优先级。具有较高优先级的线程优先于具有较低优先级的线程执行。每个线程可能会也可能不会被标记为守护程序。当在某个线程中运行的代码创建新的 Thread 对象时，新线程的优先级最初设置为与创建线程的优先级相等，并且当且仅当创建线程是守护程序时，该线程才是守护程序线程。

3.4 部署视图

体系结构的部署视图的描述描述了最典型的平台配置的各种物理节点。还描述了任务（从“过程”视图）到物理节点的分配。本节按物理网络配置进行组织。部署图说明了每个此类配置，然后将进程映射到每个处理器。

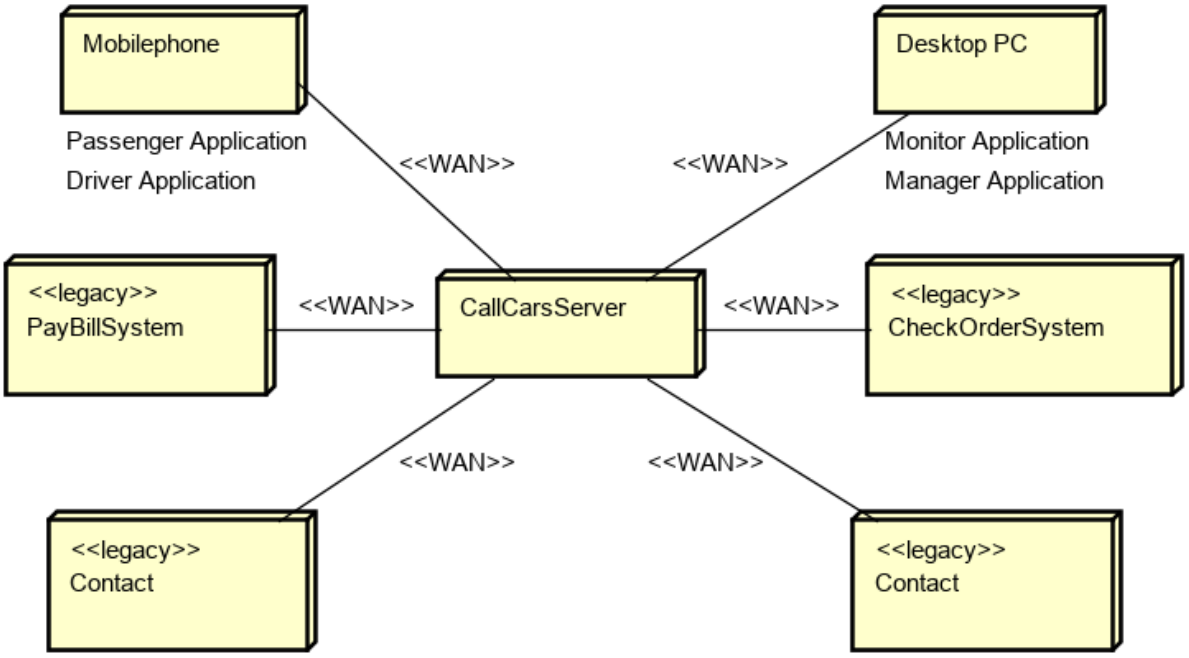

图 10 部署视图

3.5 各相关方对体系结构的要求

本打车软件系统结构视角如图 11 所示，该模型从 4 个角度（逻辑、实现、进程和部署）指出不同的相关利益方关心的事情，外加从使用者的角度对用例做观察，分析其影响系统的上下文和商业目标情况。

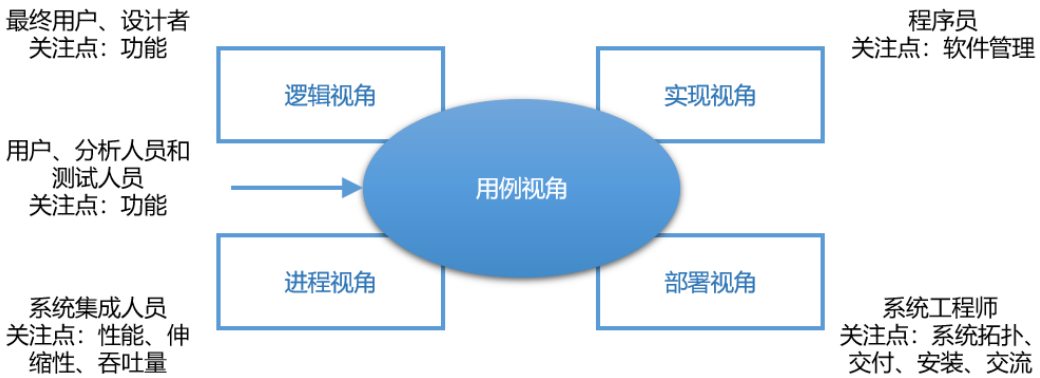

图 11 软件体系结构视角

3.5.1 系统的设计者

系统可以实现需求分析阶段的基本功能，对于本打车软件系统，要求系统能满足乘客、司机、系统管理人员和系统监测人员四种不同用户的功能需求。通过限制不同角色的使用权限，实现系统的安全与稳定，节省了人力物力与财力。

3.5.2 用户、系统分析人员

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。

系统具备良好的人机交互能力。系统提供的各种功能便于用户理解，操作简单，用户很容易掌握。拥有不同操作习惯与受教育程度的用户人群都能够对系统进行正确的操作。系统界面简洁明了，使得用户可以自己学会使用本系统。

3.5.3 系统集成人员、测试以及开发人员

系统应当满足需求分析阶段提出的性能指标，比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。

体系结构的设计应与部件无关、低风险性、高可复用性高、可度量性、高可移植性、高可维护性。

(1)与部件无关：是指体系结构的部件可相互独立地工作，无需了解其他部件的具体实现原理。

(2)低风险性：按此体系结构实施的技术和技能的风险度较低。

(3)可复用性：体系结构设计抽象，且具有高通用性。

(4)可度量性高：按该体系结构实施，系统的开发进度，人力、资金、资源调配可以估计的能力。使项目管理人员能更好地进行项目进度的管理和安排。

(5)高移植性：软件的接口易改造，可以比较容易地转移到其他移动终端上使用，可以比较方便地移植到不同平台上。

(6)高可维护性：系统的构建结构合理，采用“高内聚、低耦合”的原则，可以比较方便地进行修改、升级、测试、维护。

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。

3.6 约束条件

3.6.1 质量或可信赖性属性

(1) 资源管理：服务器部件的内存必须具有 40%以上的预留量。

(2) 易用性：客户端必须支持 Android 系统和 IOS 系统，服务端应当支持 Linux 系统和 Windows 系统。同时考虑到司机在行车途中大多使用移动终端上网，因此系统客户端需要支持主流的移动终端操作系统和市面上主流配置的移动端设备。

(3) 可修改性：系统软件接口易改造，因此转置到其它计算机上的能力很强。又由于本系统可维护性强，也方便了系统的移植。同时要求使用开源库进行系统设计，以保证系统具有良好的可移植性。

(4) 密安性：所有的修改请求必须被授权，并使用认证过程进行加密。

(5) 可使用性：系统必须 24×365 运行，可使用性达到 99%以上。

(6) 法律和政策方面的限制：开发此软件时，将严格按照有关法律和政策执行。

3.6.2 业务和项目约束条件

3.6.2.1 项目基本限制：

系统预计拥有 4.5 亿用户，1.5 亿月活用户，1500 万司机。

3.6.2.2 项目技术上的限制：

(1)常用的打车软件系统采用 C/S 开发模式，系统应采用移动客户端开发的基本技术。

(2)Web 服务器部署于 Linux 系统之上的系统一般运行稳定性高。

(3)服务器系统硬件的配置能支持服务器高效稳定的运行。

(4)为了系统将来的可扩展性，系统在硬件的使用上需尽可能减少针对性。整个系统也应尽量减少模块间的调用，尽量做到松耦合。

(5)数据文件、系统配置文件应当安全可靠的存储。

(6)系统的数据需要具有足够的可靠性，才会保证系统正常运行。

3.6.2.3 项目开发要求

(1)软件开发小组提供相应的开发阶段的文档，用户提供相适应的行业标准，使软件开发与典型实例考核相结合。

(2)要按照操作规程运行本系统，不得进行恶意破坏性操作。

(3)用户必须提供相关运行软件有效的数据库接口标准，并在改动的过程中及时通知本软件开发商，以保证从中正确读取预决算参数，进行成本预算。

(4)客户端采用 Android 开发和 IOS 开发技术，并且所有变量的命名规范符合相应语言命名规范。

(5)开发过程中使用到的第三方库或商业组件都是可用的，可实现的。

(6)人员约束：所有员工必须签署数据保密协议。

(7)密安性约束：使用登陆对用户进行分权限操作，对应用户名和密码存储至数据库中。

(8)运行打车软件系统手机客户端的移动设备具有定位功能并且提供了接口可供调用，可以连接到网络。

(9)用户所在的网络环境能够访问到打车软件系统所部署的服务器。

(10) 严格使用面向对象的开发模式，使用 MVC 框架实现前后端的分离，保证系统代码的清晰。

3.6.3.4 项目进度要求

(1)产品的第一个版本必须在 6 个月内交付，产品后续版本需要每三个月进行一次迭代，在 15 个月内交付完整版本。

(2)假设开发人员需要在 20 天内完成系统概要设计、详细设计和编码的部分，然后测试人员需要在 10 天内完成测试任务。如果其中任一个环节未在规定时间内完成，则将会导致项目完成时间延长。

3.7 非功能性需求

所谓非功能性需求，是指软件产品为满足用户业务需求而必须具有且除功能需求以外的特性。非功能性需求在需求分析阶段常常被忽略或没有被足够重视。软件产品的非功能性需求包括系统的质量需求和工程需求。对于本系统，主要通过以下方面对本系统的非功能性需求进行描述。

3.7.1 质量要求

3.7.1.1 性能

本系统的非功能性能指标可以分为时间性能、空间性能和并发量。

(1)时间性能：

a.系统执行速度：系统响应快。数据库单表操作时间不大于 0.5 秒。

b.系统响应时间（以下需求均无视网络延迟，而只考虑系统的延迟）：

- 当用户登陆时，系统必须在 0.5 秒内给出登陆成功或失败的提示。
- 乘客应当可以查看个人信息和司机信息，司机可以查看个人信息和资质审查详情，上述查询流程的响应速度最佳为 0.1 秒，平均为 0.5 秒，不超过 1 秒。系统应当可以接受 3 万人同时操作。
- 乘客和司机应可以查看历史订单详情和历史投诉详情，上述查询流程的响应速度最佳为 0.1 秒，平均为 0.5 秒，不超过 1 秒。系统应当可以接受 3 万人同时操作。
- 乘客和司机应可以修改个人信息，上述修改的响应速度最佳为 0.1 秒，平均为 0.5 秒，不超过 1 秒，且必须返回修改是否成功。系统应当可以接受 3 万人同时操作。
- 乘客支付车费的响应速度最佳为 0.1 秒，平均为 0.5 秒，不超过 1 秒，且必须返回支付是否成功。系统应当可以接受 50 万人同时操作。
- 乘客和司机查看行程信息时，定位更新响应速度最佳为 0.05 秒，平均为 0.1 秒，不超过 0.2 秒，系统需要实时刷新行程相关信息。系统应当可以接受 50 万人同时操作。
- 乘客一键报警响应速度最佳为 0.05 秒，平均为 0.1 秒，不超过 0.2 秒，系统及相关人员需要迅速响应。系统应当可以接受 1 万人同时操作。
- 系统产生警告信息到将警告消息显示在屏幕上的延时不超过 10 秒。
- 系统更新数据时，最大延时为 2 小时。
- 在系统因为访问人数过多而发生宕机时，本系统应该在 10 分钟之内可以进行恢复。

(2)空间性能：

- 本系统应该存储至少 500,000 人的信息的记录，用于不同身份背景的人进行系统的使用。

- 本系统的磁盘容量应该至少为 15TB，其中应该有至少 2TB 的空闲空间，用于信息的扩充。

(3)并发量

系统月活用户数量为 1.5 亿，同时在线用户数的期望值为 100 万，但每一时刻打车的用户数目会随时间变化（比如工作日早晚高峰要多余非高峰时段和非工作日），因此其同时在线人数的最大值为 300 万 人，因此系统期望正常运行在 100 万人的并发访问数上，最多支持 300 万人的并发访问数。

3.7.1.2 可靠性

- 系统测试时间至少为 500 小时，测试参与用户至少为 1000 人。本系统应该避免因为故障而导致的失效的能力，在进行测试的过程中，本系统应该尽可能遍历所有的可能故障情况，保证系统的成熟。
- 网络设备的可靠性由公用网和局域网及其设备组成，其中局域网及其设备的可靠性是整个系统的重要指标之一，在设备选择时应考虑到所选设备的可靠性及其售后服务质量。
- 服务器的可靠性是整个系统最基本的可靠性指标，对服务器的选择应充分考虑其可靠性和售后服务质量，更重要的是服务器应该有冗余配置或备份设备，以保障数据库服务器可每天 24 小时运行,其平均故障恢复时间不应大于每 3 个月 1 小时。
- 系统具有一定的容错能力和故障恢复能力，在发生硬件或者软件异常时，应该具有依然具有服务能力。服务能力虽然下降，但不会导致系统崩溃无法使用。
- 本系统在发生停机故障之后，应该保证十分钟之内可以修复并且正常运行。
- 考虑到此软件系统的应用场景为交通，软件系统必须是可靠的。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。软件系统必须定期备份数据以防止。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。选择数据库产品时，要考虑一定的数据负载能力。由于在处理员工信息、客户信息、账户信息等信息时，系统需要做大量的数据统计和处理，因此要具备相应的数据负载能力。 此外软件系统本身也需要提供一定的容错机制，比如必要的提示和确认机制以防止用户误操作。

3.7.1.3 可用性

- 系统支持在多种操作系统和移动终端下运行和使用。
- 系统界面美观，简洁明了，使用户能够自己学会使用本系统，有较好的用户体验，能吸引使用者。
- 本系统易于使用。新用户在初次使用系统时，系统会自动跳出操作指南指导用户使用系统，用户在不超过 3 分钟的学习时间之内便可以明白整个系统的基本功能并能熟练使用系统。
- 系统提供相应的操作指南文档。本系统提应该提供给使用者相应的文档使用手册，便于使用者获得系统功能的详细解读和使用。
- 打车软件系统面向的用户年龄层、残障情况、手机及互联网熟悉情况范围极大，因此系统的界面设计和逻辑处理必须人性化，具备良好的人机交互能力，符合人们长久以来打车的习惯。操作的设计应尽可能简易实用。
- 系统需要提供必要的撤销机制，并且当执行某些不可撤销操作时必须予以提示以防止用户误操作。
- 在需要多次重复操作的地方要提供必要的批处理功能和存储功能以提高用户体验。
- 同时打车软件系统还面向司机，使用场景主要为行驶中。打车软件系统必须支持文字交互、语音交互等多种交互方式，同时内置详细操作手册，界面设计尽量降低误触率，按键明显，自动化程度高。

3.7.1.4 密安性

- 系对不同用户的权限进行控制，且用户使用系统前应先使用注册手机号和密码进行系统的登录，不同的类型的用户具有不同的系统使用权限。通过系统管理人员管理实现对于不同角色开放不同功能的行为，来防止不合法的操作的产生，保证了系统的安全性和数据的保密性。例如司机不可以查看或修改乘客的个人信息，而系统监测人员可以查看司机的信息。
- 要求具有安全可靠的用户认证机制，防止用户通过伪造身份的手段伪造他人的身份获取到其不该知道的信息或对数据进行修改。

- 系统对于重要数据进行数据加密。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。
- 系统数据定期备份。为了本打车软件系统的长远考虑，系统数据应该进行相应的备份，同时拥有冷备份和热备份，在系统停止工作时能保证进行数据恢复。
- 防止系统出错意外崩溃时用户信息以及日志数据的丢失，并且能够持续为用户提供可靠服务。
- 该系统应该通过设置防火墙确保数据传输的安全。使用可靠的操作系统来保证系统的操作安全，确保系统在一个安全可靠的环境下运行。
- 系统应保证用户信息不泄露，系统配置文件和数据库存储文件应当进行加密处理。
- 系统应保证不会因恶意攻击而崩溃，系统开发过程不存在明显漏洞。
- 系统应当能够保证选取的开发方不存在商业竞争对手或类似的恶意对手。

3.7.1.5 可维护性

- 软件的可维护性是指改进软件的难易程度。开发人员在实现此系统时必须充分了解此系统地业务流程和运行周境，精心设计各个部分的接口和实现方法，保证该系统的结构、接口、功能以及内部过程在开发以及跟踪阶段容易被维护人员理解。
- 系统的代码和文档结构应该保证清晰易读，便于在系统出现错误之后程序员能够进行快速的恢复抢修，并且能准确的找出系统错误的原因。
- 利用先进的软件开发技术和工具，提高可维护性。
- 在软件开发每个阶段结束前的技术审查和管理复审中，着重对可维护性进行审查。
- 选择可维护的程序设计语言。
- 改进程序文档。文档是影响软件可维护性的重要因素，好的文档是建立可维护性的基本条件。在软件维护阶段，利用历史文档可以大大简化维护工作。历史文档有 3 种：系统开发日志、错误记载、系统维护日志。
- 软件的可维护性是指改进软件的难易程度。系统部件遵守“高内聚，低耦合”原则。该系统的结构、接口、功能以及内部过程在开发以及跟踪阶段，容易被维护人员理解，便于完成后期系统测试。

- 系统开发完毕后提供必要的操作维护手册和技术手册。
- 系统应当拥有良好的测试和诊断系统错误的功能。
- 系统开发完毕后提供必要的操作维护手册和技术手册。
- 当系统应用于不同城市的公共交通系统之下时，应该具备良好的适应性。不需要通过大幅度的接口与内部过程修改，就能使用户进行使用。

3.7.1.6 可移植性

- 易安装性：该打车软件系统能够跨平台移动运行，包括 IOS 和 Android。
- 共存性：打车软件系统应当能够和其他软件共存于一个平台上。
- 易替换性：打车软件系统可被容易地卸载，也容易升级到更高版本的系统。
- 系统能市面上常用的操作系统环境下方便地安装和使用，不会因为环境的不同而导致系统崩溃或者系统功能缺失。
- 尽可能使程序适用于所有的编译程序，并把不可移植的代码分离出来。
- 系统接口设计合理，便于进行某些模块功能代码的重用，方便被当作模板框架来进行相应其他系统的二次开发。
- 系统应当尽可能使用标准库函数，而不依托于特定的操作系统环境。
- 系统应当使用跨平台框架与语言进行设计，保证系统能够在任何主流操作系统上运行。

3.7.2 工程需求

3.7.2.1 设计约束

(1)本系统在进行代码编写的时候要求给出详细的代码注释，每一个功能函数都要给出函数的执行功能，输入和返回值，便于后期进行代码的重用。

(2)系统要求服务器采用 Java 语言进行编写，客户端采用 Android 开发和 IOS 开发技术，并且所有变量的命名规范符合相应语言命名规范。

(3)本系统移动端应用框架采用 MVVM-Chameleon 框架进行系统开发

(4)基于系统安全和保密性的考虑，系统的配置文件、数据存储文件等应进行加密处理，采用国际通用的加密算法，防止意外泄露或恶意攻击。

(5)系统代码设计要严格使用模块化设计，便于后期进行代码的重用，减少代码冗余。

(6)Web 服务器部署于 Linux 系统之上以保证提供稳定可靠的服务。

(7)Web 服务器系统硬件配置需满足服务器能够高效稳定地与逆行。

(8)基于系统可靠性的考虑，系统的数据存储文件应进行冗余备份，比如磁盘冗余阵列存储 RAID 等。

(9)为了系统将来的可扩展性，系统硬件使用国际通用的硬件，不应使用具有针对性的硬件。整个系统也应尽量减少各模块间的调用，尽量做到松耦合。

3.7.2.2 逻辑数据库要求

3.7.2.2.1 基本需求

- 系统并发量较大，应使用 Oracle 等大型数据库，可靠性较高，稳定性较高。
- 数据库的字段定义应具有一定的灵活性，保证一定的可修改性。
- 数据库表的设计应保证一致性、完整性，避免出现数据冗余，出现数据不一致现象应进行及时的调整。
- 满足以上的条件下，数据库数据存储尽量节省空间。
- 本系统数据库设计满足完整性约束: 实体完整性约束 、域完整性约束、 引用完整性约束、 自定义完整性约束。
- 本系统对于数据库的访问操作要有相应的并发处理和恢复机制，并且保证访问获取到数据信息的正确性。

3.7.2.2.2 数据库需求

数据库需求设计分为两部分：概念结构设计和逻辑结构设计。

概念结构设计指的是画出 E-R 模型。将概念结构进一步转化为某一 DBMS 所支持的数据模型，然后根据逻辑设计的准则、数据的语义约束、规范化理论等对数据模型进行适当的调整和优化，形成合理的全局逻辑结构，并设计出用户子模式。这就是数据库逻辑设计所要完成的任务。

3.7.2.2.3 E-R 图

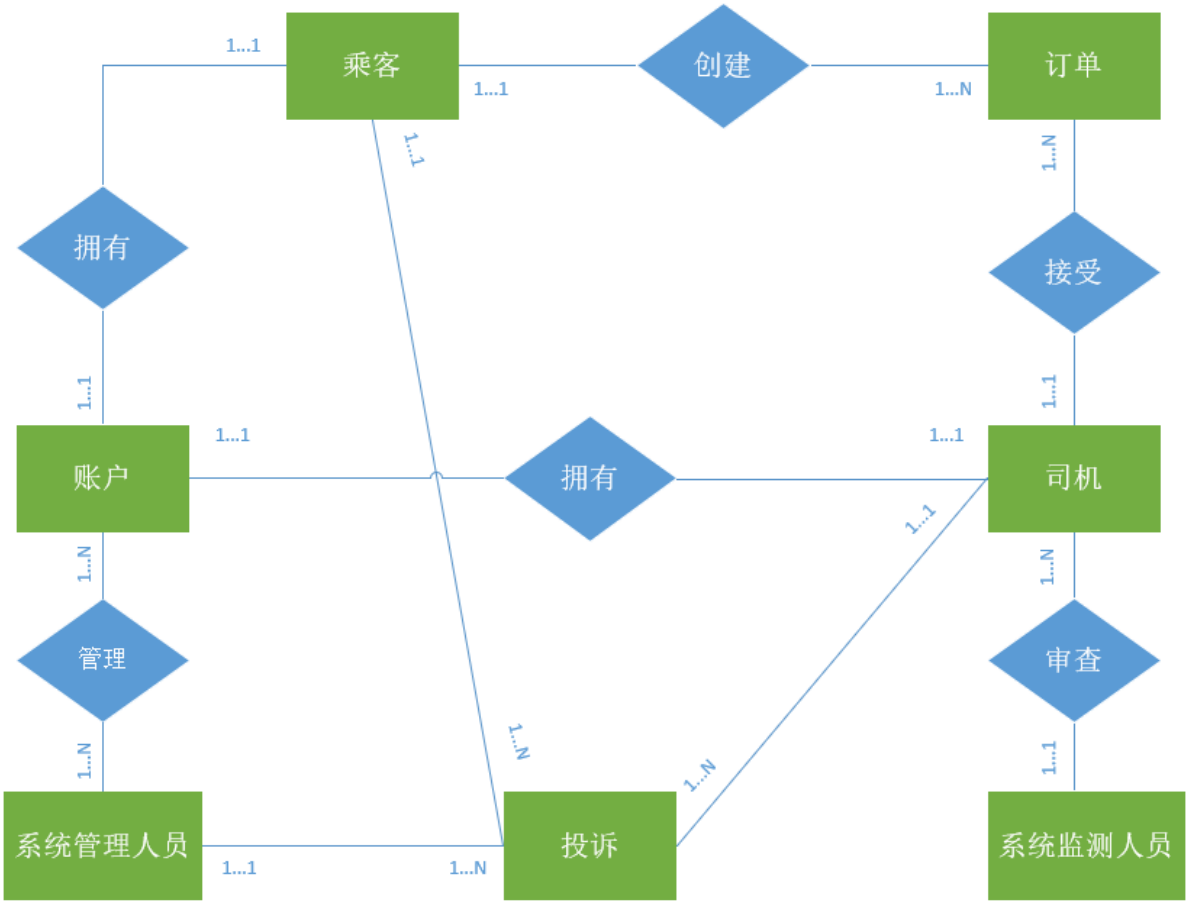

图 12 打车软件系统 E-R 图

3.8 风险

（一）如果系统使用人数在某个时候大幅增加，可能会导致服务器负载过重，而影响高并发时系统的性能和响应时间，甚至可能造成系统崩溃。

解决方案：

- 添加备用的服务器，提升系统对 Web 请求的响应性能。
- 使用 KVM 技术进行云主机虚拟化，在高峰时间段使用云服务器进行扩容，提高系统处理能力，以满足系统对 Web 请求的响应性能。

（二）如果系统的需求发生变更，可能会导致当前的体系结构无法再满足系统的需要。

解决方案：

在进行体系结构设计时，尽量做到部件和模块之间的分离。这样，在将来需求分析发生变更的时候，能够尽量减少代码层面上的更改，而是从部件和模块层面上对系统进行改

变。建立完善的需求变更管理体制，跟踪系统的需求变动轨迹，在确定需求的每一步中都要经过专人审核。

（三）如果系统的存储设备出现了故障，可能导致用户订单或个人信息数据的丢失。

解决方案：

使用 RAID 技术，使用磁盘阵列冗余保存数据，进行数据校验，保证用户数据的真实性和完整性。

4. 解决方案

4.1 相关的体系结构模式

为了满足打车软件系统的多种需求，在本项目中将采用四种设计模式进行开发设计。乘客和司机是我们系统的主要使用者，考虑到他们的使用场景，C/S 架构是最佳的选择。同时，我们的系统要求为系统管理人员和系统监测人员提供方便快捷的用户权限管理、异常行程监测、投诉处理等功能，而且工作场景固定，多层 B/S 架构便成为了最佳的选择。同时，我们的系统要求具有极高的时效性。每一次数据更新必须快速返回给系统以及用户，而且多个系统部件需要共享数据，选择主动式中心仓库模式进行架构设计同样是系统需要。此外，为了使得代码结构清晰，层次分明，展示层与应用层、数据层能够清晰有效地进行高效率的交互，系统还应当使用 MVC 模式进行开发。下面对于这几种模式结合系统进行简要的介绍，解释为什么本系统要采用这些模式进行设计开发。

4.1.1 三层 C/S 体系结构模式

C/S 结构（Client/Server，客户端/服务器架构），是大家熟知的软件系统体系结构，通过将任务合理分配到 Client 端和 Server 端，降低了系统的通讯开销，需要安装客户端才可进行管理操作。客户端和服务器端的程序不同，用户的程序主要在客户端，服务器端主要提供数据管理、数据共享、数据及系统维护和并发控制等，客户端程序主要完成用户的具体的业务。

而三层 C/S 体系结构是在 C/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。

层与层之间的通信是异步的“请求-响应(request-reply)”。请求是单方向的，从客户层开始，经过 Web 和业务逻辑层，再到数据管理层。每层都等待其它层的处理响应。

分层结构不限制多层应用的部署方式。所有层可以运行在一台机器上，或者，每层部署到一台独立的机器上。 分层可以实现各层的标准化，例如，在 Web 应用中，客户端可以使用不同厂家的浏览器，兼容地对不同厂家 Web 服务器访问。

表示层：客户端运行用户界面,向用户提供数据输入输出,响应用户动作并控制用户界面,把业务逻辑与用户界面分开,简化客户端的工作。

业务逻辑层：是系统实现业务逻辑与数据操作的核心部门，它的任务是接受用户的请求，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。

数据层：数据库服务器运行数据引擎,负责数据存储。其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,把运行结果返回给中间层。

图 13 是三层 C/S 结构的示意图。

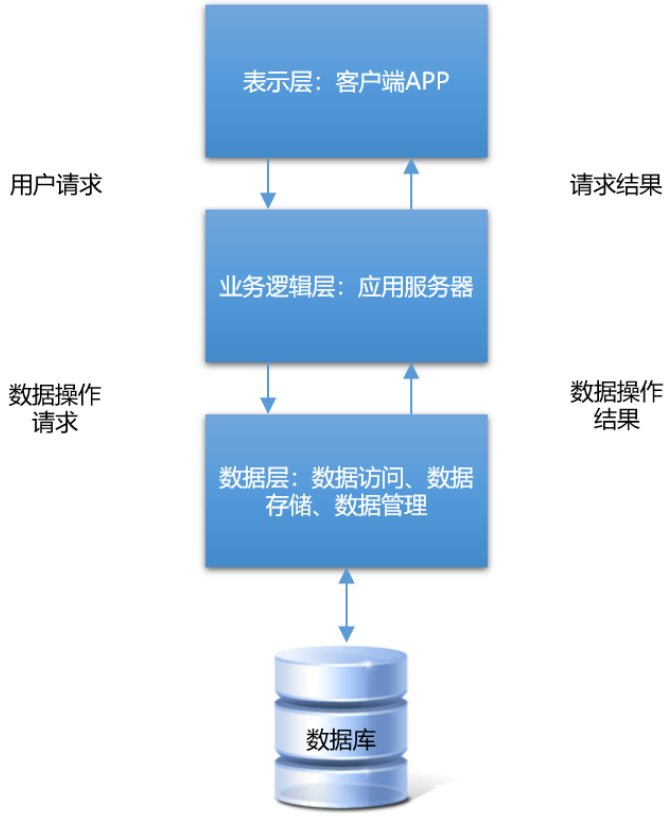

图 13 三层 C/S 体系结构示意图

4.1.2 主动式仓库模式

在需要数据被多个部件共享的情况下，共享仓库模式给予了用户很好的机制和选择。相比于顺序体系结构，其优点在于实现了数据共享，以及不相邻的两个部件之间的信息交流。在我们的系统中，数据经常是被动的访问，并经常发生更新。在这种情况下，如果仓库中的数据改变或数据访问发生变化，客户端必须立刻得到通知。

主动仓库模式是共享仓库模式化的一个变体，要求把仓库中发生的特殊事件主动地通知到客户端。主动式仓库要维护客户端的注册表，并通过适当的提示机制通知客户端。

4.1.2 MVC 模式

一个系统往往需要提供多个用户界面，而每个用户界面可能只需要反应一部分应用数据。当这些数据改变时，要能够自动和灵活地反应到不同的用户界面上。这就需要能够很容易地修改其中的一些用户界面，而不需要修改与数据相关联的应用逻辑。

解决的方法是将系统划分为三个部分：

- 一个模型(Model)：封装应用数据及其对这些数据的操作，与用户界面独立开；
- 一个或多个视图(Views)：向用户展示指定的数据；
- 一个控制器(Controller)：与每个视图关联起来，接收用户的输入，并将它翻译成对 Model 的请求。

View 和 Controller 组成了用户界面。依据 MVC 模型，更改所有 View 和 Controller 的通知机制可以用“发布-订阅(Publish- Subscribe)”模式实现。所有控制器和视图从模型接收到的订阅都要发布出通知。用户只通过 View 及其 Controller 进行交互，而不依赖于 Model，反过来，将更改情况通知给所有的不同用户。

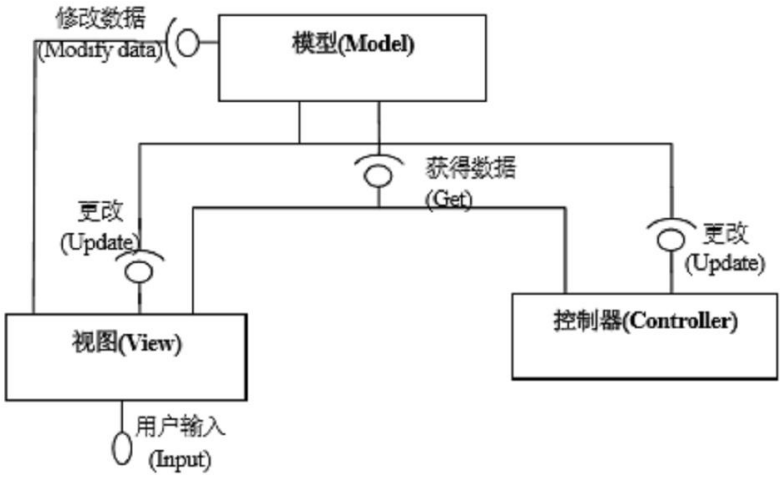

图 14MVC 模式示例

4.1.2 三层 C/S 体系结构的优点

(1) 服务器端采用 Java 服务器，客户端采用.net 的开发技术，各取其所长。Java 技术长于服务器，服务器的稳定性业界公认，常用于大型的系统，但 java 技术在客户端的开发则劣势明显。微软的.net 开发框架则恰恰相反，微软的中间件服务器，也就是 IIS，其稳定性和安全性一直为业界所诟病，但其开发的客户端拥有良好的操作性和高效的工作效率。

(2) 响应速度快，能充分发挥客户端 PC 的处理能力，很多工作可以在客户端处理后再提交给服务器，所以 CS 客户端响应速度快。

(3) 个性化设计，软件设计个性化，操作界面漂亮、形式多样，可以满足用户个性化的需求 。因为 C/S 结构有着自己的客户端，在客户端的设计上，可以根据客户的需求进行个性化的设计，可以满足客户复杂的个性化需求，特别是在操作界面、报表工具等方面。

(4) 事务处理能力强大，C/S 结构的管理信息系统具有较强的事务处理能力，能实现复杂的业务流程。C/S 结构充分利用客户端的硬件设施，将很多的数据处理工作在客户端完成，故数据处理能力比较强大，对一些复杂的业务流程，也容易实现。

(5) 安全性高，安全性能可以很容易保证，C/S 一般面向相对固定的用户群，程序更加注重流程，它可以对权限进行多层次校验，提供了更安全的存取模式，对信息安全的控制能力很强。一般高度机密的信息系统采用 C/S 结构适宜。

4.2 本项目体系结构概述

打车软件系统主要应具有乘客打车和司机接单两大功能，同时涉及到用户注册登录子系统，第三方支付系统，行程监测系统，订单投诉子系统等，打车软件系统需要不断更新数据库内容,支持的用户数量大，是一个非常复杂的分布式应用系统。

分层的 C/S 架构融合了分层模式的可修改、可移植、可重用的优点以及 C/S 模式易于开发，结构清晰的优点，而且十分契合我们的实际应用场景。同时，其他角色用户的功能同样适合分层的 C/S 模式开发，于是统一各个功能的结构模式，同时节约了开发成本。三层 C/S 体系结构又紧密结合了 Internet/Intranet 技术,是技术发展的大势所趋,因此基于三层 C/S 结构设计城市共享停车管理系统是一个较为理想的选择。

由于我们的系统提供的功能较多，需要多个用户界面，为了当数据改变时能够自动灵活的反映到不同的用户界面上，而不需要更改数据相关联的格式，我们的系统同样应该使用 MVC 模式进行开发，以实现不同层的分离。

同时，我们的系统具有极高的时效性，要求数据层的更新及时、准确地通知到客户端，而且多个部件之间经常会调用相同的数据。因此，我们的系统同时应该使用主动式的仓库模式进行开发。

4.2.1 概念级体系结构

概念级的体系结构是从需求工程阶段给出的系统模型直接进化过来的。在概念级体系结构中，表示层展示系统页面，应用层实现系统的主要功能，处理用户请求并返回相关数据，持久化层将用户信息和相关数据保存在数据库中，概念级体系结构的示意图如图 4-2 所示。

表示层：即在最终用户面前的界面。在这一层为用户呈现出各自不同的界面，用户在该层实现各自的功能操作，并且接受用户从客户端发送的业务请求，将相关请求发送到逻辑业务处理层，并将逻辑业务处理层反馈的处理结果输出用户屏幕上。系统的 APP 界面友好，乘客、司机、系统管理人员、系统监测人员在客户端 APP 对系统进行操作，系统响应用户操作，实现用户与系统的交互。

应用层：即为系统的业务逻辑层，也是系统服务器的所在位置。在该层接受各种各样的用户请求，并进行处理，然后将结果返回给用户。主要实现系统的核心功能，实现包括登录、提交打车请求、接受打车请求、实时更新行程信息、更新订单信息、维护订单信息、授予权限等业务逻辑。联系系统的前端与后端。

持久化层：该层主要负责提供数据信息以及存储数据的功能，能够对不同数据格式以及信息实现共享和交换，返回数据库中的数据或者是修改请求，对数据库中数据进行修改。该层主要根据用户操作与数据库进行交互和信息交换。

4.2.2 模块级体系结构

系统的模块化设计是系统进行复用的关键。模块级体系结构反应了对软件代码实现时的期望。特别是对于程序规模较大的系统。体系结构的层次表达了每层的向上一个层面提供的功能和接口，以及需要使用下一层的功能。

4.2.3 运行级体系结构

运行级的体系结构用来描述系统的动态结构，用运行级体系结构才能很好地进行性能分析、监视和调整，以及调试和服务维护。本打车软件系统的运行级体系结构如下图所示。

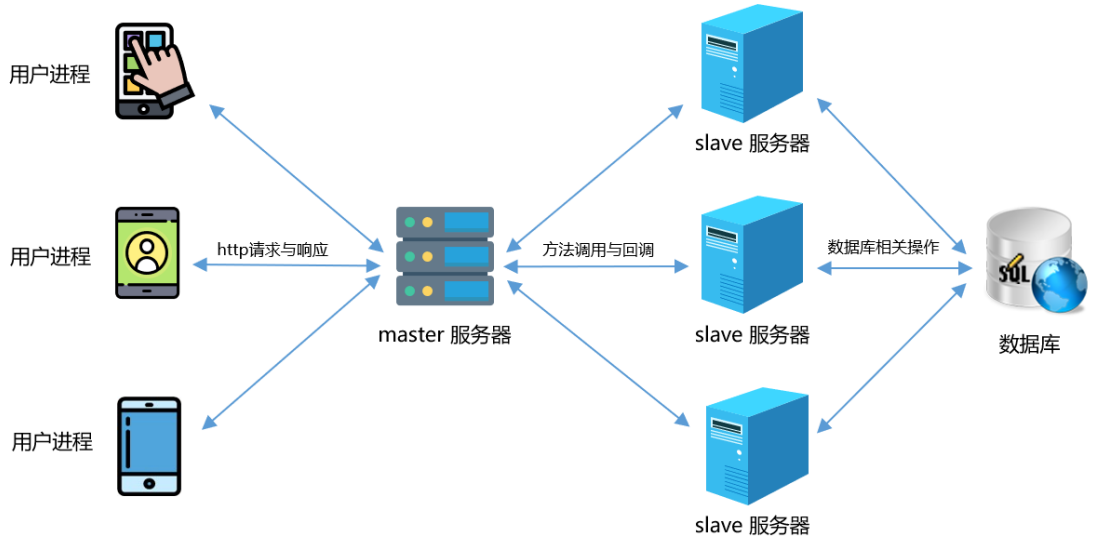

图 15 运行级体系结构示意图

系统的运行级体系结视图描述了模块在运行时，被指派到的特定运行映像运行级体系结构是系统的动态结构，主要用于系统的性能和可调度性分析，以及系统静态与动态配置的管理。

4.2.4 代码级体系结构

代码体系结构为编程实现提供方便，在编程语言层面按模块、目录、文件、和库的形式组织源代码。如果说在模块化体系结构设计时不要过多地考虑所采用的编程语言的话，这个阶段必须考虑编程语言、开发工具和环境(例如，配置管理)，以及项目和企业的组织结构。从体系结构设计角度看，如何降低代码之间、子项目之间的相互依赖性，提升代码的可重用性等，成为体系结构设计阶段必须考虑的问题。

如前所述，本打车软件系统严格遵循 MVC 架构，同时采用 C/S 与主动式中心仓库架构进行开发，因此本系统的整体代码级体系结构如下图所示。

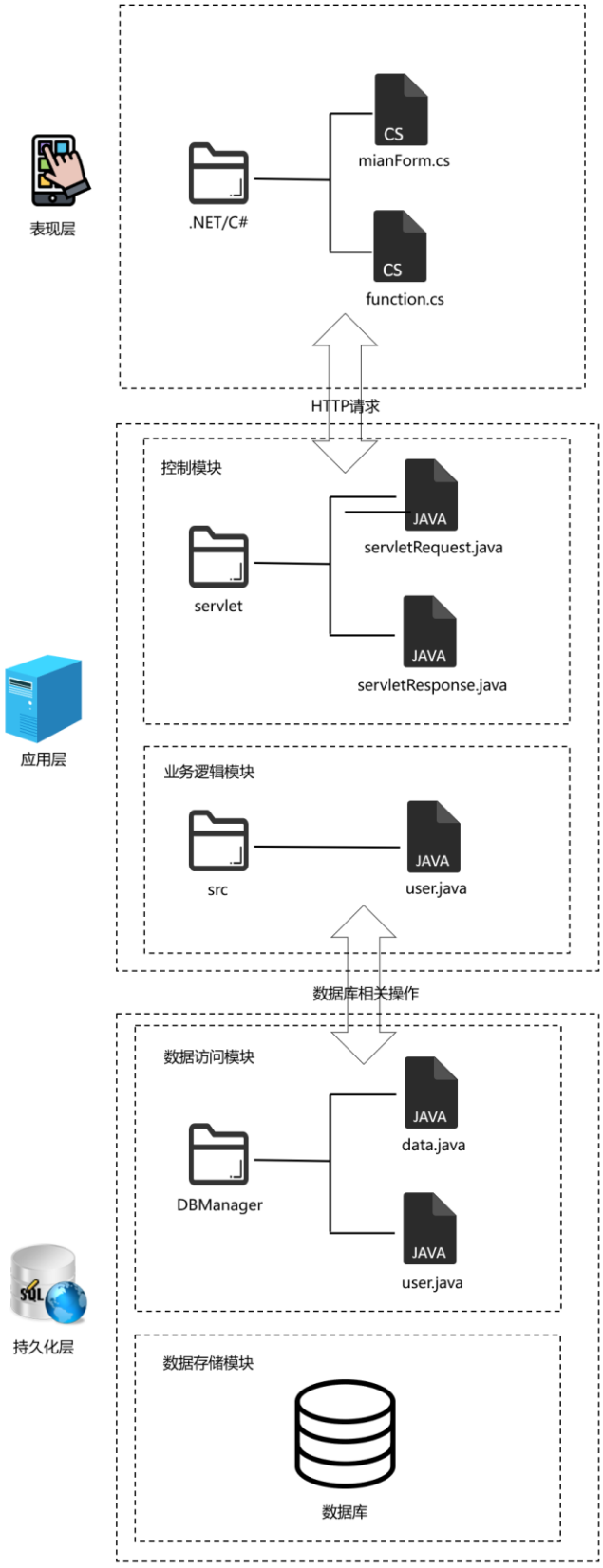

图 16 代码级体系结构示意图

4.3 结构化视图

本打车软件系统结构主要分为三层：表示层、业务逻辑层、数据层。结构化视图如下图所示。

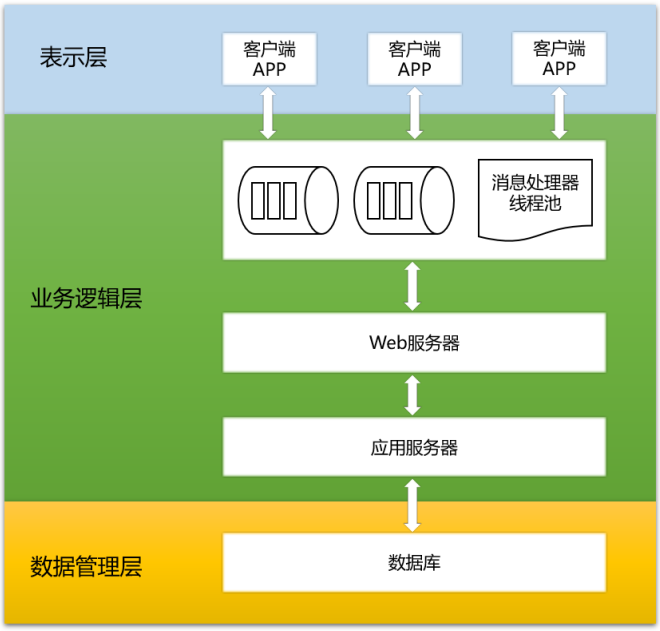

图 17 结构化视图

5. 系统的质量分析和评价

ATAM：Architecture Tradeoff Analysis Method(构架权衡分析方法），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（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。

5.1 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 方法来对城市共享停车管理系统的体系架构进行分析和评估。

5.1.1 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 1	(1) 系统支持在多种操作系统和移动终端下运行和使用。 (2) 系统界面美观，简洁明了，使用户能够自己学会使用本系统，有较好的用户体验，能吸引使用者。 (3) 本系统易于使用。新用户在初次使用系统时，系统会自动跳出操作指南指导用户使用系统，用户在不超过 3 分钟的学习时间之内便可以明白整个系统的基本功能并能熟练使用系统。 (4) 系统提供相应的操作指南文档。本系统提应该提供给使用者相应的文档使用手册，便于使用者获得系统功能的详细解读和使用。 (5) 打车软件系统面向的用户年龄层、残障情况、手机及互联网熟悉情况范围极大，因此系统的界面设计和逻辑处理必须人性化，具备良好的人机交互能力，符合人们长久以来打车的习惯。操作的设计应尽可能简易实用。 (6) 系统需要提供必要的撤销机制，并且当执行某些不可撤销操作时必须予以提示以防止用户误操作。 (7) 在需要多次重复操作的地方要提供必要的批处理功能和存储功能以提高用户体验。 (8) 同时打车软件系统还面向司机，使用场景主要为行驶中。打车软件系统必须支持文字交互、语音交互等多种交互方式，同时内置详细操作手册，界面设计尽量降低误触率，按键明显，自动化程度高。
场景 2	(1) 系统测试时间至少为 500 小时，测试参与用户至少为 1000 人。本系统应该避免因为故障而导致的失效的能力，在进行测试的过程中，本系统应该尽可能遍历所有的可能故障情况，保证系统的成熟。 (2) 网络设备的可靠性由公用网和局域网及其设备组成，其中局域网及其设备的可靠性是 整个系统的重要指标之一，在设备选择时应考虑到所选设备的可靠性及其售后服务质量。 (3) 服务器的可靠性是整个系统最基本的可靠性指标，对服务器的选择应充分

	考虑其可靠 性和售后服务质量，更重要的是服务器应该有冗余配置或备份设备，以保障数据库服务器可每天 24 小时运行,其平均故障恢复时间不应大于每 3 个月 1 小时。 (4) 系统具有一定的容错能力和故障恢复能力，在发生硬件或者软件异常时，应该具有依然具有服务能力。服务能力虽然下降，但不会导致系统崩溃无法使用。 (5) 本系统在发生停机故障之后，应该保证十分钟之内可以修复并且正常运行。 (6) 考虑到此软件系统的应用场景为交通，软件系统必须是可靠的。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。软件系统必须定期备份数据以防止。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。选择数据库产品时，要考虑一定的数据负载能力。由于在处理员工信息、客户信息、账户信息等信息时，系统需要做大量的数据统计和处理，因此要具备相应的数据负载能力。 此外软件系统本身也需要提供一定的容错机制，比如必要的提示和确认机制以防止用户误操作。
场景 3	(1) 当用户登陆时，系统必须在 0.5 秒内给出登陆成功或失败的提示。 (2) 乘客应当可以查看个人信息和司机信息，司机可以查看个人信息和资质审查详情，上述查询流程的响应速度最佳为 0.1 秒，平均为 0.5 秒，不超过 1 秒。系统应当可以接受 3 万人同时操作。 (3) 乘客和司机应可以查看历史订单详情和历史投诉详情，上述查询流程的响应速度最佳为 0.1 秒，平均为 0.5 秒，不超过 1 秒。系统应当可以接受 3 万人同时操作。 (4) 乘客和司机应可以修改个人信息，上述修改的响应速度最佳为 0.1 秒，平均为 0.5 秒，不超过 1 秒，且必须返回修改是否成功。系统应当可以接受 3 万人同时操作。

	(5) 乘客支付车费的响应速度最佳为 0.1 秒，平均为 0.5 秒，不超过 1 秒，且必须返回支付是否成功。系统应当可以接受 50 万人同时操作。 (6) 乘客和司机查看行程信息时，定位更新响应速度最佳为 0.05 秒，平均为 0.1 秒，不超过 0.2 秒，系统需要实时刷新行程相关信息。系统应当可以接受 50 万人同时操作。 (7) 乘客一键报警响应速度最佳为 0.05 秒，平均为 0.1 秒，不超过 0.2 秒，系统及相关人员需要迅速响应。系统应当可以接受 1 万人同时操作。

表 3 用例场景

5.1.2 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。

场景 1	可以增加新的数据服务器，降低远程用户的访问时间。
场景 2	通过扩充现有数据库表的规模。
场景 3	通过向数据库提供商购买更多的 license，来提升系统数据库的并发度。
场景 4	增设打车软件系统小程序的开发，方便用户直接在微信或支付宝中而不需下载软件就能使用打车软件。

表 4 增长性场景

5.1.3 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打车软件系统的修改给出更实际的需求。

场景 1	用户不仅能在移动客户端登录系统并进行相关操作，也能在小程序或其他社交平台上对系统进行操作。
场景 2	改进系统的可使用性，从 98%提升到 99.999%。
场景 3	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。

表 5 探索性场景

5.2风险

在开发新的软件系统的过程中，由于存在许多不确定因素，软件开发失败的风险是客观存在的。在系统运行过程中，由于外部环境变化或者自身设计的不足，也可能产生各种风险。因此，风险分析对于软件项目管理是决定性的。

5.3.1 技术风险

在软件项目开发和建设的过程中，战略管理技术因素是一个非常重要的因素。项目组一定要本着项目的实际要求，选用合适、成熟的技术，千万不要无视项目的实际情况而选用一些虽然先进但并非项目所必须且自己又不熟悉的技术。如果项目所要求的技术项目成员不具备或掌握不够，则需要重点关注该风险因素。重大的技术风险包括：软件结构体系存在问题，使完成的软件产品未能实现项目预定目标;项目实施过程中才用全新技术，由于技术本身存在缺陷或对技术的在掌握不够深入，造成开发出的产品性能以及质量低劣。

预防这种风险的办法一般是经常和用户交流工作成果、品牌管理采用符合要求的开发流程、认真组织对产出物的检查和评审、计划和组织严格的独立测试等。软件质量的保证体系是软件开发成为可控制过程的基础，也是开发商和用户进行交流的基础和依据。所以制定卓有成效的软件质量监督体系，是任何软件开发组织必不可少的。

5.3.2 进度风险

软件的工期常常是制约软件项目的主要因素。软件项目工期估算是软件项目初期最困难的工作之一。很多情况下，软件用户对软件的需求是出于实际情况的压力，希望项目承担方尽快开发出软件来。在软件招标时，开发方为了尽可能争取到项目，对项目的进度承诺出已远远超出实际能做到的项目进度，使项目在开始时就存在严重的时间问题。软件开发组织在工期的压力下，往往放弃文档的编写与更新，结果在软件项目的晚期大量需要通过文档进行协调时，却拖累软件进度越来越慢。此外，由于用户配合问题、资源调配等问题也可能使软件项目不能在预定的时间内完成任务。软件项目过程中有自身的客观规律性，用户对软件项目的进度要求不能与软件开发过程的时间需要相矛盾。

对于这种风险解决方案一般是分阶段交付产品、增加项目监控的频度和力度、多运用可行的办法保证工作质量避免返工。在项目实施的时间进度管理上，需要充分考虑各种潜

在因素，适当留有余地;任务分解要详细，便于考核;在执行过程中，应该强调项目按照进度执行的重要项，再考虑任何问题时，都要经保持进度作为先决条件;同时，合理利用赶工期及快速跟进等方法，充分利用资源。应该避免：某方面的人员没有到位，或者在多个项目的情况下某方面的人员中途被抽到其他项目，或身兼多个项目，或在别的项目中无法抽身投入本项目。为系统测试安排足够的时间，能使项目进度在改变之初就被发现，这对及时调整项目进度至关重要。在计划制定时就要确定项目总进度目标与分进度目标;在项目进展的全过程中，进行计划进度与实际进度的比较，及时发现偏离，及时采取措施纠正或者预防，协调项目参与人员之间的进度关系。

5.3.3 质量风险

任何软件项目实施过程中缺乏质量标准，或者忽略软件质量监督环节都将对软件的开发构成巨大的风险。有些项目，用户对软件质量有很高的要求，如果项目组成员同类型项目的开发经验不足，则需要密切关注项目的质量风险。矫正质量低下的不可接受的产品，需要比预期更多的测试、设计和实现工作。

预防这种风险的办法一般是经常和用户交流工作成果、品牌管理采用符合要求的开发流程、认真组织对产出物的检查和评审、计划和组织严格的独立测试等。软件质量的保证体系是软件开发成为可控制过程的基础，也是开发商和用户进行交流的基础和依据。所以制定卓有成效的软件质量监督体系，是任何软件开发组织必不可少的。