# 后台服务设计文档

**目录**

目录

**[1. 需求概述](#_Toc25609_WPSOffice_Level1)** **[3](#_Toc25609_WPSOffice_Level1)**

**[2. 总体设计原则](#_Toc15226_WPSOffice_Level1)** **[3](#_Toc15226_WPSOffice_Level1)**

[2.1安全稳定性原则](#_Toc15226_WPSOffice_Level2) [3](#_Toc15226_WPSOffice_Level2)

[2.2先进性原则](#_Toc15317_WPSOffice_Level2) [3](#_Toc15317_WPSOffice_Level2)

[2.3可维护性原则](#_Toc19712_WPSOffice_Level2) [3](#_Toc19712_WPSOffice_Level2)

[2.4实用性原则](#_Toc7163_WPSOffice_Level2) [3](#_Toc7163_WPSOffice_Level2)

[2.5可扩充性原则](#_Toc15369_WPSOffice_Level2) [3](#_Toc15369_WPSOffice_Level2)

**[3. 总体架构设计](#_Toc15317_WPSOffice_Level1)** **[4](#_Toc15317_WPSOffice_Level1)**

[2.1 系统逻辑架构](#_Toc14220_WPSOffice_Level2) [4](#_Toc14220_WPSOffice_Level2)

[2.2 总体网络拓扑结构](#_Toc27116_WPSOffice_Level2) [5](#_Toc27116_WPSOffice_Level2)

**[4. 后台服务主要业务流程](#_Toc19712_WPSOffice_Level1)** **[6](#_Toc19712_WPSOffice_Level1)**

[4.1 申请二维码流程](#_Toc31810_WPSOffice_Level2) [7](#_Toc31810_WPSOffice_Level2)

[4.2 手机端请码流程](#_Toc16459_WPSOffice_Level2) [8](#_Toc16459_WPSOffice_Level2)

[4.3 卡账户充值流程](#_Toc23745_WPSOffice_Level2) [9](#_Toc23745_WPSOffice_Level2)

[4.4 设备乘车数据上传流程](#_Toc158_WPSOffice_Level2) [10](#_Toc158_WPSOffice_Level2)

**[5. 接入层签名](#_Toc7163_WPSOffice_Level1)** **[11](#_Toc7163_WPSOffice_Level1)**

[4.1 签名流程](#_Toc20599_WPSOffice_Level2) [11](#_Toc20599_WPSOffice_Level2)

[4.2 必要参数](#_Toc30989_WPSOffice_Level2) [11](#_Toc30989_WPSOffice_Level2)

**[6. 乘车二维码规则](#_Toc15369_WPSOffice_Level1)** **[11](#_Toc15369_WPSOffice_Level1)**

[5.1 二维码的组成](#_Toc6042_WPSOffice_Level2) [11](#_Toc6042_WPSOffice_Level2)

[5.2　二维码的合法性](#_Toc4343_WPSOffice_Level2) [13](#_Toc4343_WPSOffice_Level2)

### 需求概述

该项目是为智慧公交出行提供后端服务支持，主要提供功能有：申请乘车二维码，二维码请码，二维码资金帐户充值，二维码资金帐户的交易记录查看，二维码收码和刷码消费等功能；

### 总体设计原则

#### 2.1安全稳定性原则

采用高可靠性的产品和技术，充分考虑整个系统运行的安全策略和机制，具有较强的容错能力和良好的恢复能力，保障系统安全、稳定、高效的运行。在安全设计中要全面考虑网络安全、数据安全和用户安全。系统的可靠性可以从系统的稳定性和数据的可靠性两个部分来着手。

系统的稳定性是要求系统能够符合365\*7\*24运行的需要。为此在所有的相关硬件系统部分中，我们都将采用系统的高可靠性方案。

数据的可靠性同时也要求数据具有可靠的备份方案，在系统发现严重故障后，备份的数据可以正确恢复。这主要指数据库具有在线备份和数据恢复。

#### 2.2先进性原则

该项目服务具有较长的产品生命力，代表客户服务系统未来发展方向，避免以后的投资浪费。

#### **2.3可**维护性原则

为在现有的人力资源的情况下，方便系统的维护。系统有完整的错误日志，在系统处理异常时能够根据日志，快捷方便的定位出错误位置、原因。

#### 2.4实用性原则

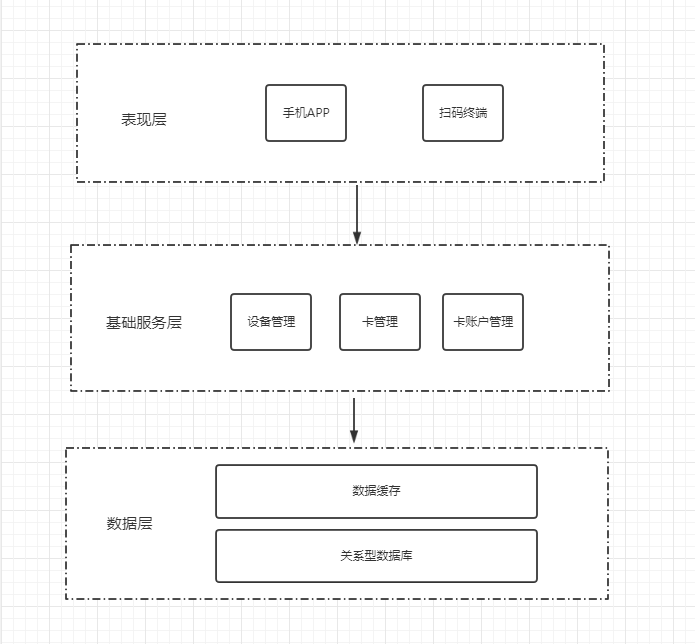
系统的可用性主要是保证系统应在任何时候都能被使用，指系统所有部分在正常工作时的可用性及系统发生改变时的的可用性，如未来在对系统进行软件或硬件的调整时，如何保证整个系统依旧可以正常对外工作，或尽可能少的中断正常工作。

#### 2.5可扩充性原则

系统服务具有良好的可扩充能力，支持系统规模的扩大和业务范围的扩展，能够满足之后业务发展的需要。能够在不更改系统的软件结构和网络结构的前提下，方便的支持系统扩容。

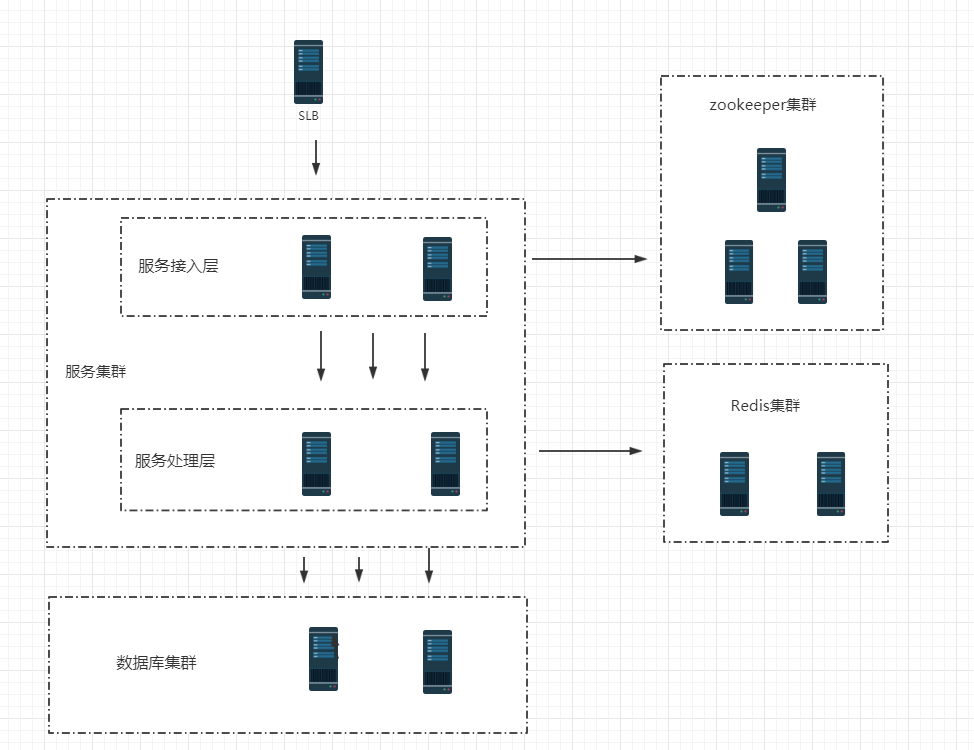
### 总体架构设计

#### 3.1 系统逻辑架构



服务主要包括两部分，一部分是手机APP的后台服务支持，一部分是公交刷卡设备的服务支持；APP的后台功能主要包括请码，账户充值等；刷卡设备的后台功能主要是收码并进行账户扣款；

#### 3.2 总体网络拓扑结构



**SLB**： 对多台云服务器进行流量分发的负载均衡服务。负载均衡可以通过流量分发扩展应用系统对外的服务能力，通过消除单点故障提升应用系统的可用性。

**服务接入层**：负责与客户端（包括手机客户端、刷卡设备）的通讯，通过接入层，服务器端向各种客户端提供统一的报文协议。该层的主要功能如下：

解析来自客户端的请求报文，并将请求数据向下一层传递，

对通讯报文进行加解密操作，使用MD5进行摘要；

获得业务处理后的数据，并将这些业务数据打包成标准格式的响应报文返回给客户端。

提供一些安全策略，如防重放攻击、非法报文的检测、防报文篡改等等。

**服务处理层**：负责处理具体的业务逻辑，该层由Service层和DAO层组成。Service层主要进行业务逻辑处理，DAO层是针对数据库的操作。

### 基础平台建设

#### 4.1 操作系统CentOS7

目前应用最广泛的操作系统有Unix、Win2003 Server、Linux，考虑到运行环境的安全性及稳定性本次将采用CentOS作为服务器操作系统。

CentOS（Community Enterprise Operating System，）是Linux发行版之一，它是来自于Red Hat Enterprise Linux依照开放源代码规定释出的源代码所编译而成。由于出自同样的源代码，因此有些要求高度稳定性的服务器以CentOS替代商业版的Red Hat Enterprise Linux使用。两者的不同，在于CentOS并不包含封闭源代码软件。

CentOS 是一个基于Red Hat Linux 提供的可自由使用源代码的企业级Linux发行版本。每个版本的 CentOS都会获得十年的支持（通过安全更新方式）。新版本的 CentOS 大约每两年发行一次，而每个版本的 CentOS 会定期（大概每六个月）更新一次，以便支持新的硬件。这样，建立一个安全、低维护、稳定、高预测性、高重复性的 Linux 环境。

服务器操作系统大多采用Unix和Linux操作系统，而Linux发行版本系统中，多使用CentOS、Redhat、Ubuntu、Gentoo、Debian。而这些发行版本可以大体分为两类，一类是商业公司维护的发行版本，一类是社区组织维护的发行版本，前者以著名的Redhat（RHEL）为代表，后者以Debian为代表。

在选择系统时，我们希望找到一个可靠的，可预测的系统，并且有强大的软件供应商和开源项目中获得强有力的支持。从可靠性、硬件兼容性和生命周期来对比Redhat与Debian：

* 可靠性

Redhat，应该称为Redhat系列，包括RHEL(RedhatEnterprise Linux，收费版本)、CentOS（Community ENTerprise Operating System，开源版本），是由红帽公司测试维护，并在Linux内核稳定分支上进行开发，系统相对稳定。Debian系列，包括Debian和Ubuntu 等，是社区类Linux的典范，分为三个版本分支（branch）： stable, testing 和 unstable。其中，unstable为最新的测试版本，其中包括最新的软件包，但是也有相对较多的bug，适合桌面用户。Ubuntu是基于 Debian的unstable版本加强而来，一个拥有Debian所有的优点，以及自己所加强的优点的近乎完美的 Linux桌面系统，界面非常友好，容易上手，对硬件的支持非常全面，是最适合做桌面系统的Linux发行版本。

* 硬件兼容性

RHEL对硬件的支持很好，主流硬件厂商早就将服务器拿过去测试，一般不存在硬件的兼容性问题。对于Debian来说，由于版权和代码纯洁性考虑，一些硬件驱动和软件被删掉了，导致安装过程有问题

* 生命周期

CentOS/RHEL的生命周期是7到10年，基本上可以覆盖硬件的生命周期，也就意味着一个新硬件安装以后，不用再次安装操作系统。如果选用了 Debian 或者 Ubuntu作为服务器，等生命周期过了以后，就没有安全补丁，服务器就会有安全风险。

由于CentOS源于 Red Hat 企业级 Linux（RHEL）的源代码，依照开放源代码规定释出的源代码所编译而成。由于CentOS开源特性，选择CentOS可以降低成本，同时又能够享受RHEL的服务支持。

CentOS7是在CentOS6基础上发布的新版本，与之前的版本相比，主要的更新包括：

* 内核更新到3.10.0
* 支持Linux容器
* LVM快照支持ext4和XFS
* 转用systemd、firewalld和GRUB2
* XFS作为缺省文件系统
* 支持PTPv2
* 支持40G 以太网卡
* 在兼容的硬件上支持以UEFI安全启动模式安装
* 这其中最令人瞩目的新特性就是支持Docker技术。作为目前流行的应用虚拟化技术之一，Docker能够将应用程序与系统完全隔离，让其在系统之间实现迁移而不需要停机，提高了应用程序的移动性和灵活性。CentOS7在内核层面支持Docker容器技术，可以提高Docker稳定性和可靠性。
* 综上，我们会选择CentOS7来作为服务器的操作系统。

#### 4.2 负载均衡SLB

负载均衡（Server Load Balancer）是对多台云服务器进行流量分发的负载均衡服务。负载均衡可以通过流量分发扩展应用系统对外的服务能力，通过消除单点故障提升应用系统的可用性。

* 什么是负载均衡

负载均衡（Server Load Balancer）是将访问流量根据转发策略分发到后端多台云服务器（ECS实例）的流量分发控制服务。负载均衡扩展了应用的服务能力，增强了应用的可用性。

负载均衡通过设置虚拟服务地址，将添加的ECS实例虚拟成一个高性能、高可用的应用服务池，并根据转发规则，将来自客户端的请求分发给云服务器池中的ECS实例。

负载均衡默认检查云服务器池中的ECS实例的健康状态，自动隔离异常状态的ECS实例，消除了单台ECS实例的单点故障，提高了应用的整体服务能力。此外，负载均衡还具备抗DDoS攻击的能力，增强了应用服务的防护能力。

* 组成部分

负载均衡由以下三个部分组成：

* 负载均衡实例 （Server Load Balancer instances）

一个负载均衡实例是一个运行的负载均衡服务，用来接收流量并将其分配给后端服务器。要使用负载均衡服务，您必须创建一个负载均衡实例，并至少添加一个监听和两台ECS实例。

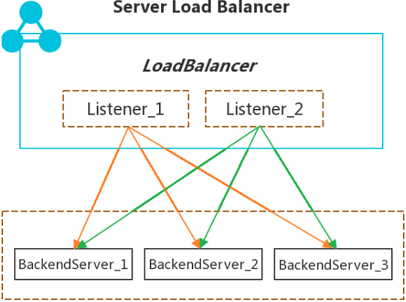
* 监听 （Listeners）

监听用来检查客户端请求并将请求转发给后端服务器。监听也会对后端服务器进行健康检查。

* 后端服务器（Backend Servers）

一组接收前端请求的ECS实例。您可以单独添加ECS实例到服务器池，也可以通过虚拟服务器组或主备服务器组来批量添加和管理。

如下图所示，来自客户端的请求经过负载均衡实例后，监听会将请求根据配置的监听规则分发给后端添加的ECS实例处理。



#### 4.3 数据库PostgreSQL RDS

PostgreSQL是以加州大学伯克利分校计算机系开发的 POSTGRES，现在已经更名为PostgreSQL，版本 4.2为基础的对象关系型数据库管理系统（ORDBMS）。PostgreSQL支持大部分 SQL标准并且提供了许多其他现代特性：复杂查询、外键、触发器、视图、事务完整性、MVCC。同样，PostgreSQL 可以用许多方法扩展，比如， 通过增加新的数据类型、函数、操作符、聚集函数、索引。免费使用、修改、和分发 PostgreSQL，不管是私用、商用、还是学术研究使用。

PostgreSQL 是一个自由的对象-关系数据库服务器(数据库管理系统)，它在灵活的 BSD-风格许可证下发行。它提供了相对其他开放源代码数据库系统(比如 MySQL 和 Firebird)，和专有系统(比如 Oracle、Sybase、IBM 的 DB2 和 Microsoft SQL Server)之外的另一种选择。

* 优点

事实上， PostgreSQL 的特性覆盖了 SQL-2/SQL-92 和 SQL-3/SQL-99，首先，它包括了可以说是目前世界上最丰富的数据类型的支持，其中有些数据类型可以说连商业数据库都不具备， 比如 IP 类型和几何类型等；其次，PostgreSQL 是全功能的自由软件数据库，很长时间以来，PostgreSQL 是唯一支持事务、子查询、多版本并行控制系统（MVCC）、数据完整性检查等特性的唯一的一种自由软件的数据库管理系统。 Inprise 的 InterBase 以及SAP等厂商将其原先专有软件开放为自由软件之后才打破了这个唯一。最后，PostgreSQL拥有一支非常活跃的开发队伍，而且在许多黑客的努力下，PostgreSQL 的质量日益提高。

从技术角度来讲，PostgreSQL 采用的是比较经典的C/S（client/server）结构，也就是一个客户端对应一个服务器端守护进程的模式，这个守护进程分析客户端来的查询请求，生成规划树，进行数据检索并最终把结果格式化输出后返回给客户端。为了便于客户端的程序的编写，由数据库服务器提供了统一的客户端 C 接口。而不同的客户端接口都是源自这个 C 接口，比如ODBC，JDBC，Python，Perl，Tcl，C/C++，ESQL等， 同时也要指出的是，PostgreSQL 对接口的支持也是非常丰富的，几乎支持所有类型的数据库客户端接口。这一点也可以说是 PostgreSQL 一大优点。

* 架构

PostgreSQL强壮的一个原因源于它的架构。和商业数据库一样，PostgreSQL可以用于C/S(客户/服务器)环境。

PostgreSQL安装核心是数据库服务端进程。它允许在一个独立服务器上。需要访问存储在数据库中的数据的应用程序必须通过数据库进程。这些客户端程序无法直接访问数据，即使它们和服务程序在同一台机器上。

PostgreSQL 是世界上可以获得的最先进的开放源码的数据库系统， 它提供了多版本并行控制，支持几乎所有 SQL 构件(包括子查询，事务和用户定 义类型和函数)， 并且可以获得非常广阔范围的(开发)语言绑定 (包括 C，C++，Java，perl，tcl，和 python)。具体的优点特性如下：

　　1. PostgreSQL 的特性覆盖了 SQL-2/SQL-92 和 SQL-3/SQL-99，是目前世界上支持最丰富的数据类型的数据库。

　　2. PostgreSQL 是全功能的自由软件数据库，PostgreSQL 是唯一支持事务、子查询、多版本并行控制系统、数据完整性检查等特性的唯一的一种自由软件的数据库管理系统。

　　3. PostgreSQL 采用的是比较经典的 C/S (client/server)结构，也就是一个客户端对应一个服务器端守护进程的模式，这个守护进程分析客户端来的查询请求，生成规划树，进行数据检索并最终把结果格式化输出后返回给客户端。

　　4. PostgreSQL 对接口的支持也是非常丰富的，几乎支持所有类型的数据库客户端接口。

#### 4.4 Redis存储系统

Redis是一个Key-Value存储系统。和Memcached类似，它支持存储的value类型相对更多，包括String(字符串)、List(链表)、Set(集合)、Zset(sorted set --有序集合)和Hash（哈希类型）。这些数据类型都支持Push/Pop、Add/Remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，Redis支持各种不同方式的排序。与Memcached一样，为了保证效率，数据都是缓存在内存中。区别的是Redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了Master-Slave(主从)同步。

Redis 是一个高性能的Key-Value数据库。Redis的出现，很大程度补偿了Memcached这类Key/Value存储的不足，在部 分场合可以对关系数据库起到很好的补充作用。它提供了Java，C/C++，C#，PHP，JavaScrIPt，Perl，Object-C，Python，Ruby，Erlang等客户端，使用很方便。

Redis支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。这使得Redis可执行单层树复制。存盘可以有意无意的对数据进行写操作。由于完全实现了发布/订阅机制，使得从数据库在任何地方同步树时，可订阅一个频道并接收主服务器完整的消息发布记录。同步对读取操作的可扩展性和数据冗余很有帮助。

#### 4.5 服务层框架Dubbo

Dubbo是一个分布式服务框架，以及SOA治理方案。其功能主要包括：高性能NIO通讯及多协议集成，服务动态寻址与路由，软负载均衡与容错，依赖分析与降级等。

* Dubbo适用于那些场景？

当网站变大后，不可避免的需要拆分应用进行服务化，以提高开发效率，调优性能，节省关键竞争资源等。   
 当服务越来越多时，服务的URL地址信息就会爆炸式增长，配置管理变得非常困难，F5硬件负载均衡器的单点压力也越来越大。   
 当进一步发展，服务间依赖关系变得错踪复杂，甚至分不清哪个应用要在哪个应用之前启动，架构师都不能完整的描述应用的架构关系。   
 接着，服务的调用量越来越大，服务的容量问题就暴露出来，这个服务需要多少机器支撑？什么时候该加机器？等等……

在遇到这些问题时，都可以用Dubbo来解决。

* Dubbo能做什么？

透明化的远程方法调用，就像调用本地方法一样调用远程方法，只需简单配置，没有任何API侵入。        
 软负载均衡及容错机制，可在内网替代F5等硬件负载均衡器，降低成本，减少单点。  
 服务自动注册与发现，不再需要写死服务提供方地址，注册中心基于接口名查询服务提供者的IP地址，并且能够平滑添加或删除服务提供者。

* Dubbo的架构



节点角色说明：

       Provider: 暴露服务的服务提供方。

       Consumer: 调用远程服务的服务消费方。

       Registry: 服务注册与发现的注册中心。

       Monitor: 统计服务的调用次调和调用时间的监控中心。

[Container](http://lib.csdn.net/base/docker" \o "Docker知识库" \t "https://blog.csdn.net/houshaolin/article/details/_blank): 服务运行容器。

调用关系说明：

0 服务容器负责启动，加载，运行服务提供者。

1. 服务提供者在启动时，向注册中心注册自己提供的服务。

2. 服务消费者在启动时，向注册中心订阅自己所需的服务。

3. 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。

4. 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。

5. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

#### 4.6 接入层框架Dubbox

[Dubbo](http://www.biaodianfu.com/dubbo.html" \o "Dubbo：来自于阿里巴巴的分布式服务框架)是一个来自阿里巴巴的开源分布式服务框架，当当根据自身的需求，为Dubbo实现了一些新的功能，包括REST风格远程调用、Kryo/FST序列化等等。并将其命名为[Dubbox](https://github.com/dangdangdotcom/dubbox)（即Dubbo eXtensions）。Dubbox主要的新功能包括：

* 支持REST风格远程调用（HTTP + JSON/XML)

dubbo支持多种远程调用方式，例如dubbo RPC（二进制序列化 + tcp协议）、http invoker（二进制序列化 + http协议，至少在开源版本没发现对文本序列化的支持）、hessian（二进制序列化 + http协议）、WebServices （文本序列化 + http协议）等等，但缺乏对当今特别流行的REST风格远程调用（文本序列化 + http协议）的支持。

dubbox基于非常成熟的JBoss RestEasy框架，在dubbo中实现了REST风格（HTTP + JSON/XML）的远程调用，以显著简化企业内部的跨语言交互，同时显著简化企业对外的Open API、无线API甚至AJAX服务端等等的开发。事实上，这个REST调用也使得Dubbo可以对当今特别流行的“微服务”架构提供基础性支持。 另外，REST调用也达到了比较高的性能，在基准测试下，HTTP + JSON与Dubbo 2.x默认的RPC协议（即TCP + Hessian2二进制序列化）之间只有1.5倍左右的差距。

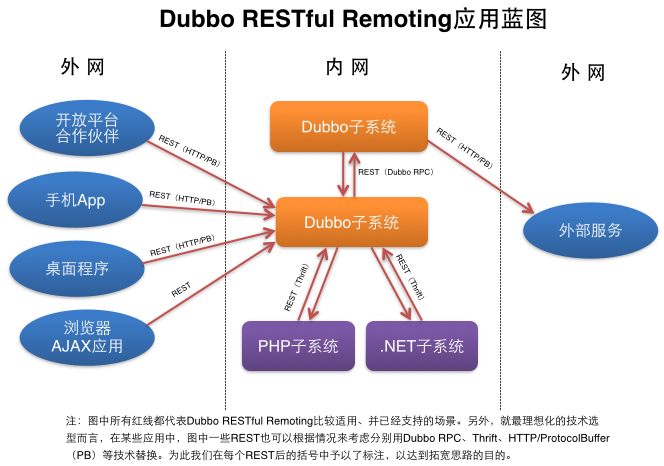
REST的优点（摘自维基百科）：

* 可更高效利用缓存来提高响应速度
* 通讯本身的无状态性可以让不同的服务器的处理一系列请求中的不同请求，提高服务器的扩展性
* 浏览器即可作为客户端，简化软件需求
* 相对于其他叠加在HTTP协议之上的机制，REST的软件依赖性更小
* 不需要额外的资源发现机制
* 在软件技术演进中的长期的兼容性更好

基于简单的文本格式消息和通用的HTTP协议，使REST具备极广的适用性，几乎所有语言和平台都对它提供支持，同时其学习和使用的门槛也较低。在dubbo中支持REST，可以为当今多数主流的远程调用场景都带来好处：

* 显著简化企业内部的异构系统之间的（跨语言）调用。此处主要针对这种场景：dubbo的系统做服务提供端，其他语言的系统（也包括某些不基于 dubbo的java系统）做服务消费端，两者通过HTTP和文本消息进行通信。即使相比Thrift、ProtoBuf等二进制跨语言调用方 案，REST也有自己独特的优势（详见后面讨论）
* 显著简化对外Open API（开放平台）的开发。既可以用dubbo来开发专门的Open API应用，也可以将原内部使用的dubbo service直接“透明”发布为对外的Open REST API（当然dubbo本身未来最好可以较透明的提供诸如权限控制、频次控制、计费等诸多功能）
* 显著简化手机（平板）APP或者PC桌面客户端开发。类似于2，既可以用dubbo来开发专门针对无线或者桌面的服务器端，也可以将原内部使用的 dubbo service直接“透明”的暴露给手机APP或桌面程序。当然在有些项目中，手机或桌面程序也可以直接访问以上场景2中所述的Open API。
* 显著简化浏览器AJAX应用的开发。类似于2，既可以用dubbo来开发专门的AJAX服务器端，也可以将原内部使用的dubbo service直接“透明”的暴露给浏览器中JavaScript。当然，很多AJAX应用更适合与web框架协同工作，所以直接访问dubbo service在很多web项目中未必是一种非常优雅的架构。
* 为企业内部的dubbo系统之间（即服务提供端和消费端都是基于dubbo的系统）提供一种基于文本的、易读的远程调用方式。
* 一定程度简化dubbo系统对其它异构系统的调用。可以用类似dubbo的简便方式“透明”的调用非dubbo系统提供的REST服务（不管服务提供端是在企业内部还是外部）

需要指出的是， 1～3是dubbo的REST调用最有价值的三种应用场景，并且为dubbo添加REST调用，其最主要到目的也是面向服务的提供端，即开发REST服务来提供给非dubbo的（异构）消费端。归纳起来，所有应用场景如下图所示：



* 支持基于Kryo和FST的Java高效序列化实现

dubbo RPC是dubbo体系中最核心的一种高性能、高吞吐量的远程调用方式，简单的说：

* 长连接：避免了每次调用新建TCP连接，提高了调用的响应速度
* 多路复用：单个TCP连接可交替传输多个请求和响应的消息，降低了连接的等待闲置时间，从而减少了同样并发数下的网络连接数，提高了系统吞吐量。

dubbo RPC主要用于两个dubbo系统之间作远程调用，特别适合高并发、小数据的互联网场景。而序列化对于远程调用的响应速度、吞吐量、网络带宽消耗等同样也 起着至关重要的作用，是我们提升分布式系统性能的最关键因素之一。在dubbo RPC中，同时支持多种序列化方式，例如：

* dubbo序列化：阿里尚未开发成熟的高效java序列化实现，阿里不建议在生产环境使用它
* hessian2序列化：hessian是一种跨语言的高效二进制序列化方式。但这里实际不是原生的hessian2序列化，而是阿里修改过的hessian lite，它是dubbo RPC默认启用的序列化方式
* json序列化：目前有两种实现，一种是采用的阿里的fastjson库，另一种是采用dubbo中自己实现的简单json库，但其实现都不是特别成熟，而且json这种文本序列化性能一般不如上面两种二进制序列化。
* java序列化：主要是采用JDK自带的Java序列化实现，性能很不理想。

在通常情况下，这四种主要序列化方式的性能从上到下依次递减。对于dubbo RPC这种追求高性能的远程调用方式来说，实际上只有1、2两种高效序列化方式比较般配，而第1个dubbo序列化由于还不成熟，所以实际只剩下2可用， 所以dubbo RPC默认采用hessian2序列化。但hessian是一个比较老的序列化实现了，而且它是跨语言的，所以不是单独针对java进行优化的。而 dubbo RPC实际上完全是一种Java to Java的远程调用，其实没有必要采用跨语言的序列化方式（当然肯定也不排斥跨语言的序列化）。

最近几年，各种新的高效序列化方式层出不穷，不断刷新序列化性能的上限，最典型的包括：

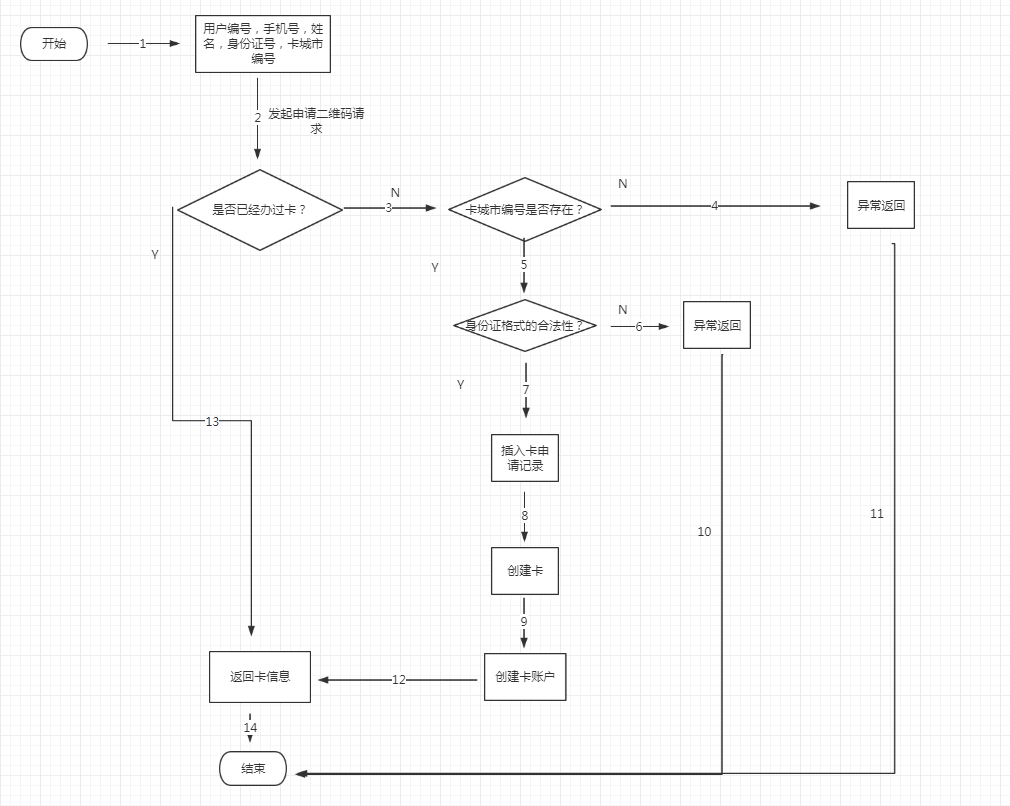
* 专门针对Java语言的：Kryo，FST等等
* 跨语言的：Protostuff，ProtoBuf，Thrift，Avro，MsgPack等等

这些序列化方式的性能多数都显著优于hessian2（甚至包括尚未成熟的dubbo序列化）。有鉴于此，我们为dubbo引入Kryo和FST这 两种高效Java序列化实现，来逐步取代hessian2。其中，Kryo是一种非常成熟的序列化实现，已经在Twitter、Groupon、 Yahoo以及多个著名开源项目（如Hive、Storm）中广泛的使用。而FST是一种较新的序列化实现，目前还缺乏足够多的成熟使用案例，但它还是非 常有前途的。

* 其他新功特性
* 支持基于嵌入式Tomcat的HTTP remoting体系：基于嵌入式tomcat实现dubbo的 HTTP remoting体系（即dubbo-remoting-http），用以逐步取代Dubbo中旧版本的嵌入式Jetty，可以显著的提高REST等的远 程调用性能，并将Servlet API的支持从5升级到3.1。（注：除了REST，dubbo中的WebServices、Hessian、HTTP Invoker等协议都基于这个HTTP remoting体系）。
* 升级Spring：将dubbo中Spring由x升级到目前最常用的3.x版本，减少项目中版本冲突带来的麻烦。
* 升级ZooKeeper客户端：将dubbo中的zookeeper客户端升级到最新的版本，以修正老版本中包含的bug。

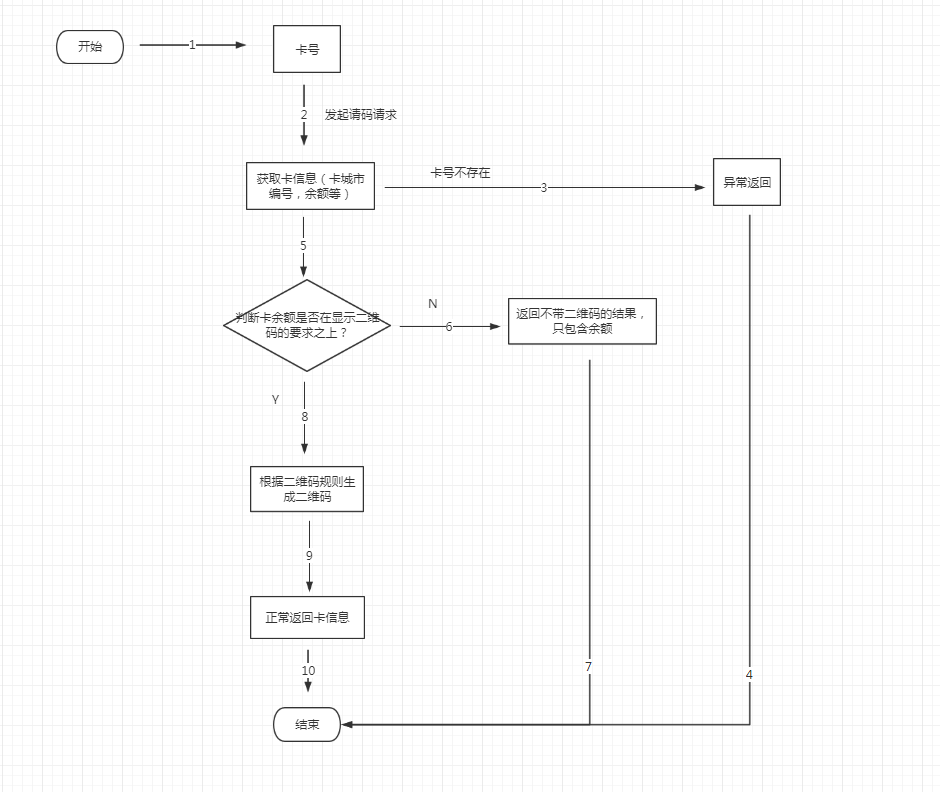
### 后台服务主要业务流程

#### 申请二维码流程



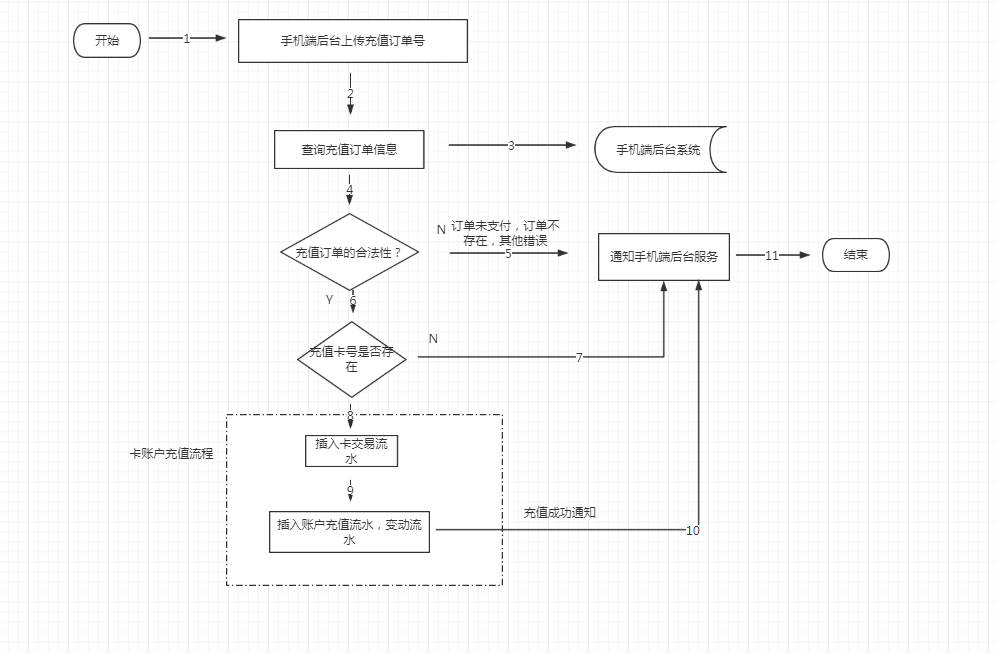
1. 开始；
2. 手机app端通过请求参数用户编号、手机号、姓名、身份证号和卡城市编号，向卡服务平台申请办理公交卡；
3. 卡服务平台通过用户编号判断该用户是否在该城市下办理过卡，如果办理过卡，跳到步骤13；如果没有办理过，判断卡城市编号是否存在；如果不存在，跳步骤4，存在，这跳步骤5；
4. 卡城市编号不存在，这异常返回；
5. 判断身份证格式的合法性，不合法，跳步骤6，否则，跳步骤7；
6. 身份证号码格式不合法，异常返回；
7. 插入卡申请记录；
8. 创建卡；
9. 创建卡账户；
10. 结束；
11. 结束；
12. 创建卡，卡账户之后返回卡信息；
13. 该用户已经在该卡办理过卡，这直接返回卡信息；
14. 结束；

#### 手机端请码流程



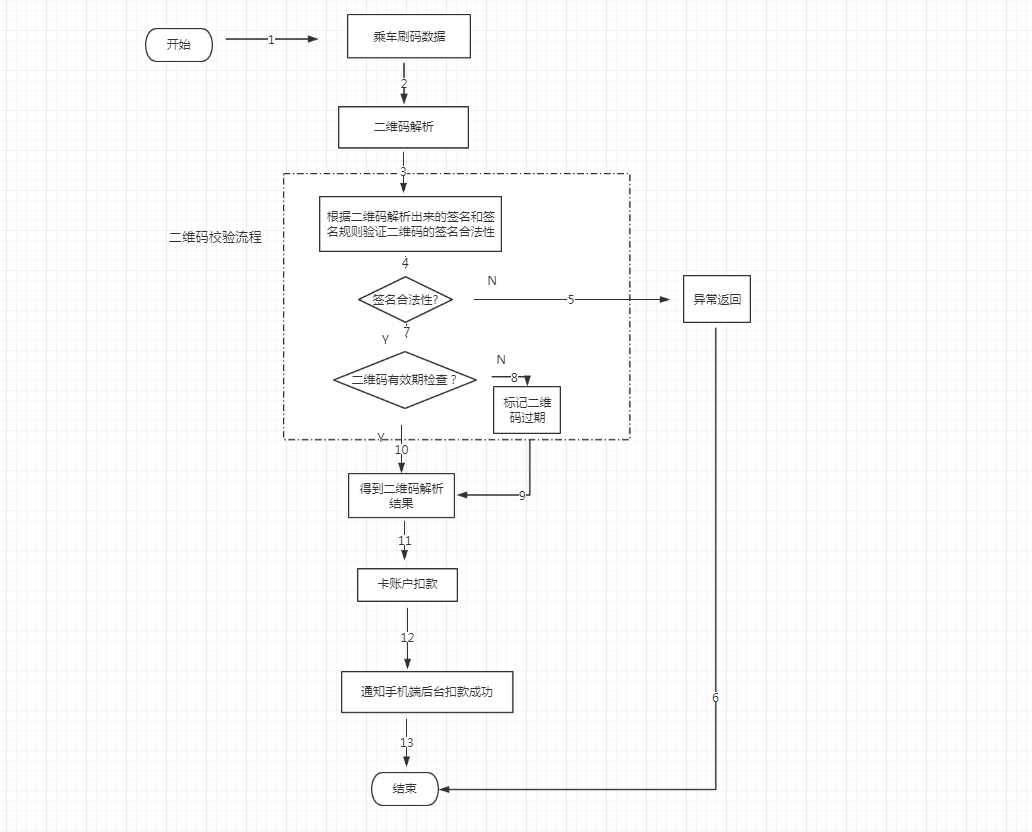
1. 开始；
2. 手机app端通过请求参数卡号发起请码请求；
3. 通过卡号获取卡信息，卡余额；卡号不存在，异常返回；
4. 结束；
5. 获取卡信息，卡余额之后，获取二维码显示的余额的最低配置；判断二维码的余额是否大于该配置，如果小于等于该配置，跳步骤6，否则跳步骤8；
6. 返回不带二维码，状态是余额不足的响应信息；
7. 结束；
8. 根据二维码生成规则生成二维码；
9. 返回二维码，卡余额信息；
10. 结束；

#### 卡账户充值流程



1. 开始；
2. 手机app端通过充值订单请求卡服务平台充值；
3. 卡服务平台向手机app后台请求充值订单信息；
4. 判断充值订单的合法性，如果订单未支付，订单不存在和其他错误等异常，跳步骤5，否则跳步骤6；
5. 通知手机app后台，充值订单异常；
6. 判断充值的卡号是否存在，如果不存在，跳步骤7，否则，跳步骤8；
7. 通知手机app后台，充值卡号不存在；
8. 插入卡交易流水；
9. 进行卡账户充值；
10. 通知手机app后台，卡充值成功；
11. 结束；

#### 设备乘车数据上传流程



1. 开始；
2. 公交设备上传刷码数据；
3. 卡服务平台进行二维码解析；
4. 根据二维码的签名规则校验签名，如果签名不通过，调步骤5，否则跳步骤7；
5. 二维码验签失败，异常返回；
6. 结束；
7. 检测二维码是否过期，如果是，条步骤8，否则跳步骤10；
8. 标记二维码已经过期；
9. 返回解析出的过期的二维码信息；
10. 返回解析出的正常的二维码信息；
11. 卡账户扣款；
12. 通知手机app后台扣款成功；
13. 结束；

### 接入层签名

#### 6.1 签名流程

签名生成的通用步骤如下：

第一步，设所有发送或者接收到的数据为集合M，将集合M内非空参数值的参数按照参数名ASCII码从小到大排序（字典序），并对key和value进行URL编码，将编码后的key和value值使用URL键值对的格式（即key1=value1&key2=value2…）拼接成字符串stringA。

特别注意以下重要规则：

◆ 参数名ASCII码从小到大排序（字典序）；

◆ 如果参数的值为空不参与签名；

◆ 参数名区分大小写；

◆ 需要请求参数是Bean结构数据时，bean结点仍以JSON串组装，且整个json串做为value参与签名

第二步，在stringA最后拼接上secret得到stringSignTemp字符串，并对stringSignTemp进行MD运算，取长度为32的加密结果，再将得到的字符串所有字符转换为大写，得到sign值signValue。

#### 6.2 必要参数

以下必要字段参与签名，每个交易都需要添加，不在交易接口描述中再做说明, 这些字段在协议设计中需要避开使用

|  |  |
| --- | --- |
| oauth\_appid | 接入商唯一编号(卡平台发放) |
| oauth\_method | 签名方法，目前支持MD5，SHA1 |
| oauth\_timestamp | 时间截，前后误差不超过5分钟，单位 秒 |
| oauth\_nonce | 随机数，建议使用纳秒数做为随机数 |

### 乘车**二维码规**则

#### 7.1 二维码的组成

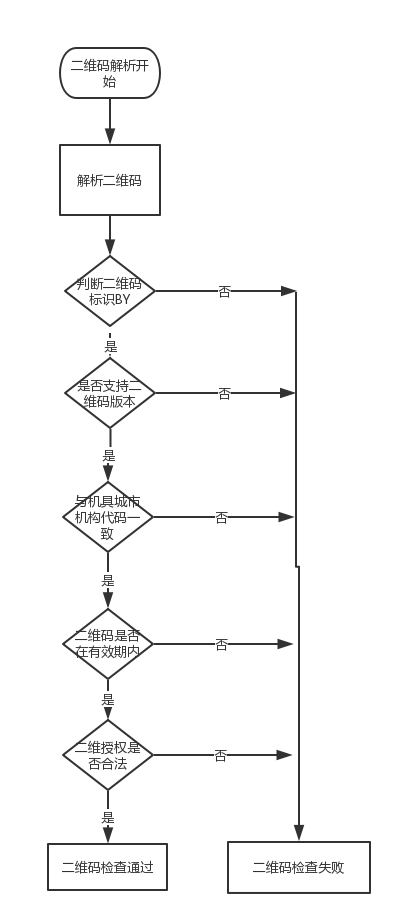
公交二维码由二维码头、联机授权域组成，公交二维码数据结构见表A。

表A-1二维码数据格式

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 数据域 | 序号 | 字段名 | 长度 | 描述 | 格式 | 必须 |
| 二维码头 | 1 | 二维码标识 | 2 | 二维码标识 取值BY | ANS | M |
| 2 | 二维码版本 | 1 | 二维码结构版本, 当前版本=1 | B | M |
| 授权机构授权域 | 3 | 城市机构代码 | 3 | 潍坊城市机构代码370700,实际转化为  0x37 0x07 0x00 | BCD | M |
| 4 | 二维码有效时长 | 2 | 二维码从创建开始到失效间的时间，以秒为单位 | B | M |
| 5 | 二维码生成时间 | 4 | 二维码生码的时间，单位秒,  Unix时间戳，对应1970的时间差 | B | M |
| 6 | 支付卡号 | 5 | 虚拟电子卡号 | BCD | M |
| 7 | 卡类型 | 1 | 卡类型  0:普通卡 | B | M |
| 8 | 自定义数据长度 | 1 | 自定义数据长度不超过256 | N | M |
| 9 | 自定长数据 | N |  | B | 0 |
| 签名域 | 10 | 授权签名 | 64 | 授权机构私钥对3~9项数据进行签名 | B | M |
|  |  |  | 83+ |  |  |  |

受理终端读取二维码信息进行验证，流程图A-2

图A-2二维码验码流程



二维码验证流程应符合下列要求：

a)   判断码源基本属性：检查二维码标识、版本信息；

b)   验证码源正确性：检查二维码是否在有效期内，验证机构授权签名是否正确，保证内容不被破坏；

二维码检查通过后，受理终端执行本地交易流程。交易成功后受理终端将生成交易记录，提示交易成功，并为用户提供乘车服务。

#### 7.2　二维码的合法性

##### 7.2.1　安全性保证

二维码的安全性保证由发卡机构私钥签名保证

##### 7.2.2　实时性保证

二维码的实时性由‘二维码有效时长’及‘二维码生成时间’保证，二维码发码平台与受理终端均采用世界标准时间，应按以下方式保证时间同步：

a)   二维码发码平台应具备定期与世界时间服务器时间同步的机制，保证二维码的生码时间与世界标准时间保持一致；

b)   受理终端应具备定期或必要时与收单机构后台系统完成时间同步的机制，保证受理终端在验码时效性。

### 阿里云服务器配置

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 内存(G) | CPU(核数) | HDD(G) | SSD(G) | 阿里云服务器配置 | 服务器数量(台) |
| SLB |  |  |  |  |  | 1 |
| 应用API | 8 | 4 | 500 |  |  | 2 |
| 应用APP | 8 | 4 | 500 |  |  | 2 |
| redis（阿里云 云数据  库 redis版） |  |  |  |  | 按量/包年 | 1 |
| zookeeper | 2 | 1 | 40 |  |  | 3 |
| MQ（阿里云互联网中  间件 消息队列MQ） |  |  |  |  | 按量/包年 |  |
| db(阿里云 RDS  PostgreSQL版) |  |  |  |  | 按量/包年 | 1 |