****

**本 科 毕 业 论 文（设计）**

|  |  |
| --- | --- |
| 课题名称 | 基于微信公众号的发票系统前端设计和实现 |
| 学 院 | 计算机科学与网络工程学院 |
| 专 业 | 软件工程 |
| 班级名称 | 软件161班 |
| 学生姓名 | 李倬帆 |
| 学 号 | 1606100107 |
| 指导教师 | 金政哲 |
| 完成日期 | 2020.05.12 |

教 务 处 制

**基于微信公众号的发票系统前端设计和实现**

**摘要** 本系统的设计主要是面向公司在用户购买产品（如： python课程）之后，方便用户在线进行自助开出公司发票，以方便于后续向有关部门申请补助时停供事实凭证。以及公司对用户购买的课程通过该发票系统进行存根的依据。

本设计前端使用Web网页开发技术，联合使用微信客户端，实现微信在线自助开票入口、查票（在微信卡包中查询），后端使用Golang语言，通过MySQL将数据持久化存储，使用RabbitMQ消息系统进行服务解耦，通过RPC远程调用技术与其他服务通信。在部署上使用Docker+Gitlab-Runne进行自动化部署。

本系统重点实现了微信公众号回复关键字入口、微信授权、领取发票到微信卡包、开票成功推送等功能。

**关键字** 微信公众号； 发票； OAuth2.0授权；模板推送

**ABSTRACT** The design of this system is mainly aimed at the company, after the user purchases the product (such as: python courses), to facilitate users to self-issue online company invoices, to facilitate the subsequent application of subsidies to relevant departments to stop providing fact proof. And the company’s basis for stubbing courses purchased by users through the invoice system.

The front-end of this design uses Web page development technology, combined with WeChat client, implement WeChat online self-service invoice entrance, by replying keywords and check invoice in WeChat card package. The system performs service decoupling and communicates with other service through RPC remote call technology. Use Docker and Gitlab-Runner for automatic deployment.

This system mainly implements the functions of invoice entrance by replying keywords in WeChat public account, WeChat authorization, receipt of invoices to WeChat card package, successful invoicing and other functions.

**KEYWORDS**  WeChat public account; invoice; OAuth2.0; template pushment

目录

[1. 前 言 4](#_Toc40189509)

[1.1 项目的开发目的和意义 4](#_Toc40189510)

[1.2 发票申请流程 4](#_Toc40189511)

[2. 系统开发环境和主要技术 4](#_Toc40189512)

[2.1开发环境 4](#_Toc40189513)

[2.2主要技术 4](#_Toc40189514)

[3. 需求分析 5](#_Toc40189515)

[3.1系统设计目标 6](#_Toc40189516)

[3.2系统目标用户分析 6](#_Toc40189517)

[3.3可行性分析 6](#_Toc40189518)

[3.4核心功能点分析 7](#_Toc40189519)

[4. 系统的总体设计 7](#_Toc40189520)

[4.1系统概要设计 7](#_Toc40189521)

[4.2系统业务模块设计 7](#_Toc40189522)

[5. 系统的详细设计与具体实现 9](#_Toc40189523)

[5.1详细设计简要说明 9](#_Toc40189524)

[5.2项目设计结构 9](#_Toc40189525)

[5.3 模块功能的实现 9](#_Toc40189526)

[5.4公用功能的实现 20](#_Toc40189527)

[5.5 自动化部署的实现 23](#_Toc40189528)

[总结 26](#_Toc40189529)

[参考文献 26](#_Toc40189530)

[致谢 27](#_Toc40189531)

**基于微信公众号的发票系统前端设计和实现**

1. 前 言

* 1. 项目的开发目的和意义

在微服务项目架构之下，每一个服务的终止在解决一个关键的问题。本项目的设计初衷是解决产品的售后发票申请问题­——提供用户自助发票申请的入口地址以及通知用户相应的申请进度。本项目依赖于微信这个拥有巨大用户量的社交平台，既贴合大众，又可以利用微信提供的成熟的技术，在用户使用方面既有方便性又可以降低生产和维护成本。

通过本项目，用户不再需要通过主动地去询问才知道发票最新进度，而是通过官方公众号推送地一条公众号消息而被第一时间通知发票的最新进度。

同时，在本项目中，发票项目只是作为一个使用实例。该功能可以被扩展到更多的项目当中，例如可以使用本项目获得通往其他项目的入口地址，也可以使用该项目来推送其他的通知，比如产品购买成功通知、安全报警通知等等。该项目可以拓展到更多的项目中去，而不仅仅是发票系统。

* 1. 发票申请流程

购买了公司产品的用户，可以微信搜索公司官方微信公众号，向公众号发送关键字（例如：“发票”），公众号在收到用户发送的关键字后将会回复用户消息，该消息含有自助发票申请系统的入口链接（如：https://example.invoiceticket.com.cn），用户点击链接进行跳转，在通过用户授权之后将跳转至发票自助申请页面，用户可以根据的真是情况进行填写并调交，平台会对用户的提交的信息进行审核并承诺7个工作日内给与用户一个通知，通知使用消息推送的形式。如果发票申请成功并得到用户授权，系统将在后台中自动插入电子发票到用户的微信卡包当中。

1. 系统开发环境和主要技术

2.1开发环境

服务器端开发语言：Golang

服务端数据库：MySQL

服务端缓存数据库：Redis

Web 端框架： 微信客户端

容器化部署技术：Docker

自动化部署：CI/CD流水线

2.2主要技术

2.2.1 Web微信客户端

微信客户端使用的用户数量超过10亿个，它是一个同时支持发送语音，文字，视频和图片等信息的社交软件，它还支持微信支付，卡卷，微店，发票等具有商业色彩的功能。这个项目客户端使用微信不仅稳定而且更贴合大众用户，同时也降低了开发的成本，无论是从用户体验出发还是从企业考虑，都是非常理想的选择。

2.2.2 服务端开发语言——Golang

Golang 语言是由Google公司开发的一种静态强类型、编译型、并发型，并具有垃圾回收功能的编程语言。它广泛应用于云领域的开发，在这方面的发展起着不可比拟的作用，例如Docker、Kubernetes、Etcd这些有关于云开发、分布式开发工具都是基于Golang开发的应用程序。Golang具有着类似于C语言的语法特色，但是自己却有着C语言没有的垃圾回收机制，C语言开发者可以更快的进行转型，它也避免C语言的复杂性，不必过分担心内存泄漏等与开发语言相关特性带来的问题。Golang还被称为“云时代的C语言”，它的编译速度快，执行效率高，支持交叉编译，即一份代码可以编译成多个平台的可执行文件，做到了类似于Java的“Write once, run anywhere”。Golang同时还拥有者许多简单、易用的标准库，如http库、IO库、JSON处理库以及文件库等等。

在并发处理上，Golang采用CSP并发模型，通过Channel机制和协程管理机制，提倡“通过通讯实现内存共享”来实现高并发，解决了C++编写网络并发编程、调试困难的问题。

2.2.3 MySQL数据库

MySQL是一种关系型数据库管理系统，在Web应用方面，他是最好的关系数据库管理系统应用软件之一，它具备多种数据处理功能：事务功能，数据库备份恢复功能，非关系型（JSON），全文本数据处理功能，通过Next-Key，MVCC等技术支持解决事务ACID特性，在并发事务中，可以保证执行性能。

2.2.4 Redis内存数据库

Redis是一个开源的、基于内存、支持持久化的非关系型数据库，正是因为Redis对数据的操作都是基于内存，使用IO多路复用模型，通过事件驱动，成功的实现单线程处理并发的问题，从而避免了并发带来的数据竞争等问题。Redis支持多种数据结构和数据功能。

2.2.5 Docker容器技术

Docker是一个容器虚拟化技术，与传统的虚拟机虚拟化不同的是，Docker虚拟化操作系统层的虚拟化（传统虚拟机虚拟化硬件），因此Docker具有便捷性，可以高效地利用服务器。由于容器地标准化，它可以无视基础设施的差异，部署到任何地方，这也使得使用Docker部署的服务具备更好的隔离兼容。

2.2.6 CI/CD流水线

CI/CD是持续集成和持续部署的简称。使用持续集成可以使代码提交到主分支前自动进行一系列测试，降低了应用引入错误。持续部署是将主分支部署至产品环境。因为这些都是使用脚本自动化进行，可以降低人力资源，从而降低开发成本，提高开发效率。

Gitlab Runner是Gitlab的一个持续化部署的插件。他要求在项目根目录中配备有“.gitlab-ci.yml”文件，在项目提交时，符合配置的runners就会自动执行其中描述的任务，而无需人工执行。

1. 需求分析

3.1系统设计目标

本系统的核心是对接微信和公司内部的接口请求。其他服务在需要创建模板消息、发送模板消息时，无需向微信服务器请求，而是通过向本系统提出请求，由本系统完成与微信的对接工作，并在完成之后返回给服务调用者。提供可以自定义的关键字自动回复功能（发票入口地址获取方式即为向公众号发送申请发票得到的自动回复，在回复中提供了入口）。此外，系统提供了微信发票插卡接口，方便用户在微信卡包中使用和查看发票。

3.2系统目标用户分析

经过分析后，系统中的用户角色被定义为如下：

系统管理员：系统管理员可以录入或修改公众号回复接口功能、可以对推送模板进行录入更改，可以对接入用户进行统计管理。

产品购买者用户：这部分用户主要就是购买了公司产品的用户，该系统要服务的主体对象，可以时刻关注自己发票的申请流程，可以发起发票请求、可以查看发票。

系统用户：本系统是基于微服务架构的项目，其他的服务模块视为本系统的系统用户，可以查看用户的授权信息，修改用户的权限，发起通知事件等。

3.3可行性分析

a. 业务平台可行性：为落实“互联网+”的发展，提高会计信息的传递效率，节约企业成本等，十二届全国人大四次会议将“推行电子发票”正式写入“十三五”规划纲要。这说明开具电子发票的必要性。发票作为会计账务的重要屏障，它可以方便公司对于公司人员的衣食住行等消费报销提供便利，电子发票作为传统纸质发票的替代品，适应了时代的变化。微信平台拥有着10亿的用户数量，无论从技术和用户量都可以了解到其实力不可小觑，借助这个强大的平台，可以更加方便地进行系统开发同时给广大用户提供便利（用户可以不用为了申请发票而特别地去安装注册一个软件，而是在原有的、日常经常使用的应用中进行），从而赢得了用户的内心接受程度。这不管是在站在公司角度还是站在用户角度，都做足了充分的考虑。因此该项目是可行的。

b. 技术可行性：本系统采用微服务架构、微信客户端、微信公众平台API调用。微服务是一种开发软件的架构和组织的方法，软件之间通过明确定义的API进行通信的小型独立服务组成。这种架构使应用程序更易于拓展和开发，从而缩短新功能的上市时间。使用微服务架构的应用体系具有敏捷性、灵活拓展、轻松部署、技术自由，代码重复使用等特性，这些使得每个微服务系统可以拥有不同的开发途径，语言但是最后可以协调在一起地对外进行服务。微信公众平台提供的API调用经过时间的检验，可以安全地、高可用地提供服务，通过简单的Restful接口让开放人员可以方便快捷的调用。系统使用Golang进行开发，对外通过HTTP进行交互，服务之间通过RPC远程调用协议，通过Redis作为缓存。

以上的技术点是当前较为热门、常见的技术，对于计算机专业的学生应该熟练、掌握其基本原理并用于真实的开发当中。

c. 经济可行性：依附于公司原有的微服务架构体系，本系统是对原有的微服务体系的一个功能拓展，并不需要花费很大的成本从零到一建立一个微服务体系，作为一个单独的服务进行开发，可以与其他服务隔离开，运行维护成本都得到有效的降低。经济投入只有初期的开发投入成本和后期的维护成本，但是可以减少更多的人工服务成本并提高用户的服务质量和服务效率，从长远上讲可以为公司减少额外开支从而获得更多收益。

3.4核心功能点分析

1. 公众号回复关键字入口

创建关键字回复接口：创建一个创建关键字接口命令，输入关键字信息，设置回复信息类型，不同类型的回复关键字信息，需要填写不同的信息，如回复音乐消息类型，需要填写回复音乐的ID。

更新关键字回复信息接口：对现有的关键字的修改可以先删除关键字的回复信息，然后重新添加一个关键字。

2. 发票申请成功推送消息模板管理

模板消息的模板使用数量有一定限制。并且模板的种类在微信模板库中已经确定下来。系统可以做的是根据模板类型实例出每一个模板对象。

创建模板信息：微信模板库中提供了模板类的一个ID，根据这个ID我们可以在微信服务端实例化一个模板消息对象。

查看模板信息：系统用户可以通过模板ID获取自己申请的模板信息的具体内容，根据返回的模板信息中，携带的example信息的到推送模板消息需要的信息。

推送模板信息：输入需要推送的模板ID，和通过查看模板信息获取的需要填写的相关参数，向本系统提出推送请求。

3. 微信卡包查看发票

插卡：在发票申请成功之后，会收到模板推送消息，通过该消息下的一个接口，用户可以授权领取微信发票，授权之后系统会收到一个事件消息，系统根据这个消息将发票插入到用户的卡包当中。这张电子发票可以提供给用户作为有效凭证进行报销等的用途。

4. OAuth2.0授权登录认证

用户使用电子发票自助申请系统需要使用OAuth2.0进行授权才可以进行后续的操作。OAuth2.0使得用户可以在不暴露微信账号密码的情况下，本系统可以获取微信用户的一部分信息进行身份识别。

1. 系统的总体设计

4.1系统概要设计

本系统主要分为前端调用、后端逻辑处理、数据存储三个部分。

前端调用：通过微信客户端，Web页面，向用户展示业务信息，用户根据页面信息和自己的需求，在系统的指引下完成操作，最后通过HTTP请求到后端逻辑处理部分进行处理，通过JSON协议将数据进行打包传递。

后端逻辑处理：后端逻辑处理提供前端处理HTTP协议的API，通过监听HTTP请求，对前端发起的请求分发到特定的处理器处理，如果涉及数据交互讲调用数据存储部分进行操作；后端逻辑处理还提供RPC接口，提供给后端其他服务调用，传递如用户信息等涉及业务的信息。

数据存储：本系统使用缓存数据库Redis和关系型数据库MySQL存储发票请求结果、系统数据。

4.2系统业务模块设计

4.2.1 微信公众号回复接口管理模块

产品购买者用户可以通过在微信公众号回复关键字信息获取某个微信功能的接口地址，通过点击该接口进行后续微信授权，如果通过将可以进行后续的业务操作。系统用户可以新添加回复接口关键字，并修改和更新本系统用户设置的回复关键字信息，系统管理员可以新添、修改和更新回复关键字信息。

对于新添加微信公众号回复接口，按回复类型可以分为6种可能的消息：文本，图片，语音，视频，音乐和图文。对于公共基础部分系统自动完成，系统用户和系统管理员（以下统一成为用户）只需要按不同类型的定制功能进行填写。

文本：填写回复的文本信息

图片：填写图片ID，需要用户上传图片至系统，系统会返回图片ID。

语音：语音消息ID，需要用户上传语音消息至系统，系统会返回语音ID。

视频消息：

视频ID，需要用户上传视频至系统，系统会返回视频ID。

视频标题

视频消息描述

音乐消息：

音乐标题

音乐描述

音乐链接

音乐高质量播放链接

媒体ID（缩略图），需要用户上传视图片到系统，系统返回视频ID

图文消息：（支持回复图文消息为1）

图文消息信息

图文消息标题

图文消息描述

JPG、PNG图片链接，推荐使用的分辨率为360\*200/200\*200

跳转链接

特别的，当设置的用户是系统管理员时，还需要指定本次设置的回复功能提供给哪一个服务进行使用。

4.2.2 微信模板推送管理模块：

注意：在使用本功能之前必须确定是商业公众号、组织号。个人订阅号和未认证的公众号是不能使用该功能的。

微信在使用模板之前，会要求用户填写公众号的使用行业。在设定行业之后，微信会提供相对应行业的模板库，系统用户只能在这个模板库中选择一个模板进行实例化，得到模板ID。为提供方便，本系统提供收集官方微信模板库，系统用户可以通过调用API获得，并选择其中一个模板库进行实例化。微信模板的使用是有限制的，并不能无限使用，需要符合微信官方的“模板消息运维规范”。

模块按功能划分：

微信模板申请：

提供模板库中的模板ID

模板消息推送：

需要推送的用户ID

模板跳转链接，可选

模板插值数据信息，可选属性有字体颜色

如果想要跳转到小程序，需要提供小程序的信息（小程序ID，跳转页）

推送的模板ID

模板撤销：

模板ID

4.2.3 微信发票插卡模块：

当微信用户在收到发票申请成功的通知之后，点击通知跳转的页面当中，提供“领取发票到微信卡包”。在收到用户的授权操作之后，系统将调用相关的操作，在用户的微信卡包的发票栏目中插入电子发票。

该功能正常情况下，是被动调用的。不需要系统用户主动调用，在必要的时候可以通过API进行调用，但需要系统用户提供完整的发票信息。

1. 系统的详细设计与具体实现

5.1详细设计简要说明

在这一章，我将会对系统的多个核心功能模块进行详细设计以及会附带有相同功能的伪代码展现，将其实现原理进行说明。本设计中实现的主要功能包括微信公众号回复接口管理模块，微信模板推送模快功能，微信发票插卡模块等，同时会对构成系统的基础构件进行介绍。

系统中使用Restful API时统一使用JSON进行数据交互，

5.2项目设计结构

项目采用MVC的经典设计模式，即将系统的操作一分为三请求API Controller，前端渲染（微信客户端），数据库操作模块。采取前后端分离的形式进行开发。后端架构中还配备有基础构件，比如缓存设计，日志记录系统，以及在不同服务之间进行数据传递的数据结构兼容系统。

项目包结构一览表：

|-- adaptor 数据结构适配器库，为了数据在系统中的处理进行透明化

|-- cache 缓存工具实现库

|-- conf 配置文件库

|-- constant 静态数据库，包含业务错误码等静态数据

|-- controller 业务逻辑处理库，即API Controller

|-- logger 日志控件库，提供方便易用的日志功能

|-- model 数据库处理库，即系统对数据库的CRUD等操作

|-- session Session处理库，维护一个用户的连接状态

`-- test 测试工具库

5.3 模块功能的实现

5.3.1 微信公众号回复接口管理模块

本功能设计初衷是为了给产品购买者提供“公司提供的电子发票申请的官方公众号在线自助申请电子发票”的入口，即产品购买者可以通过关注并回复官方公众号之后，通过公众号的自动回复功能获取到电子发票自助申请系统的链接入口。通过公众号入口进行发票系统带来的好处有：1. 非首次进入的用户只要通过微信授权既可使用系统，无需用户登录 2. 首次登录的用户仅需进行用户信息绑定，即可使用系统，无需复杂的注册流程 3. 系统可以减少用户登录注册流程的开发，降低成本。本功能的实现分为：添加关键字到自动回复功能、删除关键字回复功能、在微信公众号中回复关键字的功能。

添加关键字到自动回复的功能：

回复关键字功能可以分为文本、语音、视频、图文消息等类型。当关键字不为功能时，需要调用系统提供的文件上传功能，上传相应的图片、语音消息等。文件上传成功后会返回一个文件ID，接口创建时使用该ID进行传递即可。对于添加不同类型的关键字，使用的是同一个API进行调用。

Restful接口描述

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API名称** | AutoReplyKeyWords | | **方法** | POST |
| **请求路径** | /api/auto\_reply\_key\_words | | | |
| **描述** | 新增关键字到自动回复接口 | | | |
| 权限 | 系统用户、系统管理员 | | | |
| **API params** | | | | |
| **参数** | **是否必须** | **类型** | **描述** | |
| type | 是 | string | 回复消息的类型，可选text, image,voice,movie,music,news | |
|
|
| options | 是 | object | 额外选项，根据type不同，有不同的选项 | |
|
| server\_id | 是 | string | 标识消息的属主 | |
| key | 是 | string | 回复的关键字信息 | |
|  | | | | |
| options | type = text | image | voice | | | |
| id | 是 | string | text: 文本 | |
| image: 图像ID | |
| voice: 语音ID | |
|  | | | | |
| options | type = video | | | |
| id | 是 | string | 视频ID | |
| title | 否 | string | 视频消息标题 | |
| content | 否 | string | 视频消息描述 | |
|  | | | | |
| options | type = music | | | |
| title | 否 | string | 音乐消息标题 | |
| content | 否 | string | 音乐消息描述 | |
| url | 是 | string | 音乐链接 | |
| hq\_url | 否 | string | 高品质音乐链接 | |
| music\_image | 是 | string | 媒体缩略图id | |
|  | | | | |
| options | type = news | | | |
| title | 是 | string | 图文消息标题 | |
| content | 是 | string | 图文消息描述 | |
| image\_id | 是 | string | 图片链接id | url | |
| url | 是 | string | 图文消息跳转链接 | |
| **成功返回(JSON)** | | | | |
| code | 是 | string | "success" 成功 | |

（更多见后表）

（接上表）

|  |  |  |  |
| --- | --- | --- | --- |
| **失败返回(JSON)** | | | |
| code | 是 | string | "not exists" 不存在资源ID |
| "no permission" 权限错误 |
| “no serverID" 服务ID不存在 |
| "bad request" 请求参数失败 |
| "key repeat" 关键字重复 |
| msg | 是 | string | 具体的错误信息 |

当使用其他服务使用RPC调用时，必选参数server\_id无效，server\_id始终为服务调用者的ID。

原则上，本模块没有提供修改自动回复消息的功能，用户若有修改需要用户自动回复的功能，需要先执行删除添加的关键字自动回复信息，然后再次执行添加的功能即可。

删除关键字的自动回复的功能：

上文提到，当需要修改一个已经设定的关键字的自动回复信息或者当特定的关键字不再被需要时，需要提供关键字信息向系统提交删除请求。当提交删除请求为系统用户时，需要server\_id与系统中记录的server\_id一致时才允许删除，否则没有用户没有权利操作。系统管理员则可以对一切关键字进行删除，而无需检查其server\_id。

Restful接口描述如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API名称** | AutoReplyKeyWords | | **方法** | DELETE |
| **请求路径** | /api/auto\_reply\_key\_words | | | |
| **描述** | 删除关键字到自动回复接口 | | | |
| 权限 | 系统用户、系统管理员 | | | |
| **API params** | | | | |
| **参数** | **是否必须** | **类型** | **描述** | |
| key | 是 | string | 回复关键字信息 | |
| Server\_id | 是 | string | 标识消息的属主 | |
|  | | | | |
| **成功返回(JSON)** | | | | |
| code | 是 | string | "success" 成功 | |
| data.options | 是 | object | 返回注册的关键字回复信息 | |
| **失败返回(JSON)** | | | | |
| code | 是 | string | "no keys" 不存在该关键字 | |
| "no permission" 权限错误 | |
| “no serverID" 服务ID不存在 | |
| "bad request" 请求参数失败 | |
| msg | 是 | string | 具体的错误信息 | |

同理，当使用RPC调用时，server\_id是一个非必选的值，即使设置了该ID也是无效的，server\_id始终是服务调用者的ID。

回复关键字自动回复功能

该功能的入口是微信公众号中，用户回复关键字后，由微信服务器一个文本消息的形式转发给本系统，系统收到微信服务器发来的这个消息之后，将查找出该关键字回复的消息类型是文本、语音还是其他的类型，然后根据类型调用相应的类型处理器进行回复。

伪代码逻辑如下

*## 监听微信服务器发来的事件、消息等功能， 本功能是被微信异步调用的，并非服务器主动提出  
## 微信的每一次异步请求会被解析为一个事件  
def listenFromWeChatServer(event)  
 ## 对微信消息进行去重  
 ## 微信在5s内接收不到系统的回复时，会连续发起重试的请求，总共三次，为了避免处理相同的消息，需要进行去重处理  
 extractRepeatEvent(event)  
 ## 文本消息  
 if event.type == "text"  
 then  
 ## 将文本内容传递给文本处理器  
 resp = handleTextEvent(event.msg)  
 endif  
 ## 其他类型的消息处理  
 ## ...  
 ## 将结果返回给微信服务器  
 return resp  
end  
## 文本消息处理器  
def handleTextEvent(keyword)  
 # 根据关键字查找回复的关键字信息  
 textObj, exist = getReplyKeyWords(keyword)  
 ## 回复的关键字不存在，调用默认的消息进行回复  
 if !exist  
 then  
 return defaultAutoReply(keyword)  
 endif  
 ## 根据textObj的类型（即将要回复的消息的类型返回相应的处理器）  
 handler = getAutoReplyHandler(textObj.type)  
 ## 将textObj中的元数据解析为相应的回复类型的消息  
 textObj.Options = handler(textObj)  
 ## 将textObj转化为微信服务器支持的数据类型  
 resp = adaptor.transformToWeChatServer(textObj)  
 return resp  
end*

为了满足方便其他微服务的调用，系统也支持RPC的调用方式，接口的Proto3约定如下

*syntax = "proto3";  
// 查询自动回复消息  
message QueryAutoReplyRequest {  
 // 当keywords为空时，回复所有属主为调用者ID的所有关键字回复信息  
 // 若keywords不为空，则回复特定关键字的回复信息数据  
 repeated string keywords = 1;  
}  
message QueryAutoReplyResponse {  
 // 返回回复状态  
 string code = 1;  
 // 返回回复关键字信息  
 repeated AutoReplyMsg list = 2;  
}  
// 回复消息结构体  
message AutoReplyMsg {  
 // 回复关键字  
 string key = 1;  
 // 关键字回复消息类型  
 string type = 2;  
 // options在使用的是时候需要反序列化JSON，这是一个JSON字符串  
 string options = 3;  
}*

5.3.2 微信模板推送管理模块

模板推送功能用于公众号向用户发送重要的通知，在本系统相关联的业务中，就是向用户推送发票的申请结果。在微信公众号中可以找到模板库，开发者只能在这里面选择一个模板ID进行实例化，得到实例化的ID，用该ID进行微信模板消息推送。推送的模板信息隐藏有跳转连接，用户可以通过点击模板信息，进一步得到更多的相关信息。

导入模板库信息

模板库信息获取方式这里不作叙述，方法多样。模板库信息一般比较稳定，不需要频繁的进行更新修改。该接口导入的信息就是提供实例化模板的源信息。

Restful接口描述如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API名称** | TemplateMsg | | **方法** | POST |
| **请求路径** | /api/template | | | |
| **描述** | 导入模板库接口 | | | |
| **权限** | 管理员 | | | |
| **API params** | | | | |
| **参数** | **是否必须** | **类型** | **描述** | |
| list | 是 | object | 模板列表 | |
|  | | | | |
| list | list是一个对象列表，每个对象包含以下属性 | | | |

（更多见后表）

（接上表）

|  |  |  |  |
| --- | --- | --- | --- |
| **API params** | | | |
| **参数** | **是否必须** | **类型** | **描述** |
| short\_id | 是 | string | 模板库的ID |
| title | 是 | string | 模板名 |
| **查询参数** | | | |
| type | 是 | string | 值为"import" |
| **成功返回(JSON)** | | | |
| code | 是 | string | "success" 成功 |
| **失败返回(JSON)** | | | |
| code | 是 | string | 错误信息短数据 |
| msg | 是 | string | 具体的错误信息 |

创建（实例化）模板信息接口

创建模板需要提供模板类ID，该ID可以通过上述API获得模板类列表，并从中获取一个ID进行实例化。模板类ID不能进行模板推送，只能使用实例化的ID才能进行消息推送。

Restful接口描述如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API名称** | TemplateMsg | | **方法** | POST |
| **请求路径** | /api/template | | | |
| **描述** | 实例化模板 | | | |
| **权限** | 系统用户 | | | |
| **API params** | | | | |
| **参数** | **是否必须** | **类型** | **描述** | |
| short\_id | 是 | string | 模板类ID | |
|  | | | | |
| **查询参数** | | | | |
| type | 是 | string | 值为"add" | |
| **成功返回(JSON)** | | | | |
| code | 是 | string | "success" 成功 | |
| data.template\_id | 是 | string | 模板实例ID | |
| **失败返回(JSON)** | | | | |
| code | 是 | string | 错误信息短数据 | |
| msg | 是 | string | 具体的错误信息 | |

查询每个系统用户实例化的模板

因为通过实例化的模板，系统开发者是看不到模板消息中的具体内容的。（可以通过微信公众号的管理页面查看，但是微信公众号一般由运维维护，故一般看不到模板的消息格式，因此该接口有很大的必要性，可以提供系统开发者更深入的了解模板消息内容）。在模板推送时，需要获悉相应的模板参数，就可以通过该接口进行查询。

Resultful接口描述如下

|  |  |  |  |
| --- | --- | --- | --- |
| **API名称** | TemplateMsg | **方法** | GET |
| **请求路径** | /api/template | | |

（更多见后表）

（接上表）

|  |  |  |  |
| --- | --- | --- | --- |
| **描述** | 查询实例化的模板对象 | | |
| **权限** | 系统管理员、系统用户 | | |
| **查询参数** | | | |
| **参数** | **是否必须** | **类型** | **描述** |
| id | 否 | string | 模板类ID，如果没有指定而返回所有的模板实例ID对象 |
|  |
| **成功返回(JSON)** | | | |  |
| code | 是 | string | "success" 成功 |  |
| data | 是 | object | 实例模板信息列表 |  |
| **失败返回(JSON)** | | | |  |
| code | 是 | string | 错误信息短数据 |  |
| msg | 是 | string | 具体的错误信息 |  |

// 返回成功数据例子如下(JSON)

{  
 "data": [{  
 "template\_id": "rcGL7mkQpFxKKTZkPXsJFawPbnfxBeE7Md79spOxxa0",  
 "title": "领取奖金提醒",  
 "primary\_industry": "IT科技",  
 "deputy\_industry": "互联网|电子商务",  
 "content": "{ {result.DATA} }\n\n领奖金额:{ {withdrawMoney.DATA} }\n领奖 时间: { {withdrawTime.DATA} }\n银行信息:{ {cardInfo.DATA} }\n到账时间: { {arrivedTime.DATA} }\n{ {remark.DATA} }",  
 "example": "您已提交领奖申请\n\n领奖金额：xxxx元\n领奖时间：2013-10-10 12:22:22\n银行信息：xx银行(尾号xxxx)\n到账时间：预计xxxxxxx\n\n预计将于xxxx到达您的银行卡"  
 }]  
}

模板消息推送

当系统用户完成用户的某一事件时，比如发票系统。当处理完发票自助申请，系统用户可以通过本接口向产品用户推送一条消息，告知处理结果。

Resultful接口描述如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API名称** | TemplateMsg | | **方法** | POST |
| **请求路径** | /api/template | | | |
| **描述** | 推送模板消息接口 | | | |
| 权限 | 系统用户 | | | |
| **API params** | | | | |
| **参数** | **是否必须** | **类型** | **描述** | |
| touser | 是 | string | 需要被推送的用户ID | |
| template\_id | 是 | string | 实例化的模板ID | |

（更多见后表）

（接上表）

|  |  |  |  |
| --- | --- | --- | --- |
| **API params** | | | |
| **参数** | **是否必须** | **类型** | **描述** |
| url | 否 | String | 用户点击模板消息要跳转的链接 |
|  |
| miniprogram | 否 | object | 跳转到小程序 |  |
| miniprogram.appid | 是 | string | 小程序识别ID |  |
|  |
| miniprogram.pagepath | 是 | string | 需要跳转的页面 |  |
|  |
| data | 是 | object | 模板对象中需要进行插值的数据结构,是一个Map对象 |  |
|  |
|  | | | |  |
| **查询参数** | | | |  |
| notify | 是 | string | 推送消息结果通知地址 |  |
| **成功返回(JSON)** | | | |  |
| code | 是 | string | "success" 成功 |  |
| data.msg\_id | 是 | string | 本次的推送消息ID |  |
| **失败返回(JSON)** | | | |  |
| code | 是 | string | "no permission" 权限错误 |  |
| “no serverID" 服务ID不存在 |  |
| "bad request" 请求参数失败 |  |
| msg | 是 | string | 具体的错误信息 |  |

Data是一个Map对象，键是我们通过查询模板信息中，插值表达式中{{XXXX.DATA}}的XXXX，值是一个对象，对象定义如下

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **是否必须** | **类型** | **描述** |
| value | 是 | string | 需要插值的值 |
| color | 是 | string | 以#开头的6位色值，控制插值字体的颜色 |
|  |

对于查询参数notify，系统会对成功发出模板推送请求的结果进行异步通知推送结果。请求的地址就是指定的notify。要求使用Restful的POST方法，请求结构描述如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API名称** | ResultCallback | | **方法** | POST |
| **请求路径** | 系统用户指定 | | | |
| **描述** | 异步通知结果 | | | |
| **权限** | 无，由系统主动请求 | | | |
| **API params** | | | | |
| **参数** | **是否必须** | **类型** | **描述** | |
| msg\_id | 是 | string | 推送的消息ID | |
| status | 是 | string | 推送结果 | |

（更多见后表）

（接上表）

|  |  |  |  |
| --- | --- | --- | --- |
| **API params** | | | |
| **参数** | **是否必须** | **类型** | **描述** |
| msg | 否 | string | 推送结果为false时存在 |
|  | | | |
| **成功返回(JSON)** | | | |
| code | 是 | string | "success" 成功 |
| **失败返回(JSON)** | | | |
| code | 是 | string | 错误信息 |

实例模板的删除

由于微信官方的限制，实例模板数量不能无限增加（上限25），因此对于不再使用的模板必须及时删除，释放更多的实例模板供其他用户使用。删除的时候要求系统用户提供自己申请的实例模板ID。

Restful接口描述如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API名称** | TemplateMsg | | **方法** | DELETE |
| **请求路径** | /api/template | | | |
| **描述** | 删除模板实例 | | | |
| **权限** | 系统用户 | | | |
| **查询参数** | | | | |
| **参数** | **是否必须** | **类型** | **描述** | |
| template\_id | 是 | string | 需要删除的模板实例ID | |
|  |
| **成功返回(JSON)** | | | | |  |
| code | 是 | string | "success" 成功 | |  |
| **失败返回(JSON)** | | | | |  |
| code | 是 | string | 错误信息短数据 | |  |
| msg | 是 | string | 具体的错误信息 | |  |

5.3.3电子发票插入微信卡包模块

当产品用户收到系统推送的发票申请成功的消息时，通过点击查看更多时可以授权系统，当通过了产品用户的授权之后，系统将在后台自动地执行发票插卡操作。将用户地发票信息插入到微信卡包地发票项目中。

该接口正常情况下不应该被调用，当自动插卡异常时可以使用该接口进行插卡，但需要提供完整的参数信息。

参数的结构体定义如下（Golang）

type Pack struct {  
 *// 发票order\_id, 商户给用户授权开票的订单号* OrderID string `json:"order\_id"`  
 *// 发票的card\_id* CardID string `json:"card\_id"`  
 *// 订单授权的App ID* AppID string `json:"appid"`  
 *// 发票具体信息* CardExt CardExt `json:"card\_ext"`  
}  
  
*// 发票具体信息*type CardExt struct {  
 NonceStr string `json:"nonce\_str"`  
 *// 用户信息结构体* UserCard `json:"user\_card"`  
}  
  
type UserCard struct {  
 InvoiceUserData InvoiceUserData `json:"invoice\_user\_data"`  
}  
  
type InvoiceUserData struct {  
 *// 发票的金额* Fee int `json:"fee"`  
 *// 发票的抬头* Title string `json:"title"`  
 *// 发票的开票时间, 十位时间戳* BillingTime int `json:"billing\_time"`  
 *// 发票的发票代码* BillingNo string `json:"billing\_no"`  
 *// 发票的发票号码* BillingCode string `json:"billing\_code"`  
 *// 商品详情结构* Info List `json:"info"`  
 *// 不含税金额,单位: 分* FeeWithoutTax int `json:"fee\_without\_tax"`  
 *// 税额,单位: 分* Tax int `json:"tax"`  
 *// 发票pdf文件上传到微信发票平台后,会生成一个s\_media\_id,该id可以直接用于关联PDF和发票卡卷* SPDFMediaID string `json:"s\_pdf\_media\_id"`  
 *// 其他消费附件的PDF* STripPDFMediaID string `json:"s\_trip\_pdf\_media\_id, omitempty"`  
 *// 校验码* CheckCode string `json:"check\_code"`  
 *// 购买方纳税人识别号* BuyerNumber string `json:"buyer\_number,omitempty"`  
 *// 购买方地址,电话* BuyerBankAndPhone string `json:"buyer\_bank\_and\_phone,omitempty"`  
 *// 购买方开户行即账号* BuyerBankAccount string `json:"buyer\_bank\_account,omitempty"`  
 *// 销售方纳税人识别号* SellerNumber string `json:"seller\_number,omitempty"`  
 *// 销售方地址,电话* SellerAddressAndPhone string `json:"seller\_address\_and\_phone,omitempty"`  
 *// 销售方开户行及账号* SellerBankAccount string `json:"seller\_bank\_account,omitempty"`  
 *// 备注,发票右下角初* Remarks string `json:"remarks,omitempty"`  
 *// 收款人,发票左下角处* Cashier string `json:"cashier,omitempty"`  
 *// 开票人,发票下方处* Maker string `json:"maker,omitempty"`  
}  
  
type Info struct {  
 *// 项目的名称* Name string `json:"name"`  
 *// 项目数量* Num int `json:"num,omitempty"`  
 *// 项目的单位, 如 个* Unit string `json:"unit,omitempty"`  
 *// 项目单价* Price string `json:"price"`  
}  
  
type List []Info  
  
type Invoice struct {  
 *// 发票的 code* Code string  
 *// 获得发票用户的OpenID* OpenID string `json:"openid"`  
 *// 联合ID，当用户将公众号绑定到微信开放平台账号后，才有该字段* UnionID string `json:"unionid,omitempty"`  
}

Restful接口描述如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API名称** | InsertToUserPackage | | **方法** | POST |
| **请求路径** | /api/userpack | | | |
| **描述** | 微信发票自动插卡 | | | |
| **权限** | 系统用户 | | | |
| **API params** | | | | |
| **参数** | **是否必须** | **类型** | **描述** | |
| 参数在结构体中已经给出，不在赘述(其中json tag带有omiempty的参数为可选) | | | | |
|  |

（更多见后表）

（接上表）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **成功返回(JSON)** | | | |  |
| code | 是 | string | "success" 成功 |  |
| **失败返回(JSON)** | | | |  |
| code | 是 | string | 错误信息短数据 |  |
| msg | 是 | string | 具体的错误信息 |  |

微信插卡结果查询

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API名称** | InsertToUserPackage | | **方法** | GET |
| **请求路径** | /api/userpack | | | |
| **描述** | 微信发票自动插卡 | | | |
| **权限** | 系统用户 | | | |
| **查询参数** | | | | |
| **参数** | **是否必须** | **类型** | **描述** | |
| order\_id | 是 | string | 商户给用户授权开票的订单号 | |
|  |
| card\_id | 是 | string | 发票的card\_id | |  |
| **成功返回(JSON)** | | | | |  |
| code | 是 | string | "success" 成功 | |  |
| **data.code** | 是 | string | 发票code | |  |
| data.userid | 是 | string | 获得发票的用户 | |  |
| **失败返回(JSON)** | | | | |  |
| code | 是 | string | 错误信息短数据 | |  |
| msg | 是 | string | 具体的错误信息 | |  |

5.4公用功能的实现

5.4.1日志模块的设计

日志的设计是采用了单例的设计模式，全局使用的是同一个日志对象。本系统的日志分为系统日志，系统用户操作日志，管理员日志和产品用户操作日志。不同的日志类别记录不同用户的操作记录。

日志的使用主要是为了方便运维人员定位和发现问题，因此对于日志的处理主要是能供提供简单易用的查询方式。同时，为了保证系统的性能，也要保证日志系统在记录的过程中不会影响当前的进程。通过对比不同的IO执行方式，最终决定的是异步的日志记录方式。

日志系统的设计分为了3个部分：记录器、缓存分配器、输出器。分别承当着异步记录日志，缓存日志数据，发送日志数据。

实现原理：

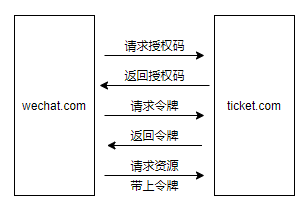
日志记录器维护一张日志表和一个定时器，每一条日志都会被记录到这张表中。每当表被充满或者定时器时间到达，记录器将向缓存分配器置换一张新表。

缓存分配器：缓存分配器维护两个大表，每一个表项就是一张记录表。这两个达标一个记录空表，一个记录满表。当满表非空时，将通知输出器进行消费。

输出器：输出器接收并消费缓存分配器中的满记录表，将置回空表项中。

*# 代码中展示的属性仅为部分有关属性，并非全部  
# 记录器  
def recorder {  
 # 定时器  
 timer  
 # 表  
 table  
}  
  
# 缓存分配器  
def cacheDispatcher {  
 // 满表  
 fullTable  
 // 空表  
 emptyTable  
}  
  
# 输出器  
def exporter {  
 // 消费的外部链接，可以是文件、数据库等  
 conn  
}  
  
# 初始化日志  
# recorderTimeout 记录器定时器时长   
def initLogger (recordTimeout, conn)  
 # 缓存分配器器  
 # initTable 创建一个空的大表，在空表中记录有曾经使用的表的缓存，如果没有则开辟新空间来创建表  
 \_cache\_ = cacheDispatcher{fullTable: initFullTable(), emptyTable: initEmptyTable()}  
 # 记录器, 新建一个定时器,在缓存分配器中申请一个空表  
 \_recorder\_ = recorder {timer: new timer(recordTimeout), table: \_cache\_.newTable()}  
 # 输出器  
 \_output\_ = exporter {conn: conn}  
  
 # 设定定时回调  
 \_cache\_.setTimeEvent(timeEvent)  
   
 # 当\_cache检测到有非空时的回调  
 \_cache\_.setHasFullCall(sync)  
end  
  
# 当调用日志记录的函数时被调用，用于传递日志数据  
def logRecord(recorder, cache, log)  
 # 如果表满了，则更换表  
 if recorder.Full()  
 then  
 # 申请新表，旧表会被置入缓存分配器的满表中  
 recorder.ApplyNewTable(cache)  
 endif  
  
 recorder.table.write(log)  
end  
  
def timeEvent(recorder, cache)   
 // 如果记录器没有记录，则不更新  
 if !recorder.isEmpty()  
 then  
 recorder.ApplyNewTable(cache)  
 endif  
end  
  
def sync(exporter, cache)  
 # 从缓存分配器中获取满表  
 table = cache.get()  
 # 消费日志表，可以是写入文件  
 exporter.consume(table)  
 # 消费完的表设置为空表，并回缓存分配器的空表中  
 cache.put(table.reset())  
end*

5.4.2 OAuth2.0授权认证

授权过程如下图所示

开放平台的核心问题就是用户验证和授权问题，OAuth就是解决授权的一种方式，它要求可以不通过用户提供的账号和密码来获取用户存储在其他服务器的信息。

授权过程原理

引导用户点击授权链接

用户打开授权链接，并且同意授权

用户同意授权获得授权码

通过授权码向服务提供方提出申请授权令牌

服务提供方验证授权码正确并发送授权令牌

通过授权令牌向服务提供方申请资源

服务提供方验证令牌正确并提供资源

5.5 自动化部署的实现

本项目采用Docker打包，并配合使用GitLab Runner进行自动化部署。采用Docker容器化进行，使得系统更容易在多种主机上面执行，也是支持微服务部署的基本条件。

Makefile可以为项目的源文件定义一套规则进行编译，Dockerfile定义了一个容器镜像的编排方式，Shell脚本可以完美的结合Bash执行脚本，项目使用这三个工具配合gitlab-runner进行自动化部署。

// Dockerfile

FROM alpine:3.11 as builder

RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.aliyun.com/g' /etc/apk/repositories && \

apk update && apk add go

ARG OUTPUT=./bin

ARG TARGET=ticket

ENV GOOS=linux

ENV GOARCH=amd64

ENV GOPROXY=https://goproxy.cn

COPY . /go

WORKDIR /go

RUN ash -c 'go build -o /ticket'

FROM alpine:3.11

# use environment value to change default link to redis or mysql

# ENV REDIS\_ADDR=$HOSTNAME

# ENV REDIS\_PASSWORD=$PASSWORD

# ENV REDIS\_DB=$DB\_NUM

# ENV MYSQL\_USER=$MYSQLUSER

# ENV MYSQL\_PASSWORD=$PASSWORD

# ENV MYSQL\_ADDR=$ADDR:$PORT

COPY --from=builder /ticket /ticket

ENTRYPOINT ["/ticket"]

EXPOSE 80

使用Dockerfile创建一个Golang项目打包的环境，然后利用Docker的多阶段构建技术，提取项目打包的结果，有效剔除编译打包阶段产生的中间产物。

// Makefile

# 项目打包移动到docker container中进行，因此过陈在dockerfile中

# Makefile文件只是执行命令

# use environment value to change default link to redis or mysql

# ENV REDIS\_ADDR=$HOSTNAME

# ENV REDIS\_PASSWORD=$PASSWORD

# ENV REDIS\_DB=$DB\_NUM

# ENV MYSQL\_USER=$MYSQLUSER

# ENV MYSQL\_PASSWORD=$PASSWORD

# ENV MYSQL\_ADDR=$ADDR:$PORT

REDIS\_ADDR ?= localhost:6379

REDIS\_PASSWORD ?= 123456

REDIS\_DB ?= 0

MYSQL\_USER ?= root

MYSQL\_PASSWORD ?= 123456

MYSQL\_ADDR ?= localhost:3306

.PHONY: build run

build:

@docker build -t ticket:1.0 .

run:

@ chmod a+x ./deploy.sh

@./deploy.sh $(REDIS\_ADDR) $(REDIS\_PASSWORD) $(REDIS\_DB) $(MYSQL\_USER) $(MYSQL\_PASSWORD) $(MYSQL\_ADDR)

Makefile可以设定目标，在make指令中指定要执行的目标时，才会执行Makefile中指定的目标，利用Makefile执行时变量注入的方式，可以设置动态参数，更具灵活性。

// Shell脚本: deploy.sh

#!/bin/bash

REDIS\_ADDR=$1

REDIS\_PASSWORD=$2

REDIS\_DB=$3

MYSQL\_USER=$4

MYSQL\_PASSWORD=$5

MYSQL\_ADDR=$6

hasServer=`docker ps -a | grep ticket`

if [[ $hasServer ]]

then

docker rm -f ticket

fi

docker run -e REDIS\_ADDR=$REDIS\_ADDR \

-e REDIS\_PASSWORD=$REDIS\_PASSWORD \

-e REDIS\_DB=$REDIS\_DB \

-e MYSQL\_USER=$MYSQL\_USER \

-e MYSQL\_PASSWORD=$MYSQL\_PASSWORD \

-e MYSQL\_ADDR=$MYSQL\_ADDR \

-p 80:80 -d \

--name ticket \

ticket:1.0

sleep 1

isOk=`docker ps | grep ticket`

if [[ -z $isOk ]]

then

echo "ticket start failed"

exit 1

fi

echo "ticket start"

使用Shell脚本可以方便的检查项目执行前的系统环境，创建项目执行时必须的环境。为项目的更新，运行作检查。

以上提到的是项目为了能够实现自动化部署的基本条件，本项目能够自动化部署的决定性条件是正确部署了Gitlab-runner。Gitlab-runner的执行需要“.gitlab-ci.yml”的脚本文件。该文件指导了自动化部署的流水线工作。该文件将部署划分为多个时期(stages)，每一个时间都可以执行多个任务，只有当任务全都执行成功了，才可以进入下一个时期。只有全部执行成功，才确定项目被成功部署。

// .gitlab-ci.yml

stages:

- build

- deploy

job1:

stage: build

tags:

- test

only:

- master

script:

- make build

job2:

stage: deploy

only:

- master

tags:

- test

script:

- /bin/bash -c "REDIS\_ADDR=172.17.0.1:6379 REDIS\_PASSWORD=1211 REDIS\_DB=1 MYSQL\_USER=root MYSQL\_PASSWORD=1211 MYSQL\_ADDR=172.17.0.1:3306 make run"

总结

在本项目的实践里，我首次真正尝试了一个项目从需求分析->系统设计->详细设计->编码实践->测试->上线使用的过程。也了解到了一个项目从0到1从实现上的不容易，也认识到了程序员要通过编程去解决问题的时候，需要从问题的根源出发而不是随便动手码字。这也是我第一个使用到了微服务进行开发的项目，也清楚地认识到微服务所带来地好处和弊端。服务不是越细越好，而是要选择适合就好。前沿技术不是用来彰显技术，而是用来解决问题，当使用的技术不是用来解决问题的时候，往往会带来更多问题，这是得不偿失的。

通过这一次实践，团队协助的重要性再一次凸显出来了，一个高效率开发出来的产品往往离不开一个能够紧密协作的团队。对软件工程理论的掌握，算法基础的掌握也影响着我们单体自身的开发数量和质量。只有不断地学习，才能不断地进步。

参考文献

[1] 上野宣.图解HTTP[M].人民邮电出版社,2014:250.

[2] 郝林 .Go并发编程实战（第2版）[M].人民邮电出版社,2017.

[3] 艾伦 A. A. 多诺万 .Go程序设计语言[M].机械工业出版社,2017.

[4] 马文龙,朱妤晴,蒋德钧,熊劲,张立新,孟潇,包云岗. Key-Value型NoSQL本地存储系统研究[J]. 计算机学报,2018,41(08):1722-1751.

[5] Josiah L. Carlson.Redis实战[M].人民邮电出版社,2015:300.

[6] 谢希仁.计算机网络（第六版)[M].北京:电子工业出版社,2013.

[7] 时子庆,刘金兰,谭晓华. 基于OAuth2.0的认证授权技术[J]. 计算机系统应用,2012,21(03):260-264.

[8] 基于Docker的应用部署管理平台研究-邹保平[J];-期刊-2017-06-20.

[9] Linux下基于Epoll+线程池简单Web服务器实现[J]-陈亮平;-期刊-2019-04-25

[10] 基于MySQL数据库的优化[J]-吴沧舟;-期刊-2013-09-15

[11] ZAITSEV P, TKACHENKO V. High Performance MySQL: Optimization, Backups, and Replication [M]. O'Reilly Media, 2012.

[12] Vinoo Das.Learning Redis[M].Packt Publishing - ebooks Account,2015:412.

[13] Judicael A. Zounmevo. A fast and resource-conscious MPI message queue mechanism for large-scale jobs,Future Generation Computer Systems[J],Volume 30,2014:265-290.

[14] C.A.R.Hoare. CommunicatingSequential Processes[M], 2015.

[15] Raja SehrabBashir. UML models consistency management[M]: Guidelines for software quality manager. International Journal of Information Management, Volume 36, Issue 6, Part A, December 2016: 883-899.

[16] Sau Sheong Chang.Go Web Programming[M].Manning,2015:325.

致谢

岁月如梭，仿佛昨天我才刚刚踏进大学的校门。殊不知，现在的我即将结束我四年的大学时光，开始步入未知的社会。我希望前程似锦，等待我的是更好的明天。大学生活的点点滴滴我都将铭记于心，带着我的座右铭”keep on going, never give up”迈向更远的远方。

人的成长需要一个社会团体，首先我很感激陪伴在我身边的所有人，是他们组成了一个让我成长的殿堂。

感谢我的导师，金正哲老师。他是我们实验室的指导老师，他总是会在大方向上给一个指引，而从不会直接告诉我们在项目中出现的问题该如何解决，正式他的这种指导方式培养了我独立思考的能力，让我在项目开发过程中遇到问题的解决能力、反应能力总是很快很高效。

感谢实验室的同学，感谢实习过程中的同事，是他们让我可以学到团结协作的魅力，也是这种团结协作的魅力，激励我们更加愉快的工作、学习。

感谢我的父母亲，是他们这几十年来的不懈努力，才让我有这个荣幸接受这十多年的教育。

感谢母校，感谢广州大学这四年里给我的教育。希望在未来里，母校更加辉煌!