西安交通大學

硕士学位论文

校园个人支付平台的设计与实现

申请人：贺欢

学科领域：软件工程

指导教师：邸德海教授

2012 年 3 月

The Design and Implemention of the Campus Personal Payment Platform

A thesis submitted to
Xi'an Jiaotong University
in partial fulfillment of the requirement
for the degree of
Master of Software Engineering

By
Huan He
Software Engineering
Supervisor: Prof. Dehai Di
March 2012

论文题目：校园个人支付平台的设计与实现
学科领域：软件工程
申 请 人 ：贺欢
指导教师：邸德海教授

摘 要

随着信息化技术的迅猛发展，特别是近几年来数字校园的发展，教师学生对在校期间数字化支付的需求也越来越高，这一切都对现有的业务服务提出了新的需求。目前的业务流程不但对工作人员有很大压力，也给学校财务部门带来了大量的额外工作。使用统一的支付平台和相关技术可以解决业务过程中的多个问题，所以需要根据现实情况提出解决方案并设计实现。

本文首先分析了现有业务与业务系统的特性，明确了平台需要满足业务在支付方面的需求。通过采用 UML 的用例图、时序图、活动图等方法对业务过程进行需求分析，根据这些分析，并结合 SOA 的思想提出了构建一个基于服务的架构的平台架构设计。通过对架构模型设计的不断细化，得到了多层的平台逻辑架构与数据架构。随后根据逻辑架构设计了实现平台框架的核心服务与核心组件，以及业务辅助服务的接口。结合 WebService 技术以及各种动态语言技术，实现了所设计的各项系统功能和服务，并详述了搭建整个平台的软硬件环境的过程。最后说明了基于该平台实现的缴费应用与支付平台的测试过程，并结合已经上线运行的支付业务，列举了基于该平台所实现的各项业务在实际中的运行情况。

该平台已经应用到校内多个业务的支付活动中，将这些支付过程统一在一起，简化了业务部门的业务流程，并对财务管理提供了支持，确保所有交易数据的准确性。实践证明，该平台不仅解决了以往存在的支付能力不足、难以统计查询等问题，而且具有新应用开发容易、新业务扩展方便、系统易于维护等特点，能够满足校内各项业务对于支付的需求。平台实现了可管理可扩展的统一支付，在数字校园的建设中发挥了重要的作用。

关 键 词：SOA 架构；Web 服务；动态语言技术；校园卡
论文类型：软件开发

Title: The Design and Implemention of the Campus Personal Payment Platform
Professional Fields: Software Engineering
Applicant: Huan He
Supervisor: Prof. Dehai De

ABSTRACT

With the rapid development of information technology, especially the development of the digital campus in recent years, the demand of teachers and students for digital payment management during the school is highly increasing, which bring new demands for the existing campus financial platform. Working pressure not only to the business process, but also brought a lot of extra work to the school's financial sector. The use of a unified payment platform and related technologies can solve lots of business issue, so that a solution needs to be designed and implement.

Firstly, the characteristics of the existing business and business systems are analyzed to clear the the functional requirements which the platform should meet. By the use case diagram, sequence diagram and active diagram in UML, the business process can be analysised for requirements, according to which, adopted SOA to modeling the system, and designing the whole structure and framework of the platform based on services. After specify the architecure model design in detail, a multi-layer logic architecure and data architure are designed. Then the core services, components, and the interfaces of bussiness assistance service were designed according to logic architecure. Combined with the Web Service and dynamic programming technologies, the system functions and services has been achieved, and detailed the process of building the hardware and software environment of the entire platform. Finally, with the online running projects, explained the test process and operation of the application based on this platform.

The platform has already been applied to the school business payment activities, which have been unified to simplify the business processes of different departments, and the financial management has been supported to ensure the accuracy of the data of all transactions. Practice has proved that the platform not only solved the lack of capacity to pay and difficulty to statistic, but also has characteristics like the facility in development of new applications and system maintenance, convenience in new business expansion, and can meet the demand for payment. The platform achieved controllable, scalable, unified payment, and played an important role in the construction of digital campus.

KEY WORDS: SOA architecture; Web service; Dynamic Programming; Campus card
TYPE OF THESIS: Software Development

目 录

CONTENTS

1 绪论

1.1 研究背景与意义

随着信息化技术的迅猛发展，特别是近几年来数字校园的发展，教师学生对在校期间数字化支付的需求也越来越高，这一切都对现有的业务过程提出了新的需求。同时数字校园平台也完成了多项基本平台的建设工作，使得技术能力和开发工具能够在一定程度上支撑起更加复杂架构的系统设计与实现。

所以为了满足学校老师、学生以及各个相关单位的金融需求，应当设计实现一个校园个人支付服务平台。平台不单面向教师、学生或者各个相关单位的管理人员使用，同时也提供程序接口面向应用系统开放使用，使数字校园的个人信息与支付活动信息能够进行关联，进而解决用户的支付活动信息需求。并且通过平台，可以促进校园内各项业务更好的开展，节省用户的时间与精力，提高工作人员的工作效率，改善服务的整体质量。

通过调研了解到，国内众多高校都已经在校园内的各个部门和场所推行了校园一卡通项目[1]，通过校园卡将校内的餐饮、购物和图书借阅等多种需要进行支付或者身份验证的场合实现了统一支付与统一认证。在这些典型的校园卡应用项目中，采用国内外各家公司的校园卡解决方案都可以实现较好的效果，实现便捷的支付过程和认证过程。但是对于校园内其他突发性或者非日常的消费，如考试报名费和代收费，则存在一定困难。在调研过程中，各个高校采用了不同的方法解决问题。主要包括以下 5 种解决方案：

（1）收取现金。这种方法与以往做法相同，只是利用了校园卡的身份识别功能，在收费时检查校园卡，根据照片和姓名确认身份[2]。这实际上和采用学生证检查身份的方法并不不同。并没有利用到校园卡的支付功能。但在业务高峰期，由于业务工作人员较少，常会发生排长队的现象。

（2）使用校园卡 POS 机收费。这种方法利用了校园卡的支付与身份识别功能，并且避免现金可能引起的假币、误找零等问题。但是当学生校园卡额度不足又不知道余额时，无法改用现金，还需先去充值方可来缴费，如果他人代缴则又无法统计实际的缴费人数。并且 POS 机所需的网络环境通常在安装点并不具备，专门为一个业务点架设专用网络环境成本很高，在学年大部分时间都可能处于闲置状态。与前一种情况同样的是，在业务高峰期，业务工作人员的工作量并未减轻，仍会发生排长队的情况。

（3）由公司开发业务收费服务系统。有高校的校园卡项目完全由公司提供整套解决方案，其中也包括了解决部分业务的临时性收费问题[3]。其思路是利用校园卡平台中的自助服务终端进行业务的缴费工作，这样将收费工作从业务部门的柜台转移到了自

助服务终端的设备上。这种方法很好的解决了排队问题，因为自助服务终端的数量较大，学生无需排队，也可以在自己任意方便的时间进行缴费。但是通常这种方案的成本较高，公司需要派调研人员了解业务需求，安排驻现场开发人员实地开发，并且已知需要有专人在业务运行过程中提供技术支持和维护。而业务过程时常有发生调整的可能，所以开发与维护的成本较高。

（4）联合开发专用缴费系统。有些高校的技术开发能力较强，所以联合公司开发专用的缴费系统进行缴费[4]。通过采用公司提供的开发工具和设备，可以自行定制开发符合本学校需求的专用缴费系统。但是缺点是这些系统只能针对具体业务使用，不具备业务的替换性。

（5）联合第三方支付公司开发网络支付平台。采用业内的一些第三方支付公司的产品，构建一个运行在互联网上的支付平台，实现采用银行卡或者校园卡电子现金的网上支付[5]。这种方案的开发规模较大，需要学校的财务部门在多个方面进行配合方可实现，也需要为第三方支付公司提供一些服务费用[6]。

结合上述的调研分析可以发现，各个学校对于其业务根据信息化规划以及资金、技术水平等考虑，采用了不同的方案解决问题。本校的业务过程与管理体制与其他院校既有类似之处也有不同之处，所以一方面可以借鉴其他高校的建设经验和设计方案，另一方面也需要根据自身的信息化水平与业务特点来设计符合自身需求的支付平台，这样方可在业务需求、技术能力以及资金预算等多个考虑的情况下，解决业务过程中的提出的问题和要求。

1.2 研究内容与方法

通过分析调研，西安交通大学的各项财务相关活动都是通过各种不同的渠道来完成的，交易的方式包括现金、刷卡、校园卡；交易的则可能发生在网上、圈存机上、各业务部门内；票据包括手写小票、校内收据、刷卡凭证、或者不提供票据；有些要在网站上查询，有些要在消费地点查询，有些要去指定部门查询；有些票据会保留，有些不保留；另外费用报销和相关信息的查询也是各有不同的地方提供，用户必须单独记住多种支付方式和业务办理过程以及账单查询的方法，并且办理业务和相关事宜的时间也各不相同，由于办理不及时或者时间不当，可能造成其他业务的办理不顺。根据这些实际情况，在分析和总结国内外校园支付解决方案的基础上，自主设计开发一套符合本校自身特点的个人支付平台，不但可以解决以上的问题，还能够提高信息的准确性和安全性。

所以本文的主要研究内容和开发工作包括以下几个方面：

1）确定用户的需求与平台开发目标

通过对西安交通大学个人金融服务现状的分析研究，对个人支付平台的开发目标和功能需求进行分析。

2）对系统总体框架进行设计

在对平台开发目标和功能需求分析的基础上，完成个人支付平台的总体架构设计。基于业务过程研究平台功能实现中的接口规范，并对业务信息传递安全性、消息通知等关键技术点的实现进行分析，满足用户需求和系统要求。整个系统的框架应具有以下的结构以实现功能，如图 1-1 所示：

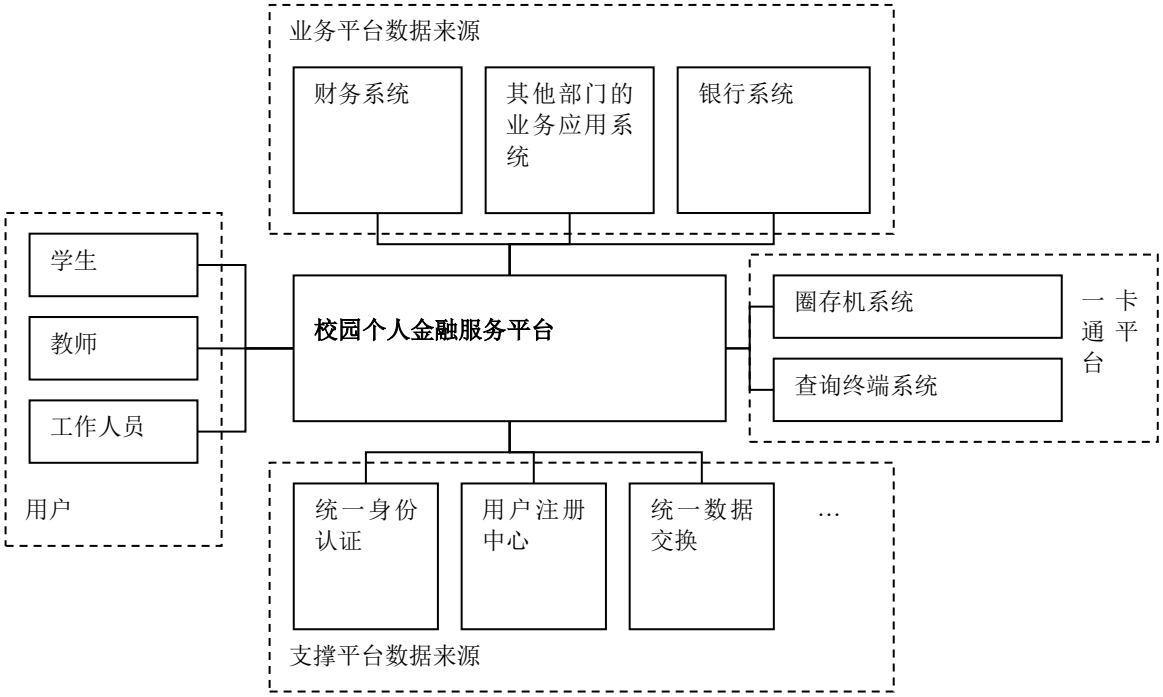


图 1-1 校园个人支付平台与外部系统的关系图

3）设计个人支付平台

从体系结构、系统用例、业务模型、交互设计等方面着手，完整、完善的设计个人支付平台。满足框架结构和用户需求对于系统的设计要求。

4）编码实施实现系统

采用能够支持 SOA 相关技术规范的开发工具和方法，实现前期所设计的系统框架和各项功能，并安装、配置和管理有关的服务器和网络环境，确保所有服务能够正常运行。

1.3 本文结构

本文在大量工作实践和阅读相关文献的基础上，深入研究了实现为已有系统实现支付功能的各种方案，首先分析了现有业务与业务系统的特性，明确了平台需要满足的功能需求；接着提出了一个基于 SOA 思想的平台架构方案实现需求。随后结合这个架构设计，给出了实现平台架构各个组成部分的子系统架构和技术方案，详细分析了各个业务所需要的功能组成，结合已经实现的系统功能和服务，详述了搭建整个平台的软硬件环境的过程。最后结合已经上线运行的项目，列举了基于该平台所实现的应用的运行情况。

本文内容安排如下：

第一章 介绍该平台的背景和必要性，并阐述了目前的现状和研究的内容。

第二章 讲述了 SOA 的概念、技术、应用情况。对平台主要应用的动态语言技术和校园卡技术进行了说明。

第三章 分析目前业务的实际情况，采用 UML 的方法对需求进行建模，通过用例图、时序图、流程图等方法分析业务对于平台的需求，

第四章 将 SOA 思想和技术与需求相结合，设计基于 SOA 思想的平台架构。并设计这个平台架构内各个服务的接口。

第五章 按照支付平台的层次结构，详细介绍了每个层次内的各个服务的实现技术，以及软硬件环境的建设。

第六章说明软件的测试过程，介绍基于支付平台实现的缴费应用系统，并对平台上线运行后的情况进行了介绍。

第七章 对本文整个研究的总结和展望，对全文所述的各项工作进行了回顾、总结。并针对目前开发的现状，对系统提出了进一步的设想和完善方向。

2　相关技术与系统

2.1 SOA 架构与实现技术

SOA，即面向服务的架构（Service-Oriented Architecture），是以可交互服务的形式设计和构建软件的一组原则与方法论[7]。SOA 的原则从理论上提供了实现完整 SOA 的要求，即对服务的要求，如基本的可重复使用、可交互、可识别等。在实施 SOA 的过程中，需要考虑所实现的服务的生命周期、性能以及工程中其他非功能需求方面的限制。在技术上，SOA 通常表现为通过 Web 服务（Web Service）协议所提供的服务[8]。

通过使用 SOA 架构，已经构建的应用服务信息资源可以通过最小的代价实现多次复用，达到融入到新平台、新应用的功效，可以将开发工作中的重复部分得到显著的缩减，并且通过改进重复部分，使所有应用服务的系统能够同等享受到服务的升级。

2.1.1 SOA 的特征与角色

SOA 平台是一个抽象的平台，并没有强制要求用某种指定的技术来实现。实现构建一个 SOA 平台，主要是按照 SOA 的思想来构建的这个平台，使平台具有 SOA 架构的一些基本特征：

（1）低耦合，高复用。SOA 架构具有开发性，架构中的部件可以是完全不同的系统，在地理位置上也可能不在一起。因此在结构上这种架构一定呈现低耦合的特征，这种特征保证了架构的灵活性和可扩展性。根据用户的需求，松耦合的服务可以组合调整为新的系统，或者只需局部替换即可改变整体行为[9]。这样可以显著提高开发的效率和程序的稳定性。这些低耦合的服务在基于 SOA 平台的多个应用系统中使用，并且可以将其重新组合作为新的服务发布，被新的应用所使用。

（2）接口标准开放。用户通过发布 WSDL 实现服务的对外暴露，服务调用者通过 UDDI 可以查询到服务，或者直接访问 WSDL 所在的服务地址[10]。所有的服务交互过程都遵循 SOAP 协议，这样服务之间可以协作，服务使用者也可以简单的访问所有服务无需增加新的协议开销。

（3）支持多种消息格式与模式。不用的服务提供者自身按照其规则和格式进行工作，SOA 平台能够支持这些格式与模式，并且将这些独立的格式与模式封装在所提供的服务之中，而不将这些复杂性暴露给服务调用者[11]。

在 SOA 中，通常包括三种角色，即服务的提供者，服务的使用者，以及服务的注册中心，SOA 中的每个实体都扮演着服务提供者、使用者和注册中心这三种角色中的某一种（或多种）[8]。这三者之间的关系可以用图 2-1 来表示：

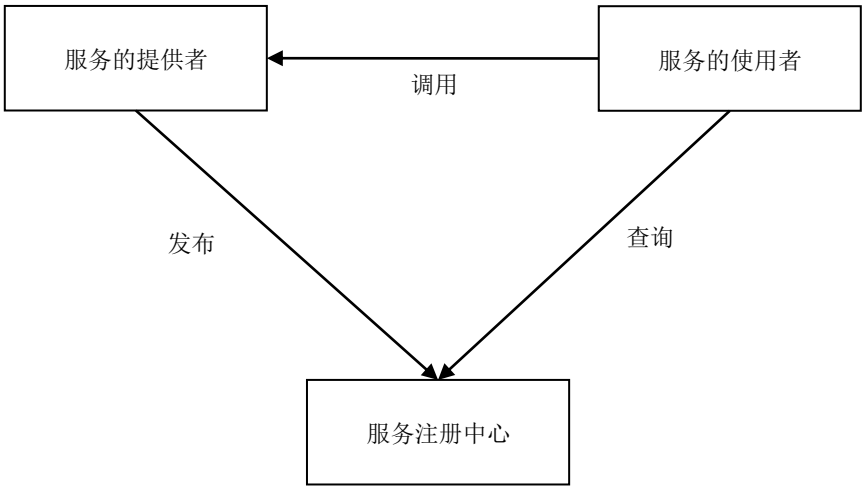


图 2-1 SOA 架构中三种角色的关系

如图 2-1 所示，服务的提供者将所提供的服务发布在服务注册中心中。服务的使用者在服务注册中心查询服务，并调用服务的提供者所提供的服务。服务注册中心为这两者提供发布和查询的服务。每个角色的主要功能如下：

（1）服务提供者：服务提供者是一个可通过网络寻址的实体，它接受和执行来自使用者的请求。它将自己的服务和接口契约发布到服务注册中心，以便服务使用者可以发现和访问该服务。

（2）服务使用者：服务使用者是一个应用程序、一个软件模块或需要一个服务的另一个服务。它发起对注册中心中的服务的查询，并可以根据查询结果调用服务的提供者所提供的服务，并且执行服务功能。服务使用者根据接口契约来执行服务。

（3）服务注册中心：服务注册中心是服务发现的支持者。它包含一个可用服务的存储库，并允许感兴趣的服务使用者查找服务提供者接口。

2.1.2 SOA 的常用技术实现

基于 SOA 的原则，软件系统的开发可以通过构建服务来实现，通常的做法是采用 Web 服务。目前业界有多种方式实现 Web 服务的具体协议，如 SOAP（Simple Object Access Protocol，简单对象访问协议）、也有一些架构风格，如 RESTful（Representational State Transfer）[12]。SOAP 协议是通过 http 或者 https 协议，以 XML 格式进行数据交换的一种 Web 服务实现方式。由于 SOAP 只是实现数据交换的一种通讯协议，不依赖任何平台、硬件，所以各个编程语言都有各自的工具来实现 SOAP 协议的服务端与客户端。在 Java 编程语言中，由 Apache 基金会提供支持的 Apache CXF 是一个开源的服务框架，可以实现 SOAP 协议。.Net 平台也提供了实现 SOAP 协议的组件，使用 C#语言可以简单的实现基于 SOAP 协议的 Web 服务。

SOA 的思想可以采用了一系列的具体协议来实现，基于 SOAP 的 Web 服务包括了以下的协议：

（1）UDDI。UDDI 是 Universal Description, Discovery, and Integration 的缩写，通

常表示为服务查找协议，用以在网络上查找所需要的 Web 服务[13]。

（2）WSDL。WSDL 是 Web Services Description Language 的缩写，是 Web 服务描述语言，是为描述 Web 服务而发布的一种 XML 格式的文档。

（3）SOAP。SOAP 是 Simple Object Access Protocol，简单对象访问协议，是一种基于 XML 的用于交换结构化信息的协议。

（4）XML。XML 是 extensible markup language 的缩写，是用于标记电子文件使其具有结构性的标记语言。

（5）HTTP，HTTPS。HTTP 是 hypertext transport protocol 的缩写，HTTPS 是 Hypertext Transfer Protocol over Secure Socket Layer 的缩写，是浏览器与服务器之间通信的数据传输格式。

首先，Web 服务发布的过程中会使用 WSDL 描述各个服务的接口信息，其次，服务查找过程中会使用 UDDI 来让服务请求者查找、发现服务。然后，服务请求者和服务提供者之间的使用 SOAP 协议进行通信。以上的过程都是基于 XML 进行数据描述，底层采用 HTTP 或者 HTTPS 进行传输。

2.1.3 SOA 的安全

在 SOA 中，服务之间的松耦合特性带来了系统开发的灵活性与敏捷性，但是同时这也给服务安全带来了挑战[14]。在 SOA 架构中，需要建立一个容易控制管理、高度重用又灵活的应用安全框架。按照 IBM 对 SOA 安全所提出的模型[15]，一个 SOA 架构下的系统，应当自下而上提供多重安全保障，其部分结构如图 2-2 所示：

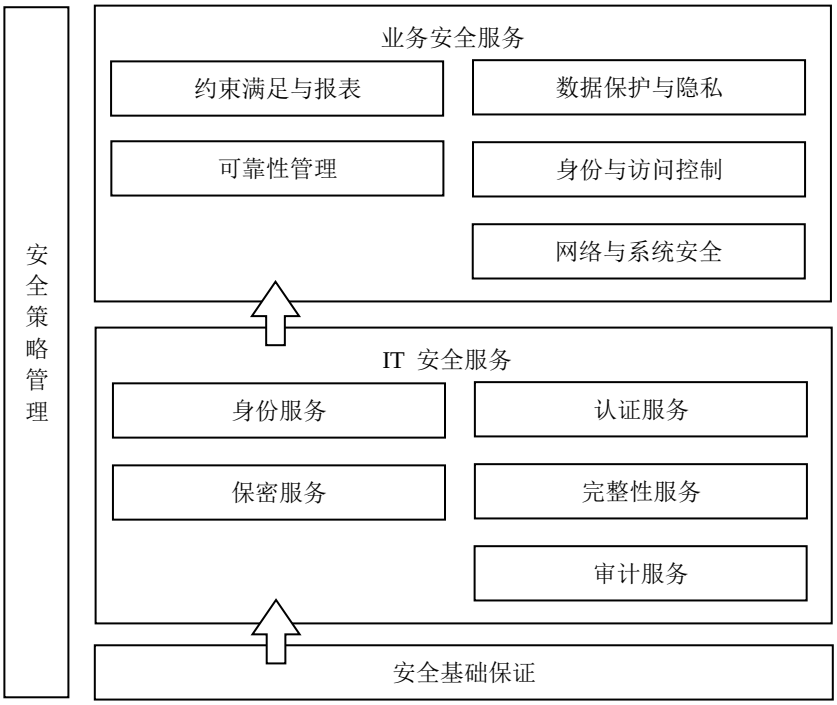


图 2-2　IBM 的 SOA 安全模型部分结构图

如图 2-2 所示，安全模型的基础是安全基础保证层，包括了加解密算法、目录技

术和密匙机制等支持 IT 安全服务完成业务功能的各种技术。在此之上是 IT 安全服务，作为 SOA 基础架构的组成部分，这个模型中把安全本身作为服务提供给应用服务，即是说安全也是服务的一部分，包括实现身份认证、访问授权、保密性、完整性和审计等服务。IT 安全服务支撑着业务安全服务，在业务安全服务中，需要管理对约束满足、报表、身份认证、访问管理、数据保护、隐私、可靠性管理、以及系统安全和网络安全的要求。

而安全策略管理是 SOA 安全的核心部分之一。在 SOA 架构和各个服务的支撑下，策略驱动的服务安全解决方案可以使业务需求产生出来的目标在操作层面上能够得到很好的贯彻和执行。在策略驱动的服务管理中，策略被分为三个层面：

（1）业务层面，把策略描述为业务某种形式的说明。

（2）架构层面，把策略明确为架构风格和开发模式。

（3）实施层面，把策略细化为各种配置条目。

这三个层面自上而下逐渐细化，根据这种方式，可以提出多种 SOA 的安全策略，例如以下的安全策略：

（1）消息安全策略，消息内容的加密和签名，使得消息内容不能被偷窥和篡改。

（2）访问控制策略，细粒度的访问控制，使得能够灵活控制敏感信息不会泄露。

（3）可靠消息递送策略，消息不会由于传递路径中某些节点的故障而丢失，能够保证消息传递到目的地。

（4）服务生命周期策略，为服务在生命周期转换的时候指定一个或多个验证器，以保证服务满足某些规程。

（5）WS-I 符合策略，WSDL 都满足 WS-I 规范，使得 J2EE 平台的 WEB 服务能够和.Net 平台的 WEB 服务互操作。

（6）原创认证策略，应用的代码都要原创，不能抄袭任何拥有知识产权的系统的代码，以免带来法律上的风险。

根据自身业务的需求和系统的规模，可以有选择的根据 IBM 的安全模型选择适合自身系统的安全架构和策略。例如可以采用多级的安全策略结合，结合 XML 加密解密还有客户端的安全技术以及代理网关，再控制服务粒度，都可实现 SOA 的安全[16]。

2.2 动态语言技术

动态语言是一类高级编程语言，通常是指在运行期执行代码的编程语言[17]。可以在运行时进行添加代码或者修改类型定义等操作。

2.2.1 语言特征

通常动态语言各具特点，每种语言都有一些与众不同的特征，这些特征使得这些语言在实现某些功能的时候具有相当的便捷。动态语言与静态语言不像面向过程和面向对象编程语言那样区别明显，有些动态语言起初也是面向过程的，但是在不断的发展过程加入对面向对象的支持，也有些动态语言从设计之初就是按照完全的面向对象

思想进行设计的。以下是动态语言的一些特点：

（1）运行期的对象修改。在动态语言中，一个对象或者类型能够在运行期能够被动态的修改。即在运行期，新对象可以随时生成或者组合现有的对象，甚至修改已有对象内部的结构关系，调整对象方法函数等等各个组成。

（2）函数式编程。函数式编程是种编程典范，它将电脑运算视为函数的计算。函数编程语言最重要的基础是 λ 演算（lambda calculus）。而且 λ 演算的函数可以接受函数当作输入（参数）和输出（返回值）。和指令式编程相比，函数式编程强调函数的计算比指令的执行重要。和过程化编程相比，函数式编程里，函数的计算可随时调用。

（3）闭包。闭包是可以包含自由（未绑定到特定对象）变量的代码块；这些变量不是在这个代码块内或者任何全局上下文中定义的，而是在定义代码块的环境中定义。“闭包”一词来源于以下两者的结合：要执行的代码块（由于自由变量被包含在代码块中，这些自由变量以及它们引用的对象没有被释放）和为自由变量提供绑定的计算环境（作用域）。

2.2.2 实际开发中的应用

通常的企业应用或者大规模数据应用都是采用.Net 或者 J2EE 这样的静态编程语言实现，开发过程必须按照编码、编译、本地测试、服务器测试等过程进行。当需求发生变化时，必须在开发环境中定位、开发、编译、测试，最终上传到服务器上。整个过程牵扯到多个环境的切换和开发工具平台的管理，对于小型开发团队甚至单人开发团队而言，是非常沉重的开发方法，会为整个开发过程带来大量与开发无关的工作量。

动态语言技术是与静态编程语言技术相对的编程语言和相关技术，在早期就有 shell 编程和批处理技术等雏形，主要功能是作为对日常事务与系统维护方面的使用，随着用户更进一步的要求，出现了更多解释型的编程语言，代码更具可读性，通用性也更好，除了胜任日常事务和系统维护，也可以用于复杂应用系统的开发。在各个领域都有动态语言的身影，如科学计算、复杂信息系统、网站、游戏等开发工作，随着解释器以及设备性能的提高，动态语言与静态语言之间的速度差别不再成为应用系统的主要瓶颈。

PHP 语言在网络应用编程方面具有很广泛的应用，经过大量的实践案例证明，其在高并发高负载高复杂度的功能实现上具有良好的性能表现，同时在开发方面也对开发人员非常友好，语法简洁容易构建应用。在 5.0 版本以后，增加了对面向对象特性的支持，使开发的应用可以有更多的实现方法和手段。如 Yahoo，Facebook，WordPress，Discuz!，Drupal 等知名网站和应用系统均采用 PHP 作为实现语言，同时有大量的开源项目和社区可供开发者学习。

Python 语言在多个领域都具有非常广泛的应用，如网络应用系统方面，具有 Twisted 框架可以实现高性能高并发的 Web 服务器，具有 NumPy 等工具实现科学计算，也有 ctypes 等工具库实现对底层库的调用，也支持使用 C 语言实现特殊功能进行扩展[18]。

通过使用动态语言，可以将开发过程明显缩短，代码长度相对于传统应用明显缩

短，只需不到一半的代码量即可实现原有的复杂功能，无论作为原型展示还是具体的应用系统都是非常合适的。

2.3 本章小结

本章就校园个人支付平台所采用的基本技术进行了说明，包括 SOA 思想、特征和相关的实现技术、如基于 SOAP 的 Web Service 的 Java 实现和.Net 实现以及 SOA 的安全方面的内容。随后介绍了动态编程语言的特征和在实际开发中的应用。

3　支付平台需求分析

3.1 业务背景

3.1.1 技术平台

目前学校的信息化经过多年的建设，已经形成了良好的平台支持，包括数字校园平台和一卡通平台。同时各个职能部门也建立了自己的信息管理系统，在近几年都实现了基础数据的数字化，业务办理过程的信息化，这为建立一个校园个人金融平台提供了良好的基础支持。

数字校园平台是西安交通大学校园信息化的基础平台，是实现校园信息多层次多角度多功能可以被复用、实现各种业务应用系统需求、满足师生工作学习中需求解决各种问题的最基本的核心支撑平台。这个平台的主体是用户注册中心、统一身份认证和统一数据交换，如图 3-1 所示：

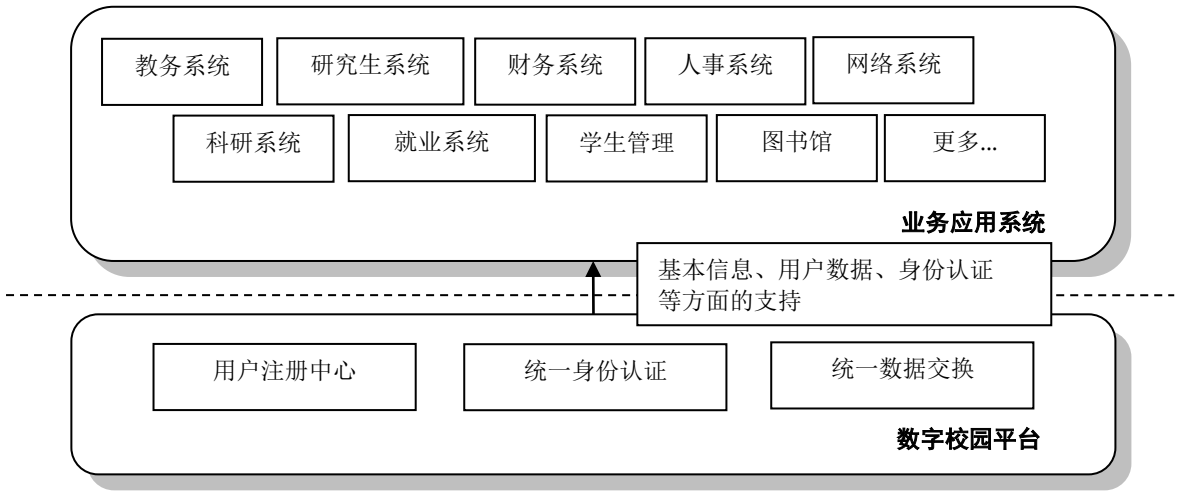


图 3-1　数字校园平台与支撑的业务应用系统

用户注册中心担负着数字校园统一用户管理的职责，在用户中心注册的用户可以通过统一身份认证在不同的应用中漫游。使用统一身份认证，可以在校内信息门户、学生报到注册管理系统、数字迎新系统、研究生系统、教务系统、OA 系统、书院管理、邮件系统以及多个部门的门户网站系统实现身份认证和单点登录。统一数据交换打破了校内信息孤岛，使各种信息在校园内流动起来。目前连接了，教务、研究生、人事、学生管理、财务、一卡通、网络综合业务、科研、OA、图书馆等 10 几个学校的核心业务系统数据库，实现基础数据在全校的共享。

一卡通平台的主要载体是校园卡。校园卡采用了基于 MIFAREONE S70 规范的非接触式卡片，相较以往的 IC 卡或者光电卡，具有安全性好、可靠性高、操作简便、适

用于多种应用的特点。校园卡内可以存储持卡人的多项基本信息，如学号、姓名、班级、证件号码等，结合各种专用设备，可以应用于多种场合。

校园卡包括新生卡、学生卡、教工卡、贵宾卡等等多种外观和身份，其中新生报到后统一领取没有照片新生卡并采集照片，等统一制卡完成后换为带有照片的学生卡。有照片以后，常作为鉴别身份的证件与学生证基本等同[19]。

除了校园卡，一卡通平台还提供了圈存机系统。圈存机系统是一种无人值守的自助服务系统，包括圈存机终端和圈存机服务器，其终端具备触摸操作屏幕、校园卡读卡器、银行卡读卡器和热敏打印机等外设[21]。作为校园卡系统的重要支撑，能够实现充值、修改密码、余额查询、流水查询、补助发放、学费转账、相关通知公告发布等等各种功能。圈存机系统的架构如图 3-2 所示：

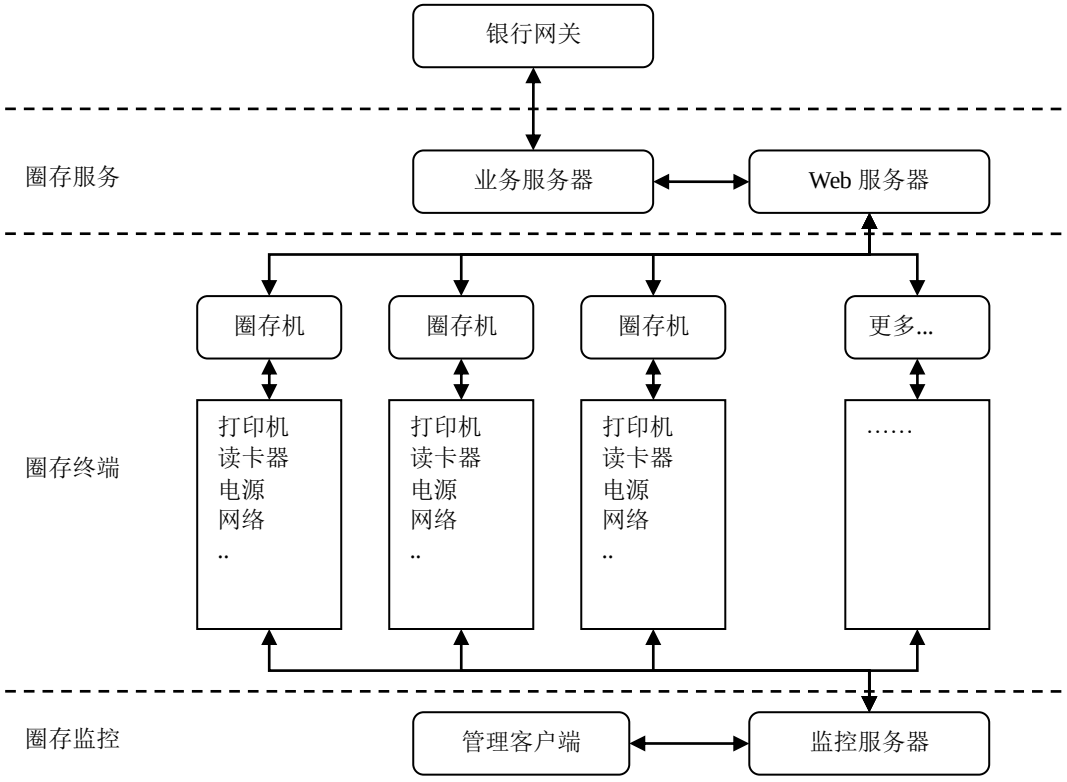


图 3-2　圈存机系统架构图

圈存机采用了 B+C/S 的模式进行架构，对于直接面对用户的功能采用了 B/S 架构，便于功能的快速部署和管理，图 3-2 中，圈存服务和圈存终端所构成的银行转帐、流水查询、密码修改等服务都采用了 B/S 架构实现，方便动态的为用户提供服务，并且一台 Web 服务器可以为多台圈存机提供服务。而对于面向管理人员的监控和管理工具则采用了 C/S 的架构，由圈存监控和圈存终端上的伺服程序配合，实现远程查看圈存机工作状态，查看打印机缺纸，读卡器故障，系统下电，网络异常等等多种设备管理的功能。

校园卡系统除了第一方提供的 POS 解决方案和各种支持系统以外，还提供了 SDK

（Software Development Kit）用以实现第三方开发[3]，SDK 的框架如图 3-3 所示：

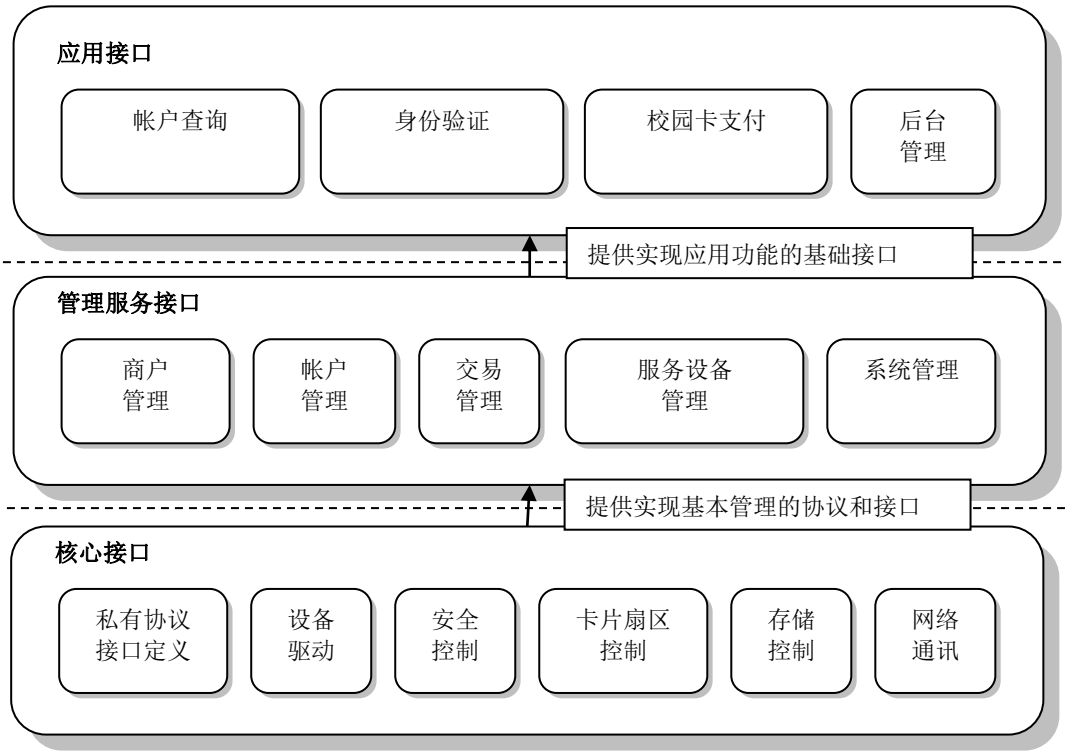


图 3-3　校园卡 SDK 框架结构图

从上图的 SDK 接口可知，校园卡应用开发的 SDK 主要包括三个层次，自下而上分别是核心接口、管理服务接口和应用接口。核心接口是校园卡系统的底层接口，包括实现交易和通讯的所有协议的定义、专用硬件设备驱动、安全芯片与授权机制的管理、校园卡扇区字段设计与读写规则、数据字典与数据库模型以及对应的逻辑关系等等底层实现接口。

管理服务接口是第一方开发应用所实现的各种子系统的接口与实现，包括了校园卡所有相关联的实体对象的管理，如商户、帐户、卡、交易、设备等等，通过管理服务接口都可以实现查询、调用、配置、交互等各种操作。核心接口和管理服务接口都是面向第一方开发使用的，从公开的接口中无法直接调用。对第三方开放的接口是最上层的应用接口，包括帐户查询中的基本信息查询接口、身份验证接口、校园卡支付中的联机支付接口，以及对于后台管理的查询部分接口。基于应用接口，可以实现对支付所需要的基本功能，实现校园内常见业务的支付功能。其中校园卡支付是最主要的功能，能够实现校园卡的读写和一卡通数据库的更新操作。

3.1.2 复杂的业务和支付过程

西安交通大学的各项财务金融相关活动都是通过各种不同的渠道来完成的，根据业务的不同，通常在办理业务的时候进行支付。图 3-1 所示的各个业务应用系统中，大多包含了需要进行支付的业务，如图 3-4 所示：

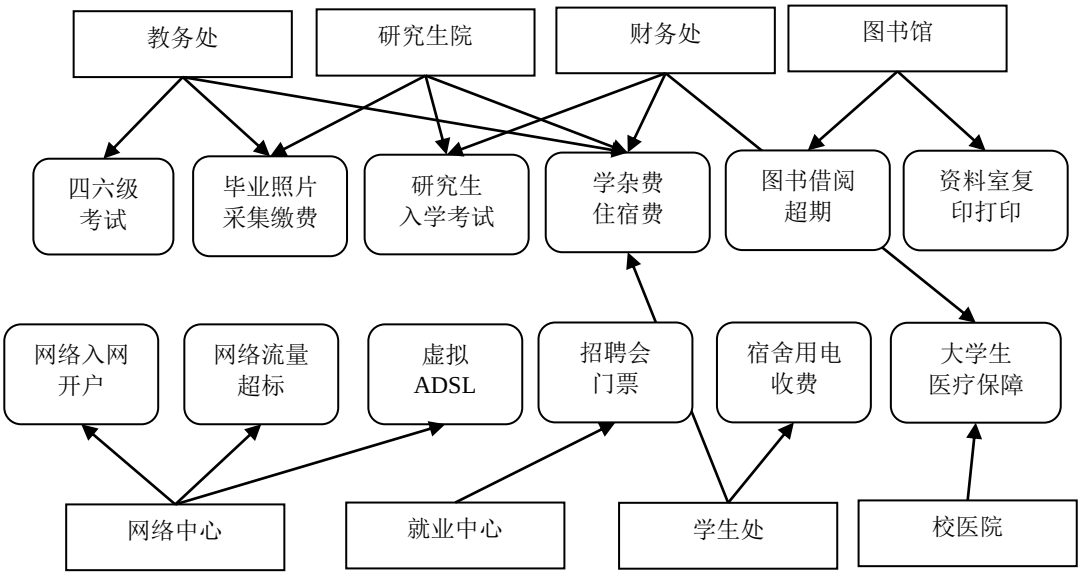


图 3-4　各部门涉及收费方面的业务

在图 3-4 所列举的部门收费项目中，有些是需要使用现金的，如教务处的四六级考试报名，还有与研究生院共同开展的毕业生照片采集支付。有些是面向非学生的，如就业中心的招聘会门票不仅是现金收费并且面向外校学生，研究生入学考试也涉及到校外人员，并且该业务还需要财务处的配合；有些业务需要多个部门协作配合，如学杂费、住宿费需要财务处、学生处、教务处、研究生院共同配合，校医院的大学生医疗保险也需要和财务处配合。另外还有一些业务收费都有时间限制，如图书馆的图书借阅超期，资料室复印打印，网络中心的网络开户收费、虚拟 ADSL 充值都只在工作日的上班时间办理。

随着数字校园建设的深入，各个业务部门都建立了管理自身业务的应用信息系统，但是业务办理的过程并没有发生变化，过去需要在柜台办理的各项业务，现如今仍需要用户在柜台办理。所以在业务高峰期，用户仍然不得不排队等待。而对于业务的工作人员，系统虽然提供了一定的便利，但是手工的工作量也没有显著减少。

3.2 需求分析

3.2.1 业务与流程分析

通过上述的背景分析可以发现，各个业务部门都对支付有需求，并且支付的过程可能需要其他部门的配合，如图 3-4 中所示的学杂费、住宿费收取，就需要财务处、教务处、学生处还有研究生院在数据和业务流程上，通过各自的业务系统以及工作流协同配合方可完成。目前虽然业务可以工作，但是学生得不到业务完成的通知，只能在晚些时候通过银行提供的流水查询服务查询缴费情况，所以学生对于这种大额交易的通知提醒有需求。类似的，在毕业照片采集缴费的工作中，学生需要手工填写多个表格，不但容易出错而且所采集的数据大部分都在基础数据平台已经具有，无疑造成了学生的重复录入。在网络开户的工作中，学生不但需要手工填写表格，还需要多次

往返于学院和网络中心进行填表盖章才能完成办理，办理人员也需要进行手工查询、开通、开票等重复劳动。

业务的具体过程不同，但是整体而言都应具有一定的规律。采用 UML（Unified Modeling Language，统一建模语言）可以刻画出所有这些业务，以及这些业务的过程，并且通过对这些过程的分析和引入支付系统，可以分析出支付系统需要实现的功能和业务系统的边界。

为了简化分析的难度，本文以最具有代表性的两个业务：毕业照片采集缴费和网络入网开户这两个业务用例进行分析，如图 3-5 所示：

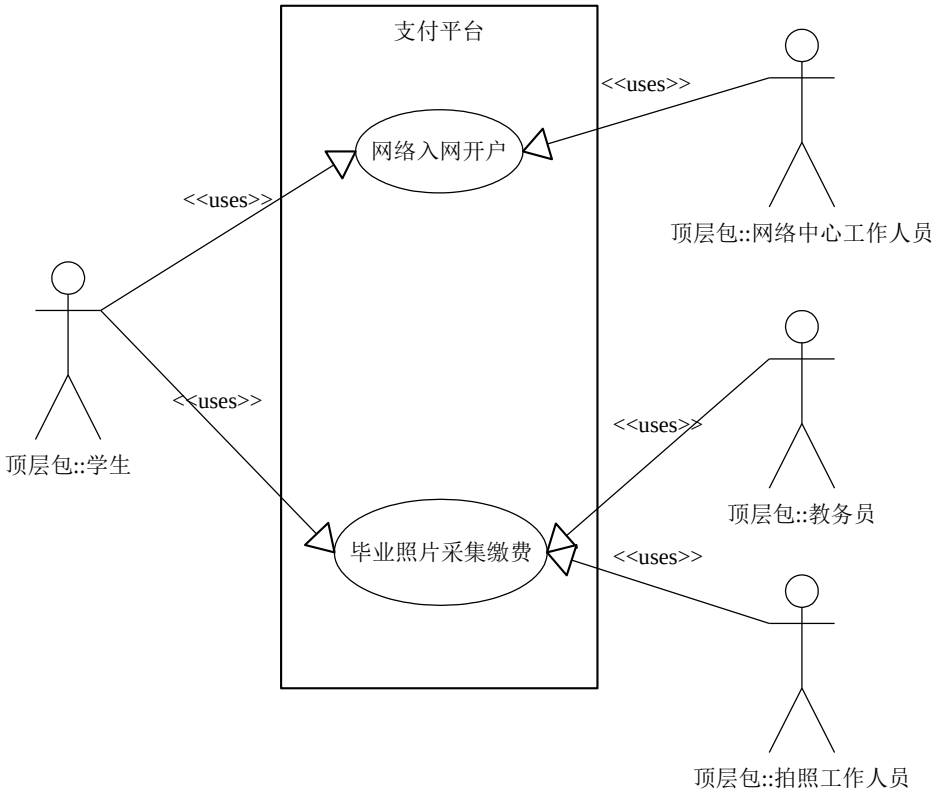


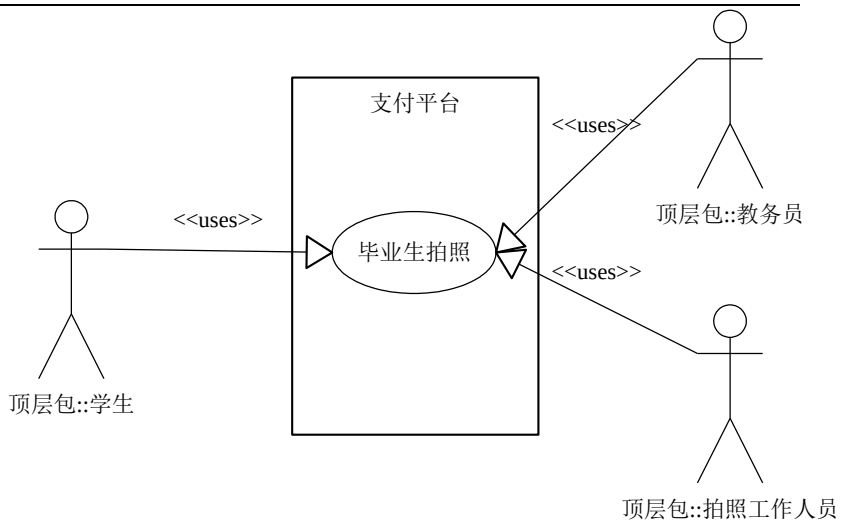
图 3-5　学生常见缴费业务的用例图

在图 3-5 可以看出，网络入网开户这个用例需要学生和网络中心工作人员参与，而毕业照片采集缴费需要学生、教务员和拍照工作人员共同参与。在这个用例图中，无法明显看出支付系统在其中的作用，所以需要进一步的通过 UML 的时序图、活动图等图进行分析，与图 3-4 中的其他业务相似，这些过程都需要有支付方面的支持。

3.2.2 用例分析

图 3-4 和图 3-5 的业务具有共性，如毕业生照片采集支付，教务处和研究生院都需要参与其中，完成收费和管理的工作，而这项业务并没有应用系统可用，只能采用手工的方式进行。该项业务既没有收费系统也没有业务系统，对于图 3-5 中所示的用例毕业生照片采集支付，这个用例的详细信息如表 3-1 所示：

表 3-1 毕业生照片采集支付用例详细说明

用例名称	毕业生照片采集支付
用例图	
用例主流程	1) 学生首先完成报名机读卡申请表的填写 2) 然后将报名费和申请表交给教务员 3) 教务员逐个开具收据给学生 4) 学生给工作人员出示收据，然后进行拍照， 5) 拍照完成后填写排队名单再交给工作人员， 6) 工作人员录入整理排队名单上报上级单位。
替代流程	1a) 学生去年已经拍照，则无需再拍照 4a) 学生没有收据，则回到主要流程 1)

如表 3-1 所示，学生首先完成报名机读卡申请表的填写，然后将报名费和申请表交给教务员，教务员逐个开具收据给学生。在拍照时，学生给工作人员出示收据，然后进行拍照，拍照完成后填写排队名单再交给工作人员，最后工作人员录入整理排队名单上报上级单位。如果学生去年已经拍照，则无需拍照。而在拍照的时候没有收据的话，则需要重新填写报名卡并且缴费。

根据这个用例的过程，可以分析得到该用例的时序图，如图 3-6 所示：

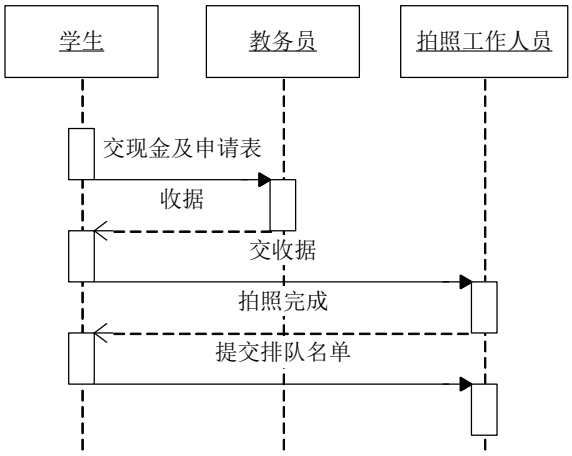


图 3-6 毕业生照片采集支付工作流程时序图

在图 3-6 中，学生首先完成报名机读卡申请表的填写，然后将报名费和申请表交给教务员，教务员逐个开具收据给学生。在拍照时，学生给工作人员出示收据，然后进行拍照，拍照完成后填写排队名单再交给工作人员，最后工作人员录入整理排队名单上报上级单位。整个过程中，学生需要将自己的个人信息和排队手工填写，教务员需要手工收费并记录收费名单，工作人员需要手工录入排队名单并整理。在这个过程中，学生的基本信息本身是存在于数字校园平台数据库中的，手工二次填写无疑是重复工作，并且容易造成错误。收费过程和排队过程也需要手工进行记录，既浪费时间也没有利用起校园卡的身份识别与认证优势。

所以，这个业务需要引入支付管理和对业务本身进行辅助的排队登记等功能，对表 3-1 的用例进行扩展，加入支付系统的支持，该用例的过程可以修改为表 3-2 所示的内容：

表 3-2　使用支付平台的毕业生照片采集支付用例详细说明

用例名称	使用支付平台的毕业生照片采集支付
用例主流程	1) 学生刷卡登记
	2) 学生将报名费交给教务员
	3) 学生刷卡自动排号，然后进行拍照
	4) 工作人员下载排队名单上报上级单位
替代流程	1a) 学生去年已经拍照，则无需再拍照
	3a) 学生刷卡提示没有缴费，则回到主要流程 1)

可见，相对于表 3-1，采用支付平台的情况下，在表 3-2 中明显简化了这个业务的流程以及业务办理过程中相关人员的工作量，并且可以减少不必要的重复录入数据工作和手工录入错误带来的问题，也可以减少拍照工作人员的工作量。根据表 3-2，可以得到如图 3-7 所示的时序图：

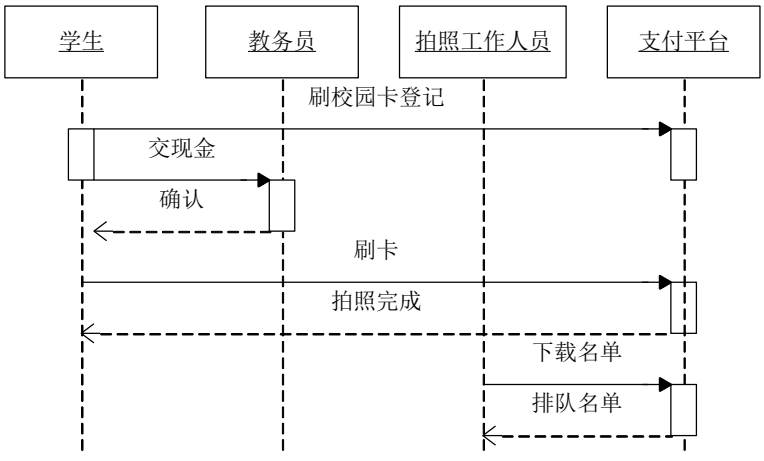


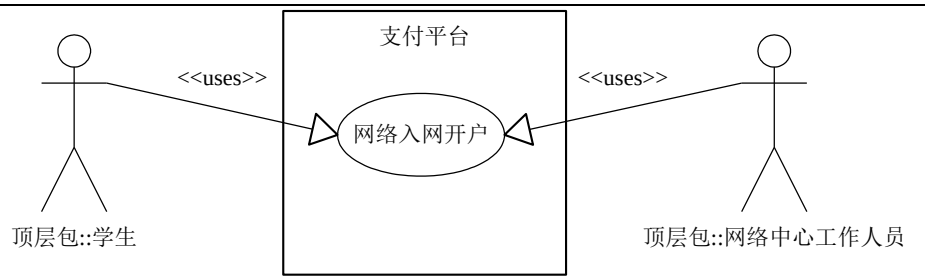
图 3-7　引入支付平台后的毕业生照片采集支付业务流程时序图

从图 3-7 中可以看出，学生原本的手工录入工作都转移给了支付平台，教务员除了收取现金以外也不需要开收据和登记缴费名单，这些工作都交给支付平台完成。拍

照过程也是使用刷卡代替收据和填写排队表，学生不用再填写表，工作人员也不需要录入和整理名单，只需在支付平台下载即可。

对于图 3-5 中所示的网络入网开户用例，这个用例的详细信息如表 3-3 所示：

表 3-3　网络入网开户用例详细说明

用例名称	网络入网开户
用例图	
用例主流程	1）学生领取申请表并填写 2）辅导员签字盖章 3）学生将申请表交给网络中心 4）网络中心工作人员检查申请表、收取开户费用 5）网络中心工作人员在业务系统中录入信息、开通网络，开具发票
替代流程	4a) 尚未签字盖章，返回主流程 2）

如表 3-3 所示的网络入网开户的业务中，学生领取申请表并填写，然后由辅导员签字盖章，学生将申请表交给网络中心，由网络中心工作人员检查申请表、收取开户费用、在业务系统中录入信息、开通网络，再开具发票，具体过程可以用图 3-8 表示：

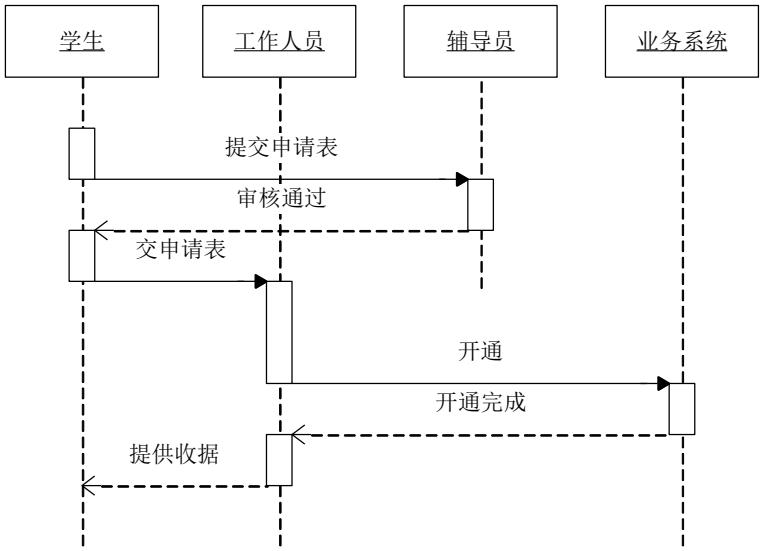


图 3-8　传统网络开户业务的时序图

从图 3-8 可以看出，学生和业务部门工作人员的手工工作占了整个业务的大部分工作量，工作人员需要检查申请表、录入数据、开通网络、手工开票等工作。使用平台实现自助的支付服务，并且使用平台增强业务系统的对用户服务的功能，采用支付

平台后，如表 3-3 所示的用例详细过程就可以变为表 3-4 所示：

表 3-4　使用支付平台的网络入网开户的详细说明

用例名称	使用支付平台的网络入网开户
用例主流程	1）学生在线申请 2）辅导员在线审批 3）学生进行支付，由支付平台通知系统自动开通网络
替代流程	3a) 尚未审批，无法支付，返回主流程 2）

将表 3-4 转化为时序图，可得到图 3-9 所示的时序图：

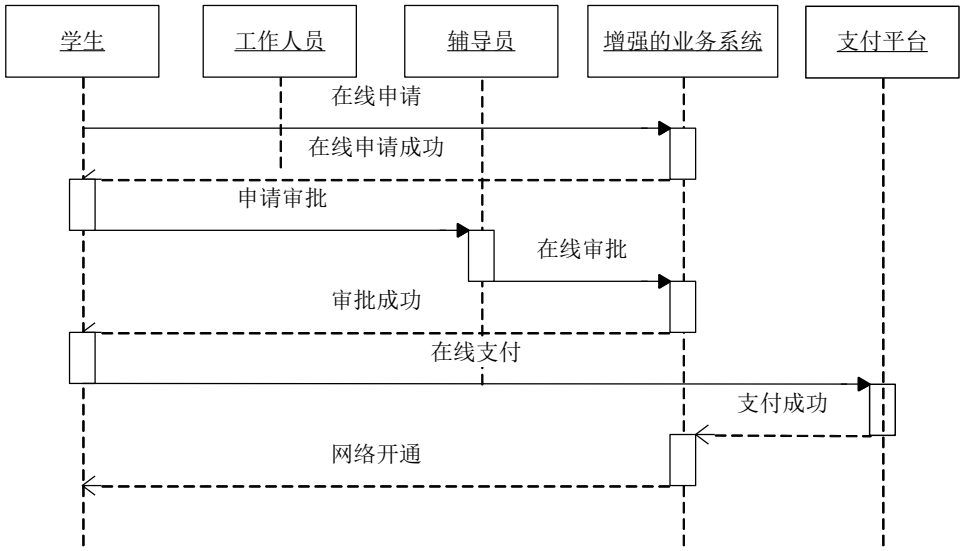


图 3-9　使用平台之后的网络开户业务时序图

可以看出，在平台的支持下，工作人员的工作量全部转移给业务系统和支付平台，其工作量可以明显减轻，同时学生也可以避免以往的排队等待问题。所以通过对业务进行分析，支付系统可以在对业务系统不产生改变，不增加工作人员的工作量的前提下，改进业务流程。

3.2.3 静态结构分析与 SOA 的应用

将图 3-4 中的业务进行归纳，再总结图 3-7 和图 3-9 中支付平台的功能，可以归结出用户对于支付平台的要求包括：

（1）对原本不具备收费功能的业务系统提供数字化的收费功能。

（2）对数字化的收费的内容实现查询、统计、报表等功能。

（3）实现业务过程的改进，办理过程的自动化。

根据这些要求，并结合图 3-7 和图 3-9 的过程可知，支付在整个业务过程中起到重要的作用。在图 3-5 中所示的用例中，学生用户和工作人员用户的所需要的功能包括业务的查询、支付、业务的通知和反馈这四个步骤。这四个步骤可用如图 3-10 所示的活动图说明：

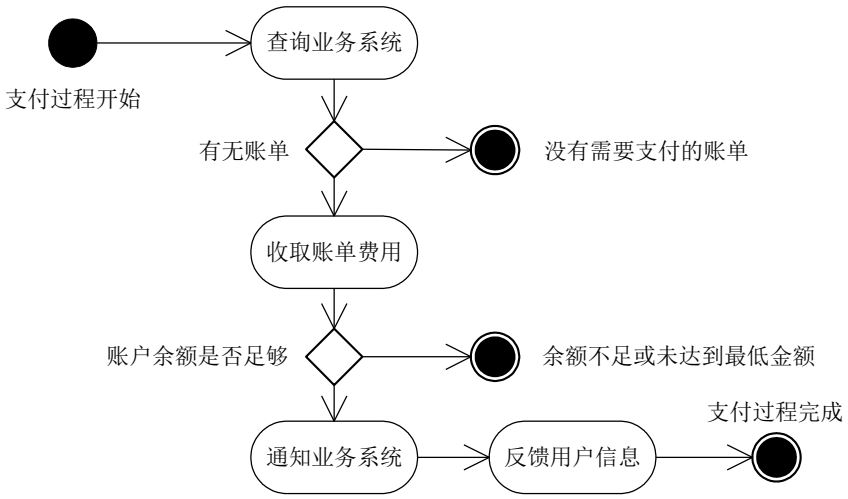


图 3-10　支付活动的抽象活动图

如图 3-10 所示，首先，学生向业务办理者进行查询，判断是否需要进行支付业务。如果无需办理则结束。然后进行支付，按照一个通用的支付过程完成支付。最后，通知业务系统已经完成支付，学生得到完成支付的反馈，整个支付过程完成。业务的支付过程主要包括四个步骤：

（1）业务查询：业务查询即获取学生需要进行支付的账单。

（2）支付收费：支付收费即使用现金或者校园卡等方式进行支付的活动。

（3）业务通知：即在支付完成后告知业务主体支付过程完成。

（4）反馈用户：就支付过程的操作过程和结果反馈给用户。

这四个步骤在图 3-4 和图 3-5 的业务与用例中都是能够体现出来的，如果建立一个支付系统解决具体的业务，则该系统应具有支付、业务、用户、账单等多个基本类，如图 3-11 所示：

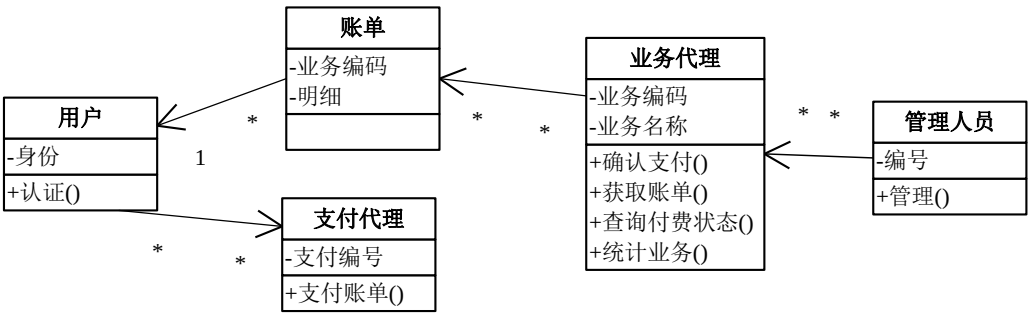


图 3-11　基本支付系统的类图

如图 3-11 所示，作为支付系统，最基本需要具备用户类，提供对用户认证的功能。一个用户可以对应多个账单，同时用户可以在不同的支付代理处进行账单的支付。需要支付代理类，完成支付账单的功能。需要业务代理，由业务代理提供账单，每个账单根据业务的不同对应于不同的业务业代理，业务代理完成在业务系统中确认支付、获取业务账单、查询付费状态、统计业务办理情况等功能。并且业务代理都具有不同的管理员可以对其进行管理操作。

根据图 3-4 所示的业务情况，如果为每一个业务都建立一个独立的系统，则需要建立多个支付系统，如图 3-12 所示：

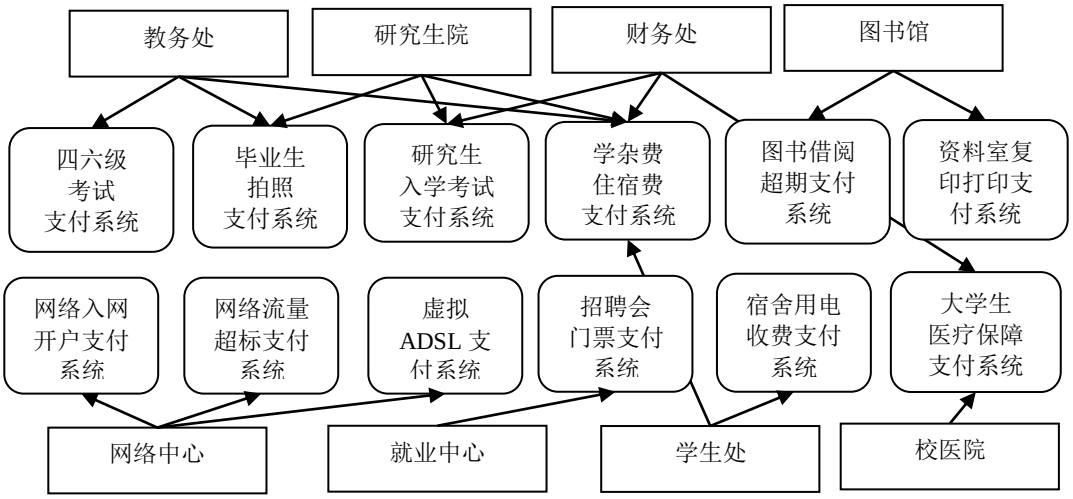


图 3-12　单独建立支付系统的情况

如图 3-12 所示，每个支付系统都根据业务的不同而独立设计。而在图 3-10 的活动图中已经说明所有的支付都具有一定的相似性，都可以分解为一定的相似过程。所以如果开发单独的系统，必然带来代码管理的复杂和应用系统多样的不易管理等问题。

根据这一情况，结合在 2.1 节中讨论的 SOA 技术，可以发现，SOA 思想是适合解决这个问题的重要工具。如 2.1 节所述，通过使用 SOA 架构，可以将开发工作中的重复部分得到显著的缩减。在这个系统的建设中，可以采用 SOA 来实现统一的支付过程，将所有业务的支付都采用服务来进行实现[20]。如图 3-13 所示：

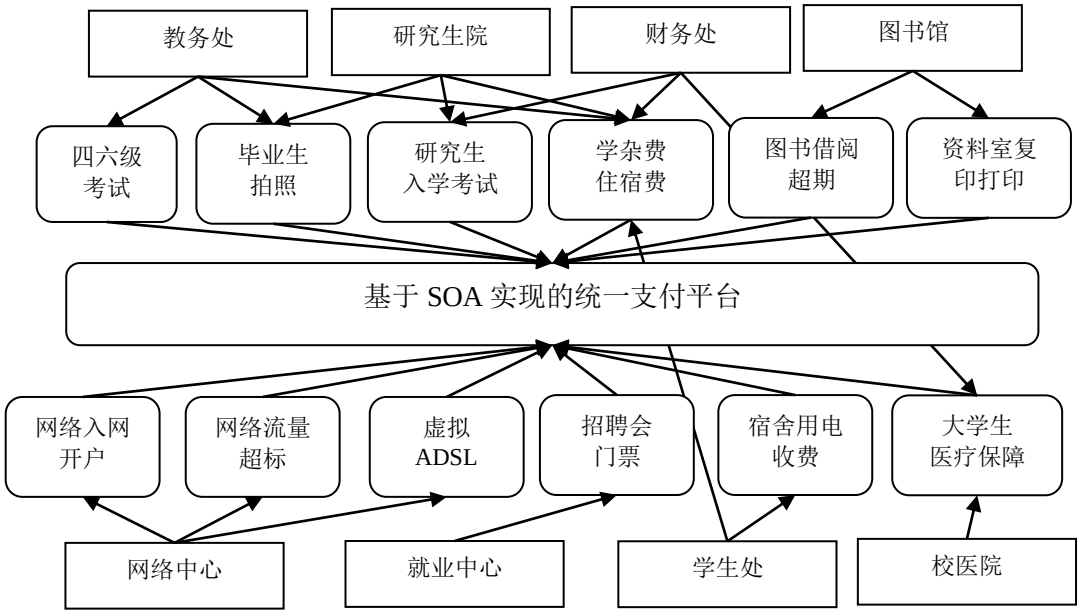


图 3-13　使用统一支付平台解决业务的支付问题

如图 3-13 所示，使用 SOA，可以将图 3-11 所示的类功能都转化为服务，从而将

图 3-12 中的单独的支付系统构成一个统一的支付平台来实现这些业务的支付，各项业务都可以通过基于 SOA 的平台实现支付，基于 SOA 的平台可以将支付作为服务提供给各项业务并且也可以在业务方面提供流程改进的支持。所以系统将对功能按照 SOA 的服务的思想进行构建，实现低耦合、高复用等特点。

3.2.4 非功能需求

对于平台的非功能需求主要包括安全性、可靠性、可维护性和易用性。

交易过程本身是涉及用户多项隐私的过程，如密码、账户金额、账户信息、银行卡信息等等，所以平台的安全性在设计时是首先需要考虑的。

其次，交易过程涉及用户资金的变化，必须保证用户的利益不会受到损害，所以系统必须能够处理各种异常情况而不造成交易过程的失败对用户造成影响。

另外，平台面向用户和开发人员的接口各有不同，但是都必须具有足够的易用性。对于最终用户，系统的操作过程必须简单直接，符合用户心理预期。对于平台应用的开发人员，接口应当简洁容易理解，没有副作用。

最后，平台应当具备一定的可维护性，保证管理人员能够迅速排查发生异常的问题，或者能够通过简单的配置实现功能的调整。

3.3 本章小结

本章首先分析了目前各项业务的背景情况，包括数字校园与一卡通平台的技术背景，以及目前各种业务的实际情况。然后根据这些背景和具体的业务过程，采用 UML 中的用例分析方法，采用用例图、时序图、活动图、类图等手段对业务用例进行分析，进而得到系统的基本组成，并采用 SOA 的思想分析了实现这些业务支付的合适方法，即构建一个基于服务架构的平台结构。最后提出了一些非功能方面的需求。

4　支付平台设计

4.1 支付平台架构设计

根据 3.2.3 节中的分析，用户需求在不同的业务过程中有不同的体现，如果单独建立系统完成支付，则需要修改这些系统的功能和流程方可将支付加入其中，并且仍需要工作人员适应新的系统，并没有减轻工作人员的工作量。用户需要的功能具有一定的通用性，但是具体的需求内容由于其业务办理过程的不同而各有特点，不能采用一个产品或者一个系统解决所有用户的需求。所以需要采用 SOA 思想，设计一个统一的支付平台，提供完整的服务，从软件、硬件的各个层面为业务提供支持[22]。

通过这个平台提供的服务或者技术支持，各个业务的办理都可以获得业务能力的提升，进而带来更好的服务水平和质量。平台必须提供多样的服务形式和多种不同的能力以满足业务的需求，并且随着业务过程的发展，服务应当具有一定的适应能力，否则会让业务过程来适应平台服务甚至最终放弃平台服务。

平台的架构应当从不同的角度进行刻画，方可将其特征展现的更加清楚，各种不同的架构视图。

4.1.1 实现 SOA 的系统服务逻辑架构分析

分析图 3-7 和图 3-9 的业务过程以及图 3-13，可以得到以下支付平台应当提供的服务，在具备这些服务的情况下，支付过程方可实现，同时，服务应当是按照一定的层次结构来划分的[23]，整个支付平台的基本结构如图 4-1 所示：

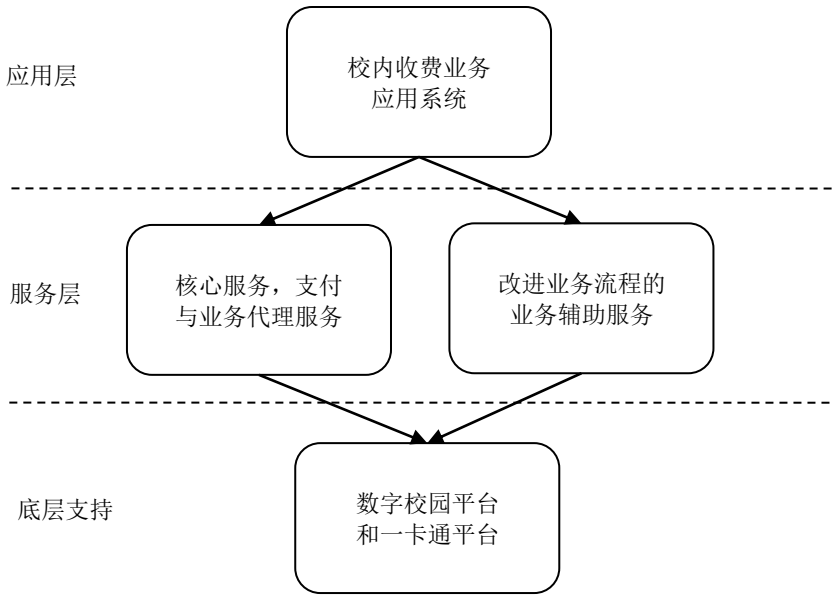


图 4-1　支付平台的基本组成

如图 4-1 所示，平台自上而下分为应用层、服务层以及底层的平台支持。在应用层中，实现了校内收费业务的应用系统。在服务层，通过提供核心服务与业务辅助服务，为应用的实现提供基础。在底层，通过数字校园平台和一卡通平台提供技术和数据的支持。其中，服务层应当包括以下五个服务：

（1）身份识别：识别进行支付活动的学生身份，判断进行交易的主体是否符合要求。将不符合的要求的从支付活动一开始就剔除，避免有异常的数据在系统中进行。图 3-7 中，通过教务员完成了这个服务，但是支付平台刷卡的过程也辅助实现了这个服务。

（2）支付收费：按照业务规则完成学生资金账户和业务账户的交易过程，完成对校园卡的支付和一卡通数据库读写等多个操作。在图 3-7 中由教务员收取现金和学生刷卡共同完成支付，在图 3-9 中是由支付平台完成支付。

（3）业务代理：接手业务的支付过程，处理支付对应的业务过程和规则，完成在业务系统中体现支付完成的操作。在图 3-7 中排队过程即使用了支付平台的业务代理服务查询刷卡的学生是否进行了缴费，在图 3-9 中支付平台的业务代理通知了业务系统所以实现了网络开通。并且根据 3.2.4 节对于安全性的考虑，设计代理可以保护业务系统的网络环境安全，避免暴露过多资源给外界，并且可以较好的隔离不同的业务功能，使每个功能在各自的范围内进行工作[24]。

（4）活动调度：协调完成从其他各个服务的执行，为面向用户的终端提供简化的服务支持工作。在图 3-7 和图 3-9 中都无法直接看到活动调度的存在，但是从实现的角度，所有服务都应当在一个调度下实现，否则无法良好的协同工作。

（5）辅助功能：为业务过程提供必要的功能支持，如名单查询、数据转换等等不需要在支付系统内完成的操作。在图 3-7 中，支付平台提供了名单下载辅助功能，虽然不是支付业务的必须，但是能够确保整个业务更好的完成。在图 3-9 中，支付平台用身份识别和信息查询增强了业务系统的能力，使之提供了在线申请的功能。

这五个主要服务按照功能的重要程度，可以划分为核心服务和辅助服务，据此可以将支付平台划分为如图 4-1 所示核心服务与业务辅助服务。其中，支付收费、活动调度、业务代理是核心服务，身份识别、辅助功能是辅助服务。基于这个划分，可以将支付平台的重心放在核心服务的实现上，根据业务的需求，再实现所需的辅助服务。

如图 3-4 所示，由于多个业务都存在支付需求，并且各个业务系统，平台从架构上需要能够适应异构系统，并且需要满足可复用与便于维护与开发的要求，所以适合采用 SOA 的架构思想进行设计。SOA 思想提倡高复用、低耦合，适合在具有通用性的应用中使用。基于这样的思想，平台将各个功能设计为服务，对外暴露服务接口，同时保留部分提供服务的系统的独立性，使之可以单独运行，为平台整体提供数据或者功能的服务。

将图 4-1 进一步细化，可以到如图 4-2 所示的详细架构设计图：

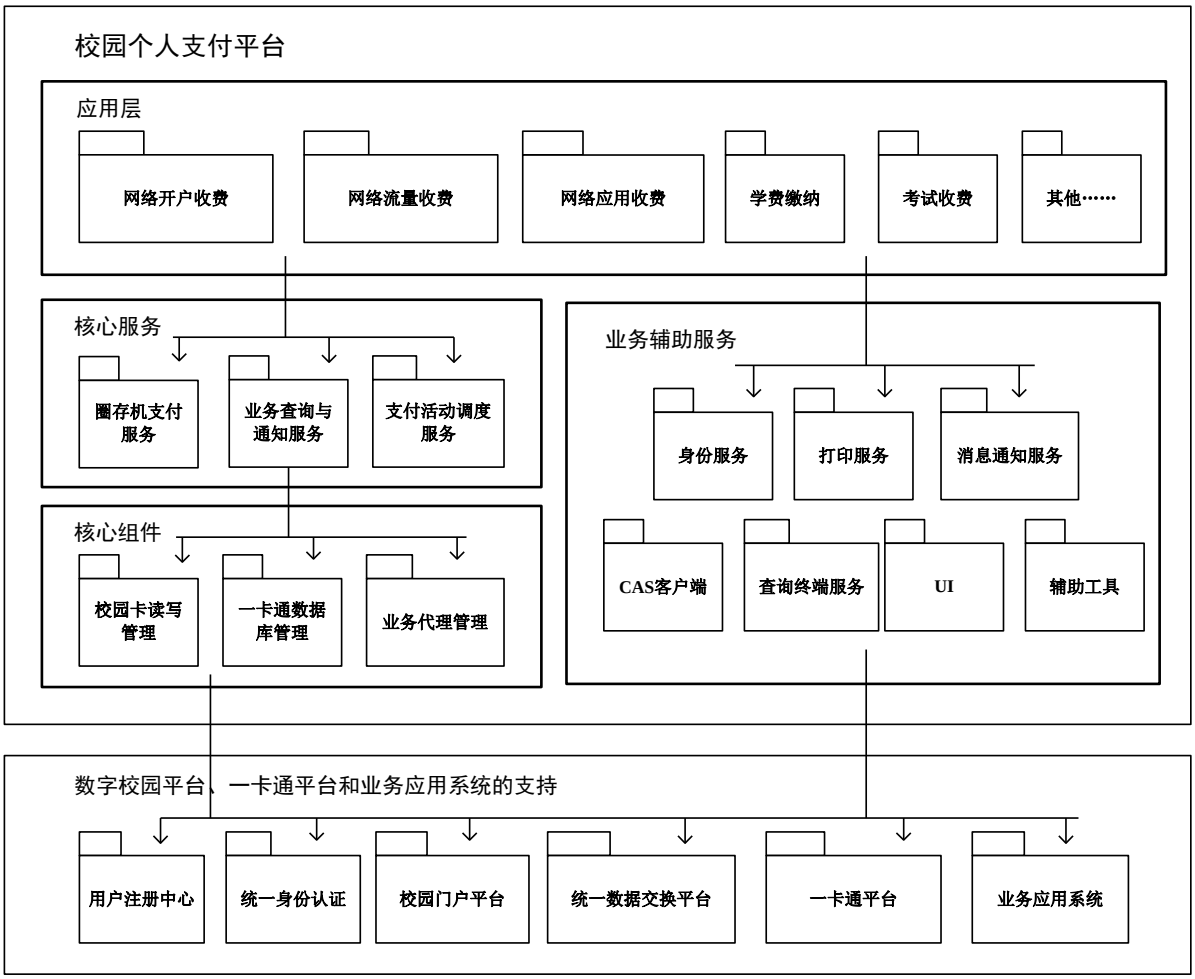


图 4-2　校园个人支付平台的逻辑架构图

图 4-1 中的核心服务粒度较大，不适合用 SOA 直接实现，应当进行更细的划分。首先分为核心服务和核心组件两层，基于组件实现具体的服务。如图 4-2 所示，核心服务包括圈存机支付服务、业务查询与通知服务、支付活动调度服务，核心组件包括校园卡读写管理、一卡通数据库管理、业务代理管理这些组成。根据 3.1.1 节中对技术平台的说明可知，一卡通平台适合实现支付的过程，采用图 3-2 所示的圈存机系统可以很好的实现支付过程的自助化，这样可以将工作人员从收费中解放出来。所以图 4-2 中核心服务的支付收费采用圈存机实现。这样将支付收费服务定义为圈存机支付服务，根据图 3-3 所示的第三方 SDK，可以将圈存机支付服务分为校园卡读写管理和一卡通数据库。这两者都可以作为组件在核心组件层实现。

业务代理服务可以通过图 3-7 和图 3-9 的业务过程分析和图 3-10 业务过程抽象处理过程可以知道，业务代理主要完成业务查询和业务通知的服务，所以图 4-2 中将业务代理服务具体化为业务查询与确认服务，并在核心组件中实现了具体的业务代理服务功能。

如前所述，在图 3-7 和图 3-9 的业务过程中并没有直接看到调度的存在，但是时序图本身即代表了各个活动的调度过程，所以活动调度仍是核心服务的组成部分。并

且根据 3.2.4 节中非功能需求对于安全性的考虑，采用圈存机系统在进行支付时，不能同时连接多个其他网络的业务系统以免造成安全漏洞，所以使用活动调度作为支付服务和业务服务的接口是合适的。

具体而言，核心组件包括校园卡读写管理负责提供对校园卡扇区内容的控制，一卡通数据库管理提供对商户和账户等交易信息的控制，业务代理管理实现对业务系统支付业务的接口实现。

除了上述的核心服务以外，图 4-2 中的辅助服务可以根据图 3-7 和图 3-9 的需求进行扩展，业务辅助服务是平台的重要组成，为具体的业务系统提供一些额外的功能或者数据方面的支持，如图 3-7 业务中所使用的身份识别和票据打印、还有 4.1.1 节分析学费缴纳业务需要的消息通知、CAS 客户端认证、查询终端、UI 元素以及其他各种辅助工具。

业务辅助服务为具体的业务系统提供一些额外的功能或者数据方面的支持，如身份服务提供验证用户身份和有效性的服务、票据打印提供打印符合学校财务要求的票据打印服务、消息通知服务提供短信等手段为用户提供支付的信息，CAS 客户端为使用 CAS 服务的系统提供登录的入口，这些分布式的服务为整个平台可以提供支持[25]。

另外，查询终端服务是根据图 3-9 业务过程中提示用户网络已经开通这个活动而设计的，在支付的过程中，详细的业务信息不便完全显示，所以可以提供一个专用的查询终端实现对业务的查询详细结果的工作。以图 3-9 而言，开户后需要提供 IP 地址、网关、账户名、密码等业务信息，使用查询终端可以完整详细的展示给用户这些信息，并且还可以提供热敏打印的服务，将信息打印在纸上提供给用户。参考图 3-2 的圈存机架构，查询终端服务的架构也采用 B/S 结构。

基于这些核心服务和业务辅助服务，就可以按照图 3-13 实现图 3-4 中的业务收费业务的具体应用，能够实现的业务包括图 3-4 中的网络开户、网络流量、网络应用、毕业生照片采集支付等等多种与支付相关的业务。

从图 4-2 中可以看出，平台是依托数字校园平台、一卡通平台和业务应用系统设计的，所以从整体上，数字校园平台、一卡通平台和业务应用系统是平台的基础，为平台的所有功能提供支持和保证。平台基于数字校园平台的核心系统，即统一身份认证，用户注册中心、统一数据交换和校园门户。其中统一身份认证平台提供了对平台的安全性保障，用户注册中心提供了学校师生的基本信息可以用于授权管理以及部分查询功能的实现，统一数据交换提供了从各个业务部门获取数据的统一渠道和数据支持，校园门户为所有用户提供了展示数据的用户界面。校园一卡通平台作为支付的载体，提供了和支付相关的服务。同时业务应用系统也为平台提供了业务数据的支持。

按照 SOA 的思想，图 4-2 中的核心组件、核心服务、辅助服务中每个功能，都可以封装为独立的 Web 服务，按照标准实现数据通信。

4.1.2 SOA 服务的数据架构

从图 3-7 和图 3-9 可知，平台自身不直接产生业务数据，都是从业务部门获取支

付的账单或者需求，转交给支付服务进行账户的交易。所以平台的数据架构来自于图 4-2 中的核心组件、核心服务和辅助服务中每个服务在工作过程中收到的业务数据或者支付数据或者其他数据。

从 4.1.1 节逻辑架构的分析和图 4-2 的各个服务，以及图 3-7 和图 3-9 的具体业务过程看出，具有数据存储需求的服务包括圈存机支付服务、业务查询与确认服务、活动调度服务、身份服务、消息服务、票据服务。按照 SOA 的思想，各个服务对外只暴露服务接口，所以自身的数据在逻辑上都存储在各自服务的数据库中。这种逻辑上的隔离可以实现较好的可靠性也不会因为单点故障引起其他数据库异常[26]。

数据架构如图 4-3 所示：

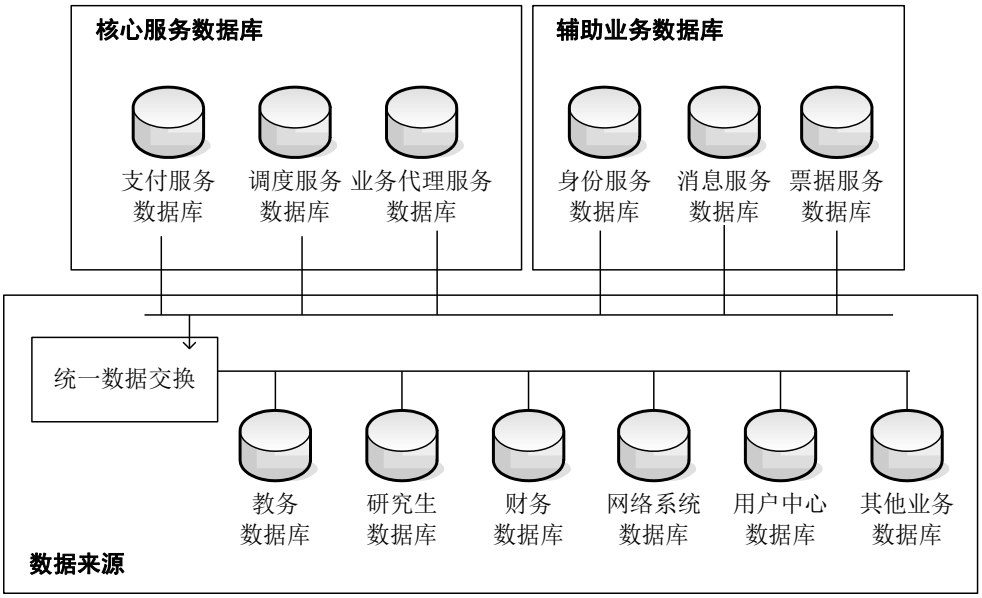


图 4-3　校园个人支付平台的数据架构图

圈存机支付服务、业务查询与确认服务、活动调度服务、身份服务、消息服务、票据服务分别各自具有独立的数据库，各自按照实现过程中的需求实现具体的数据库表和字段。根据需求分析和图 3-7、图 3-9 的业务过程分析，平台的数据架构底层应当是来自数字校园平台中统一数据交换得到的数据。

在图 4-3 中，统一数据交换得到的数据即图 3-1 中数字校园平台所支撑的那些应用系统中的数据，包括教务数据库、研究生数据库、财务数据库、网络系统数据库、用户中心数据库以及其他各种业务系统的数据库。具体而言，教务数据库提供了本科生的个人信息，研究生数据库提供了研究生的个人信息，财务数据库提供了部分缴费的流水记录，网络系统提供了所有网络业务的数据信息，用户中心提供了所有注册用户身份的信息，等等。这些都通过统一数据交换，为平台提供了数据的来源。

4.1.3 支付平台的安全架构

校园个人支付平台作为实现校园支付业务的载体，其安全性是业务的基础保证。在以往的业务办理过程中，如图 3-6 所示的毕业生照片采集缴费过程中，支付业务的

安全是由工作人员来保证的，由工作人员完成现金的验证和保存。但在支付平台上，所有支付的过程都是数字化的，所以必须由支付系统保障支付过程的安全性[27]。

根据 2.1.3 节中对于 SOA 平台的安全考虑和 IBM 的安全模型，支付平台可以选择一些符合自身需求的安全方案和策略来确保业务的安全。按照 IBM 的安全模型，系统的安全包括了多个层次和方面的安全，但是完整实施所有安全模型的话，势必造成大量的资源用在与业务功能不大相关的其他功能的实现上。所以出于开发成本的考虑，平台的设计主要考虑数据的安全和支付过程的网络安全。整体的安全保障技术架构如图 4-4 所示：

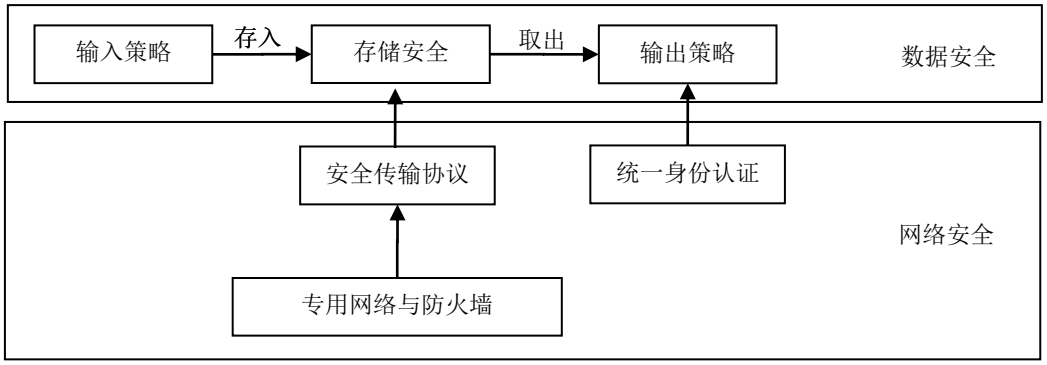


图 4-4　支付平台的安全架构

支付平台的安全主要是针对交易数据的安全，如图 4-4 所示，数据安全主要包括数据从输入到存储再到输出的过程的安全，针对这个流程，采用输入策略和输出策略控制数据的输入输出过程，由存储所在的服务器即一卡通数据库服务器和业务系统的服务自身来保障存数据存储的安全，数据并不保存在支付平台内。所有的数据交易都是在网络环境下完成的，所以必须要对网络环境的安全性做出考虑，通过确保网络环境的安全可以实现对支付平台安全最基本的保证。

在网络安全方面，如图 4-4 中所示，网络安全主要包括三个部分，首先建立了专用的局域网来运行支付平台的所有服务器，该网络与外部教育网或者其他公用网络隔离，并用一个物理防火墙设备以及软件的防火墙共同进行边界的管理，避免了外部通过网络入侵造成数据安全的可能性[28]。在此之上，所有的查询、交易都是基于加密的 https 协议进行通讯，即使在网络内部也不能通过监听网络流量获得交易信息[29]。另外，基于数字校园平台的统一身份认证服务，要求所有的数据查询都是需要通过身份认证方可访问，这样可以避免用户个人的信息被他人所看到[30]。

这网络安全的基础上，数据安全的输入策略主要包括：

（1）账户资金、业务账单和校园卡三者绑定。通常的业务办理过程中，根据现场情况可以由他人代为缴费或者充值。但是在自助服务的情况下，如果允许代缴，则可能发生使用未挂失的遗失卡恶意刷卡代缴的情况，所以为了在这种情况下能够也保护遗失卡的用户卡内余额不被滥用，则限定了该卡的对应账户的余额只能用于支付该卡用户的业务账单，不能为其他任何人代付账单。即账户的资金、业务账单以及校园卡

对应的身份这三者之间绑定在一起，不能与其他账户或业务有关联。

（2）只接受正常状态的校园卡作为身份识别。所有发生挂失、冻结、销户等异常状态的校园卡，在平台内均不可使用。

数据安全中的存储安全是由一卡通平台的数据库和业务系统的数据库共同来完成的，即通过一卡通平台来管理交易数据的存储，通过业务系统数据库来管理业务系统的数据。在支付平台内部，只保存交易流水的单号和基本信息，并且这些信息都保存在专网内的数据库服务器中，由防火墙和数据库自身的安全设置保证安全。

数据安全的输出策略主要是要求所有业务数据的访问都必须在统一身份认证的保护下。任何访问者在查询数据时，都必须通过统一身份认证方可查看本人的业务和交易数据，不能查看他人的。

通过这两个方面的安全考虑，可以在技术与资源可实现的程度上实现对支付平台的安全考虑。

4.2 服务接口设计

4.2.1 核心组件

校园个人支付平台的核心服务是支付，采用校园卡的支付是其中一种手段。根据 4.1.1 节的逻辑架构设计分析，使用圈存机的系统的支付包括读写卡和更新服务器数据库两个操作，所以在图 4-2 中将圈存机的支付服务分为了校园卡读写管理和一卡通数据库管理两个部分，根据图 3-3 的 SDK 框架与 3.1 节中的分析，校园卡的交易过程核心是对卡内余额和账户余额的操作，这两处操作共同构成了校园卡的支付服务，所以支付的流程如图 4-5 所示：

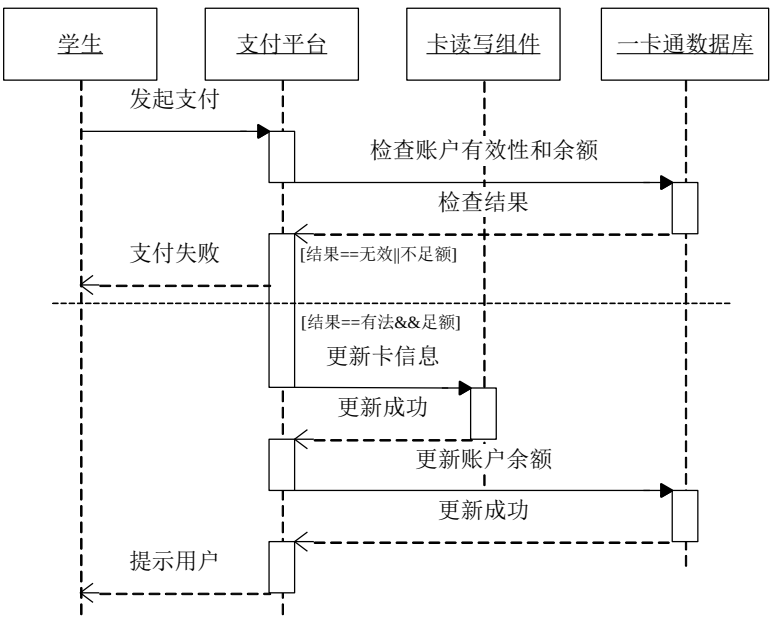


图 4-5　支付服务实现过程的时序图

从图中可以看出，支付的过程主要包括检查账户有效性和余额、更新校园卡，更

新服务器账户信息。检查账户的有效性是验证校园卡用户身份是否满足业务要求，验证账户是否冻结、过期、挂失或者其他异常情况的步骤。如果无法通过该检查，则无法进行后续步骤。检查账户余额是否足够是判断卡内余额和账户余额是否足够支付账单或者完成指定金额的充值，如果余额不足，则无法进行后续的步骤。

当两项检查完成之后，首先读写校园卡的相关扇区，更新卡内的余额和用卡次数等信息，完成卡内余额的更新。

然后通知服务器更新服务器端账户内的余额。两项操作都必须联机进行，如果发生脱机，则无法更新卡内余额也无法更新账户内的余额。

当上述更新操作完成之后，支付过程完成。

根据图 4-2 的逻辑架构和上述分析，支付服务主要包括两个组件，其接口应该包含如图 4-6 所示的内容：

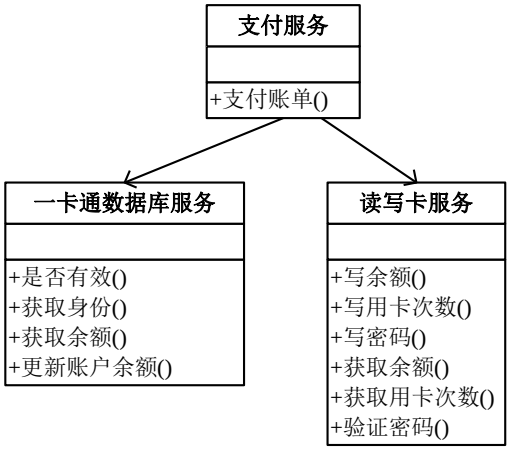


图 4-6　支付服务内需要实现的组件接口

一卡通数据库服务提供的接口包括是否有效、身份、余额、更新余额，读写卡服务包括写余额、用卡次数、密码、获取密码、获取用卡次数、验证密码等功能。通过这两个组件，即可实现对支付功能的支持。

业务代理接手业务的支付过程，处理支付对应的业务过程和规则，完成在业务系统中体现支付完成的操作。在图 3-7 中排队过程即使用了支付平台的业务代理服务查询刷卡的学生是否进行了缴费，在图 3-9 中支付平台的业务代理通知了业务系统所以实现了网络开通。并且根据 3.2.4 节对于安全性的考虑，设计代理可以保护业务系统的网络环境安全，避免暴露过多资源给外界。

从图 3-7 与图 3-9 的支付过程时序图以及图 3-10 的抽象支付过程可以知道，学生发起业务之后，支付平台向业务代理服务查询账单，业务代理服务返回结果，如果无需支付，则通知用户无需支付。如果需要支付，则首先向支付服务发起支付，收到支付完成的结果后，向业务代理服务发起更新业务记录的请求，返回更新完成后，支付平台通知学生业务完成。业务代理主要完成业务查询和业务通知的服务。

业务代理服务管理在这个过程中主要实现的就是查询账单和确认支付这两个内

容，查询账单在图 3-7 中是缴纳的拍照费时先行检查是否缴费以及排队时检查票据的过程，在图 3-9 中是审批成功后需要缴纳的开户费的查询。而确认支付在图 3-7 中是缴费人员确认支付完成，在图 3-9 中体现为网络开通。

根据图 3-10 的抽象支付过程，可以进一步细化业务代理在整个支付过程中的行为，该过程如图 4-7 所示：

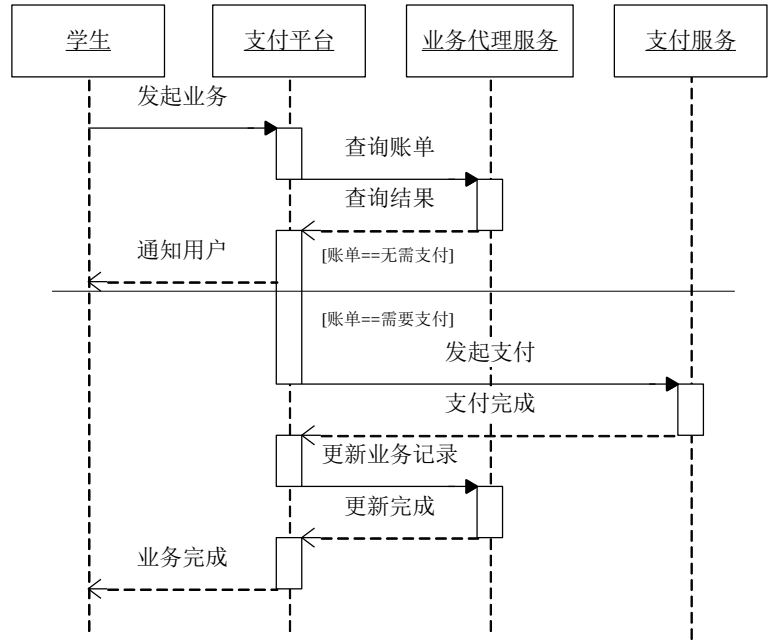


图 4-7　业务代理服务工作过程的时序图

所有支付都是围绕具体的业务进行开展的，对于个人用户而言，所有的业务可以简化为查询与确认支付过程。所以，根据之前的分析以及图 4-7 的中业务代理服务完成的工作，业务代理服务需要提供的接口如图 4-8 所示：

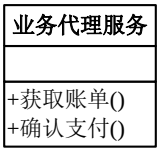


图 4-8　业务代理服务实现的接口

查询就是通过身份识别，查看在业务中该用户的欠费或者余额情况，并据此做出是否需要支付的凭据。而确认支付就是从用户的资金账户向业务的账户中进行充值，既可以是清缴欠费，也可以充值累计余额。所以，业务代理服务器只要提供这两个接口就可以完成绝大多数的业务支付工作。

对于用户具体操作的终端，如果分别独立与各个Web Service进行交互则非常复杂，所以需要设立一个调度服务器处理与各个Web Service进行交互的工作终端只需要按照约定的简单格式发送请求给调度服务器，由调度服务器完成业务的具体工作即可，无需独立与各个 Web Service 进行交互。这样也避免暴露过多的服务地址和网段给不必要的外部系统，便于安全性的管理。所以，调度服务需要实现的就是对支付服务和业务

代理服务的接口的访问，在图 4-2 中和图 5-17 中，分别就调度服务的逻辑架构和物理架构进行了分析，可以知道调度服务被与圈存机系统相连的 Web 服务器所调用，代理了平台内的支付服务和业务代理服务，所以调度服务的接口是封装支付服务的支付接口，以及业务代理服务的查询账单接口和确认支付接口，如图 4-9 所示：

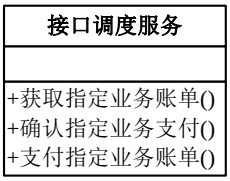


图 4-9　接口调度服务实现的接口

接口调度服务实现了获取指定业务账单、确认指定业务支付、支付指定业务账单的功能。

4.2.2 核心服务

实现了核心组件之后，即可按照图 4-2 实现核心服务。系统核心服务可以用来构建具体的缴费应用系统，所以在应用层面可以看到核心服务所提供的主要接口。这些接口采用 Web 服务的方式发布，系统可以调用 Web 服务实现对核心组件功能的使用。

核心服务主要提供以下的接口，如图 4-10 所示：

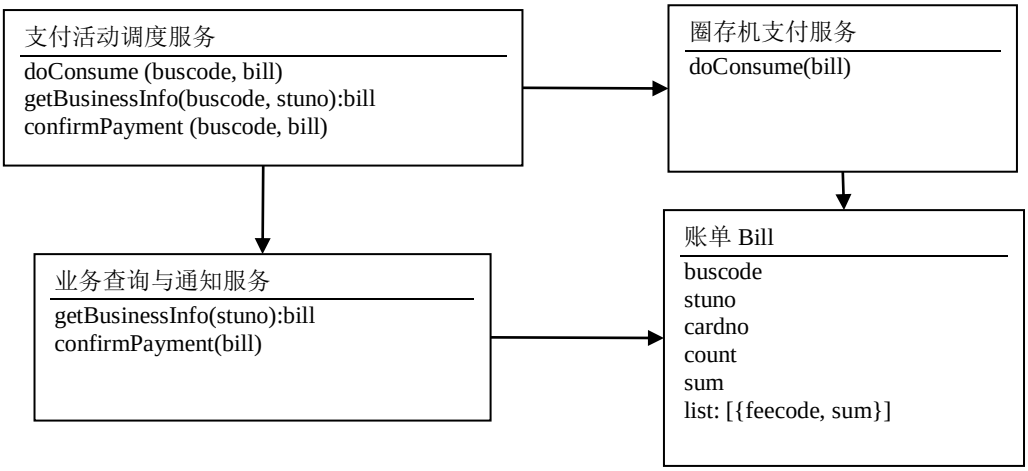


图 4-10　核心服务之间调度接口

可见，圈存机支付服务将校园卡读写管理组件与一卡通数据库管理组件进行了分装，提供了两个封装后的 Web 服务接口。业务查询与确认服务也采用 Web 服务封装了业务代理管理组件提供的服务。支付活动调度服务访问圈存机支付服务和业务查询与确认服务，并将这些服务的接口重新封装对外提供。在这三个服务之间，除了基本的学号信息以外，增加了 buscode 和账单 Bill 的定义，其中 buscode 业务代码是全局共享的字符串，在系统内每个服务中都有配置，采用预先约定的字符串标识。调度服务根据业务代码选择相应的业务查询与确认服务，同时圈存机支付服务也要根据业务代码选择支付的账户。在账单中的 feecode 费用代码也是全局共享的字符串，每个业务的不

同收费都用不同的费用代码标识，在圈存机支付服务进行具体交易的时候使用。

支付活动调度服务的接口具体定义如表 4-1 所示：

表 4-1　调度服务的接口具体定义表

接口名称	功能与参数说明
doConsume (buscode, bill)	功能：支付账单 buscode：业务代码 bill：账单
getBusinessInfo(buscode, stuno):bill	功能：获取业务信息账单 buscode：业务代码 stuno：学号
confirmPayment (buscode, bill)	功能：确认支付 buscode：业务代码 bill：账单

圈存机支付服务的接口如表 4-2 所示：

表 4-2　支付服务的接口具体定义表

接口名称	功能与参数说明
doConsume (bill)	功能：支付账单 bill：账单

业务查询与确认服务的接口如表 4-3 所示：

表 4-3　业务代理服务的接口具体定义表

接口名称	功能与参数说明
getBusinessInfo(stuno):bill	功能：获取业务信息账单 stuno：学号
confirmPayment (bill)	功能：确认支付 bill：账单

账单的字段定义如表 4-4 所示：

表 4-4　账单的字段具体定义表

字段名称	字段含义与类型
buscode	业务代码：字符串
stuno	学号：字符串
cardno	卡号：字符串
count	明细数量：整数
sum	金额：整数
list : [(feecode, sum)]	明细：数组。费用代码：字符串　金额：整数

服务器系统之间的通讯采用了基于 HTTP 协议和 SOAP 协议的 Web Service 进行实现，具体的交互协议内容格式是针对通用业务而设计。业务查询与确认服务和圈存机支付服务实现 Web 服务的技术采用了 Apache 基金会发布的基于 Java 的 Apache CXF

技术和微软公司的.Net 框架，对外接口都采用一致的基于 SOAP 的 WSDL。

4.2.3 业务辅助服务

（1）CAS 服务客户端

用户访问基于该平台的应用首先需要进行身份识别、即通过统一身份认证网关 CAS。CAS 采用了基于 HTTPS 的认证的方式，保证用户名称和密码的安全，在校园信息门户等系统中都有使用[31]。基于校园个人支付平台的应用系统在对最终用户提供网络信息查询的时候，都需要采用 CAS 作为用户身份认证的手段，所以在平台中，采用多种技术实现了 CAS 服务的客户端。在使用相应的技术开发应用时，可以直接应用客户端实现认证。

所有客户端主要实现 CAS 的两个功能，一个是访问拦截重定向，一个是凭据验证。整个过程如图 4-11 所示：

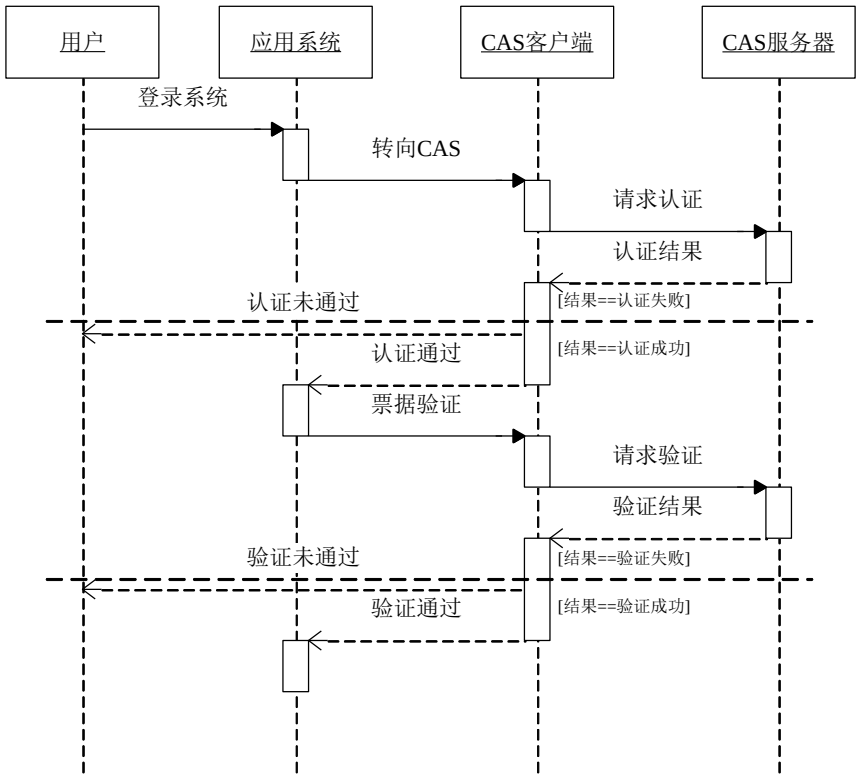


图 4-11　CAS 客户端实现认证与验证过程的时序图

用户登录系统以后，应用系统调用 CAS 客户端，转向指定的 CAS 服务器进行认证，如果用户无法通过认证，则直接由客户端提示用户登录失败。认证成功后，应用系统保留登录凭据，当访问应用系统内受保护资源时，系统会转向 CAS 客户端对凭据进行验证，CAS 验证后返回结果，如果验证失败则提示用户验证未通过需要重新登录。如果验证通过则应用程序继续进行。具体的接口如图 4-12 所示：

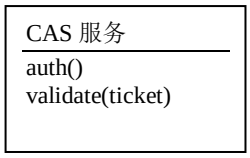


图 4-12　CAS 的服务接口

CAS 服务的接口详细说明如表 4-5 所示：

表 4-5　CAS 服务的接口表

接口名称	功能与参数说明
auth()	将用户重定向到 CAS 认证进行认证 如果通过认证则继续后续程序，否则程序终止
validate(ticket)	验证客户端提供的凭据，判断是否有效 ticket：通过认证后 CAS 服务器提供的凭据

（2）消息通知服务

在业务办理过程中和业务办理完成后，用户可能需要得到业务办理相关的通知信息，在整个数字校园中，可以提供邮件和短消息的方式来通知用户，并且提供了消息中心来处理消息请求[32]。通过平台提供的接口，实现了业务办理结果的通知，并且业务系统只需提供学生学号和需要发送的内容即可。基本过程如下图 4-13 所示：

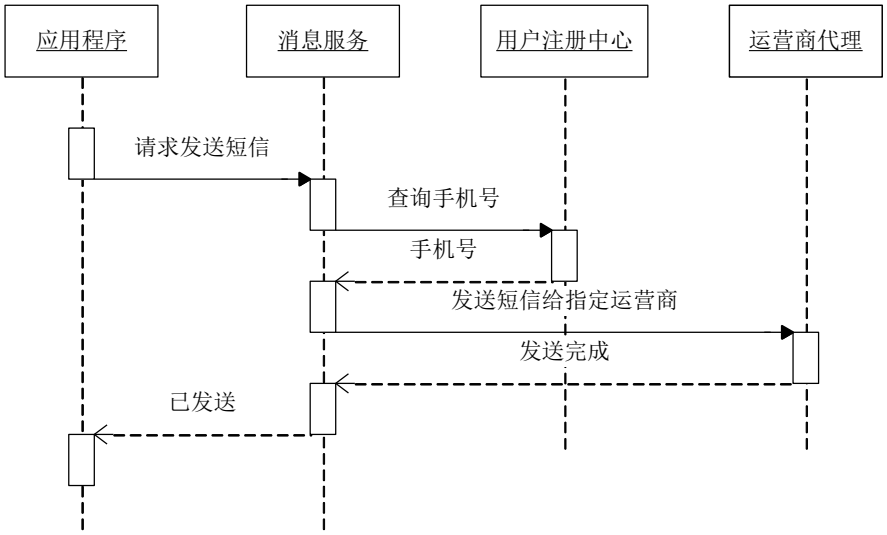


图 4-13　消息通知服务实现短信发送过程的时序图

应用程序根据需求向消息服务发出给某学生发送短信的请求，消息服务向用户注册中心查询该学生的手机号，然后根据手机号选择相应的运营商的代理进行发送，收到代理回执后告知应用程序发送完成。根据这个业务过程，可知消息服务对外实现的接口如图 4-14 所示：

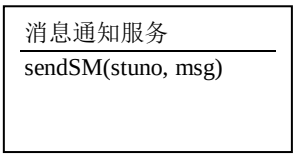


图 4-14　消息通知服务的接口

消息通知服务的接口详细说明如表 4-6 所示：

表 4-6　消息通知服务的接口表

接口名称	功能和参数说明
sendSM(stuno, msg)	向指定学号发送消息 stuno：学号 msg：短消息内容

在缴纳学费的业务中，当学费缴纳之后就可以给学生发送短信让其了解到业务已经成功办理的消息。

（3）票据打印服务

根据财务要求，部分业务需要保存纸质的票据。如果所有业务都由业务人员手工开具票据的话，则会极大的增加工作人员的工作量。所以在平台内专门提供了票据查询和票据打印的服务，其过程如图 4-15 所示：

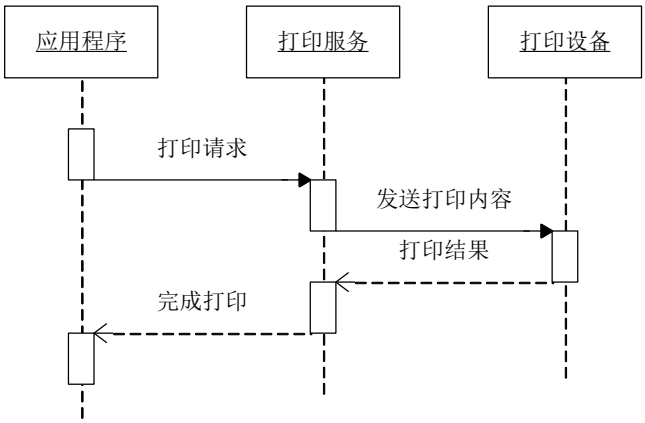


图 4-15　票据打印服务工作过程的时序图

应用程序向打印服务发送打印请求，打印服务根据内容将打印任务发送给设备，由设备完成打印。返回打印结果后服务提示打印完成。所以票据打印服务的接口如图 4-16 所示：

图 4-16　票据打印服务的接口

票据服务的接口详细说明如表 4-7 所示：

表 4-7　票据服务的接口表

接口名称	功能和参数说明
printReceipt(receipt)	打印指定票据 receipt：票据内容或者编号

使用票据打印，可以打印标准的财务票据或者小票。

（4）卡号与学号识别转换工具

在业务人员为用户办理业务的过程中，经常需要使用用户的学号。为了减少手工收入的时间和可能造成的错误，用户可以使用校园卡刷卡。但是专用读卡器需要工作在内部网络并且价格昂贵，无便于普及应用，所以在日常工作中，常使用只读卡号的通用读卡器。由于卡号具有唯一性且和学生的学号之间是建立了一对一的对应关系，所以通过卡号可以查找对应的学号，根据图 3-3 中的 SDK，可以获取得到有关校园卡与学生的基本信息。但是面对一个输入框，如果要让用户选择输入类型，通常会有操作失误，所以采用类似“框计算”的思想，可以解决这种问题[33]。

通过 JavaScript 技术或者服务器端识别技术，用户无需自己区分输入项目，就可以在一个输入框内输入学号或者刷卡，系统会自动识别输入类型，并统一返回标准的学号格式信息，这样即可将校园卡的身份服务充分利用。

目前提供服务器端识别技术与客户端识别技术两种方案，分别可以适用于基于 Web 的网站项目开发和基于桌面应用程序的本地程序开发中。服务器端的识别过程如图 4-17 所示：

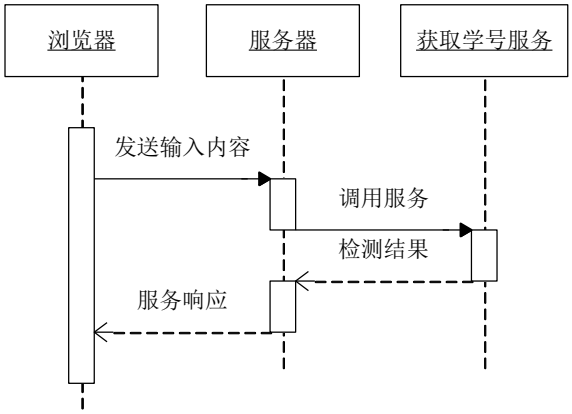


图 4-17　服务器端识别卡号学号并转换为学号的服务时序图

如图 4-17 所示，用户访问指定的页面，并在输入框中进行刷卡或者输入学号，浏览器或者应用将内容发给服务器后，服务器调用获取学号服务，服务按照匹配算法识别传入的字符串，最后返回对应的学号或者错误提示，这样即可在浏览器中显示对于与指定卡号的学号的相关信息了。

获取学号的字符串匹配的算法如图 4-18 所示：

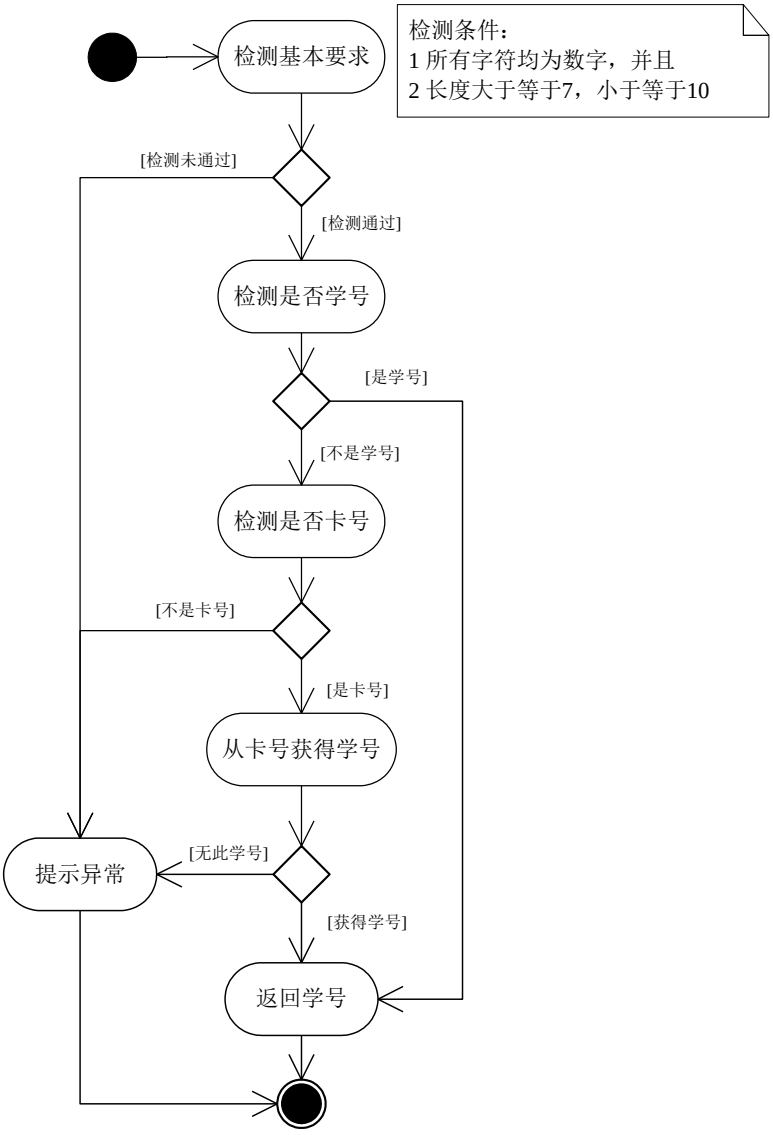


图 4-18　字符串匹配学号卡号的算法活动图

如算法所示，服务将用户输入看作未知字符串，采用正则表达式等技术分析字符串是否符合基本格式，如果不是则提示异常返回。然后查学号表判断是否是学号，如果是学号则直接返回，如果不是学号则查卡号表判断是否是在校注册的校园卡号，如果不是则提示异常返回。如果是卡号则返回对应的学号。

另外一种方案是在客户机上安装伺服程序，在读卡器将卡号写入键盘缓冲区之前先行将输入的信号从系统中拦截，然后将这个输入的卡号在显示在界面之前先由伺服程序进行处理，待得到对应的学号信息后再输出到本地机器的键盘缓冲区中。这样可以不用修改业务系统本身即可实现刷卡输入学号的功能，为无法支持服务端获取学号的系统提供便捷的业务能力改进。

该种方案的工作过程如图 4-19 所示：

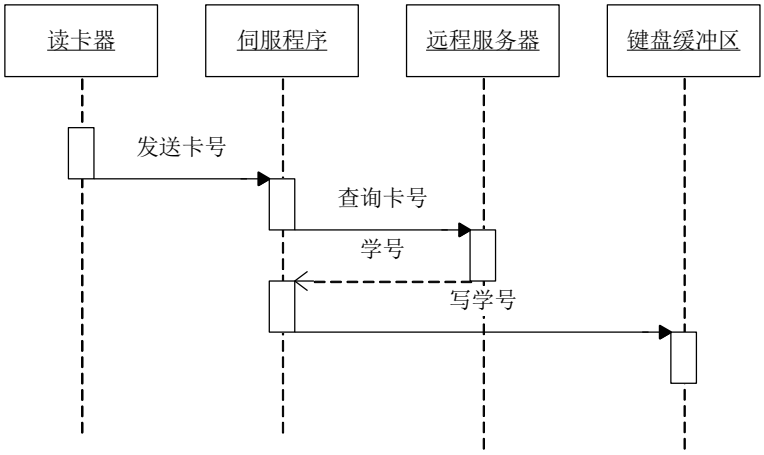


图 4-19　客户端伺服程序获取学号过程的时序图

如时序图所示，通过过伺服程序，将刷卡操作输入的卡号进行拦截获取，首先发送给远程的卡号学号转换服务器进行查询，得到返回的学号后再将其写入本地机器的键盘缓冲区。整个操作过程对用户而言相当于刷卡后直接填写了学号而非卡号，无需用户关心具体的实现细节。

（2）触摸屏 UI 设计参考规范

用户最终用户在使用平台时，能够接触的唯一接口就是系统的交互界面，根据 3.2.4 节中对于易用性的需求，为了让用户具有更好的操作体验，平台提供了一套人机界面的元素与整体设计。

在圈存机体系上进行支付时，业务的整个交易过程都可以分解为如图 4-20 的过程页面，每个页面配合后端的服务完成支付过程的一部分业务：

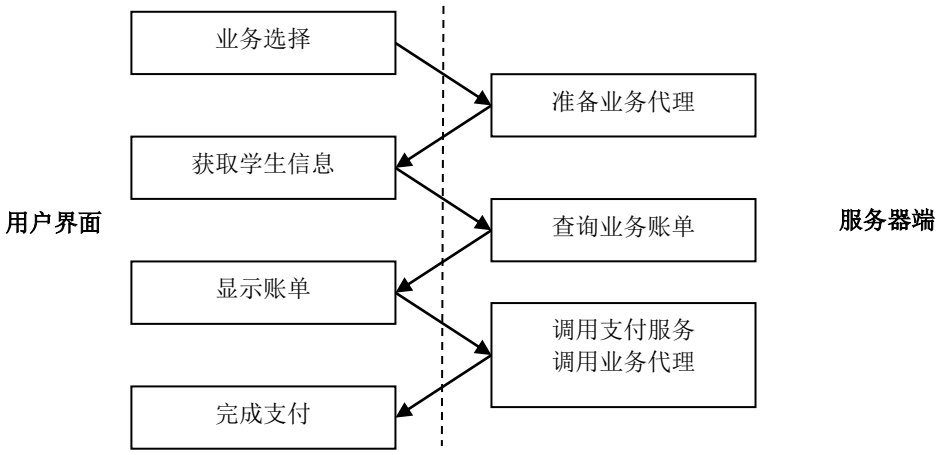


图 4-20　支付过程的页面与服务器端交互过程

如上所示，用户首先进行业务的选择，将选择的业务发送到服务器，服务器准备业务代理；然后界面进行读卡获取学生信息，将学生信息发送给服务器，服务器据此查询此学生的业务账单；然后界面显示学生账单，学生选择支付后，服务器调用支付服务和业务代理完成业务，最后显示业务办理结果。

4.3 本章小结

本章通过对校内异构业务系统以及现有数字校园平台的分析，以及对具体业务用例的分析，提出了一个基于 SOA 思想和相关 Web 服务技术的校园个人支付平台的解决方案，并结合具体的业务流程和抽象的支付模型，设计了平台的逻辑架构和数据架构，并对读写卡组件、业务代理管理、支付活动调度服务、CAS 客户端、消息通知服务、票据打印服务等多个服务的接口进行设计。

5 平台实现

5.1 开发技术

从图 3-4 和图 4-2 可知，平台需要为多个在体系架构、实现方式、数据标准、业务流程都完全不相同的异构系统的不同业务提供支付服务，并且自身也需要有不同的功能实现不同的服务，所以势必采用多种不同的技术来适应各不相同的业务环境。

根据图 4-2 所示的结构，从使用者可以将系统服务按照面向用户和面向系统进行划分。在面向用户的圈存机系统上的开发是面向最终用户的，在图 3-2 和相关的分析中可知，由于圈存机系统在此方面采用了 B/S 架构，即 Browser 浏览器端采用了 HTML 和 JavaScript 语言，Server 端采用基于 Java 的 JSP 技术。为了兼容原有圈存机系统的生产环境，减少不必要的兼容问题，所以对圈存机的接口和嵌入的服务方面，支付平台也必须采用 HTML 和 JavaScript 语言实现前端，用 Java 和 JSP 实现后端。另外，打印票据的需求必须开发桌面应用程序，并且所采用的组件是基于.Net 体系的，所以采用 C#技术实现。

而面向系统的服务，根据服务的不同，支付平台也需要根据实际情况选择合适的技术进行实现。开发技术的选择有多个影响因素，如开发周期、生产环境、开发人员技术水平、技术支持等等因素，综合这些因素，在实现不同的系统实现时选择技术。由于该项目的开发人员只有一人，同时负责支付平台的维护和支持，所以开发工作应首先以开发人员为优先考虑，在项目的开发周期内实现功能。因为动态语言技术具有代码量少，调试方便的主要优点都适合在开发人员少开发周期紧的情况下进行快速开发的要求，所以除了由于业务系统原因只能选择指定技术的情况以外，首先考虑使用动态语言技术实现功能。

在图 4-2 中，核心服务的校园卡支付方面由于一卡通平台采用了 C++/C#技术，并且所有开发案例和开发包都采用 C++/C#实现，所以支付平台在实现相关服务的时候也出于兼容性考虑采用 C#。网络费用的业务代理为了兼容和复用以前的系统代码，也采用了 C#技术。而另外一个虚拟 ADSL 缴费的项目由于对方系统提供的技术资料基于 Java，所以采用 Java 和相关的 Web 服务技术。调度服务实现请求转发，面向 Web 服务进行开发，所以不受以往系统的限制可以采用 Python 语言技术实现。

综合上述说明可以看出，采用动态语言技术实现基于 Web 实现的服务具有代码量少，开发便捷，部署方便，服务器易于管理的优点，并且动态语言如 PHP 和 Python 可以免费获得，且具有开源框架和项目的支持，减少开发成本和开发时间，所以多个辅助服务均采用动态语言实现。

结合上述分析与 2.1.2 节中所介绍的动态编程语言在快速开发、快速响应需求变更

等方面的优势，平台选择了一种多语言多框架跨平台的开发架构，整体如图 5-1 所示：

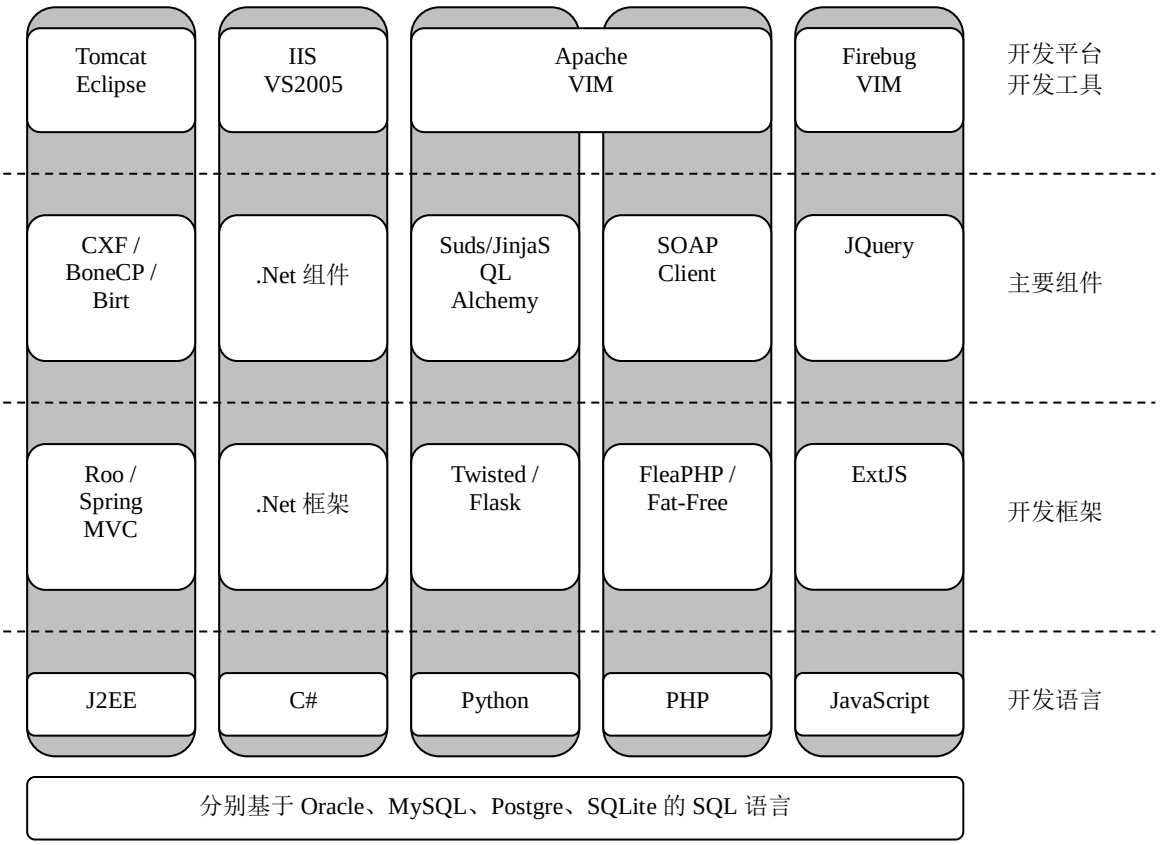


图 5-1　校园个人支付平台的开发架构图

从上图可知，纵向是一种语言所采用开发框架、主要组件、开发平台和开发工具。为了兼容以往的系统和开发符合 SOAP 规范的 Web 服务，除了采用动态编程语言，系统也采用了静态编程语言来实现这部分工作。

J2EE 目前仍然是企业级应用开发的主流技术，其在企业的应用案例能够充分证明其安全性和稳定性方面的考虑。主要采用 Roo 和 Spring 作为 Web 项目开发的框架，用 CXF 组件提供 Web 服务，BoneCP 技术处理底层的数据库操作，Birt 技术实现报表，采用 Tomcat 作为生产和测试的服务器，使用 Eclipse 和相应功能扩展作为开发工具。

.Net 架构也是企业级开发的主流技术，并且校园卡开发的官方 SDK 和应用系统也是采用微软的 Win32 组件技术实现的，所以采用.Net 技术能够平稳的对接校园卡的相关资源，减少问题的产生。C#语言采用.Net 架构的整套方案，包括其架构、组件以及 IIS 服务器实现 Web 服务和独立的功能。另外部分业务采用了 C#的桌面控件进行开发。整个开发工作采用 Visual Studio 2005 作为开发工具。

如 2.2.2 节中的说明，Python 语言在各个方面都具有非常成功的应用案例，能够满足高并发高负载等性能方面的要求，并且同时具有开发速度快，代码清晰简洁的特点。所以平台采用 Python 语言做核心调度服务的开发，使用 Twisted 框架，使用 Suds 实现

Web 服务访问。另外部分网站项目采用 Flask 作为框架，使用 Jinja 作为界面模版引擎，SQLAlchemy 封装数据库操作。另外还有部分桌面应用采用 wxPython 作为桌面程序的 GUI 实现。开发工作采用 VIM 作为编辑器，运行在 Twisted 或者 Flask 自带的服务器下，生产系统运行在 Apache 服务器下。

在 2.2.2 节中也说明了 PHP 语言用作辅助服务的开发，采用 Fleaphp 框架或者 Fat-Free 框架，用 SOAPClient 实现 Web 服务的访问和调用。测试和开发均采用 Apache 服务器，使用 VIM 编辑器开发。

目前浏览器端的开发最普遍的就是 HTML + CSS + JavaScript 技术组合，具有良好的适应性和表现能力，所以面向最终用户的交互采用 JavaScript 技术实现，部分业务的管理端采用了 ExtJS 框架，使用 JQuery 组件实现界面交互效果。在 Firefox 的扩展插件 Firebug 中进行调试开发。

数据库开发方面，根据系统的不同，分别使用了基于 MySQL、Oracle、Postgre、SQLite 的 SQL 语言进行数据库方面的操作。

5.2 核心组件

5.2.1 校园卡读写组件

读写卡服务是对卡内信息的读写，底层实现读写卡内信息是使用 C++实现的，对服务器通信的模块采用 Java 封装实现，在页面上通过 Java Script 调用 C++控件实现对校园卡的读写功能。传统业务的读写卡过程直接采用底层函数操作，非常繁琐且不容易复用，所以在实现平台的过程中，对读写卡组件进行了封装，便于后续的开发。读写卡服务的部分 JavaScript 代码如下：

```
function CardOperator(){
   参数配置;
}
CardOperator.prototype.init = function(xmlhttp, posplugin, keyapi){
   初始化();
}
CardOperator.prototype.readCardSimple = function(secnum, blocknum){
      if(!this.openCardPort()){ return false; }// 第一开端口
      if (!this.findCard()) { return false; }// 首先寻卡
      if (!this.selectCard()) { return false; }// 然后选卡
      if (!this.authoriseCard(secnum)) { return false; }// 然后验证扇区
      return this.readCard(secnum, blocknum); // 最后读卡，直接返回
}
CardOperator.prototype.writeCardSimple = function(secnum, blocknum, content){
      if(!this.openCardPort()){ return false; }// 第一开端口
      if (!this.findCard()) { return false; }// 首先寻卡
```

```
        if (!this.selectCard()) { return false; }// 然后选卡
        if (!this.authoriseCard(secnum)) { return false; }// 然后验证扇区
        return this.writeCard(secnum, blocknum, content); // 最后写卡，直接返回
    }
```

在 JavaScript 的代码开发过程中，采用了面向对象的实现方式，将所有对卡的操作封装在 CardOperator 类中。如读写卡过程所示，每次读卡或者写卡，都需要完成打开端口、寻卡、选卡、验证扇区的操作，等扇区可以访问后，方可读写。如图 5-2 所示：

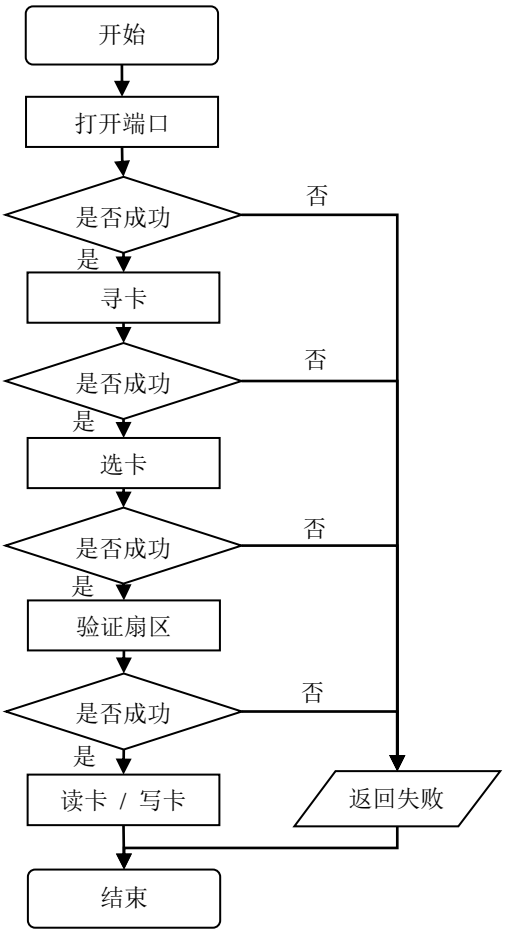


图 5-2　实现读写卡的函数的流程图

如图 5-2 所示，当任何一个步骤发生失败时，函数返回失败并结束。在整个过程中，验证扇区的步骤必须有密钥服务器的配合。通过对扇区的读写和详细扇区的设计规划，就可以实现对卡片内账户余额与其他个人信息的读写。

5.2.2 一卡通数据库服务

一卡通数据库服务是对账户信息的查询，底层使用 C++实现控件，平台使用 C#封装了对控件的调用，实现了更新卡账户余额以及其他辅助功能。数据库服务的部分 C#代码如下：

```
[DllImport("AIO_API.dll")]
public static extern int TA_UpJournal(ref CardConsume pCardCons, short Timeout);
// 调用 SDK 提供的一卡通交易函数
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
public struct CardConsume
public string doConsume(string accountno, string stuno, string cardno, string opor,
string balance, string sum, string times, string buscode, string feecode)
{
    CardConsume cc = this.generateCardConsume(accountno, cardno, opor, balance,
sum, times); // 构造 SDK 所需的数据类型
    int nRet = TA_UpJournal(ref cc, 10); // 调用 SDK 函数
    if (nRet == 0 && cc.RetCode == 0)
    {
        return "支付成功";
    }
    else
    {
        return "支付失败";
    }
}
```

更新一卡通数据库的操作需要调用图 3-3 的 SDK 框架中应用接口部分的函数，如上所示，首先在 C#里，从 SDK 的动态链接库中导入所需要的函数和类型定义，然后通过实现了一个 doConsume()的函数，构造所需的类型并调用函数实现对数据库的操作。整个函数的流程图如图 5-3 所示：

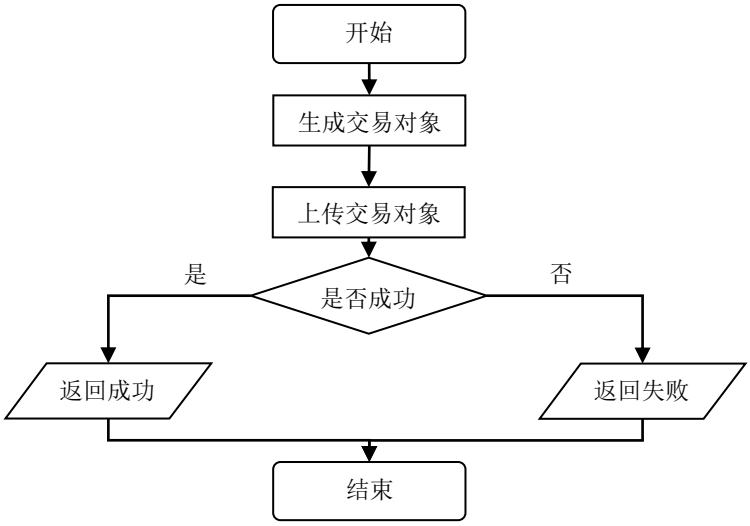


图 5-3　实现与一卡通平台数据库通讯的函数的流程图

5.2.3 业务代理管理

抽象的业务代理服务器对外提供查询和支付这两个接口，而对于业务的办理而言，具体的操作过程都交由业务代理服务器实现，访问这两个接口的用户无需了解具体的实现细节。为了实现这一特性，可以采用 Web Service 技术实现业务代理服务器，所有的业务代理服务器使用相同的接口，即相同的 WSDL，只要用户具有 Web Service 的地址和授权，即可按照相同的规范调用服务。

在具体的开发过程中，一个业务封装了业务系统和相关的数据库，如图 5-4 所示：

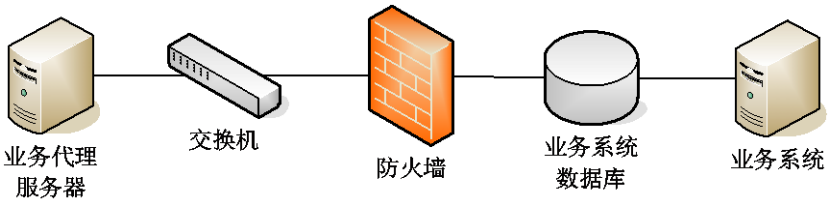


图 5-4　业务代理服务实现访问业务系统数据库的物理架构图

由于该业务代理工作使用了一些 Win32 体系下的组件，所以其服务的开发采用了.Net 框架的 C#语言，实现业务代理功能的部分代码如下：

```
public string getCheckByStuNo(string stuno)
{
    string sql = "select ..." //  此处根据学号构建查询语句
    try
    {
        DataTable dt = this.db.doSelect(sql);
        foreach (DataRow dr in dt.Rows)
        {
            string tmp_str_fee = dr["WJJE"].ToString(); //  获取每笔费用明细
        }
        ret =  构造返回账单字符串();
        return ret;
    }
    catch (Exception e)
    {
        处理异常();
    }
}
```

业务代理实现了 getCheckByStuNo()函数实现查询账单，通过用 SQL 语句从数据库的指定表查询记录，获得学号对应的账单。该函数的流程图如图 5-5 所示：

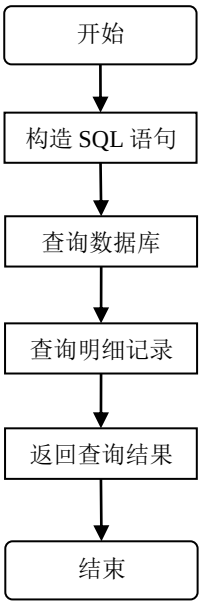


图 5-5　查询账单的函数的流程图

如图 5-5 所示，查询账单的函数通过 SQL 语句查询业务系统的数据库，并根据明细记录构造返回结果，最后返回给调用该服务的用户。

另外，业务代理也实现了 confirmPayment()函数，通过 SQL 完成在业务系统中确认业务支付的过程。其代码如下：

```
public bool confirmPayment(string stuno, string cardno, string feeid, string feemonth, string feecode, string feesum, string BackJnl, string consumeTime, string ip)
{
    string sql_paycheck = "";   // 按照业务规则需要实现的 SQL 语句
    try
    {
        DataTable dt = this.db.doSelect(sql);
        确认支付();
    }
    catch (Exception e)
    {
        处理异常();
    }
    return true;
}
```

另一个业务封装了其他业务系统的 Web Service 并搭建了专用虚拟链路的服务器，如图 5-6 所示：

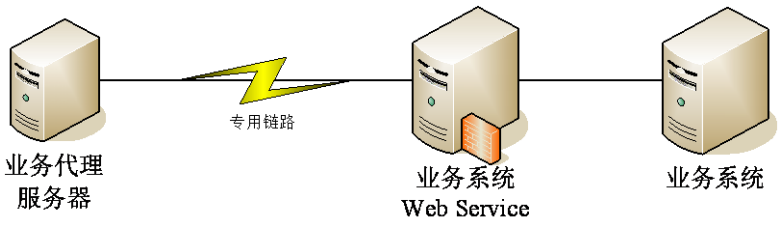


图 5-6　业务代理服务实现访问业务系统 Web Service 的物理架构图

对数据库进行底层操作的业务代理按照业务规则对数据库相应的数据表和字段进行读写，使整个过程对于业务系统是透明的，但业务系统可以看到操作结果。而业务系统提供了 Web 服务的业务代理则只需要按照 Web 服务的规范调用即可实现业务过程。该服务由于配置了 Linux 服务器进行通信，所以开发工作也就采用 J2EE 体系来实现 Web 服务，部分代码如下：

```
public UserReturnPack callQueryUserInfo(String account){
   try{
      AuthReturnPack arp = callAuth(); // 首先进行认证
      String request = generateQueryUserInfoString(arp,account); // 生成查询参数
      String result = client.schoolService(request); // 查询
      return analysisUserReturnPack(result); // 返回查询结果
   }catch(Exception e){
      处理异常();
   }
}
```

如代码所示，该业务代理实现了 callQueryUserInfo()函数进行用户信息的获取，通过调用业务系统提供的 Web 服务接口 schoolService()和构造的参数，实现业务的查询。由于业务系统的要求，业务代理首先要进行认证，代理与业务服务器之间通过建立可信的会话状态确保代理能够符合认证要求。然后业务代理根据查询的请求信息构造查询参数和相应的协议报文。通过将这个报文发送给业务服务器实现查询的功能，随后即可得到所要查询的账户的信息。

另外该业务代理也实现了确认账单的函数，在该业务中，确认账单即表示对账户进行充值操作。当已经完成从一卡通的账户给业务账户的转账后，业务代理即发起对业务系统账户的充值操作。其核心代码如下：

```
public ChargeAccountPack callChargeAccount(String stuno, String account, String
balance, String posid){
   try{
      AuthReturnPack arp = callAuth(); // 首先进行认证
      String request = generateChargeAccountString(arp,account,balance,posid); // 生成
充值的参数
      String result = client.schoolService(request); // 进行充值操作
```

```
        ChargeAccountPack cap = analysisChargeAccountPack(result); // 返回充值结果
        return cap;
    }catch(Exception e){
        处理异常();
    }
}
```

该业务代理实现了 callChargeAccount()函数处理在业务过程中的支付。该业务系统的 Web 服务名称均为 schoolService，只在传递参数的时候根据参数内容不同来识别所进行的操作。

除此以外，还有一类业务过程是不存在数据库也不存在业务系统的，如图 3-6 和图 3-7 所示的毕业生照片采集支付收费业务，这种业务完全是按照人工方式进行的，所以对这种业务，需要使用业务辅助系统提供一个将业务持久化为数据的服务，通过 Excel 文档或者其他方式将业务名单存入数据库，实现一个专用的业务代理，如图 5-7 所示：

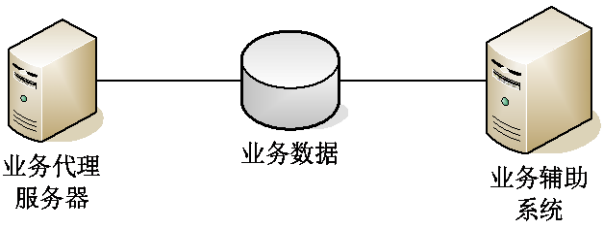


图 5-7　业务代理服务实现访问无业务系统的业务数据的物理架构图

该业务与图 3-9 的业务有所不同，其无需调用支付服务，只要按照图 4-8 所示的接口实现业务代理服务即可。另外由于图 3-7 还有排队的需求，所以在业务辅助系统中可以实现这个功能。该业务由于不涉及支付过程也无需使用通用的调度服务，所以实现上采用 PHP 技术可以快速的进行设计开发并支持大规模并发的使用，以下是业务代理接口实现的部分代码：

```
function actionAjaxSearch(){
    $stuno = getStunoFromInput($_POST['in']);
    $student = $this->_modelStudents->find($id);
    if($student){
        echo json_encode($student);
        return;
    }
    $result['success'] = false;
    $result['msg'] = '请重新刷卡或者输入学号';
    echo json_encode($result);
}
```

actionAjaxSearch()函数实现了查询学生缴费信息的功能，其流程图如图 5-8 所示：

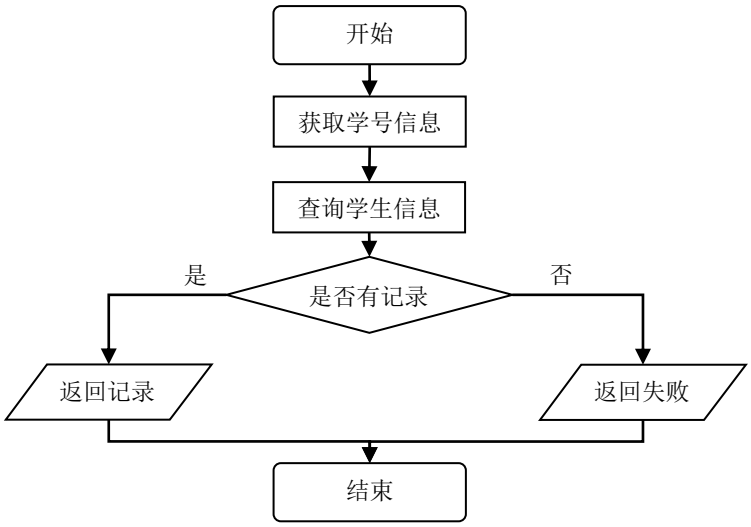


图 5-8　查询学生缴费信息的函数的流程图

如图 5-8 所示，该函数在获取学号信息后，查询学生的信息，如果有记录则返回记录，如果没有则返回失败。另外该业务代理还实现了确认缴费的功能，代码如下所示：

```
function actionAjaxConfirm(){
    $stuno = $_POST['stuno'];
    $student = $this->_modelStudents->find(array('stuno'=>$stuno));
    if ($student) {
        $student['state'] = 'CONFIRMED';
        $student['msg'] = "缴费成功，{$student['name']}目前的状态是
[{$student['state']}]";
        echo json_encode($student);
        return;
    }
    $result['success'] = false;
    $result['msg'] = "缴费失败";
    echo json_encode($result);
}
```

该函数的流程图如图 5-9 所示：

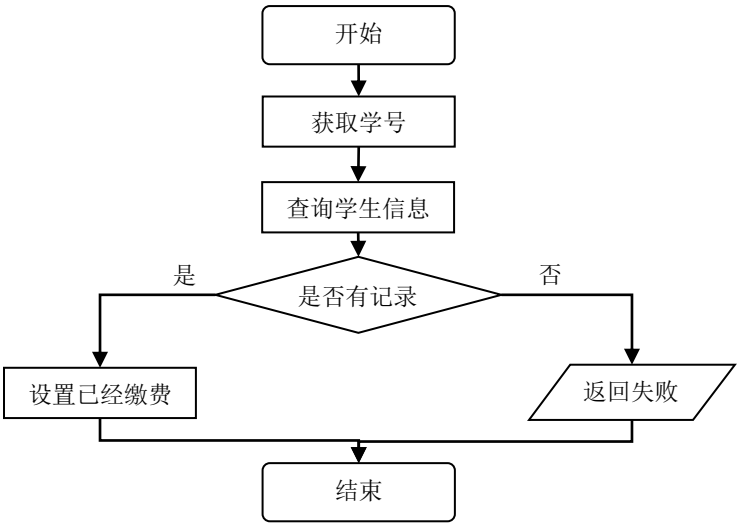


图 5-9　确认缴费的函数的流程图

如图 5-9 所示，actionAjaxConfirm()函数用于确认缴费。从上述代码和流程图可以看出，这两个由 PHP 实现的服务采用了浏览器异步通讯技术 AJAX 和 JSON 结构的数据格式。

通过以上不同业务的实现业务代理的方法的说明，可以看出使用业务代理实现 Web 服务可以将业务的网络复杂型、业务复杂性等都封装起来，避免平台内的其他应用由于要重复实现相似的过程而变得过于复杂。

5.3 核心服务

5.3.1 支付活动调度

从图 5-17 的物理架构上可知，调度服务不直接面对最终用户，所以对于调度服务器，自身无需提供人机界面，只要对支付服务和业务服务提供进一步的封装，实现通用的接口即可。为了实现的通用性和便于终端访问，采用基于 Python 的 Twisted 框架实现了一个 HTTP 服务器，使用 HTTP 协议规范的请求来调用接口。调度服务器将各个配置信息保存在自身的配置文件中，可以动态的加载和调整，终端只需提供业务的名称、操作名称和操作参数即可从调度服务器处得到相应的响应返回给用户。实现功能的部分代码如下：

```
def getBusinessInfoByStuNo(self, request):
    try:
        stuno = request.args['stuno'][0]
        buscode = request.args['buscode'][0]
        busagent = Client(cfg.configs[buscode]['bs_wsdl'])
        return ret
    except:
        处理异常();
```

```
    def pay(self, request):
        try:
            siosagent = Client(cfg.configs[buscode]['ss_wsdl'])
            siosret =
siosagent.service.DoConsume(accountno,stuno,cardno10,oper,balance,feesum,times,buscode
,feecode)
            return siosret
          except:
            处理异常();

    def confirmPay(self, request):
        try:
            busagent = Client(cfg.configs[buscode]['bs_wsdl'])
            busret = busagent.service.confirmBusinessPayment(stuno, cardno10, feecode,
feesum, extra)
            return busret
          except:
            处理异常();
```

如上所示，核心调度服务器分别将圈存机支付服务和业务查询与确认服务的接口在自身内部实现，包括业务查询与确认服务的 getBusinessInfoByStuNo()用于查询业务，confirmPay()用于业务确认支付，以及圈存机支付服务的一卡通数据库交易 pay()函数。调度服务的函数过程本质上是调用相应代理完成功能，本身并不实现业务功能或者支付功能。所以从其流程图可以看出，调度服务只是将参数转发给相应的代理，然后返回代理计算的结果。

5.3.2 核心服务的 Web Service

所有的服务都是采用 Web Service 的方式来实现，具体的 WSDL 如下所示：

圈存机支付服务的部分 WSDL 片段如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions>
    <wsdl:types>
        <s:schema elementFormDefault="qualified"
targetNamespace="http://www.xjtu.edu.cn/">
            <s:element name="DoConsume">
                <s:complexType>
                    <s:sequence>
                        <s:element minOccurs="0" maxOccurs="1" name="stuno"
type="s:string"/>
                    </s:sequence>
```

```
            </s:complexType>
          </s:element>
          <s:element name="DoConsumeResponse">
            <s:complexType>
              <s:sequence>
                <s:element minOccurs="0" maxOccurs="1" name="DoConsumeResult"
type="s:string"/>
```

如 WSDL 的定义中所述，该 Web 服务提供了名为 DoConsume 的函数，将具体的圈存机支付活动封装在其中。

业务查询与确认服务的 WSDL 如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions>
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://tempuri.org/">
      <s:element name="getBusinessInfoByStuNo">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="stuno"
type="s:string"/>
              ….
      <s:element name="getBusinessInfoByStuNoResponse">
        …
      <s:element name="confirmPayment">
        …
```

如 WSDL 片段所示，该 Web 服务实现了 getBusinessInfoByStuNo()函数获取业务信息，confirmPayment()函数确认支付完成。

支付活动调度服务的面向 Web 服务器，所以采用了基于 HTTP 协议的实现，将接口定义为 URI（Uniform Resource Identifier），格式如下：

https://192.168.1.2:12580/gbi：即 getBusinessInfo 的缩写。实现获取业务信息的功能。

https://192.168.1.2:12580/cp：即 confirmPayment 的缩写。实现确认支付的功能。

https://192.168.1.2:12580/dc：即 doConsume 的缩写。实现选择响应支付服务的功能。

以上 url 只接受 POST 方式的 http 请求，参数规格如表 4-1 和表 4-4 所示。

5.4 业务辅助服务

5.4.1 CAS 客户端

已经实现了基于 Python，PHP，Java 的 CAS 客户端，分别为多个不同的异构应用系统提供 CAS 服务。以 Python 为例，实现的代码如下：

```
class CASClient:
```

```
def __init__(self,returl):
    self.CAS_URL = 'https://cas.xjtu.edu.cn/'
    self.returl = returl

def Authenticate(self):
    login_url = self.CAS_URL + 'login?service=' + urllib.quote(self.returl)
    return redirect(login_url)

def Validate(self, ticket):
    validate_url = self.CAS_URL + 'validate?service=' + urllib.quote(self.returl) +
'&ticket=' + urllib.quote(ticket)
    ret = urllib.urlopen(validate_url).readlines()
    if len(ret) ==2 and re.match("yes", ret[0]) != None:
        return ret[1].strip()
    return None
```

实现过程使用了 Python 的面向对象技术，构造了一个 CASClient 类，实现了 Authenticate 函数实现认证过程，以及 Validate 函数实现验证过程。

从实现过程可知，Authenticate 函数通过将页面重定向到指定的服务器实现业务的终止，同时保留了认证完成后返回的地址。

Validate 函数的流程图如图 5-10 所示：

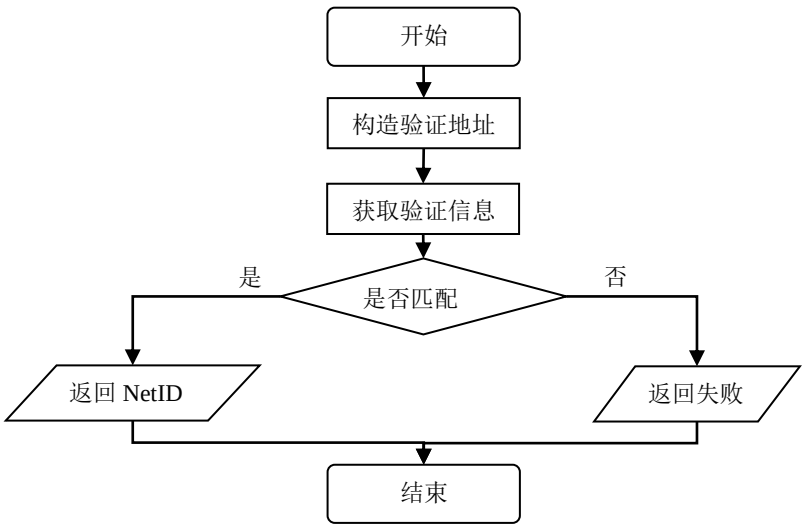


图 5-10　CAS 验证函数的流程图

如图 5-10 所示，Validate 函数将票据作为参数传入 CAS 服务器的指定 URL 实现验证，并将返回服务器验证的结果。

5.4.2 消息通知服务

该项服务采用 Python 技术通过监控业务日志实现，在无须修改业务自身流程的情

况下就可以实现通知服务。消息通知服务的接口实现的 Python 代码如下：

```
def sendSM(stuno,msg)
    pattern_cm = r'(^134|135|136|137|138|139|150|151|158|159|188)\d{8}$'
    pattern_cu = r'(^130|131|132|133|155|156|186)\d{8}$'
    pattern_ct = r'(^189)\d{8}$'
    uc = WSClient(uc_wsdl)
    cm = WSClient(cm_wsdl)
    cu = WSClient(cu_wsdl)
    ct = WSClient(ct_wsdl)
    user = uc.service.getInfoByStuno(uc_token,stuno)
    ret = None
    if user.mobile!='':
        if re.findall(self.pattern_cm, mobile):
            ret = cm.service.sendSM(user.mobile, msg)
        elif re.findall(self.pattern_cu, mobile):
            ret = cu.service.sendSM(user.mobile, msg)
        elif re.findall(self.pattern_ct, mobile):
            ret = ct.service.sendSM(user.mobile, msg)
    return ret
```

该函数的流程图如图 5-11 所示：

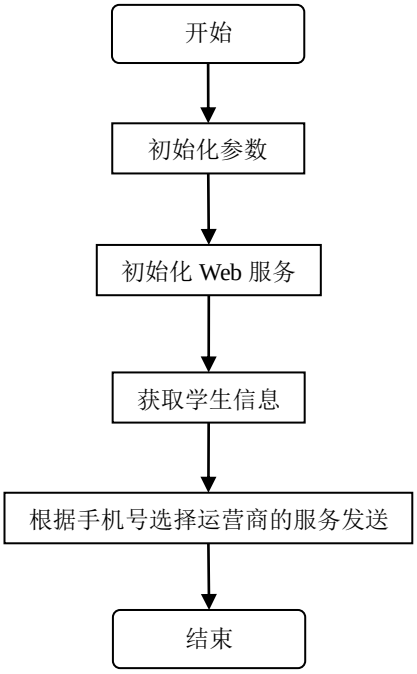


图 5-11　消息通知函数的流程图

如图 5-11 和代码所示，消息发送服务首先从用户中心的 Web 服务获取学生详细信息，得到手机号，并根据手机号查询对应的运营商。再调用对应运营商的服务接口实现短信发送的过程。

5.4.3 票据打印服务

该服务采用 C#实现专用发票的套打，配置了财务专用的 EPSON LQ-630 打印机，实现代码片段如下：

```
public class Printer
{
    private PrintDocument pd;
    private PrinterSettings ps;
    public void print_Check(List<PrintItem> pis)
    {
        this.piList = pis;
        this.pd.Print();
    }
}
public class PrintItem
{
    private string text;
    private Font f;
    private Brush b;
    private float margin_left;
    private float margin_top;
    private StringFormat sf;
}
```

如代码所示，系统实现一个 Printer 打印机类，包含了由.Net 框架和操作系统提供的 PrintDocument 类。然后实现一个 PrintItem 类表示需要打印的每个项目细节，具体到每个字符串需要在打印输出时的具体位置，通过 margin_left 和 margin_top 两个参数控制在页面上的位置。通过 Printer.print_Check()函数可以实现打印 PrintItem 数组的功能，将一个票据看作一组 PrintItem，逐个打印之后即完成整个票据的打印。

5.4.4 终端查询服务

校园内部分用户不具备使用网络的条件，所以平台提供了专用的查询终端用以查询业务办理的详细情况。同时也采用了在校园信息门户的形式可以查询这些业务信息，进一步扩展了查询功能的使用方式[34]。

用户发起查询业务的请求，终端机将请求发给服务器，由服务器在指定的业务代理上查询，然后再逐级将查询到的业务结果返回并最终显示给用户。

终端服务器从不同的业务代理服务器获得需要查询的业务数据，用户可以通过专用的查询终端实现查询。所有的查询终端都配置了读卡器和触摸操作屏幕。图 3-9 所示的网络开户业务在办理完成后，就可以使用查询终端实现办理结果的查询。

终端采用 B/S 架构，前端基于浏览器技术采用 HTML、CSS 和 JavaScript 技术开发，后端使用 J2EE 技术实现。实现的物理架构如图 5-12 所示：

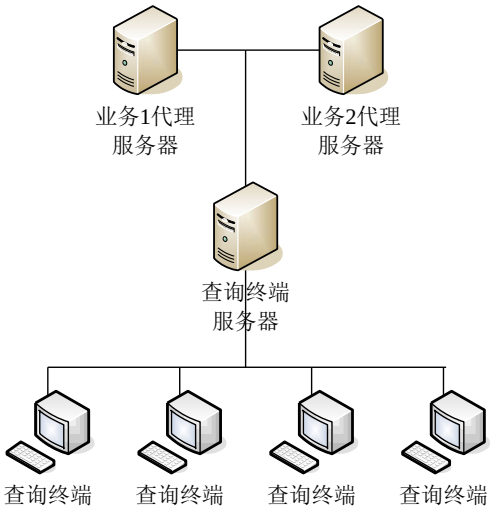


图 5-12　查询终端实现查询的物理架构图

5.4.5 平台内通用工具集

平台开发的过程中，根据业务需要实现了一些工具，有些是在服务内使用的，如类型转换、格式化输出等等。有些可以通过进一步的完善实现普遍通用的功能，如识别卡号学号并进行转换的工具，以及通用的触摸屏 UI 设计参考规范。识别卡号学号的功能在无法使用专用读卡器的网络环境下可以提供对校园卡刷卡进行业务的支持，使工作人员无需手工输入学号。同时也可以使用校园卡作为身份识别的工具，简化业务过程。通用的触摸屏 UI 设计参考规范则为多种服务提供了人机界面的参考，如圈存机操作和查询终端操作都可以使用触摸屏进行工作，这两者采用相同参考规范的设计可以让用户无需学习新系统的操作，并且具有较高可用性的设计可以提高用户操作的准确性避免不必要的操作错误引起问题。

（1）卡号与学号识别转换工具

目前服务器端的采用 PHP 实现了该工具，代码如下：

```
function getStunoByCardno($no){
   if(preg_match("/^\d{7,10}$/", $no)==0){
      return null; //  去除完全不可能的输入
   }
   $stu = $this->searchCardDatabase($no);
   if($stu){ return $stu['stuno']; }
   $stu = $this->searchStudentDatabase($no);
   if($stu){ return $stu['stuno']; }
   return null;
}
```

如代码所示，getStunoByCardno()函数实现将$no 变量转为学号的功能，按照图 4-18 所示的算法，首先用正则表达式检测是否正规，然后查询卡数据库判断是否是卡号，如果是卡号则返回对应学号，如果不是则继续查询学号库，如果是学号则返回学号，

如果不是则返回空。

这个服务需要客户端和服务器端配合实现，客户端采用了 C#技术调用专用驱动和 Win32 组件控制读卡器和键盘缓冲区的读写，服务器端采用 Python 技术实现一个简易的 Socket 服务器，代码片段如下：

客户端的 C#代码：

```
public partial class Form1 {
   [DllImport("dcrf32.dll")]
   public static extern int dc_read(int icdev, int adr, [Out] byte[] sdata);
   WinIoSys m_IoSys = new WinIoSys();
   private void GetCardNo(object sender, ElapsedEventArgs e)
   {
      thisICCardNo = 0; char str = (char)0;
      dc_card((Int16)_icdev, str, ref thisICCardNo);
      TcpClient tcpclnt = new TcpClient();
      tcpclnt.Connect("192.168.1.6", 12580);
      发送卡号();
      ret =  读取服务器响应数据();
      m_IoSys.写入键盘缓冲区(ret);
   }
```

该函数的流程图如图 5-13 所示：

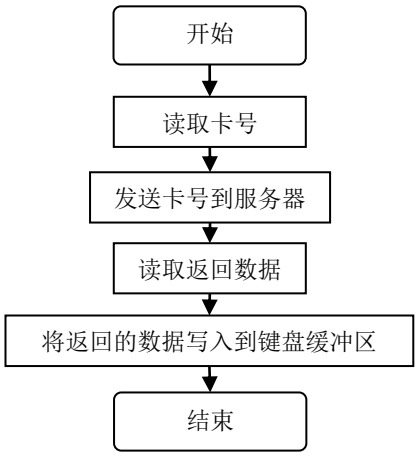


图 5-13　客户端 C#实现卡号转为学号并写入的流程图

如图 5-13 和代码所示，通过导入读卡驱动的动态链接库和相应的函数，还有 Windows 系统的输入输出库 WinIO，实现读取校园卡的功能和写如缓冲区的功能。读取卡号后，建立 Socket 连接并发送卡号，服务器返回数据后写入键盘缓冲区。

服务器端采用 Python 实现，代码片段如下：

```
import socket
from suds.client import Client
```

```
ip = ‘0.0.0.0’
port = 12580
maxClient = 10
serv = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
serv.bind((ip,port))
serv.listen(maxClient)
client = Client(carddata_wsdl)

while(1):
   conn, addr = serv.accept() #  收到请求
   msg = conn.recv(1024) #  读取请求内容
   ret = client.service.getStuByCardno(msg) #  查询学号
   if ret!=None:
      发送(ret['stuno']);
   else:
      发送错误消息();
serv.close()
```

该服务的主要过程的流程图如图 5-14 所示：

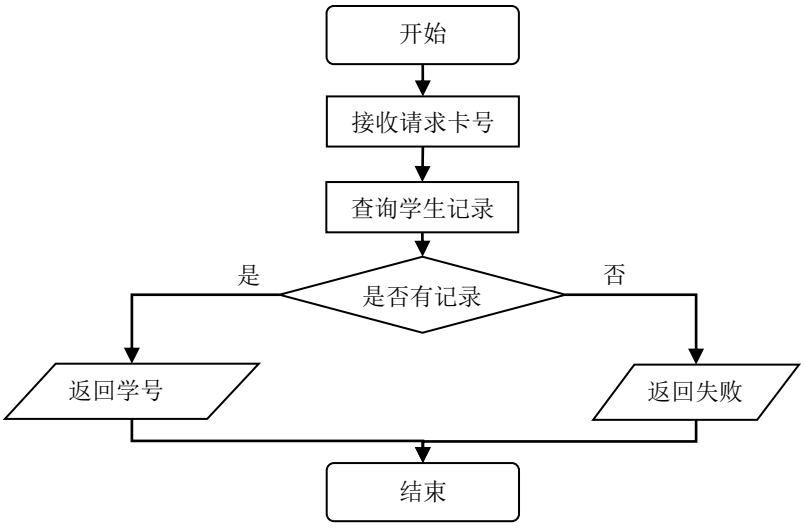


图 5-14　服务器端的主要过程的流程图

如图 5-14 和代码所示，服务器端使用 Python 建立一个 Socket 服务器，监听指定端口，当收到请求后调用卡号查询服务得到学号，通过连接写回请求的客户端。

（2）触摸屏 UI 设计参考规范

根据视觉效果和用户的操作感受，具有高对比度的大图形设计在各种环境下都容易识别。按照这个方式设计的界面如图 5-15 所示：

(a) 业务选择　　(b) 获取学生信息

(c) 显示账单　　(d) 完成支付

图 5-15　触摸屏操作的主要页面设计图

圈存机采用了触摸屏作为人机交互的接口，这和以往银行的自动柜员机只有物理按键的操作方式有很大差别。在触摸操作的过程中，由于屏幕校准或者用户操作习惯的差异，如果按键较小则无法准确点击，而且当点击界面的操作反馈不明显时，用户不知道是否点击成功。

所以为了解决用户操作的问题，特别设计了专用的操作界面。如上面的系统界面所示，系统具有一致性的观感和适应业务流程的操作，所有的返回或者退出按键都在屏幕右下方，并且当下按时底色变为红色。所有的业务操作过程按钮交互时变为绿色。系统提示信息一直保持在界面左侧，以蓝色区域表示。

参考 3.2.4 节中对于易用性的需求，所有按键均采用较大的尺寸设计，按键指示文字置于左上方，采用黑体加粗。

具体规格如图 5-16 所示：

图 5-16　操作按键的设计图

上图即实际按键的尺寸，用户在使用触摸屏进行操作的时候可以准确的点击并得到明显操作反馈，避免操作失误。

5.5 运行环境建设

整个平台涉及多个不同的服务系统，各自面向不同的业务，运行在不同的网域中，所以服务器搭建的过程相对较为复杂，以下逐个说明。

5.5.1 服务器结构

平台的物理架构由各个服务的物理边界进行划分，按照功能区域划分成不同的服务器组，包括核心服务器组和业务辅助服务器组。

不同的业务运行在不同的服务器上，这些服务器共同构成了平台的硬件基础。服务器之间按照业务的逻辑关系划分边界，提供服务接口或者同时提供独立的业务能力。

核心服务的服务器包括了支付服务、业务代理服务和调度服务。从图 3-4 和图 4-2 可知，各个业务都需要支付服务，所以从逻辑上设计的业务代理服务器应当是每个业务对应一个业务代理服务器，这样可以确保业务之间的隔离和可靠性，也符合 3.2.4 节对于安全性的考虑。所以业务代理服务器不只是一个服务器，而是一个服务器群，可以一个业务一个服务器，也可以将使用较少的服务合并在同一个物理服务器上。通过这样的配置，可以充分利用服务器的资源，也可以为负载高的业务选择单独的服务器以满足需求。

同样的，支付服务器也是一个服务器群，可以为每个支付业务设置一个服务器，也可以将使用较少的服务合并在一个物理服务器上。

整体的物理架构如图 5-17 所示：

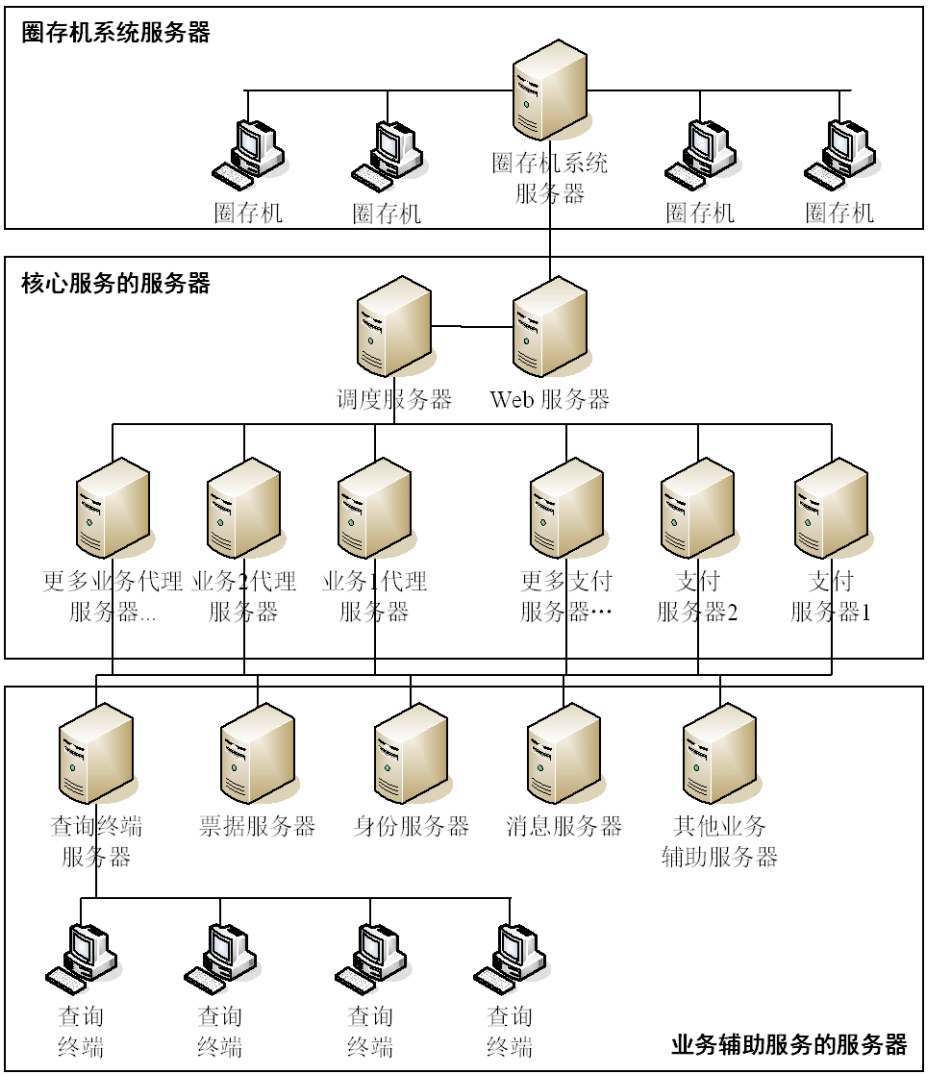


图 5-17　校园个人支付平台的物理架构图

支付服务需要通过图 3-3 所示的 SDK 来连接校园卡的交易支付服务器，由于交易服务器不止一个，所以基于和业务代理服务器同样的原因，支付服务也采用了多个支付服务器的设计方案，当一个支付服务器无法满足多个业务需求时，可以基于业务将一个单独的支付服务器专门配置给一个业务，进而也解决了 3.2.4 节对于可靠性的考虑。由于图 3-2 所示的圈存系统在面向用户服务方面采用了 B/S 架构的设计，所以为了利用这种方式，调度服务器也通过一个 Web 服务器与圈存机系统服务器相连，这样圈存机可以无缝的从圈存机业务切换到平台提供的支付服务中。业务辅助服务均采用单个服务器实现各自的服务，其中终端服务使用了 B/S 架构。

5.5.2 网络结构

根据图 5-17 对于整体服务器物理结构的设计，可以得到整体的网络拓扑结构如图 5-18 所示：

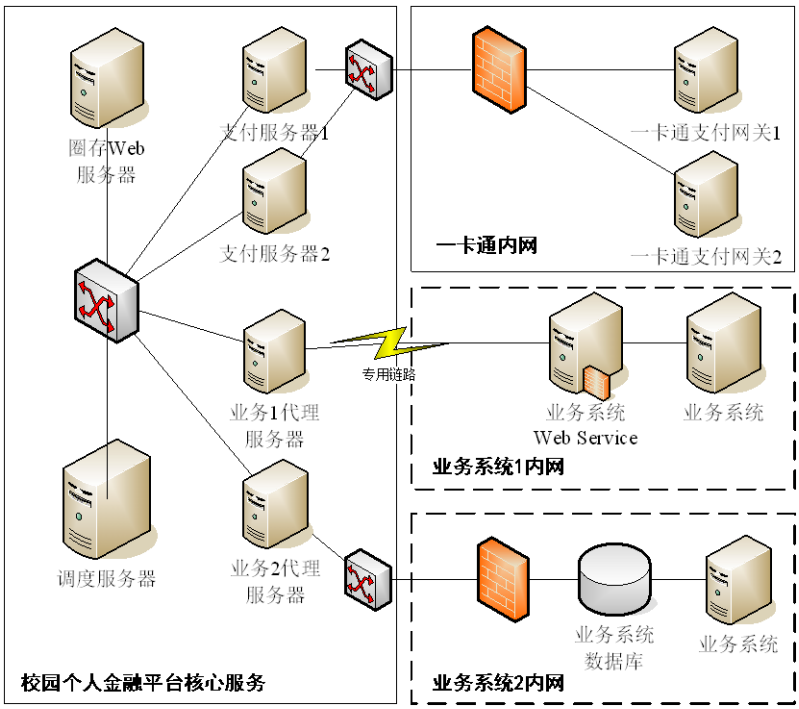


图 5-18　硬件服务器的拓扑结构图

如图 5-18 所示，核心的服务器包括支付服务器，业务代理服务器，接口调度服务器。根据架构设计，所有的服务在逻辑上都是可以分离的，在硬件上也可以分离，所以根据这个特点，不同的服务运行在不同的服务器上。支付服务器通过防火墙连接到一卡通支付网关，业务代理服务器根据业务的不同也通过防火墙连接到不同的服务器上，并且各自运行在完全不同的网段，接口调度服务器直接与这些服务器相连。

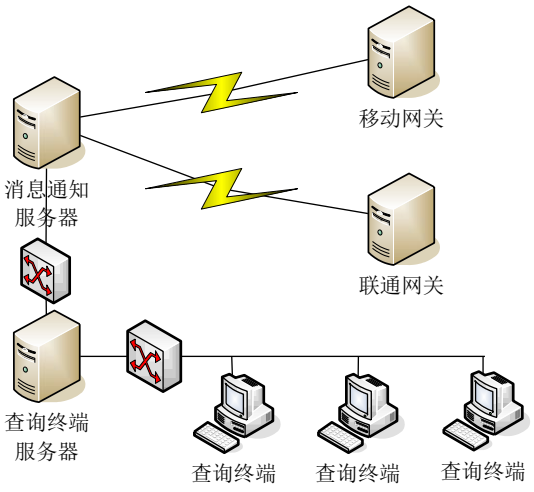


图 5-19　业务辅助服务器的拓扑结构

如图 5-19 所示，各个不同的服务都运行在各自独立的服务器上。CAS 客户端部署在使用了 CAS 服务的应用系统中，通过代码级别的调用实现认证功能。如在信息门户系统、综合信息服务系统等系统中都有使用。照片服务器、消息通知服务器以及查询终端服务都运行在独立的服务器上。

5.5.3 软件平台搭建

核心平台分别采用.Net 技术、Java 技术和 Python 技术实现，如表 5-1 所示：

表 5-1 核心平台服务器的软件环境

服务平台	操作系统	服务器	数据库	其他
支付服务器	Windows Server 2003	IIS	MySQL	无
业务代理服务器 1	Windows Server 2003	IIS	MySQL	无
业务代理服务器 2	RedHat Linux	Apache	MySQL	无
调度服务器	CentOS Linux	Python	MySQL	无

支付服务器安装了 Windows Server 2003 操作系统和 IIS 服务器软件，数据库采用 MySQL，并为后端的一卡通网关安装了第三方开发接口。

一个业务代理器安装了 Windows Server 2003 操作系统和 IIS 服务器软件，数据库采用 MySQL，并配置了连接业务系统数据库的组件库。

一个业务代理服务器安装了 Red Hat Linux 操作系统和 Tomcat 服务器软件,数据库采用 MySQL，并为该服务器配置了双网卡，其中一个网卡通过虚拟链路直接连接到外地的服务器的 Web 服务上。

调度服务器安装了 CentOS Linux 操作系统，数据库采用 MySQL，安装了 Python 实现 HTTP 服务器的框架 Twisted 和实现 Web 服务访问的组件 Suds。

另外，所有的服务器都安装了 Nagios 监控系统，当系统或者服务发生异常或者错误时，监控服务会以短信形式通知管理员进行排查。

其他业务辅助服务的服务器与终端的软件平台配置如表 5-2 所示：

表 5-2 业务辅助服务的服务器与终端的软件平台配置

服务平台	操作系统	服务器	数据库	其他
消息服务器	Ubuntu Linux	Python	MySQL	无
查询终端服务器	CentOS Linux	Tomcat	Oracle	无
查询终端	Ubuntu Linux CentOS Linux	Apache	无	热敏打印机
圈存机终端	Fedora Linux	无	无	热敏打印机 专用读卡器 银行卡读卡器
PC 刷卡缴费终端	Windows XP	无	无	针式打印机 专用读卡器

除了 PC 刷卡缴费终端采用 Windows XP 操作系统，其他均采用 Linux 操作系统，只在发行版上有所区别。

5.6 本章小结

本章针对目前已有的业务过程和进一步的业务需求，采用相应技术架构，根据设计的架构实现了平台的读写卡、业务代理、调度服务、Web Service、各项业务辅助系统功能。安装了核心服务所在的服务器，包括支付服务、业务代理服务和调度服务所运行的服务器，并配置了所有服务器的软件硬件环境。

6 支付平台的应用、测试以及运行情况

6.1 基于支付平台实现的应用

目前基于支付平台，已经实现了多个缴费业务的应用系统，包括圈存机缴费系统、PC 刷卡缴费系统、学杂费缴纳通知系统、毕业生照片采集缴费系统和查询终端系统等多个业务的缴费应用系统，以下就部分业务进行说明。

6.1.1 圈存机缴费系统

圈存机缴费是利用校园个人支付平台上的多个服务构、工具和接口，采用 SOA 思想和相关技术实现的一个自助交费服务。在这个系统中，采用了平台提供的支付服务、业务代理服务，调度服务以及圈存机缴费环境和相关业务辅助服务。如图 6-1 所示：

(a) 操作界面

(b) 缴费业务选择

图 6-1 圈存机缴费系统

目前实现的应用包括网络开户收费、网络流量收费、虚拟 ADSL 业务收费，都可以在圈存机缴费系统上实现。

6.1.2 毕业生照片采集缴费系统

除了用校园卡进行缴费以外，部分业务根据需求也采用收取现金的方式进行工作。比如在毕业生照片采集缴费的工作中，收取现金后开具收据并手工登记个人基本信息，带收集完成后再进行汇总计算，并在现场凭收据再次填涂机读卡登记信息排队照相。所以利用校园卡的身份识别功能和配发的通用读卡器，可以将除现金收取以外的所有手工操作变为刷卡处理。

基于平台提供的业务代理服务和业务辅助服务提供的各项服务功能，实现了刷卡登记收费和刷卡查询收费状态。其中收费功能使用了业务代理服务收费确认的接口，并且使用了基于 PHP 的 CAS 客户端实现身份认证，使用卡号学号识别转换工具。在拍

照排号的时候也使用了业务代理的支付查询功能判断是否缴费，同时也使用了卡号学号识别转换工具，可以刷卡排号，系统会自动调用平台接口实现查询，使排队过程不再需要手工录入任何信息，提高了工作效率。

在系统中使用了服务器端识别卡号的技术，使得在一个输入框内兼容两种不同的输入，在收费和排队中都使用了这项技术，如图 6-2 所示：

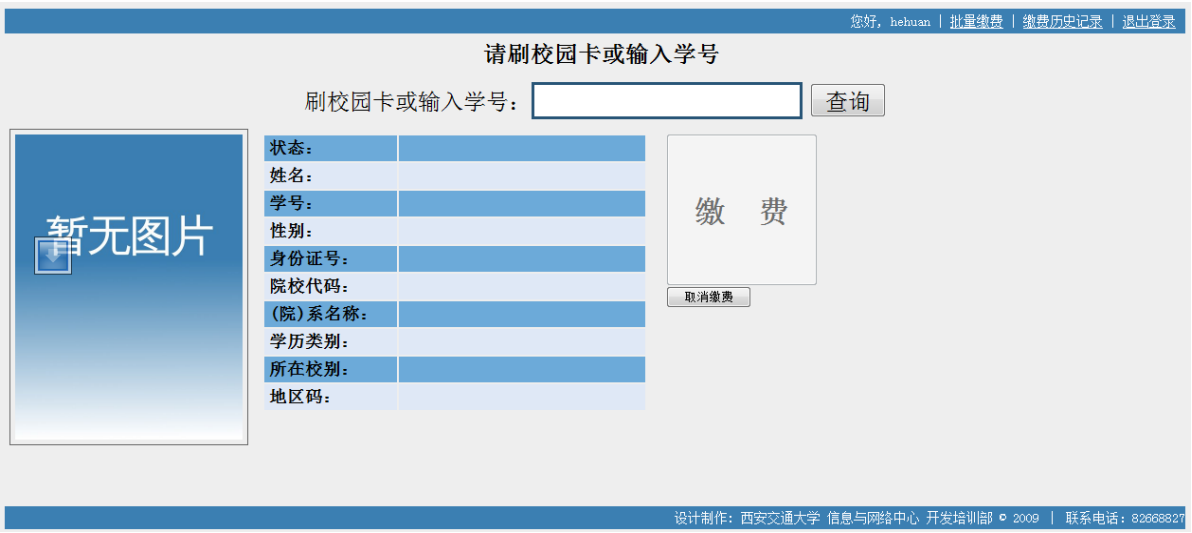


图 6-2　收费登记界面

使用该技术，在收费的时候可以刷卡或者直接输入学号，由服务器端来识别用户的输入并返回学生的信息。在排队的时候也使用这个技术，可以刷卡或者直接输入学号，这样学生可以在没有携带校园卡的情况下使用学生证或者自己报学号来排队。学生刷卡后系统自动会将信息添加到排队名单中，或者输入学号也可以实现。

操作界面如图 6-3 所示：

图 6-3　排队界面

6.2 支付平台测试

6.2.1 测试计划

由于开发资源的限制，特别是开发人员数量的限制，测试的种类和规模都较少。为了确保基本的软件质量，测试主要包括

单元测试：由开发人员自己在开发时完成，对所用关键函数进行测试。目标是确保函数按照预期返回计算结果。

功能测试：根据测试用例对系统的功能进行测试。目标是确保功能按照预期工作。

6.2.2 单元测试

在开发过程中，采用相应语言的测试框架，对部分重点函数进行了单元测试，具体如表 6-1 所示：

表 6-1　单元测试项目

语言	测试框架	测试对象	测试数量
Java	JUnit	测试业务代理的主要函数： callQueryUserInfo() callChargeAccount()	合计： 20 个 全部通过
C#	NUnit	测试业务代理的主要函数： getCheckByStuNo() confirmPayment() 测试一卡通数据库的主要函数： doConsume()	合计： 30 个 全部通过
Python	PyUnit	测试调度服务器： getBusinessInfoByStuNo() pay() confirmPay()	合计： 30 个 全部通过

以 Java 的测试为例，以下是部分测试代码：

```
[TestMethod]
public void callQueryUserInfo_Normal_Test(){
        BusAgent agent = new BusAgent();
        UserReturnPack expect = new UserReturnPack(“213456789”);
        Assert.AreEqual(expect, agent.callQueryUserInfo (“213456789”));
}
```

上述测试对 callQueryUserInfo()查询用户信息的功能进行了基本的测试，确保可以查询到指定帐号的用户信息，类似的还有异常数据测试。在开发期间，已经对所有函数进行测试，确保能够通过所有的单元测试。

6.2.3 功能测试

该测试采用测试数据，实际模拟业务的缴费过程，确保涉及到的功能都是正常可用，每个业务都采用以下测试用例进行基本的功能测试，如表 6-2 所示：

表 6-2　缴费功能基本测试用例

测试功能	业务缴费	
测试目的	确认支付过程正常，可以完成业务的办理	
是否通过	输入与期待	
	点击屏幕上的业务	进入业务画面提示插卡
	插入校园卡	显示指定业务的账单信息
	选择缴费	扣费并提示已通知系统
	查询业务状态	提示费用已缴清

支付平台提供的主要功能即是为业务系统提供缴费功能，平台的所有测试也是基于该基本测试用例。测试过程在圈存机上进行，首先点击进入业务界面，系统提示要求插卡，插入校园卡后，系统读取卡信息并查询业务代理服务器，将指定帐号和业务的的账单信息显示在圈存机上。用户选择缴费后，系统扣费并通知业务系统用户已经缴费，并提示用户。最后用户可以查询到业务状态，提示费用已经缴清。

所有基于该平台的缴费应用均已通过此测试用例。

6.2.4 测试总结

系统在开发过程中通过所有了所有的单元测试，并通过了功能测试。

另外，为了确保系统能够在生产环境下承受并发访问的要求，进行了一定程度的压力测试，平台能够在两倍于实际终端数量的并发访问下保持正常的响应时间。在试运行期间，对平台的各个相关服务，也进行了集成测试，确保整体能够正常运行。

6.3 平台及应用运行情况

截至 2011 年 12 月 31 日，基于平台的各个应用的使用情况如表 6-3 所示：

表 6-3　各缴费业务运行时长

业务名称	投入使用	累计工作时长（工作日）
网络开户费（PC 刷卡）	2008.9~2009.7	200
网络开户费（圈存缴费）	2009.9 至今	852
网络流量费（圈存缴费）	2009.9 至今	852
ADSL 充值（圈存缴费）	2010.9 至今	487
毕业图像采集（现金登记）	2009.10, 2010.10, 2011.10	每次使用 10 日，共 30 日

根据运行时长和服务器数据库记录的缴费流水，可以统计出平台各项缴费业务的服务情况如表 6-4 所示：

表 6-4　各项缴费业务的运行情况

业务编号	业务名称	累计服务人次	日平均服务人次	日高峰服务人次
1	网络开户费（PC 刷卡）	5576	27	220
2	网络开户费（圈存缴费）	18265	21	287
3	网络流量费（圈存缴费）	60079	70	328
4	ADSL 充值（圈存缴费）	20689	42	192
5	毕业图像采集（现金登记）	11231	374	520

其中业务 1 在业务部门的工作时间使用，自圈存缴费运行后停止使用。运行期间，平均 20 秒内完成一笔业务办理的查询、扣费、打印票据的工作。由于该业务具有明显的集中性，在开学一个月内是办理高峰，在高峰期通常采用三台定制的 PC 协同工作，已经将往年业务办理时的高峰排队明显缩短。

基于平台的圈存机缴费服务上线后，无论工作日或者假期都在线运行，自 2009 年 9 月起，截至 2011 年 12 月 31 日，累计运行了业务 1 迁移至圈存机进行办理，同时还将业务 3 也放在圈存机上办理，用户平均 15 秒完成一次业务过程。办理过程完全自助，服务柜台从此不再有排队产生。业务 4 与以往的欠费缴费不同，是充值方式的，所以无论何时都可以进行办理。

业务 5 没有采用该平台的缴费功能，但使用平台提供的辅助功能同样可以实现缴费登记。此业务每年只运行 10 天进行收费和排号，短时间内会产生大量交易，所以日平均服务人次较高。

所有基于平台的交易，都提供了基于信息门户的查询和圈存机的查询。前者可以实现交易流水的详细查询，后者可以提供交易流水也可以提供当前业务状态的查询。除此以外，查询终端也提供了对于网络开户业务和拍照效果检查服务的支持，用户可以在查询终端机上刷卡查询这些信息。

学费缴纳通知服务运行以来，在缴费最为集中的 3 日内累计发送短信超过 20000 条，运行 1 个月后，累计发送短信超过 30000 条。毕业生图像采集排队服务在每年的活动中，平均每日完成接近 4000 人次的排队服务。

6.4 本章小结

本章介绍了基于支付平台所实现的多个缴费系统，并提供了这些系统的运行介绍。另外对支付平台的测试过程进行了说明，提供了测试计划以及单元测试和功能测试，并总结了测试结果。最后结合已经上线的缴费系统，介绍了该平台的实际使用情况。

7 结论与展望

7.1 结论

随着教师学生对在校期间金融管理的需求的提高，学校业务服务能力与管理平台都需要随之有相应的改进。由于学校各个部门之间具有相对的业务独立性，在信息化发展水平上各有差异，在系统构建上各有标准，特别是在与缴费支付相关的业务过程中，存在着极大的差异。所以迫切需要采用一个统一的支付平台和相关技术解决业务系统中没有足够支付能力、缴费难以查询统计等多种与校园内金融活动相关的业务问题，不仅为业务提供了统一的支付手段，也便于进行信息统计和管理。

本文主要工作包括：

（1）通过对校内异构业务系统以及现有数字校园平台的分析，提出了一个基于SOA 思想和相关 Web 服务技术的校园个人支付平台的解决方案，并结合具体的业务流程，抽象出通用的支付模型，并针对这个模型，设计了平台的逻辑架构和数据架构，并分析了实现架构和物理架构，提出了系统作为框架的功能结构。该平台的架构设计不仅解决了当前支付过程存在的问题，还为将来的系统扩展和新功能提供了支持，满足了校园各项业务的需求。

（2）针对目前已有的业务过程和进一步的业务需求，用统一建模语言描述和分析了实现支付、业务代理和调度服务等核心的服务、支持服务、工具集和以及这些服务所提供的接口，给出了具体的实现技术，并具体实现了这些方案。

（3）按照架构设计，安装了核心服务所在的服务器包括支付服务、业务代理服务和调度服务所运行的服务器，并配置了所有服务器的网络和相关防火墙的策略，安装配置了相应的软件环境。配置了消息服务的服务器和应用部署环境；安装和配置专用格式的针式打印机和套打格式票据，并配置相应的打印服务；定制专用的查询终端设备完成设备精简与打印、触摸屏、读卡器等外设的联调，并安装专用的服务器软硬件平台；定制排号专用笔记本电脑的软件环境；并且统计了所有业务功能的工作运行情况。

（4）对平台进行了单元测试、功能测试和压力测试并总结上线运行以来的运行情况。

7.2 展望

基于目前已经完成的框架，平台将从两个方向继续进行开发。首先是平台框架的进一步完善，除了实现对一卡通支付平台的支持，还可以通过对接第三方接口，实现对网银支付的支持，使平台服务的对象不局限在在校学生，还可以扩展到与学校业务

相关又未入校的群体，如报考学校的学生。

另一个方向是基于平台的应用，除了已有的四个缴费业务和相关业务以外，还可以将更多的可以实现自助服务的缴费业务，同时也可以将现有的周边服务进一步完善，如提供自动的对帐服务，方便业务部门进行帐务的清算；为跨部门业务提供协同工作的基础，如方便研究生院和财务处共同进行补助发放的计算；为教师学生提供更多完全在线的业务办理过程，如考试报名等。

参考文献

[1] 许鑫, 周新宇. 一卡通建设中的问题和新思路[J]. 中山大学学报(自然科学版), 2009,48: 22-24

[2] 王永明. 校园一卡通系统的基础平台建设和应用功能分析[J].智能建筑与城市信息, 2009,1: 101-105

[3] 王春雁, 白雪. 高校校园卡系统应用现状及趋势浅析[J]. 中国教育信息化, 2010,11: 83-87

[4] 万梅. 基于“一卡通”第三方校园支付平台的研究[J]. 电脑知识与技术, 2011,7(15): 3528-3530

[5] 孙琪华. 构建校园支付通收费管理平台的设想[J]. 会计之友, 2011,11: 118-120

[6] 谢靖. 基于银校互联平台的学生收费系统设计与实现[J]. 软件, 2011,32(6): 49-51

[7] 许佳, 郭玉梅, 姚妍妍. 基于 SOA 的高校学生管理信息化进程研究[J]. 中国教育信息化, 2009,11: 11-13

[8] Michael Huhns, Munindar P. Singh. Service-Oriented Computing: Key Concepts and Principles[J]. IEEE Internet Computing, 2005,5: 75-80

[9] 赵亮, 姚青. 基于 SOA 的可变业务流程管理系统[J]. 信息化技术, 2010,31(24): 5244-5247

[10] 李雯, 王慧. 基于 SOA 的 WSDL 服务契约架构设计与实现[J]. 福建电脑, 2008(5): 149-150

[11] 赵颜, 黄永中. 基于 SOA 的高校信息化进程的资源整合研究[J]. 中国教育信息化, 2008,1: 22-25

[12] 唐旭华, 邹峥嵘. 基于 RESTfulWebServices 的空间数据共享[J]. 测绘科学, 2010,35(4): 122-124

[13] 姜峰, 范玉顺. UDDI 与 Web 服务扩展元数据拓扑映射[J]. 清华大学学报(自然科学版), 2009,49(7): 1080-1084

[14] 徐烨, 曾浩. 基于 WS-Security 的 SOA 安全协议框架设计[J]. 合肥工业大学学报(自然科学版), 2011,34(4): 506-508

[15] 肖波, 于世伟. 刚柔并济，SOA 安全策略最佳实践系列[EB/OL]. [2010-01-25] http://www.ibm.com/developerworks/cn/webservices/1001_xiaobo_soasecurity/index.html

[16] 曾轩, 孔志印, 汤光明. SOA 架构下的强制访问控制模型研究与实现[J]. 计算机工程与设计, 2011,32(12): 3983-3988

[17] 霍泰稳. 乱花渐欲迷人眼——回顾动态语言的2007[J]. 程序员. 2008,2: 50-51

[18] 蒋崇武，刘斌，王轶辰. 基于 Python 的实时嵌入式软件测试脚本[J]. 计算机工程, 2009,35(15): 64-67

[19] 金培莉, 岳江红, 曹东亚. 校园一卡通网络支付平台研究[J]. 实验技术与管理, 2011,28(4): 307-310

[20] 何亮, 王纯. 基于 SOA 的第三方移动支付平台的设计[J]. 北京工商大学学报(自然科学版), 2009,27(2): 50-54

[21] 贺欢, 韩博, 李一鸣等. 基于一卡通圈存机的外挂式缴费系统的设计与实现[J]. 实验技术与管理, 2011,28(5): 229-231

[22] Wen-Hsien Tsai, Bor-Yi Huang, Jau-Yang Liu, et al. The application of Web ATMs in e-payment industry: A case study[J]. Expert Systems with Applications, 2010,37: 587-597

[23] 徐遇霄. 基于 Java 的 SOA 分层研究与设计[J]. 舰船电子工程, 2010,30(6): 124-129

[24] Hassan Gomaa, Koji Hashimoto, Minseong Kim, Sam Malek, Daniel A. Menascé. Software Adaptation Patterns for Service-Oriented Architectures[J]. Proceedings of the 2010 ACM

Symposium on Applied Computing, 2010,10: 462-469
[25] Guoli Li, Vinod Muthusamy, and Hans-Arno Jacobsena. Distributed Service-Oriented Architecture for Business Process Execution[J]. ACM Transactions on The Web, 2010,4:
[26] 李颂华, 陶丽红, 高栋. 基于SOA架构的物流信息系统的相关技术研究与实现[J]. 北京科技大学学报, 2009,1-31(135)
[27] 刘丹. 电子商务支付平台的安全问题刍议[J]. 商业经济, 2011,9: 56-57
[28] 崔俊杰. 网络安全的关键技术[J]. 煤炭技术, 2011,30(2): 103-105
[29] 李志芳, 潘军, 孙辉. SSL 协议及 WEB 安全实现[J]. 煤炭技术, 2011,30(5): 116-117
[30] 谷松, 张月琳. 统一身份认证在数字化校园中的应用[J]. 中山大学学报(自然科学版), 2009,48(3): 256-259
[31] 胡建鹏. 基于 Portal 的统一身份认证与系统集成研究[J]. 计算机工程与科学, 2010,32(12): 30-33
[32] 胡加艳, 陈秀万, 陶迎春等基于室内外定位的校园 LBS 研究[J]. 计算机工程, 2010,36(8): 254-257
[33] 王朗. 框计算模式对图书馆机构的影响分析[J]. 图书馆论坛, 2011,31(5): 79-81
[34] 杜丰, 邸德海, 杨洁. Portlet 与 Web Service 实现校园门户的信息聚合[J]. 中山大学学报（自然科学版）, 2009,3-48(293)

致　谢

我要感谢我的导师，他为我提供了参与众多实际项目的机会，每个项目的开展和完成都离不开他对我工作的支持和肯定。在项目遇到重大难题时，他帮助我协调各方人员和资源，解决问题，使项目能够继续推进。在开题、中期以及论文写作方面，也教会我如何构思、编排、设计内容架构。在做项目设计和实现等方面，总能给我更多的思考以让工作更加完善。

感谢韩博老师和李一鸣老师，他们有着深刻的技术功底和广泛的知识领域，在工作中经常帮助我学会解决各种问题，教会我梳理复杂的业务逻辑，架构复杂问题的技术方案，以及各种工作中应当注意的良好习惯。也感谢他们在生活上对我的各种帮助，使我能够得到很多工作以外的收获。

感谢张亚娟老师对我在数据库方面的指导和帮助，使我学会很多数据库方面的指示和技巧。感谢杨帆老师在 Java 框架方面对我的帮助，杜丰老师在.Net 开发和校园卡开发技术方面对我的支持，以及杨洁老师和谭薇老师在日常事务方面对我的帮助和提点，还有刘宏磊老师、韩磊老师、朱晓芒老师对我的帮助！

也感谢在工作期间，其他部门的同事、还有研究生院、教务处、就业中心、财务处、档案馆、校医院、保卫处、软件学院等多个单位的老师们对我的帮助和支持！

最后，感谢我的父母一直以来对我的无私奉献和支持。

攻读学位期间取得的研究成果

[1] 贺欢, 韩博, 李一鸣, 邸德海. 基于一卡通圈存机的外挂式缴费系统的设计与实现[J]. 实验技术与管理, 2011,5-28(40)

[2] 李一鸣, 贺欢. 西安交通大学虚拟校园的设计与实现[J]. 实验技术与管理, 2011, 5-28(40)

[3] 刘宏磊，李一鸣，贺欢，韩博. 掌上迎新系统的设计与实现研究[J]. 中国教育信息化，2012,2(57)

[4] 贺欢（第9完成人）. 数字校园及应用. 陕西省高等学校科学技术奖,二等奖，陕西省教育厅，2011.6

[5] 贺欢（第1完成人）. 西安交通大学学生照片采集刷卡排号系统. 西安交通大学首届创意大赛优秀创意奖，校级，西安交通大学，2010.7

[6] 贺欢（第1完成人）. 西安交通大学 PC 机刷校园卡缴纳网费系统. 西安交通大学首届创意大赛优秀创意奖，校级，西安交通大学，2010.7

[7] 贺欢（第8完成人）. 西安交通大学电子离校系统. 西安交通大学首届创意大赛优秀创意奖，校级，西安交通大学，2010.7

[8] 西安交通大学（作为第1完成人）.西安交通大学学生照片采集刷卡排号系统计算机软件著作权: 中国, 2010SR040581［P］.2010-08-11. www.copyright.com.cn

[9] 西安交通大学（作为第1完成人）.西安交通大学 PC 机刷校园卡缴纳网费系统计算机软件著作权: 中国, 2010SR040582［P］.2010-08-11. www.copyright.com.cn

[10] 西安交通大学（作为第1完成人）.西安交通大学中国移动短信通知发送系统计算机软件著作权: 中国, 2010SR040382［P］.2010-08-11. www.copyright.com.cn

[11] 西安交通大学（作为第1完成人）.西安交通大学圈存机缴纳网费系统计算机软件著作权: 中国, 2010SR040445［P］.2010-08-11. www.copyright.com.cn

[12] 西安交通大学（作为第1完成人）.西安交通大学导师自助填报系统计算机软件著作权: 中国, 2010SR040444［P］.2010-08-11. www.copyright.com.cn

[13] 西安交通大学（作为第2完成人）.一卡通设备监控报警系统计算机软件著作权: 中国, 2010SR040578［P］.2010-08-11. www.copyright.com.cn

[14] 西安交通大学（作为第2完成人）.西安交通大学虚拟校园系统计算机软件著作权: 中国, 2010SR040580［P］.2010-08-11. www.copyright.com.cn

[15] 西安交通大学（作为第3完成人）.交大兵马俑 BBS 站 WAP 应用系统计算机软件著作权: 中国, 2010SR040579［P］.2010-08-11. www.copyright.com.cn

[16] 西安交通大学（作为第4完成人）.西安交通大学职工住房分配货币化补贴计算系统计算机软件著作权: 中国, 2010SR040214［CP］.2010-08-11. www.copyright.com.cn

[17] 西安交通大学（作为第8完成人）.西安交通大学学生电子离校系统计算机软件著作权: 中国, 2010SR040215［CP］.2010-08-11. www.copyright.com.cn

学位论文独创性声明（1）

本人声明：所呈交的学位论文系在导师指导下本人独立完成的研究成果。文中依法引用他人的成果，均已做出明确标注或得到许可。论文内容未包含法律意义上已属于他人的任何形式的研究成果，也不包含本人已用于其他学位申请的论文或成果。

本人如违反上述声明，愿意承担以下责任和后果：

1．交回学校授予的学位证书；

2．学校可在相关媒体上对作者本人的行为进行通报；

3．本人按照学校规定的方式，对因不当取得学位给学校造成的名誉损害，进行公开道歉。

4．本人负责因论文成果不实产生的法律纠纷。

论文作者（签名）：　　日期：　　年　　月　　日

学位论文独创性声明（2）

本人声明：研究生　　所提交的本篇学位论文已经本人审阅，确系在本人指导下由该生独立完成的研究成果。

本人如违反上述声明，愿意承担以下责任和后果：

1．学校可在相关媒体上对本人的失察行为进行通报；

2．本人按照学校规定的方式，对因失察给学校造成的名誉损害，进行公开道歉。

3．本人接受学校按照有关规定做出的任何处理。

指导教师（签名）：　　日期：　　年　　月　　日

学位论文知识产权权属声明

我们声明，我们提交的学位论文及相关的职务作品，知识产权归属学校。学校享有以任何方式发表、复制、公开阅览、借阅以及申请专利等权利。学位论文作者离校后，或学位论文导师因故离校后，发表或使用学位论文或与该论文直接相关的学术论文或成果时，署名单位仍然为西安交通大学。

论文作者（签名）：　　日期：　　年　　月　　日

指导教师（签名）：　　日期：　　年　　月　　日