**吉林省科技发展计划项目**

“基于云计算的库存与配送管理关键技术及系统开发”

项目鉴定文件之三

技 术 报 告

长春理工大学计算机与信息技术研究所

2013年11月

目录

[1 SAAS软件应用模式 1](#_Toc373066278)

[1.1软件应用结构 1](#_Toc373066279)

[1.2实现海量数据处理 1](#_Toc373066280)

[1.3软件部署方案 2](#_Toc373066281)

[2 Windows Azure操作系统的软件架构及实现技术 3](#_Toc373066282)

[2.1 Windows Azure 3](#_Toc373066283)

[2.1.1 简介 3](#_Toc373066284)

[2.1.2 架构 3](#_Toc373066285)

[2.1.3 Windows Azure的应用 4](#_Toc373066286)

[2.2 Sql Azure 6](#_Toc373066287)

[2.2.1 简介 6](#_Toc373066288)

[2.2.2 架构 6](#_Toc373066289)

[2.2.3 Sql Azure的应用 8](#_Toc373066290)

[3 Web客户端技术 11](#_Toc373066291)

[3.1 Html5与ASP.NET应用 11](#_Toc373066292)

[3.2 css应用 11](#_Toc373066293)

[3.3 AJAX应用 12](#_Toc373066294)

[3.4 Silverlight应用 12](#_Toc373066295)

[4 海量数据分布式存储 14](#_Toc373066296)

[4.1 分布式存储框架 14](#_Toc373066297)

[4.1.1 Hadoop 14](#_Toc373066298)

[4.1.2 MapReduce 15](#_Toc373066299)

[4.1.3 HDFS 16](#_Toc373066300)

[4.1.4 Hbase 20](#_Toc373066301)

[4.1.4 集群 21](#_Toc373066302)

[4.2 分布式存储数据结构 23](#_Toc373066303)

[4.2.1 Hbase集群设计 23](#_Toc373066304)

[4.2.2 数据访问接口 25](#_Toc373066305)

[4.2.3 储存结构设计 26](#_Toc373066306)

[5 缓存数据库架构 28](#_Toc373066307)

[5.1 缓存数据库 28](#_Toc373066308)

[5.1.1 Redis 28](#_Toc373066309)

[5.1.2 Redis性能 28](#_Toc373066310)

[5.2 缓存数据库架构设计 32](#_Toc373066311)

[5.2.1 表设计，数据结构选择 32](#_Toc373066312)

[5.2.2 Redis数据导入与导出 37](#_Toc373066313)

[5.2.3 Redis接口设计 37](#_Toc373066314)

[6 仓库管理系统 39](#_Toc373066315)

[6.1库存信息管理系统系统架构设计 39](#_Toc373066316)

[6.1.1表现层（Presentation Layer） 41](#_Toc373066317)

[6.1.2业务层（Business Tier） 41](#_Toc373066318)

[6.1.3数据层（Data Tier） 41](#_Toc373066319)

[6.1.4小结： 42](#_Toc373066320)

[6.2数据库设计 42](#_Toc373066321)

[6.2.1基本业务分析 42](#_Toc373066322)

[6.2.2概念结构设计（E-R图） 42](#_Toc373066323)

[6.3物流配送流程图: 45](#_Toc373066324)

[6.3.1前台用户操作流程 45](#_Toc373066325)

[6.3.2后台管理界面： 49](#_Toc373066326)

[6.3.3入库管理 50](#_Toc373066327)

[6.3.4订单生成配送单 51](#_Toc373066328)

[6.3.5运输订单界面 53](#_Toc373066329)

[6.3.6出库管理 53](#_Toc373066330)

[6.3.7 配送管理 53](#_Toc373066331)

[6.4详细技术 55](#_Toc373066332)

[6.4.1软件功能流程图 55](#_Toc373066333)

[6.4.2软件功能模块设计 55](#_Toc373066334)

[7 实时监控 63](#_Toc373066335)

[7.1实时监控概述 63](#_Toc373066336)

[7.2物流配送管理手机移动端的介绍与应用 64](#_Toc373066337)

[7.2.1 物流配送管理系统移动端的优点 64](#_Toc373066338)

[7.2 .2物流配送管理智能终端架构介绍 65](#_Toc373066339)

[7.3实时信息查询 67](#_Toc373066340)

[7.3.1实时信息采集 67](#_Toc373066341)

[7.3.2实时地图技术 69](#_Toc373066342)

[8 海量数据分析处理 71](#_Toc373066343)

[8.1 配送分析算法 71](#_Toc373066344)

[8.1.1 算法理论 71](#_Toc373066345)

[8.1.2 MapReduce算法设计 71](#_Toc373066346)

[8.2 路径分析算法 73](#_Toc373066347)

[8.2.1 算法理论 73](#_Toc373066348)

[8.2.2 MapReduce算法设计： 74](#_Toc373066349)

[附录 76](#_Toc373066350)

# 1 SAAS软件应用模式

## 1.1软件应用结构

软件部署在Windows Azure服务器上之后，用户根据需求租用软件使用权。针对不同的用户提供相应的软件服务。达到不同用户访问相同数据中心。如图1-1所示。采用saas服务模式在效果上与企业自建信息系统基本没有区别，但节省了用户购买IT产品，技术和维护运行的资金，从而降低了中小型企业信息化的门槛与风险并且免除了用户的服务器硬件，网络安全设备和软件升级维护的支出，客户只需通过互联网就可以获得所需要软件和服务。

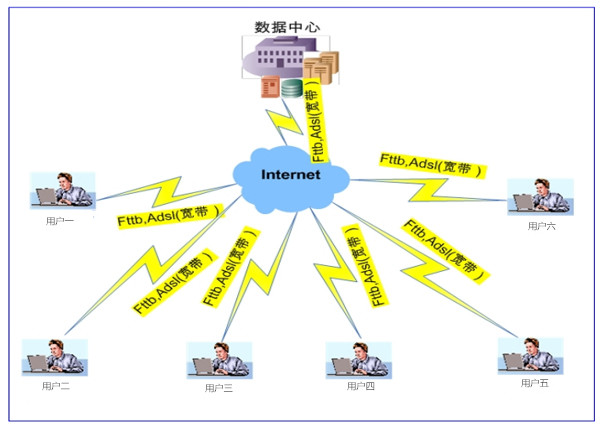


图 1-1SAAS模式

## 1.2实现海量数据处理

软件采用Hadoop分布式平台，因为Hadoop以并行的方式工作，通过并行处理加快处理速度，更有MapReduce引擎对大规模数据集进行并行运算。软件还有定位功能，对定位位置坐标进行处理，采用Redis作为缓存数据库，对信息进行实时存储与读取。软件具有实时监控模块，每一秒将向数据库存储一条信息，所以采用Hadoop中MauReduce对海量数据进行处理。如图1-2为Hadoop分布式集群。由于需要频繁的获取数据，对数据库的访问量大，Redis缓存数据库可以提高获取数据的速率。Redis是一个key-value存储系统，为保证效率，数据都是还存在内存中而且会周期性的把更新的数据写入磁盘中。

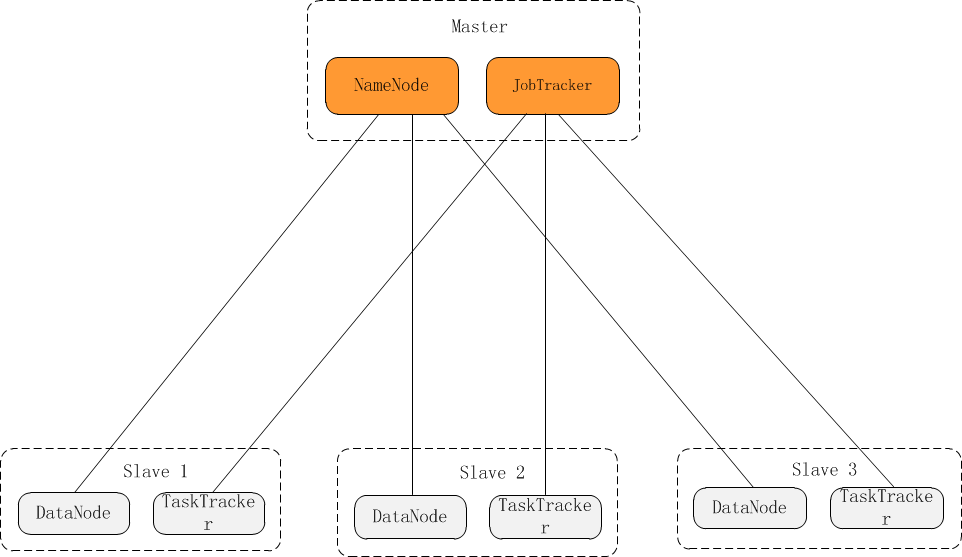


图1-2 Hadoop分布式集群

## 1.3软件部署方案

将软件部署在Windows Azure平台，通过创建云服务，可以在 Windows Azure 中部署多层应用程序，以便定义用于处理分发以及允许灵活扩展应用程序的多个角色。云服务由一个或多个 Web 角色和辅助角色组成，其中每个角色都具有其自己的应用程序文件和配置。

每个云服务都具有两个环境，可以将服务包和配置部署到这两个环境。在将云服务升级到生产环境之前，可以将其部署到过渡环境以对其进行测试。

# 2 Windows Azure操作系统的软件架构及实现技术

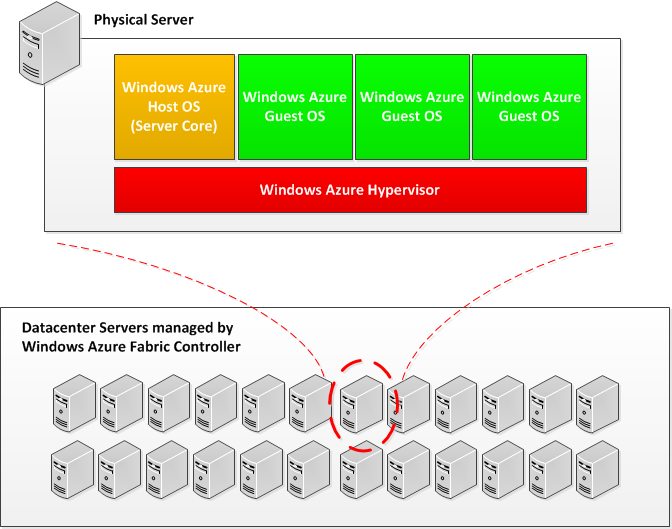
## 2.1 Windows Azure

### 2.1.1 简介

Windows Azure是微软开发的一套基于云计算的操作系统，作为当前云计算平台的主流，Windows Azure为开发者为提供了一个帮助其开发可运行在[云服务器](http://baike.baidu.com/view/3145014.htm)、[数据中心](http://baike.baidu.com/view/754520.htm)、Web和PC上的[应用程序](http://baike.baidu.com/view/330120.htm)的平台。在Windows Azure上，[云计算](http://baike.baidu.com/view/1316082.htm)的开发者能使用[微软](http://baike.baidu.com/view/2353.htm)全球的储存、计算能力和网络基础服务。

### 2.1.2 架构

Windows Azure是专为微软建设的数据中心管理所有服务器，网络以及存储资源开发的云操作系统。Windows Azure的数据中心由庞大的计算机群组成。如图2-1所示, Windows Azure采用了Hypervisor虚拟化技术，在其数据中心的每台物理服务器上安装了多个虚拟机。而这些虚拟机中一定要包含一个安装了Windows Azure Host OS的虚拟机和多个安装了Windows Azure Guest OS的虚拟机。其中Windows Azure Host OS负责对Windows Azure Host OS进行监管。虽然每台服务器上有一个Windows Azure Host OS主管服务器上的虚拟机，但是对于数据中心的众多服务器间的管理并不是件容易的事。为此，微软由Windows Azure Fabric Controller来对庞大的服务机群进行管理。Windows Azure Fabric Controller是一套中控管理系统，它能够自动化的管理数据中心内的所有实体服务器，其中包含了由用户要求的Windows Azure Guest OS的部署工作，定时的Hotfix修补以及管理不同版本的VM部署图像的复制等工作。在Windows Azure Host OS上，都包含一个Fabric Agent中控软件，它相当于Fabric Controller与每台服务器交互的接口。Fabric Agent能回报目前虚拟机的各项信息给Fabric Controller，同时Fabric Controller通过它来管理服务器上的虚拟机。

图2-1 Windows Azure计算资源与管理框架

### 2.1.3 Windows Azure的应用

软件部署在Windows Azure上，将软件部署在Windows Azure上主要有如下优点：

1．节约成本

应用程序需要部署在服务器上，而服务器价格昂贵并且服务器本身的维护费用也很高。采用Windows Azure部署应用程序，使不必为软件的部署购置更多的服务器，以及管理服务器的设施，这无疑节约了大量的硬件成本。

2．缩短了软件开发周期

利用Windows Azure能很容易的完成应用程序的部署工作，同时Windows Azure上提供了相应的服务能在应用程序的使用过程中，对应用程序进行监控与管理。这使得开发人员能专注于应用程序的开发，而不需要花费太多的精力在应用程序的部署上。

#### 2.1.3.1虚拟机

要在Window Azure上完成应用程序的部署，必须建立虚拟机。Windows Azure中的虚拟机是用户可以控制和管理的云服务器，在创建虚拟机后用户可以像访问其它服务器一样访问该虚拟机。

Windows Aurea根据用户的不同需求提供了两种创建虚拟机的方式，其一，通过使用Windows Azure管理门户映像库中的某一个映像创建虚拟机；其二，在本地上创建包含映像的虚拟硬盘文件并将其上载到Windows Azure，然后使用该映像创建虚拟机。

根据软件部署所需要的操作系统环境，选用了管理门户映像库中的Windows Server 2008 R2 SP映像来创建运行Windows Server的虚拟机。在选择完虚拟机映像后还需要对虚拟机的大小、连接的资源、DNS名称以及网络连接性进行定义。

Windows Azure提供了以下几种大小的虚拟机:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 大小 | CPU内核数 | 内存 | 磁盘大小 | 最大数据磁盘数 | 最大IOPS |
| ExtraSmall | 共享 | 768MB | 147GB | 1 | |  |  | | --- | --- | |  | 1x500 | |
| Small | 1 | 1.75GB | 197GB | 2 | 2x500 |
| Medium | 2 | 3.5GB | 262GB | 4 | 4x500 |
| Large | 4 | 7GB | 412GB | 8 | 8x500 |
| ExtraLarge | 8 | 14GB | 732GB | 16 | 16x500 |
| A5 | 2 | 14GB | 262GB | 4 | 4x500 |
| A6 | 4 | 28GB | 412GB | 8 | 8x500 |
| A7 | 8 | 56GB | 732GB | 16 | 16x500 |

鉴于软件将给多家公司提供服务，以及软件的访问量大，用户数多的情况，应当选择一个承载能力强的虚拟机。为此，虚拟机的大小选择为ExtraLarge。同时，虚拟机的大小还可以根据需要增加。

虚拟机的其它定义信息如下：

1. Virtual Machine Name(虚拟机名称): DispatchManageSystemVM
2. DNS名称：DispatchManageSystem.cloudapp.net

创建虚拟机时会生成一个包含虚拟机的云服务，定义的DNS名称即为虚拟机创建的云服务的名称。此值将作为联系虚拟机所属云服务的URI的一部分。

1. Region(区域): 将虚拟机的位置设为China，Windows Azure会根据所填位置选择最近的数据中心为虚拟机分配资源。

#### 2.2.3.2云服务

由于软件包含一个web前端以及提供数据服务的数据库，若将软件部署到单个虚拟机上，在大量用户并发访问时，很可能会出现虚拟机由于负载过大而出现故障的情况。为此，应用程序的部署采用了如图2-2所示的架构。通过创建多个虚拟机，并创建云服务来管理虚拟机，实现虚拟机间的通信。软件采用了图2-2的架构，将软件的逻辑代码运行在多个Windows Server 2008 R2的IIS上，将提供数据服务的数据库安装在多个Windows Server 2008 R2 with SQL Server 2012上。由于这些虚拟机都在同一服务中，因此可以通过配置云服务来配置硬件负载来分散请求，这样就提高了软件的访问速度。同时，由于软件可以只是用单个SQL Server实例，若当前使用的SQL Server实例发生异常，云服务会自动切换到其它SQL Server实例，任何一个单点故障都不会波及到其它虚拟机，这种方法提高了软件的可靠性。每个虚拟机都包含了一个自己公用的IP地址，而同一个服务中的所有VM都可通过一个公用的IP地址进行访问，用户只要通过这个IP地址就能访问软件。

要使用云服务，需要完成云服务的配置，这包含以下三个组件：

1.服务定义文件：云服务定义文件(.csdef)可定义服务模型，包括运行软件所需的虚拟机数量；

2.服务配置文件：云服务配置(.cscfg)提供云服务和各个虚拟机的配置设置。

3.服务包：服务包(.cspkg)包含软件代码和服务定义文件。

课题组通过Vistual 2012发布软件，并将软件发布包上载到运行软件web前端的虚拟机上，同时将本地SQL Server中的数据向Sql Azure迁移，通过Windows Azure虚拟机上的Sql Server管理Sql Azure中的数据完成了软件的发布。

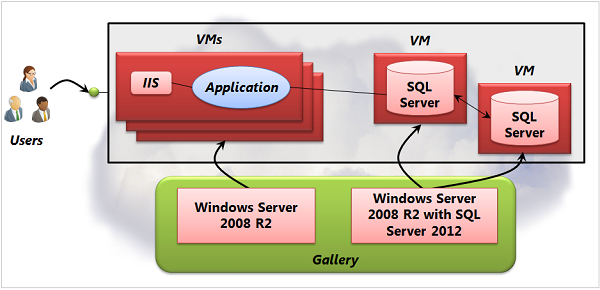


图2-2 软件部署框架

## 2.2 Sql Azure

### 2.2.1 简介

SQL Azure是微软构建在Windows Azure云操作系统之上的运行[云计算](http://baike.baidu.com/view/1316082.htm)的关系数据库，它体现了云存储，为用户提供了网络型的应用程序[数据存储](http://baike.baidu.com/view/551712.htm)的服务。

### 2.2.2 架构

SQL Azure的架构如图2-3所示可分为四个部份:

1．客户端层（Client Layer）

客户端层是唯一运行在微软数据中心之外的一层，不使用SQL Azure任何特殊的组件，而是使用了当前客户端上安装的SQL Server的功能，如ADO.Net、WCF服务、ODBC、SQL Server管理工具。客户端层使用API初始化一个TDS连接，通过1433端口连接到SQL Azure上。客户端层通过运行SQL Server来构建云应用。

2．服务提供层（Service Layer）

服务提供层是 SQL Azure 显露在[客户端](http://baike.baidu.com/view/930.htm)前面的服务接口 (Facade 模式)，负责接取所有向 SQL Azure 提交要求的TDS over SSL连接与指令，当连接进入 SQL Azure 时，SQL Azure Load Balancer(负载平衡) 会分派连接到不同的 SQL Azure Gateway中。SQL Azure Gateway负责处理 TDS 连接，管理连接层安全性以及解析指令是否有内含潜在威胁的指令，再交由连接管理员将连接分派到位于平台提供层内不同的 SQL Azure [数据库服务器](http://baike.baidu.com/view/32944.htm)中进行处理，SQL Azure Gateway 也会管理对 SQL Azure 的连接，以避免可能会封锁住服务器的连接 (例如过长的查询或过长的数据库交易等)。

3．平台提供层 (Platform Layer)

平台提供层则是以 Windows Azure Computes 的虚拟机簇 (Cluster)，每台虚拟机都安装有 SQL Server 2008及其管理的数据库，通常一份数据库会分散到三至五台的 SQL Server VM 中，而每台 SQL Server VM 也安装了 SQL Azure Fabric [中控软件](http://baike.baidu.com/view/4224988.htm)，并通过 SQL Azure Fabric 与 SQL Azure Gateway 的管控下，所有对单一数据库的连接都不一定会持续连入同一台 SQL Server VM 中。SQL Server VM 内也安装了 SQL Azure Management Service，它会负责对每个数据库间的数据复写工作，以保障 SQL Azure 的基本高可用性要求。每台 SQL Server VM内的 SQL Azure Fabric 和 Management Service 都会彼此交换健康与监控信息等，以保持整体服务的健康与可监控性。

4．基础建设层 (Infrastructure Layer)

基础建设层由Windows Azure Computes以及其高度可扩充性的运算与网络基础架构来组成，以支持 SQL Azure 所需的高可用性以及高扩充性等云特色。

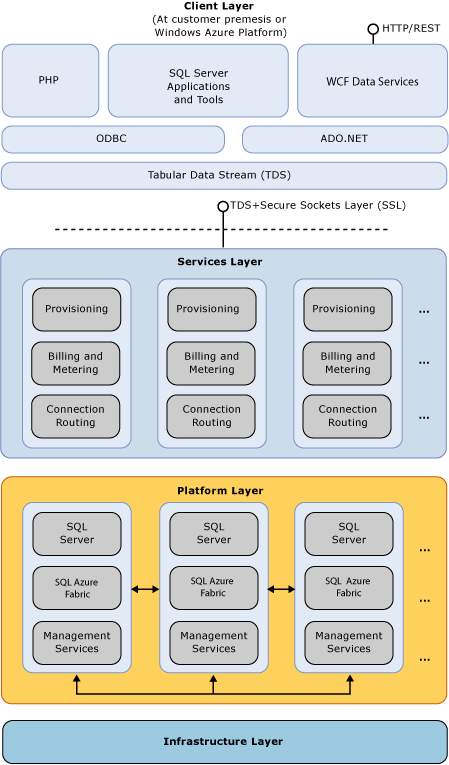


图2-3 SQL Azure的架构

### 2.2.3 Sql Azure的应用

Sql Azure具有如下优点：

1.自主管理

Sql Azure提供了企业级数据中心的规模和功能，省去了日常管理本地Sql Server实例的之处。自主管理的能力使得企业既不用增加本地IT的支持负荷，也不用分散具有良好技能的职员的精力从本职工作到维护部门的数据库应用程序。

2．高可用性

Sql Azure构建于久经考验的Windows Server和Sql Server技术之上，拥有足够的弹性以处理所有使用上和负载的变化。服务会再多个物理服务器上复制多份数据冗余拷贝以维持数据可用性和业务持续性。

3．可拓展性

Sql Azure的一大关键优势在于能够轻松拓展用户的解决方案。随着数据增长，数据库也需要纵向拓展和横向拓展。纵向拓展往往会有一个上限，而横向拓展并没有实际的限制。

鉴于Sql Server的这些优点，软件的用户登录和注册模块数据部署在Sql Azure上，利用Sql Azure分布式的数据中心，保证了数据的可用性和安全性。

#### 2.2.3.1 Sql Azure Server

Sql Azure在Window Azure上提供了关系数据存储、访问和管理服务，要完成软件登录和注册模块的数据部署需要在Windows Azure上创建用于管理Sql Azure的逻辑服务器。Sql Azure Server作为Windows Azure上用于管理和运行Sql Azure的逻辑服务器，其提供了软件访问Sql Azure的接口。创建Sql Azure Server需要完成如下配置：

1.管理员设置

在这一部分，需要设置Login Name与Login Password。Sql Azure使用Sql 身份验证进行加密连接，管理员名称与管理员密码将作为连接Sql Azure的登录凭证来保证Sql Azure安全性。

2.服务器区域：China

区域将确定服务器的地理位置，区域不能随意切换。基于软件的服务对象范围，将区域设置为China。

1）访问设置:允许Windows Azure 服务访问此服务器

2）DNS名称：DispatchManageSystem.database.windowsnet

3）软件需要使用此DNS名称连入数据库。

#### 2.2.3.2 创建数据库

完成Sql Azure Server的创建后，需要在Sql Azure Server创建数据库。Sql Azure提供了两个数据库版本，分别为Web版数据库和企业数据库。其中Web版数据库的大小最大可达5GB，企业数据库的大小可达50GB。基于软件的大数据量，选择了企业版数据库。数据库的最大大小时在首次创建数据库时指定的，在创建完数据库后仍可对数据库的大小进行更改。

#### 2.2.3.1 配置Sql Azure Server防火墙

每个Sql Azure Server都有自己的防火墙设置，Sql Azure管理人员可以自由设置不同的客户端来源模型。通过配置逻辑服务器的防火墙,将部署软件web前端的虚拟机IP地址设置为允许访问Sql Azure的IP地址。设置防火墙后，当请求访问Sql Azure的IP地址不在允许的IP地址范围中，防火墙将阻止访问请求，这提高了Sql Azure的安全性。

#### 2.2.3.2 数据库迁移

由于用户登录和注册模块数据量大且并行访问量大，因此需要创建多个数据库来保证数据库的负载平衡，使得软件前端在需要访问Sql Azure时可以选择随机访问其中一个数据库。同时，开发人员可以使用ADO,ADO.NET,Entity Framework,Linq to SQL等方式来访问Sql Azure数据库。

课题组将本地Sql Server数据库打包上载到Sql Azure Server上完成了用户登录和注册模块数据的部署。

# 3 Web客户端技术

## 3.1 Html5与ASP.NET应用

HTML5取消过时的标记，可以自动适应网页的设计，提供了更多的元素属性，比如required，email这些属性会直接验证是否为空，email地址是否有效。在软件开发中主要采用html5。

ASP.NET技术，它是一种嵌入型的脚本语言可由因特网服务器执行，运行与IIS中。在开发中在html中嵌入ASP.NET，作为数据交换的主要手段。ASP.NET支持页面与代码分离，在aspx文件中进行页面的设计，在aspx.cs文件中对页面产生的事件进行处理。在web.config文件中可以对session进行配置还可以配置数据库连接。

## 3.2 css应用

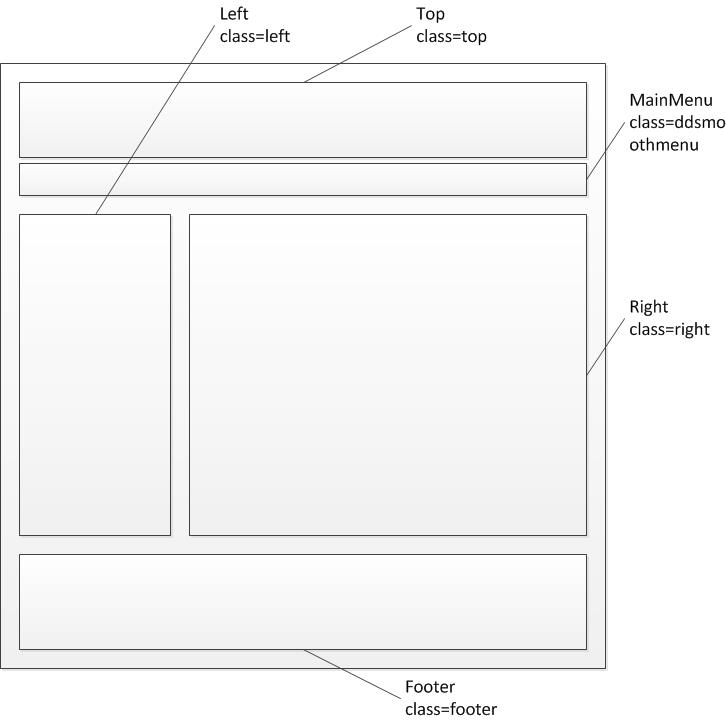
网站布局采用div+css模式，将网站样式写入css文件，在页面中去调用css文件中的样式去控制网页元素的位置，大小，颜色等等。使网站结构与样式分离，便于后期维护；可以将样式定义精确到像素的级别；可以用多套样式，使网页有任意样式切换的效果，可以降低服务器成本。页面布局分为头部，导航，左边，右边和尾部。在头部是网站的logo和宣传图片，导航为网站整体导航，左边为每一栏目的导航，右边为主要信息，尾部为版权等信息页面布局如图3-1。

图3-1页面布局

## 3.3 AJAX应用

Asp.net ajax是一个完整的开发框架，容易实现与现有的asp.net程序相结合。

ASP.NET AJAX Control Toolkit含有大量的独立AJAX控件和对ASP.NET原有服务器控件的AJAX功能扩展，实现起来也非常简单。

传统的html向服务器发送请求返回结果是会整个页面刷新，ajax的局部刷新显得跟人性化，在开发中采用ajax的局部刷新软件有更好的用户体验。其中用了很多ajax的控件比如ModalPopupExtender等，在进行局部数据交换时用ajax技术采用异步数据交换，实现局部刷新，而不是全局网页的刷新。这样不仅网页在加载时减少数据交换而且提高了软件加载速度。

软件中还用得了javascript脚本语言，通过javascript语言进行动态效果的制作，软件UI层数据校验全部由javascript编写的脚本来进行验证。

## 3.4 Silverlight应用

在Silverlight平台上实现开发WebGIS的接口，通过访问地图服务、影像服务、几何服务、地理处理服务、要素服务、网络服务等，还可以访问OGC标准的WMS、WFS、WCS等服务，实现在sliverlight地图的绘制。

在实时监控模块中，silverlight提供地图支持，通过服务将从数据库中提取的订单的地理坐标在silverlight制作的地图中显示，在实时监控模块中实现对车辆的实时监控，将车辆的地理位置信息在地图中显示，标明车辆行驶经过的路线。如图3-2实时地图监控。

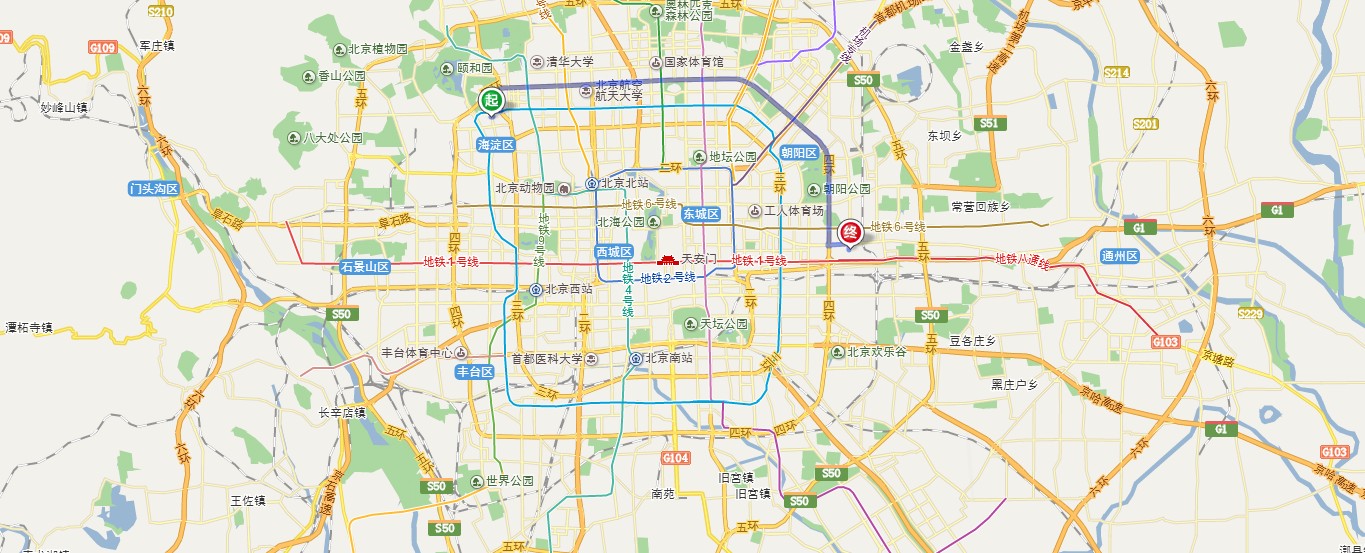


图3-2 实时地图监控4 海量数据分布式存储

## 4.1 分布式存储框架

### 4.1.1 Hadoop

Hadoop是Apache软件基金会旗下的一个开源分布式计算平台。以Hadoop分布式文件系统（Hadoop Distributed Flle System,HDFS）和MapReduce(Googel MapReduce的开源实现)为核心的Hadoop为用户提供了系统底层细节透明的分布式基础架构。HDFS的高容错性，高伸缩性等优点允许用户在不了解分布式系统底层细节的情况下开发并行应用程序。所以用户可以利用Hadoop轻松地组织计算机资源，从而搭建自己的分布式计算平台。并且可以从分应用集群的计算和存储能力。完成海量数据的处理。经过业界和学术界长达10年的锤炼，目前的Hadoop 1.0.1已经趋于完善，在实际的数据处理和分析任务中担当着不可替代的角色，如图4-1所示Hadoop框架。

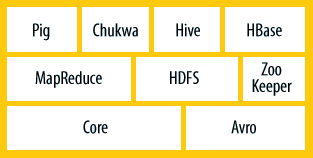


图4-1 Hadoop框架

1．Hadoop的功能与作用

为什么需要Hadoop呢？众所周知，现代社会的信息增长速度很快，这些信息中有积累着大量数据，其中包括个人数据和工业数据。预计到2020年，每年产生的数字信息中将会有超过1/3的内容驻留在云平台中或借助云平台处理。需要对这些数据进行分析处理，以获取更多有价值的信息。那么如何高效地存储管理这些数据，如何分析这些数据呢？这时可以采用Hadoop系统。在处理这类问题时，它采用分布式存储方式来提高读写速度和扩大存储容量；采用MarReduce整合分布式文件系统上的数据，保证高速分析处理数据；与此同时还采用存储冗余数据来保证数据的安全性。

Hadoop的HDFS具有高容错性。并且是基于Java语言开发的，这使得Hadoop可以部署在低廉的计算机集群中，同时不限于某个操作系统。Hadoop的HDFS的数据管理能力，MapReduce处理任务时的高效率以及它的开源特性，使其在同类分布式系统中大放异彩，并在众多行业和科研领域中被广泛应用。

2．Hadoop的优势

Hadoop是一个能够让用户轻松架构和分布式计算平台。用户可以轻松地在Hadoop上开发运行处理海量数据的应用程序。它主要有以下几个特点：

1）．高可靠性。Hadoop按位存储和处理数据的能力值得人们信赖。

2）．高扩展性。Hadoop是在可用的计算机集簇间分配数据完成计算任务的，这些集簇可以方便的扩展到数以千计的节点中。

3）．高容错性。Hadoop能够自动保存数据的多份副本，并且能够自动将失败的任务重新分配。

3．Hadoop应用现状和发展趋势

由于Hadoop优势突出，基于Hadoop的应用已经遍地开花，尤其是在互联网领域。

Hadoop目前已经取得了非常突出的成绩。随着互联网的发展，新的业务模式还将不断涌现，Hadoop的应用也会从互联网领域向电信，电子商务，银行，生物制药等领域扩展。相信在未来，Hadoop将会在更多的领域中扮演幕后英雄，为提供更加快捷优质的服务。

### 4.1.2 MapReduce

　　MapReduce采用"分而治之"的思想，把对大规模数据集的操作，分发给一个主节点管理下的各个分节点共同完成，然后通过整合各个节点的中间结果，得到最终结果。简单地说，MapReduce就是"任务的分解与结果的汇总"。

　　在Hadoop中，用于执行MapReduce任务的机器角色有两个：一个是JobTracker；另一个是TaskTracker，JobTracker是用于调度工作的，TaskTracker是用于执行工作的。一个Hadoop集群中只有一台JobTracker。

　　在分布式计算中，MapReduce框架负责处理了并行编程中分布式存储、工作调度、负载均衡、容错均衡、容错处理以及网络通信等复杂问题，把处理过程高度抽象为两个函数：map和reduce，map负责把任务分解成多个任务，reduce负责把分解后多任务处理的结果汇总起来。

需要注意的是，用MapReduce来处理的数据集（或任务）必须具备这样的特点：待处理的数据集可以分解成许多小的数据集，而且每一个小数据集都可以完全并行地进行处理。

1．MapReduce处理过程

在Hadoop中，每个MapReduce任务都被初始化为一个Job，每个Job又可以分为两种阶段：map阶段和reduce阶段。这两个阶段分别用两个函数表示，即map函数和reduce函数。map函数接收一个<key,value>形式的输入，然后同样产生一个<key,value>形式的中间输出，Hadoop函数接收一个如<key,(list of values)>形式的输入，然后对这个value集合进行处理，每个reduce产生0或1个输出，reduce的输出也是<key,value>形式的。MapReduce处理大数据集的过程如图4-2所示：

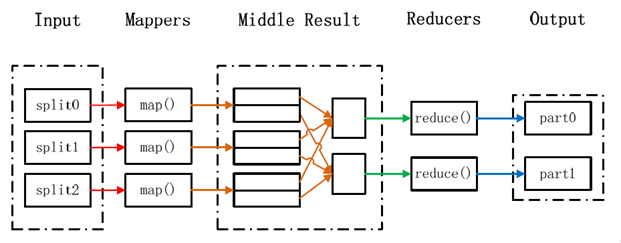


图4-2 MapReduce处理大数据集的过程

MapReduce依赖Hadoop FileSystem存储job执行过程中需要的所有资源文件。这些文件有job的jar文件、job的配置文件、job的mapper需要处理的目标文件(输入文件)以及job的输出结果。MapReduce可以根据配置文件中File System的URI判断当前是使用哪种Hadoop支持的File System，默认是local system。更关注job在TT上的表现，而TT又是依赖于DN，所以之后所说的File System都是指HDFS。

2．用途

在Google，MapReduce用在非常广泛的应用程序中，包括“分布grep，分布排序，web连接图反转，每台机器的词矢量，web访问日志分析，反向索引构建，文档聚类,机器学习，基于统计的机器翻译...”值得注意的是，MapReduce实现以后，它被用来重新生成Google的整个索引，并取代老的ad hoc程序去更新索引。

MapReduce会生成大量的临时文件，为了提高效率，它利用Google文件系统来管理和访问这些文件。

### 4.1.3 HDFS

　　HDFS是构建在PC硬件之上的分布式文件系统，非常适合需要访问海量数据的应用。它和现有的分布式系统最大的区别是：高容错性和低成本。HDFS主要为了实现以下目标：

　　1． 硬件错误

一个典型HDFS可能包括成白上千台机器，每台机器存储一部分的数据。每台机器都有出错的可能，导致HDFS总会有些机器处于不工作的状态。因此检测失败和快速恢复数据是HDFS的核心架构目标。

　　2． 流数据访问

在HDFS上运行的应用需要流式顺序访问他们的数据，不适合那种随机访问数据的模式，这样做可以提供非常高效的数据访问速度。

　　3． 海量数据

在HDFS上的应用都会有自己的海量数据。HDFS上的文件大小一般都会是G字节级别。因此，HDFS更适合支持大文件，而且支持千万个数据文件。

　　4． 简单一致性模型

HDFS只支持一次写，多次读的文件访问模式。一个文件一旦创建，写，关闭之后就不能再改变。 MapReduce和网络爬虫等大部分互联网应用都符合这个模型。

　　5． 移动计算比移动数据要容易

如果数据和计算在一台机器上，可以省去了网络IO时间，这样计算效率更高。HDFS提供了接口查询数据在集群中的分布情况，这样应用可以就近分布计算单元。

　　6．提供类似Linux的命名空间

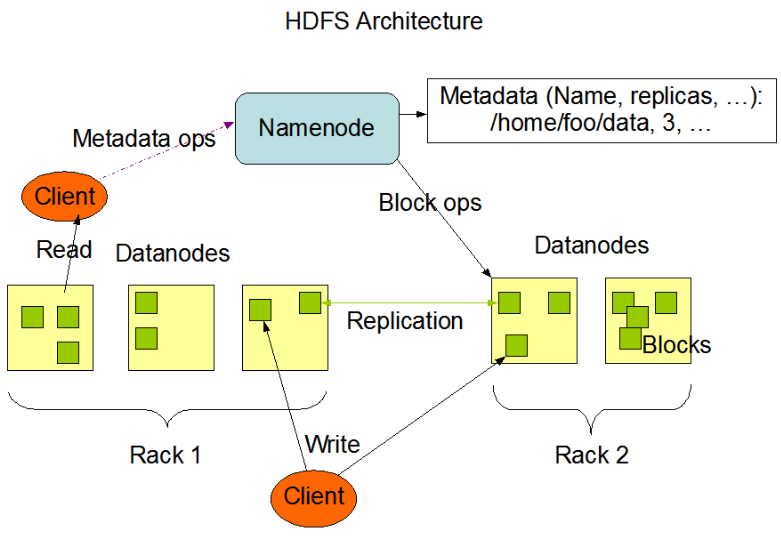
用户可以创建目录，在目录里存储文件。文件组织结构和linux非常类似。你可以创建，删除，移动，重命名文件或者目录。

　　HDFS是一个 master/slave的架构。HDFS只有一个NameNode，即master。master负责管理文件系统命名空间和client对文件的访问。此外，HDFS有很多DataNode，通常一个机器节点一个DataNode，管理这个节点上的存储。HDFS对外提供一个文件系统名字空间允许用户把数据存为文件的格式。

文件按block划分文件拆分成若干个block，这些block存放在DataNode节点上。NameNode操作名字空间比如：打开，关闭，重命名文件目录。DataNode负责为文件系统的客户提供读/写操作服务。DataNode同时还为NameNode提供block创建，删除，备份机制。

NameNode只负责元数据信息，没有数据流。NameNode维护名字空间,任何对文件系统名字空间的改动都记录在NameNode。系统的文件组织结构和linux非常类似。你可以创建，删除，移动，重命名文件或者目录。

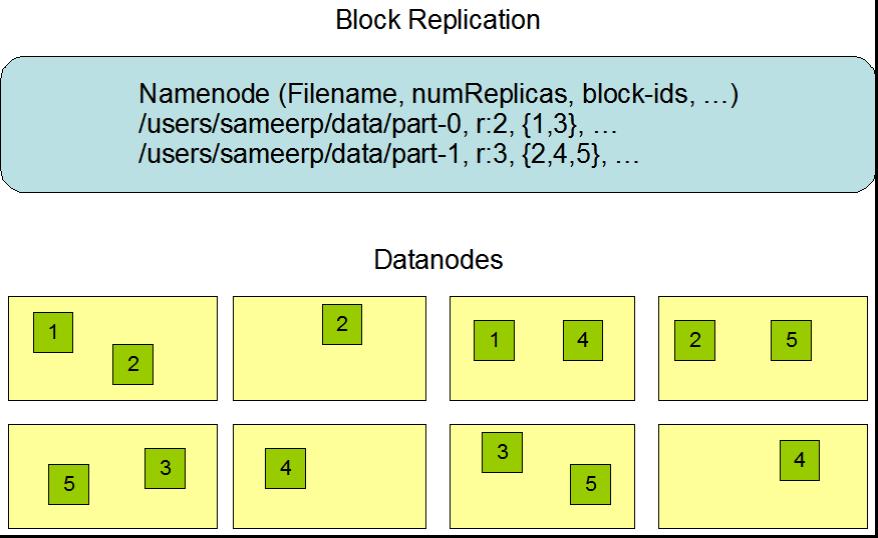
以上过程如下图4-3所示：

图4-3 HDFS处理过程

数据备份，bolck备份HDFS设计为大文件提供可靠存储。每个文件被分割为一系列的block，每个block都有相同的大小（64M），除了最后一个。为了容灾，文件的block会有备份，备份的个数可以由上传者指定，缺省是3。

block信息上报namenode周期性(缺省为一小时)的收到来自datanode的bolck上报信息，namenode如果判断某个文件的bolck备份有丢失，会产生一个新block。

数据备份分布关系如下图4-4所示：

图4-4 数据备份分布关系

HDFS的元数据持久化，dfs.name.dir这个参数设置HDFS的元数据信息存放在本地操作系统的目录，如果有多个目录用逗号分开。

目录里的文件示例如下所示：

1049092 2009-11-12 09:15 edits   
17678000 2009-11-12 08:50 fsimage  
8 2009-11-12 08:50 fstime   
100 2009-11-12 08:50 VERSION

HDFS的健壮性（容错处理）

磁盘错误，心跳和重新备份每个datanode周期性发送心跳信息给namenode。网络中断可能会导致一批datanode连不上namenode。namenode检测到一段时间没有上报心跳后，把这个datanode标识为dead，不再分配新的io请求给它。在这个datanode上的所有数据都不能访问了，这就会导致一些block的备份数量会少于指定的值。namenode会经常检查block备份数量，发起重新备份。

Rebalancing检查某个datanode的硬盘容量低，会自动把数据移到另外的datanode。（这个特性还在开发中）元数据磁盘错误FsImage和EditLog是HDFS的核心数据结构。这些文件损坏会导致HDFS失效。namenode可以配置支持多份元数据拷贝。

Namenode单点namenode机器是整个hdfs的唯一单点。如果namenode机器down了，手工干预是不可避免的。现阶段namenode自动重启恢复到另一台机器还不支持。目前简单有效的方式就是用抢IP或者DNS切换NameNode机器。

### 4.1.4 Hbase

hbase是建立的hdfs之上，提供高可靠性、高性能、列存储、可伸缩、实时读写的数据库系统。

它介于nosql和RDBMS之间，仅能通过主键(row key)和主键的range来检索数据，仅支持单行事务(可通过hive支持来实现多表join等复杂操作)。主要用来存储非结构化和半结构化的松散数据。

与hadoop一样，Hbase目标主要依靠横向扩展，通过不断增加廉价的商用服务器，来增加计算和存储能力。

1．HBase的优点：

分布式，易扩展，高性价比，运维成本低都是它的优点。HBase可以支持海量数据，单张表的数据量不上T，都不好意思出来打招呼。甚至可以拿很烂的SATA盘来作为存储，由于依赖底层的HDFS。新装的机器甚至可以不用做硬RAID。

HBase可以在任何时候随时宕掉2，3台机器，就当什么都没发生似的（当然master除外，但是hbase，hadoop的master节点负载都很”清闲“）。这是HBase本身分布式的架构，先天性的优势。

其物理存储如如图4-5所示：

1）Table中的所有行都按照row key的字典序排列。

2）Table 在行的方向上分割为多个Hregion。

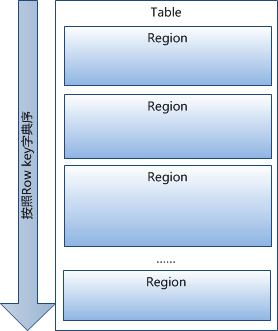


图4-5.

### 4.1.4 集群

1．软件版本

Hadoop: hadoop-0.20.2.

JDK: jdk1.7

OS: ubuntu 13.04

2．配置的机器：

主机[服务器master]（一台）： 192.168.99.1 master

从机[服务器slaves]（200台）： 192.168.99.2 slave1

192.168.99.3 slave2

……

192.168.99.201 slave200

其拓扑结构如下图4-6：

图4-6 系统拓扑结构

3．群集的工作

在集群的内部，服务器与服务器之间通过1GB宽带在局域网中进行连接 ，以此来保证海量数据处理过程中的传输速度。

在的服务器群集中，Master提供对外部系统的直接访问服务web Service ，当外部系统对集群发出数据请求，首先得到消息的是Master，Master对任务进行分析，把一个任务进行多个任务，并把任务分别发送给各个Slave，Slave安装指定的方式执行任务，并把执行结果反馈给Master，Master把结果进行合并，把最终结果返回给外部系统。

集群相应过程如下图4-7所示：



图4-7 集群相应过程

4．集群优化

为了让集群的性能得到充分的提高，通过修改配置文件来优化集群。

Hadoop通用参数调整

1） dfs.namenode.handler.count或mapred.job.tracker.handler.count

namenode或者jobtracker中用于处理RPC的线程数，默认是10，由于集群较大，设为160，以提高数据处理速度。

2） dfs.datanode.handler.count

datanode上用于处理RPC的线程数。设为8，使datanode机器资源充分利用起来。

3） tasktracker.http.threads

HTTP server上的线程数。设为40，以使tasktracker机器资源充分利用起来。

5．HDFS相关配置

1） dfs.replication

文件副本数，设为3，使数据文件的可靠性提高。

2） dfs.block.size

HDFS中数据block大小，设为256MB。（也可以通过参数mapred.min.split.size配置），使文件系统在block合并的时间减少。

6．map/reduce 相关配置

1） {map/reduce}.tasks.maximum

同时运行在TaskTracker上的最大map/reduce task数，一般设为2，防止TaskTracker任务过多，维护TaskTracker机器的硬件。

7．map task相关配置

（1） io.sort.mb

Map task的输出结果和元数据在内存中所占的buffer总大小。设为200M。

（2） io.sort.spill.percent

这个值就是上述buffer的阈值，使用默认值0.9，减少buffer合并消耗的时间。

8． reduce task相关配置

（1） Mapred.reduce.parallel

Reduce shuffle阶段copier线程数。设为25，使shuffle阶段的时间能缩短。

## 4.2 分布式存储数据结构

需要同时多人通过移动客户端对数据定时（十秒）向数据库进行提交，并需要迅速的响应，对于业务比较兴旺的系统，这个数据量是可想而知的。系统中要处理的是全国范围内的物流配送信息，对于这样一个需要数据海量处理的系统，数据库的性能要求无疑是很高的，对服务器的稳定性要求也不是一般的，面对这样高标准，找到了一个这样的解决方案，那就是Hbase。

### 4.2.1 Hbase集群设计

Hbase集群是在hadoop集群之上搭建的，所以硬件环境与前面说的hadoop集群基本一致。

OS是Ubuntu Server 13.04，HBase版本是0.20.6。

HMaster&NameNode:master,IP:192.168.99.1;（SecondaryNameNode）

 HRegionServer&HQuorumPeer:slave1,IP:192.168.99.2;

 HRegionServer&HQuorumPeer:slave2,IP:192.168.99.3;

……

HRegionServer&HQuorumPeer:slave200, IP:192.168.99.201;

在的集群中，由于数据承载量比较大，配置了有200个Region Server，以保证数据处理的效率。

本集群中，通过ZooKeeper来对Hregion进行监控维护，当出现死节点时，ZooKeeper会采集相应的应对方法，以此增强系统的稳定性。

Hbase集群中，各个部分的分布情况如下图4-8所示：



图4-8 Hbase集群的分布情况

1．Client端设置

含访问hbase的接口，client维护着一些cache来加快对hbase的访问，比如region的位置信息。

2．Zookeeper设置

1）保证任何时候，集群中只有一个master

2）存贮所有Region的寻址入口。

3）实时监控Region Server的状态，将Region server的上线和下线信息实时通知给Master

4）存储Hbase的schema,包括有哪些table，每个table有哪些column family

3．Master设置

1）为Region server分配region

2）负责region server的负载均衡

3） 发现失效的region server并重新分配其上的region

4）GFS上的垃圾文件回收

5）处理schema更新请求

4．Region Server设置

1）Region server维护Master分配给它的region，处理对这些region的IO请求；

2）Region server负责切分在运行过程中变得过大的region。

可以看到，client访问hbase上数据的过程并不需要master参与（寻址访问zookeeper和region server，数据读写访问regione server），master仅仅维护者table和region的元数据信息，负载很低。

### 4.2.2 数据访问接口

Hadoop中还给提供了很多对hbase的访问接口如Native Java API，通过Native Java API可以进行编程。然后通过web service发布在局域网内，外部Redis缓存数据库通过访问web service来达到对数据访问的目的。

在web service中主要提供了数据增删改查的操作接口，例如：

insertData，其中提供tableName，ColumnName,Values三个参数。对表名为tableName的数据表插入数据，列名ColumnName与值Values一一对应。

UpdateData，其中提供Key ，tableName，ColumnName，Values四个参数。对表名为tableName的数据表中，行键为Key，列名为ColnumName的数据更新为Value。

deleteRow，其中提供rowkey，tableName两个参数。删除tableName表中，行键为rowkey的一条数据记录。

QueryByConditionKey，其中提供tableName，key两个参数。查询数据表tableName中行键为key的一条记录。结果各列通过String[]进行存放并返回。

QueryAll，其中提供tableName两个参数。查询数据表tableName中的所有记录，结果各列通过String[][]进行存放并返回。

QueryConditonNotkey，其中提供tableName，colnumKey，columnValue三个参数。查询数据表tableName中列名为colnumKey，相应值为columnValue的所有数据。

GetDistributionByPlateNumber，其中提供strPlateNumber一个参数。通过车牌号strPlateNumber获取当前对应的配送单ID，用于Android端，web service操作Hbase数据库的过程如图4-9所示。

4-9 web service操作Hbase数据库的过程

### 4.2.3 储存结构设计

1．表结构设计

1) Row Key

系统中最近写入HBase表中的数据是最可能被访问的，所以将时间戳作为row key的一部分，由于是字典序排序，所以可以使用Long.MAX\_VALUE - timestamp作为row key，这样能保证新写入的数据在读取时可以被快速命中。

2) Column Family

由于Hbase中在一张表里定义太多的column family影响效率，目前Hbase并不能很好的处理超过2~3个column family的表，所以仅使用一个column family。因为某个column family在flush的时候，它邻近的column family也会因关联效应被触发flush，最终导致系统产生更多的I/O。

Hbase的表结构(以系统中DistributionList表为例)，如表4-1所示：

表4-1 DistributionList表结构

|  |  |  |
| --- | --- | --- |
| RowKey | Column Family | |
| 9223372036850569854 | Info | value |
| Info : CourierTime | 2013/11/5 |
| Info : CompletionStatus | 未完成 |
| Info : CompletionTime |  |
| Info : StartStationID | 9223372036848569856 |
| Info :EndStationID | 9223372036849658965 |
| Info :PlateNumber | 苏DF0008 |
| Info :OrderID | 9223372036850358968 |

2．性能优化

1) In Memory

创建表的时候，可以通过HColumnDescriptor.setInMemory(true)将表放到RegionServer的缓存中，保证在读取的时候被cache命中。

2) Max Version

创建表的时候，可以通过HColumnDescriptor.setMaxVersion(int

maxVersions)设置表中数据的最大版本，如果只需要保存最新版本的数据，那么可以设置setMaxVersions(1)。

3) Time To Live

创建表的时候，可以通过HColumnDescriptor.setTimeToLive(int timeToLive)设置表中数据的存储生命期，过期数据将自动被删除，例如如果只需要存储最近两天的数据，那么可以设置setTimeToLive(2 \* 24 \* 60 \* 60)。

(4) Compact & Split

在HBase中，数据在更新时首先写入WAL 日志(HLog)和内存(MemStore)中，MemStore中的数据是排序的，当MemStore累计到一定阈值时，就会创建一个新的MemStore，并且将老的MemStore添加到flush队列，由单独的线程flush到磁盘上，成为一个StoreFile。于此同时， 系统会在zookeeper中记录一个redo point，表示这个时刻之前的变更已经持久化了(minor compact)。

StoreFile是只读的，一旦创建后就不可以再修改。因此Hbase的更新其实是不断追加的操作。当一个Store中的StoreFile达到一定的阈值后，就会进行一次合并(major compact)，将对同一个key的修改合并到一起，形成一个大的StoreFile，当StoreFile的大小达到一定阈值后，又会对 StoreFile进行分割(split)，等分为两个StoreFile。

由于对表的更新是不断追加的，处理读请求时，需要访问Store中全部的StoreFile和MemStore，将它们按照row key进行合并，由于StoreFile和MemStore都是经过排序的，并且StoreFile带有内存中索引，通常合并过程还是比较快的。

在的实际应用中，必要时手动进行major compact，将同一个row key的修改进行合并形成一个大的StoreFile。同时，可以将StoreFile设置大些，减少split的发生。

# 5 缓存数据库架构

## 5.1 缓存数据库

### 5.1.1 Redis

Redis是一款开源的、高性能的键-值存储（key-value store）。它常被称作是一款数据结构服务器（data structure server）。和Memcached类似，它支持存储的value类型相对更多，包括字符串（strings）、哈希（hashes）、列表（lists）、集合（sets）和 有序集合（sorted sets）等数据类型。 对于这些数据类型，你可以执行原子操作。例如：对字符串进行附加操作（append）；递增哈希中的值；向列表中增加元素；计算集合的交集、并集与差集等。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。

为了获得优异的性能，Redis采用了内存中（in-memory）数据集（dataset）的方式。根据使用场景的不同，你可以每隔一段时间将数据集转存到磁盘上来持久化数据，或者在日志尾部追加每一条操作命令。

Redis同样支持主从复制（master-slave replication），并且具有非常快速的非阻塞首次同步(non-blocking first synchronization)、网络断开自动重连等功能。同时Redis还具有其它一些特性，其中包括简单的check-and-set机制、pub/sub和配置设置等，以便使得Redis能够表现得更像缓存（cache）。

Redis还提供了丰富的客户端，以便支持现阶段流行的大多数编程语言。详细的支持列表可以参看Redis官方文档：http://redis.io/clients。Redis自身使用ANSI C来编写，并且能够在不产生外部依赖（external dependencies）的情况下运行在大多数POSIX系统上，例如：Linux、\*BSD、OS X和Solaris等。

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Python，Ruby，Erlang，PHP客户端，使用很方便。.redis单点的性能也非常高效（利用项目中的数据测试优于memcache）。

### 5.1.2 Redis性能

首先介绍与Redis几个相关类似的数据库，然后与Redis进行对比。

**Memcached Memcached** 是一个高性能的分布式内存对象缓存系统，用于动态Web应用以减轻数据库负载。它通过在内存中缓存数据和对象来减少读取数据库的次数，从而提高动态、数据库驱动网站的速度。Memcached基于一个存储键/值对的hashmap。其守护进程（daemon ）是用C写的，但是客户端可以用任何语言来编写，并通过memcached协议与守护进程通信。

**Memcachedb** memcachedb是 一个开源项目，给memcached分布式缓存服务器添加了Berkeley DB的持久化存储机制和异步主辅复制机制，让memcached具备了事务恢复能力、持久化能力和分布式复制能力，非常适合于需要超高性能读写速度，但是 不需要严格事务约束，能够被持久化保存的应用场景。

**Tokyo Tyrant** [Tokyo Tyrant](http://baike.baidu.com/view/3548503.htm)提供dbm数据库Tokyo Cabinet的网络接口。它使用简单的基于TCP/IP的简单二进制协议进行通信。同时它拥有Memcached兼容协议并且可以用HTTP/1.1协议进行数据交换。所以实现了跨平台，跨语言使用Tokyo Tyrant。采用[热备份](http://baike.baidu.com/view/1231806.htm)，更新日志记录，复制（replication）来实现高可用性和高可靠性。到目前为止，Tokyo Tyrant只能运行在linux, FreeBSD, Mac OS X, Solaris。

在读写海量数据时候，希望给客户端的效果是更快响应，虽然数据库本身自带缓存，但是效率与性能远远不及Redis，当然也有其它缓存数据库比如memchead等，但是效率也不及Redis，下面是对于500万条数据在其他数据库对于Redis下进行读取的一些比较，如表5-1，5-2所示。

表5-1 Redis读写速度

|  |  |  |
| --- | --- | --- |
| **缓存数据库** | **Write（次/秒）** | **Read（次/秒）** |
| Memcached | 55,989 | 50,974 |
| Memcachedb | 25,583 | 35,260 |
| Tokyo Tyrant | 42,988 | 46,238 |
| Redis | 85,765 | 71,708 |

表5-2 服务器的平均负载

|  |  |  |
| --- | --- | --- |
| **缓存数据库** | **Write（M/s）** | **Read（M/s）** |
| Memcached | 1.80, 1.53, 0.87 | 1.17, 1.16, 0.83 |
| MemcacheDB | 1.44, 0.93, 0.64 | 4.35, 1.94, 1.05 |
| Tokyo Tyrant | 3.70, 1.71, 1.14 | 2.98, 1.81, 1.26 |
| Redis | 1.06, 0.32, 0.18 | 1.56, 1.00, 0.54 |

上表是以100 bytes大小为单位并发写入数据库，然后在读取进行测试，从图中可以看出，Redis的读写性能最高的。

下面是对服务器本身的性能的影响。

在本地测试时成功写入500万条记录，共耗时524秒，平均每秒写入数据9542笔。磁盘上的数据文件大小134M。写入过程中，服务器内存、CPU和磁盘等资源使用情况如下图5-1所示：

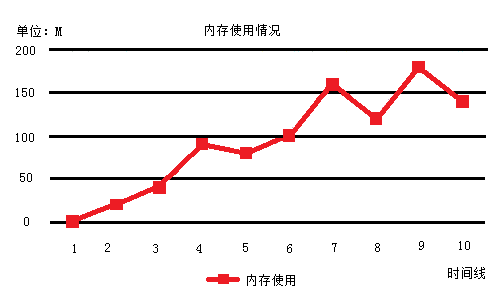


图5-1 内存使用情况



图5-2 CPU使用情况

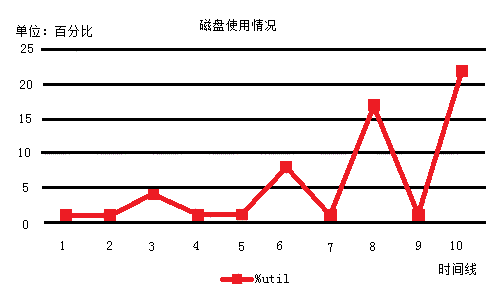


图5-3 磁盘使用情况

可见，内存使用平稳上升，最后占用140M左右，主要用来缓存数据，对比数据文件的大小可以设想，在操作系统内存可用的情况下，内存的分配和数据文件的大小是大致相当的。内存占用会有临时的超过实际占用的几个点，而且会马上释放掉，CPU和磁盘IO都表现出周期性的上下波动，估计这些和Redis刷新数据到磁盘的实现机制有关，在高点的时候正好是数据写入磁盘的过程。总体来说占用资源很少，表现也很平稳。

本地测试成功读出500万条记录，共耗时184秒，平均每秒读出数据27174笔。读数据过程中没有发生磁盘IO。CPU有所消耗，Idle值稳定在77左右，等待CPU资源的进程一直是1个。内存表现平稳没有波动。如下图5-4所示：



图5-4 CPU使用情况

通过以上测试结果可以看出，Redis数据操作都在内存完成，定期刷新到磁盘，占用的磁盘空间和内存大小由实际的数据量决定，在内存可用的情况下二者是一致的。从功能角度看Redis高效的读写效率和丰富的数据结构能满足很多互联网应用场景的需求，而且在高负载的情况下，数据内存化是趋势，总体来说Redis虽然还在不断发展之中，代码量也不多，但是一个轻量级的很有潜力的NoSQL产品，肯定会有越来越多的成功案例。

## 5.2 缓存数据库架构设计

### 5.2.1 表设计，数据结构选择

本系统有User，CompanyInfo，VehicleInfo，GPSInfo，GPSReturnsInformation ，DistributionList，CargoInfo，Order，StorageInfo，Role，PermissionAssignment，FunctionalAuthority，DeliveryStage等多个表，有些表在Redis中的存储结构类似，只选择其中的代表进行讲解。

1.GPSInfo，GPS信息

GPS信息表中只有2个字段，一个主键对应一个设备名称，对于这种单一主键对应，只需要利用Redis中最简单的数据结构类型**String**即可。

2.VehicleInfo车辆

VehicleInfo车辆信息表，属性描述如图5-5所示：



图5-5 VehicleInfo车辆信息属性描述

对于此类型的数据表主键对应多个数据类型，对应redis有2种设计方案：

1）直接利用**String**类型数据结构，结构图如图所示，这种方式的缺点是，增加了序列化/反序列化的开销，并且在需要修改其中一项信息时，需要把整个对象取回，并且修改操作需要对并发进行保护，引入CAS等复杂问题，如图5-6所示。



图5-6 redis直接利用**String**类型数据结构

1. 利用Redis数据结构**Hash**结构

为了克服这种缺陷redis引入了hash数据结构，Redis的**Hash**实际是内部存储的Value为一个HashMap，并提供了直接存取这个Map成员的接口，如图5-7所示：



图5-7利用Redis数据结构**Hash**结构

也就是说，Key仍然是VehicleInfoID， value是一个Map，这个Map的key是成员的属性名，value是属性值，这样对数据的修改和存取都可以直接通过其内部Map的Key(Redis里称内部Map的key为field), 也就是通过 key(VehicleInfoID) + field(属性标签) 就可以操作对应属性数据了，既不需要重复存储数据，也不会带来序列化和并发修改控制的问题。很好的解决了问题。

这里同时需要注意，Redis提供了接口(hgetall)可以直接取到全部的属性数据,但是如果内部Map的成员很多，那么涉及到遍历整个内部Map的操作，由于Redis单线程模型的缘故，这个遍历操作可能会比较耗时，而另其它客户端的请求完全不响应，这点需要格外注意。

3.Order订单列表，由于订单列表字段很多，就不一一列出来了，但是类似于twitter的关注列表，粉丝列表等，订单列表也需要一个集合来随时获取或者存储，所以Redis为提供了数据结构类型**List，**Redis list的实现为一个双向链表，即可以支持反向查找和遍历，更方便操作，不过带来了部分额外的内存开销，Redis内部的很多实现，包括发送缓冲队列等也都是用的这个数据结构，所以对于订单查找提供了良好的方法。

4.DeliveryStage配送阶段

DeliveryStage配送阶段属性描述，如图5-8所示：



图5-8 DeliveryStage配送阶段属性

从表面看，这个表与其他表并没有什么不同之处，但是他是一个配送阶段，所以是根据时间的先后进行查询更新，需要根据时间进行排序，这里Redis提供了**Sorted set**数据结构，其实Redis也有**Set**这个数据结构，对外提供的功能与list类似是一个列表的功能，特殊之处在于set是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时，set是一个很好的选择，并且set提供了判断某个成员是否在一个set集合内的重要接口，这个也是list所不能提供的。

set 的内部实现是一个 value永远为null的HashMap，实际就是通过计算hash的方式来快速排重的，这也是set能提供判断一个成员是否在集合内的原因。Redis sorted set与set类似，区别是set不是自动有序的，而sorted set可以通过用户额外提供一个优先级(score)的参数来为成员排序，并且是插入有序的，即自动排序。当你需要一个有序的并且不重复的集合列表，那么可以选择sorted set数据结构，比如twitter 的public timeline可以以发表时间作为score来存储，这样获取时就是自动按时间排好序的。Redis sorted set的内部使用HashMap和跳跃表(SkipList)来保证数据的存储和有序，HashMap里放的是成员到score的映射，而跳跃表里存放的是所有的成员，排序依据是HashMap里存的score,使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。

这里配送阶段是要根据时间先后来修改或插入配送状态的，所以在此用**sorted set**无论是从效率上来说还是开发者的角度都是最佳的.

其他表也与此类似，根据不同特性选择了不同的数据结构。

### 5.2.2 Redis数据导入与导出

用Redis做缓存数据，需要定时把一些数据比如数据库（Hbase）或者其他数据更新到Redis中，也需要定时把一些数据从Redis更新到数据库（Hbase）中的，在不同的数据表用到不同的Redis的数据结构，除了单个数据可以直接使用getset进行存储外，其他都是通过实体进行交互数据，最终实现数据之间的同步更新，Redis数据的导入与导出都是通过如下图所示的方式进行的，最终实现同步更新。

图5-9 Redis数据的导入与导出

通过上图5-9可以发现，Redis数据的导入导出都是通过实体进行相互之间转换来实现的，所以无论是单点不定时更新还是批量定时更新都能通过数据实体来实现。

### 5.2.3 Redis接口设计

Redis通过Jredis接口来访问它的各种数据结构，如Hash，string，Set等等，并通过这个接口实现数据结构所存数据的增删改查，具体过程见下图5-10所示。

5-10 Redis接口设计

从上图可看出，Redis的javaAPI提供了强大的接口Jredis，他能实现对他所支持的数据结构进行增删改查，所以通过设计适当的数据结构从而设计出实体进行封装，与Redis进行完美契合，最终能够完美的使用Redis 做缓存。

# 6 仓库管理系统

## 6.1库存信息管理系统系统架构设计

本系统为物流公司物流管理系统，主要面向物流管理人员，提供公司/站点信息添加、订单自动生成、为货物选择运输线路、选择配送货物车辆、出入库等功能。可以大大缓解管理人员的劳动强度，提高公司办事效率。

根据做出的需求分析以及体系结构的优缺点，决定采用B/S体系结构进行开发。采用这种结构可以很好的满足用户需求，而且容易开发和维护。在开发平台的选择上达成了共识：

1．技术上成熟且具有一定的先进性；

2．有高效、集成的开发工具；

3．应为开发人员熟练掌握；

4．软件平台提供商对该软件平台的后续支持能力。

首先在系统平台的选择上，有两点考虑：1.公司现有的各级软件系统都是基于微软Windows系列平台的，且公司没有在日后使用其他平台的打算；2、微软的Windows平台完全能满足开发、运行该系统的要求。因此此系统是基于此平台的。决定采用ASP.NET作为开发平台，它提供了为建立和部署企业级 Web 应用程序所必需的服务。ASP.NET 为能够面向任何浏览器或设备的更安全的、更强的可升级性、更稳定的应用程序提供了新的编程模型和基础结构。

数据库，选用超高性能缓存数据库Redis和用于分布式存储的数据库HBase相结合的方案，可以快速高效的处理海量数据（具体介绍请参考第四章、第五章）。其物理分布如图6-1所示：

 图6-1物理分布图

本系统所采用系统架构是基于Asp.net的三层架构，分别为**表现层**，**业务层**，和**数据层。**其结构如图所示6-2所示：



图6-2结构示意图

### 6.1.1表现层（Presentation Layer）

表现层用于用户接口的展示，以及用业务层的类和对象来“驱动”这些接口。在ASP.NET中，该层包括aspx页面、用户控制、服务器控制以及某些与安全相关的类和对象。

### 6.1.2业务层（Business Tier）

业务层用于访问数据层，从数据层取数据、修改数据以及删除数据，并将结果返回给表现层。通常该层被划分成两个子层：业务逻辑层（Business Logic Layer，BLL）和数据访问层（Data Access Layers，DAL）。业务逻辑层在数据访问层之上，也就是说BLL调用DAL的类和对象。DAL访问数据并将其转给BLL。数据访问层用数据库取的数据，把数据通过实体类（Model)或链表的形式给BLL，BLL处理数据给表现层。

### 6.1.3数据层（Data Tier）

数据层是数据库或者数据源。在.NET中，通常它是一个SQL Server或Access数据库，但本系统采用的Redis与HBase相结合方式，满足了高性能的要求和海量数据分布式存储的特点。

### 6.1.4小结：

由于采用了适当的体系结构和高效的开发工具,整个项目的进展很顺利。但也暴露了一些问题，需要改进。

首先，在编程时容易导致系统信息的扩散，如果有人恶意攻击服务器，程序执行将出现异常。这是，就会在页面上显示相应的错误信息。这些信息会暴露这台服务器的路径信息。这个漏洞往往会被利用，程序员对此也一头雾水，考虑从服务器入手，采用通用的异常说明界面，解决该问题。

其次，在设计中在用户界面定制方面，希望通过不断的实践，加强其灵活性。

## 6.2数据库设计

### 6.2.1基本业务分析

在库存管理系统中，数据库设计同样重要，设计质量的优劣，可直接影响数据库数据冗余度、数据的一致性、数据丢失等问题。下面将对本系统数据库设计进行详细说明。

**根据系统分析的结果设计该系统的数据库表设计。**

有不同的角色，如车辆管理员、配送管理员、公司管理员、分公司管理员等。

不同角色可以分配多个权限，并记录其权限分配的创建、结束时间。分配权权表对应功能权限表，每个功能权限表示可以进行不同的操作。

每个用户有用户名、角色信息、密码、所属公司、邮政编码、手机号码等信息。

每个公司有公司名称、业务名称、联系电话、联系人、传真、所在地址、公司级别、下属公司、相邻公司、经纬度等信息。

每个公司对应多辆配送车辆，每辆车车牌号，容量等级，当前配送状态（无配送，正在配送，维修），当前位置等信息。

每个订单有收货人信息、目的地信息、起始地信息、货物说明信息、容量等级、备注等信息。

一个订单对应多个配送阶段，配送阶段表明在配送的某一个阶段的配送起始站和配送终点站，以及完成状态。

配送单对应多个订单，这些订单在某个配送阶段有相同的配送起始站和配送终点站。

一个父订单可以对应多个子订单，父订单对应着总金额，收货人地址，联系方式，子订单有具体商品单价，数量，金额，订单号等信息。

### 6.2.2概念结构设计（E-R图）

下面将用E-R图表示系统各个功能模块所涉及的数据表及其联系。

1.注册管理模块E-R如图6-3所示：

图6-3注册管理模块

2.订单管理模块E-R图，如图6-4所示：

图6-4订单管理模块

3.出入库及配送模块E-R图，如图6-5所示：

图6-5出入库及配送模块

1. 系统总体E-R图，如图6-6所示：

图6-6系统总体E-R图

## 6.3物流配送流程图:

物流配送流程图，如图6-7所示：



图6-7物流配送流程图

### 6.3.1前台用户操作流程

用户操作流程图，如图6-8所示：



图6-8 用户操作流程图

具体操作流程：

首先在浏览器的地址栏中输入正确的地址和端口号进入系统登录界面，如下图6-9所示：



图6-9 系统登录

在用户名与密码栏中填入正确的信息后，“登录”系统：如果用户所填信息不符合格式要求，系统会弹出错误提示，请重新填写。如果没有用户没有账号，可以点击“注册账号”进行注册。注册页面如图6-10所示：



图6-10用户注册

用户可以根据要求进行注册，如上图所示用户所填写的内容必须符合格式要求，当用户操作成功时，系统会跳入订单界面，如图6-11所示：

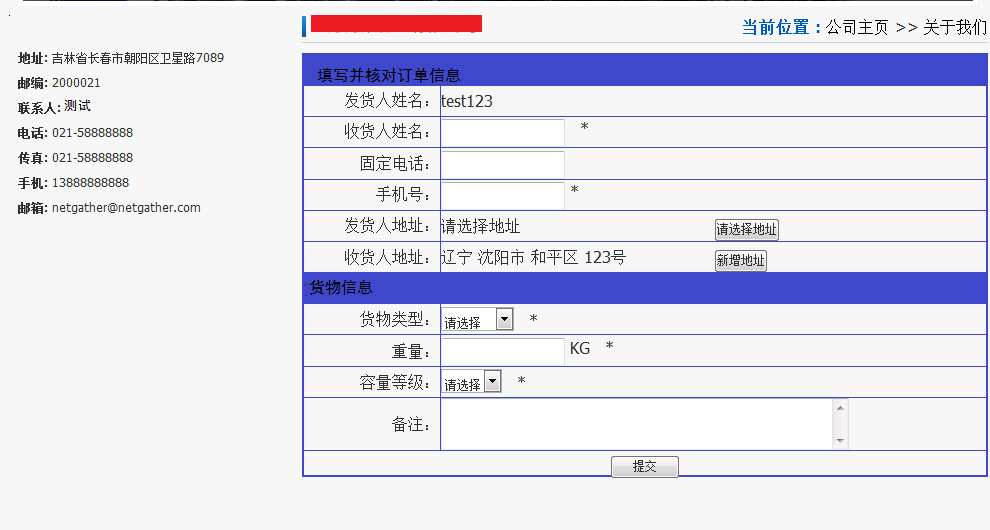


图6-11订单生成界面

如上图所示，用户可以根据自己的要求填写发货人，收货人的姓名，地址。当用户填写的格式符合要求时，此时用户提交订单，系统会跳到付款界面。如图6-12所示：

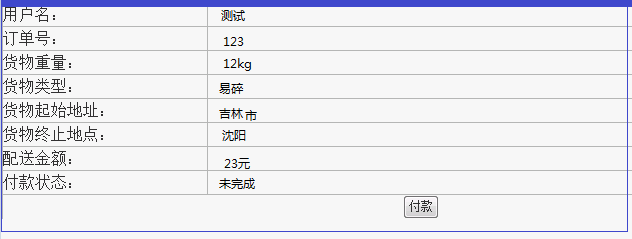


图6-12订单提交

如上图界面所示，当用户确认订单信息正确，点击付款按钮完成付款时，一个完整定单生成。

### 6.3.2后台管理界面：

当用户完成订单时，后台管理人员会根据用户的订单生成相应的配送单，工作人员根据配送单完成相应的流程。完成配送流程送到收货人的手中。流程图如图6-13所示：



图6-13 后台管理界面功能

### 6.3.3入库管理

当用户完成一条订单时，需要完成货物的入库，此时工作人员需要填写入库订单如下图6-14所示



图6-14 入库订单界面

点击源地址选择框旁的图标+ 可弹框进行客户信息的维护。当工作人员完成入库订单的操作时，可以点击订单管理按钮，查询订单入库情况如下图6-15所示：



图6-15 客户订单管理

### 6.3.4订单生成配送单

当订单完成入库操作时，此时的工作人员需要对订单拆分成配送单，如下图6-16所示：

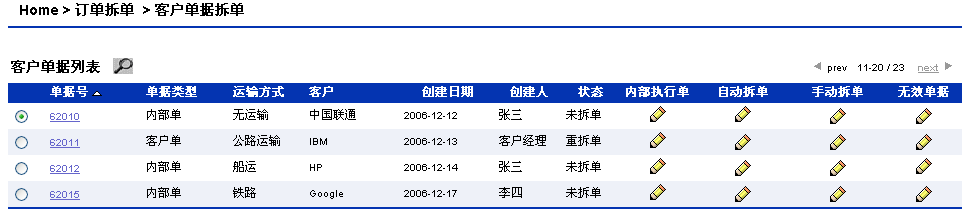


图6-16 订单生成配送单

可对未拆单或重拆单的系统订单进行手动拆单，如图6-17所示：



图6-17手动拆单

点击拆单按钮打开页面，如图6-18所示：



图6-18 拆单

如上图所示，系统会对会对用户产生的订单拆分成相应的配送单，工作人员也可以对订单进行手动拆分，来达到要求。当拆分完成时，系统会跳到运输单界面，此时需要工作人员对运输单进行填写。

### 6.3.5运输订单界面

当工作人员完成配送单时，需要对配送单对应相应的运输单如下图所示6-19所示：



图6-19 运输订单

### 6.3.6出库管理

当运输订单完成之后，工作人员会拿着运输单要求仓库管理员出库，此时仓库管理员完成出库订单操作如下图6-20所示：



图6-20出库订单界面

### 6.3.7 配送管理

1．车辆司机调配

当工作人员完成出库订单时，运输人员会根据运输方式对货物进行装车，调配司机，车辆等操作。如下图6-21所示：



图6-21 车辆司机调配

当货物到达一个配送站点时，进行中转站调度界面

2．中转站调度，如图6-22所示：



图6-22 中转站调度

3．货物追踪，如图6-23所示：



图6-23 货物追踪

最后货物到达收货人的手中。

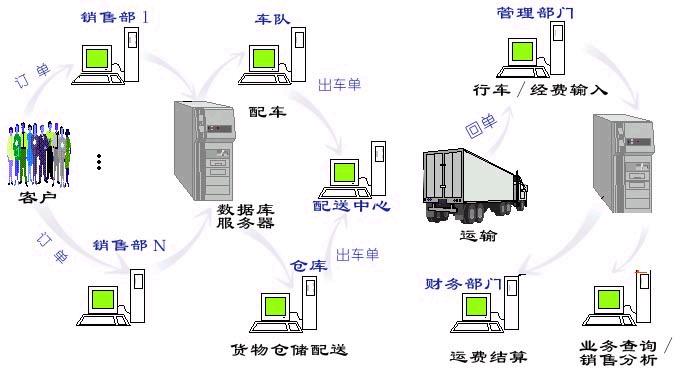
## 6.4详细技术

随着电子商务的发展和信息化的普及，人们越来越热衷于在家购物，从而导致企业对反应速度要求越来越高，对物流公司处理大量的数据提出挑战，对物流公司的速度需求也越来越高，所以物流公司的竞争力在于速度。

有一个好的货物管理系统有利于提高物流公司的速度；货物管理系统面向工作人员和客户，对货物运送，客户进行集中管理；同时客户可以根据客户号查询自己的货物在什么地方，以及实现货物安排的科学化，系统化，和自动化，最大限度地为客户，物流公司提供方便和提高管理效率.

### 6.4.1软件功能流程图

软件功能流程图，如图6-24所示：

图6-24 软件功能流程图

### 6.4.2软件功能模块设计

该项目包括7大子功能：

注册管理子功能：管理包括客户和职员的注册;

订单管理子功能：用户管理的订单生成、定单取消及修改；

配送管理子功能：管理货物的配送单跟踪、订单的拆分，车辆的调用，。

系统维护子功能：对公司信息，车辆信息及人员信息的维护。

结算管理子功能：对运输过程产生的各种费用及人工费进行结算。

入库管理子功能：处理客户的各种收货指令及进行相应的入库处理。

出库管理子功能：对货物的出库进行处理。

1．注册管理模块

1）功能说明：

注册管理模块包括用户和职员的注册等；

2）输入项：

管理人员和用户分别登陆不同的注册界面的地址，根据要求填写有效的信息

3）输出项**：**

分别将信息保存到数据库中，涉及的表有管理员表和用户表中，

4）主要算法：

（1）注册模块中，用到了向数据库中的写入新记录语句、和从输入时的条件判断语句，运用结构化模块化设计。

（2）调用了UserLogin.aspx界面，UserRigister.aspx界面来填写用户信息，

2．订单管理模块

1）功能说明

“订单管理”模块是对客户下达的各种指令进行录入、管理、查询、修改、打印等。主要包括订单类型、订单分配、订单查询、订单确认、订单打印等功能。

（1）订单类型

主要包含的订单类型有如下几种：

托运订单、入库订单、出库订单、调拨订单、退货入库订单、退货出库订单、搬场订单、速递订单、其它订单

（2）订单分配

对各分公司及营业点所接收的订单进行汇总，然后根据实际需要对订单进行分配管理。

（3）订单查询

可按日期、订单号、订单类型、业务部门、客户、受理点等条件对订单进行查询，并可对未确认的订单进行修改。

（4）订单确认

对已送达完成的订单做最终确认，主要是对订单数量、实发数量、实收数量进行确认，以便能准确地对该笔业务进行考核和费用结算。

（5）订单打印

对各种订单可根据客户需要设计不同打印格式对订单进行打印。

2）输入项：

管理人员登陆订单界面根据要求填写所要的信息。

3）输出项**：**

分别将信息保存到数据库中，涉及的表有订单表。

4）主要算法：

（1）.订单模块中，用到了向数据库中的写入新记录语句、和从输入时的条件判断语句，运用结构化模块化设计。

（2）.调用了Order.aspx界面，用户填写订单信息。调用OrderCheck.aspx界面，工作人员查询订单信息。

3．入库管理子功能：

1）功能说明：

“入库管理”模块是处理客户的各种收货指令及进行相应的入库处理，主要包括入库类型、入库方式、货物验收、收货单打印、库位分配、预入库清单打印、预入库确认、直接出库处理等功能。

（1）．入库类型

预定入库、销售入库、采购入库、调整入库、内拨入库、盘点入库、退货入库、调换入库、包装入库、报费入库

（2）．入库方式

根据进货方式的不同可分为：一次性入库、分批入库。

（3）．货物验收

根据收货通知单对入库的货物进行审核确认。验收内容主要有货主、货物名称、规格、货物等级、接收数量、破损数量、搁置数量、货物重量、货物体积、生产日期等。

（4）．收货单打印

该功能是打印出收货单据。单据内容有：收货日期、订单号、收货流水号、客户、客户通知编号、货物代码、货物名称、规格、单位、通知数量、接收数量、破损数量、搁置数量、生产日期、货物重量、货物体积等内容。

（5）．库位分配

对要入库的货物进行库位分配，分配原则有两种，按货物分配库位和按库位分配货物。

按货物分配库位：

根据货主、货物的尺寸、体积、重量、货物等级和保管等级等货物因素， 以及库位的 尺寸、体积、承重量及库位等级等库位因素，自动的从库位中找出 合适的库位。

按库位分配货物：

根据库位的尺寸、体积、承重量、库位等级以及货物尺寸、体积、重量、货物等级和保管等级等货物因素，自动的从货物清单中选出合适的货物。系统分配库位时。根据分配方式的不同，可分为人工分配和自动分配两种。

人工分配：人为的分配库位，电脑辅助人工查询空库位情况及各库位的使用情况。

自动分配：按库位使用情况，体积、重量及优化分配原则的优先级，自动排出货物的安排库位。（具体分配可能还要与其它实际情况有关）。

（6）．库位清单打印

根据预先安排的库位，打印出货物库位清单，以便保管员对号入库。其主要内容：客户、订单号、货物、规格、批号、库位、数量、实际入库数量等信息。

（7）．预入库确认

当所有的货物已经入库后，按库位清单的实际入库数量进行入库确认。同时，把预入库类型由预定入库改成销售入库。

（8）．直接入库处理

为了操作上的简便，根据实际情况有些货物可经验收后不作库位分配，而直接进行入库处理。

2）输入项：

管理人员登陆入库管理界面地址，根据要求填写有效的信息

3）输出项**：**

分别将信息保存到数据库中，涉及的表有入库表中。

4）主要算法：

（1）.入库模块中，用到了向数据库中的写入新记录语句、和从输入时的条件判断语句，运用结构化模块化设计。

（2）.调用了RukuCahxunFirst.aspx界面，在此界面上工作人员可以看到入库订单的信息，点击查询按钮跳到RuKuCahxun.aspx界面来查看订单的具体信息。

4．出库管理子功能

1）功能说明

“出库管理”模块是对货物的出库进行处理，主要有出库类型、货物调配、拣货清单打印、拣货处理、预出库调整、预出库确认、直接出库处理、送货单打印等功能。

（1）．出库类型

根据出库目的的不同可分为：预定出库、预拣出库、销售出库、领用出库、抽样出库、调整出库、内拨出库、盘点出库、退货出库、调换出库、包装出库、报废出库。

（2）．货物调配

根据发货通知单对货物进行调配处理，即从不同的仓库进行合理的拣货处理。根据处理方法可分为人工调配和自动调配两种。

人工调配：按照先进先出（批号或生产日期）及路线原则，人工安排货物的出库情况。

自动调配：严格的按照先进先出的原则（批号或生产日期），自动安排货物的出库情况。

（3）．拣货清单打印

该功能是打印出预出库的货物库位清单。根据清单用途，可分为外部拣货清单和内容拣货清单两种。

外部拣货清单：主要给承运人的拣货清单，目的是告诉承运人到哪几个仓库去提货。其内容有：承运人、客户、仓库、货物、规格、数量、单位等信息。

内部拣货清单：主要给内部操作人员的拣货清单，可以同时处理多家客户的拣货情况。其内容有：客户、仓库、货物、规格、数量、单位等信息。

（4）．拣货处理

该功能是把预出库的货物从原库位拣货到拣货区，以便提高出库的工作效率。拣货条件可按出库日期、仓库、客户、货物等内容进行查询。

（5）．预出库调整

在把货物从存储区转移到拣货区时才发现操作失误时，按先进先出法原则把该货物退回存储区。

（6）．预出库确认

当承运车辆来提货时，根据该车辆的货物配载情况，从拣货区出库，做真正出库处理。

（7）．直接出库处理

针对某些客户的要求，为了操作上的简便，可根据实际情况对有些货物不经调配而直接进行出库处理。

（8）．送货单打印

该功能是根据不同客户打印出货物配送单。其内容有：出库日期、单据编号、客户、联系人、联系电话、受货人、受货人地址、联系人、联系电话、承运人、承运车辆、出库仓别、货物、批号、数量、单位等。

2）输入项：

管理人员登陆出库界面的地址，管理人员根据要求填写有效的信息。

3）输出项**：**

分别将信息保存到数据库中，涉及的表有出库表中。

4）主要算法：

（1）.出库模块中，用到了向数据库中的写入新记录语句、和从输入时的条件判断语句，运用结构化模块化设计。

（2）.调用了ChuKu.aspx界面，工作人员查看订单的出库情况，调用ChuKuCheck.aspx界面来操作订单出库信息。

5．配送管理模块

1）功能说明

“配送管理”模块是根据客户的指令对出库的货物进行配送的安排，主要包括配送计划、承运方式选择、车辆分配、车辆调度、送货单打印、货物跟踪、车队考核等功能。

（1）．配送计划

根据客户的发货指令产生配送计划。

（2）．承运方式选择

根据客户的要求、送货量的大小、运输成本的高低等合理选择承运方式，大体分三类：陆运、水运、空运。

（3）．车辆分配

根据车辆情况、货物体积、货物重量、送货方向等状况对要配送的货物进行车辆的安排，可实行自动或人工分配。

（4）．车辆调度

主要是查询车辆的分配情况及各种汇总信息，并可对已分配好的车辆进行重新调整。

（5）．送货单打印

按车辆、客户打印出送货单。

（6）．货物跟踪

通过条形码、无线终端、GPS、Internet等通讯设备对配送的货物进行及时的跟踪查询。

（7）．车队考核

对所有运输车队的配合度、运输及时性、完整性、车况等进行考核。

6．系统维护模块

1）功能说明

（1）．用户管理

用户管理主要是对系统用户进行新增、修改及删除处理。按用户级别来分，可分为超级用户和普通用户两种。

超级用户：

超级用户对系统负责全面管理，其权限不受限制。

普通用户：

普通用户对系统负责部分管理，所以可分为各种不同的类别，其类别设置在[工作组管理]中完成。其权限受超级用户的控制。

（2）．工作组管理

工作组管理主要是对普通用户的用户组进行新增、修改及删除处理。按用户级别来分，可分为超级用户工作组和普通用户工作组两种。

超级用户工作组：

超级用户工作组对系统负责全面管理，其权限不受限制。

普通用户工作组：

普通用户工作组对系统负责部分管理，可分为各种不同的类别。普通用户工作组分为业务部工作组，综合管理科工作组，永新服务科工作组，财务科工作组，不同用户分入不同工作组。如果抽砂轻艺公司发生业务变动，只需修改工作组。

（3）．权限管理

权限管理主要是对每一普通用户工作组进行授权管理。根据普通用户工作组的职能不同，分别授于不同的权限。

（4）．密码修改

密码修改是用户可以修改自己的密码，以防止密码失密，每一用户都有权修改密码。

（5）．屏幕保护

屏幕保护是用户因事离开计算机而又不想退出系统时，为了防止别人操作，可进入[屏幕保护]功能。

（6）．公司车辆信息完善

完善公司以及车辆信息，实现对公司车辆的添加，删除，修改。

2）主要算法：

（1）.系统维护模块中，用到了向数据库中的写入新记录语句、和从输入时的条件判断语句，运用结构化模块化设计。修改完善系统信息。

（2）.调用了CompanyInfo.aspx界面，CompanyCheck.aspx界面，来维护完善公司信息，调用VehicleAdd.aspx界面，VehicleCheck.aspx界面来完善车辆信息。调用SuperAdministrator.aspx界面来添加管理用户组，超级管理员等

7．结算管理模块

1）功能说明

“结算管理”模块主要对客户的仓储费、运输费、过境过桥费等等费用进行结算处理。同时对承运单位作运费支出处理。主要包括费用种类、结算方式、收款处理、付款处理、应收款查询、应付款查询、等功能。

（1）．费用种类

费用包括出入库费（力资费、铲车费），仓储费、运输费、物流服务费和其它费用。计算要点有：

入库费的计算方法有两种：按重量(元/吨)、按体积(元/m^3)。

仓储费的计算方法有两种 ：按月、按天。

运输费的计算方法有四种 ：按重量、按吨公里、按时间、其它。

（2）．结算方式

结算方式可分为月底结算及随时结算两种。

月底结算：每月月底结算仓储费、力资费、运费等等费用。

随时结算：由客户要求，随时结算仓储费、力资费、运费等等费用。

（3）．收款处理

对已经结算客户的费用作收款记录。

（4）．付款处理

对已经结算承运人的运费作付款记录。

（5）．应收款查询

查询客户的应收款情况。

（6）．应付款查询

查询客户的应付款情况。

2）输入项：

管理人员登陆付款界面的地址，管理人员根据要求填写有效的信息完成结算管理

3）输出项：

分别将信息保存到数据库中，涉及的表有付款表中。

**4）主要算法：**

（1）.结算模块中，用到了向数据库中的写入新记录语句、和从输入时的条件判断语句，运用结构化模块化设计。

（2）.调用了PayMent.aspx界面，工作人员查看对已经结算客户的费用作收款记录，对已经结算承运人的运费作付款记录，查询客户的应收款情况等

# 7 实时监控

## 7.1实时监控概述

实时监控功能模块，由计算机系统、网络系统、MIS管理信息系统、GPS系统、GIS系统，android手机客户端组成，将GPS引入车辆管理，实现数据自动采集，自动分析，利用GPS定位功能与车辆管理MIS系统相结合，将位置信息导入车辆管理数据库，通过软件运算和分析将以往手工录入数据变为系统自动采集、分析，可以大大缓解管理人员的劳动强度，提高数据的准确性，分析的可靠性。同时将采集分析后的数据，拟合成地图的形式显示出来，更加明确清晰的给人以反馈。

主要的实时监控是由android智能终端的GPS芯片实现对位置信息的采集，通过访问数据网络、无线网络访问配置在远程服务器上的Web服务，实现对数据的实时上传和远程操作过程。以android手机为例，参与配送人员需登录使用客户端程序登录车辆信息，程序自动获取位置信息，通过网络服务上传至网络服务器，网络服务器对数据进行处理将其结果以网页的形式呈现给用户，并以服务端JSON数据返回给客户端，同时将结果上传至云端和分布式数据服务器组。同时配送人员还可对当前配送单和订单进行状态的更改 。如下图7-1和7-2所示为实时监控系统架构图。



图7-1 分布式实时监控系统架构图（1）



图7-2分布式实时监控系统架构图(2)

## 7.2物流配送管理手机移动端的介绍与应用

### 7.2.1 物流配送管理系统移动端的优点

数据采集功能使用android手机上GPS实现，其主要优点是：

1. 仪器操作简便

随着GPS技术的不断改进，GPS测量的自动化程度越来越高，而且移动端能实现随时随地，互联互通，资料更新一步到位。在观测中只需开启移动手机客户端即。位置信息的上传，跟踪观测和记录等均由仪器自动完成。不需跟踪监控时，只需关闭客户端即可。

通过数据通讯方式，将所采集的数据传送到数据处理中心实现全，自动化的数据采集与处理。另外，移动手机客户端体积也越来越小，相应的重量也越来越轻，极大地减轻了测量工作者的劳动强度。

2．应用广泛

Android智能手机普及率比较高，这样大大降低了对使用者的要求，从而也降低了经济成本。无需特殊环境，只要有数据服务即可实现相关操作。手机的优势，方便，便捷，贴生活；手机客户端的优势，展示产品，推广，及时保存性（显示在手机桌面上），再加上智能手机的普及，有利于软件的应用与发展。

### 7.2 .2物流配送管理智能终端架构介绍

android智能终端主要是使用java语言开发，实现GPS数据采集和远程访问数据库的操作。智能终端在物流配送和后期数据分析处理阶段都有着至关重要的作用。

智能终端由于需要访问数据库，整体架构依旧使用较为经典的MVC架构，对整体程序进行编写，android程序显示的功能主要以活动(Activity)为基础,在上面加以xml即可扩展标记语言控制页面布局，来实现图形界面的基本功能，通过页面上简单的逻辑代码为程序提供业务的实现基础。同时与业务相关的实体类对相关数据进行封转方便以后程序使用相关信息和业务逻辑类(命名规则:实体名+DAO)完成对数据库基本方法的调用为主要功能提供数据支持。此外为了便于程序功能的实现和拓展，还另外加了一个功能方法包，其中加入了相关持久化层基本操作数据库的函数，和一些程序常用的相关函数。使程序结构更加完整和健壮。下面是移动智能终端的详细架构分析：

1.客户端依据程序包共分为四部分，包括：

1）活动包(com.example.activities)

2）实体包(com.example.model)

3）业务逻辑包(com.example.dao)

4）公共方法包(com.example.utils)

2.详细介绍

1）活动包中包含三个活动，登录活动（LoginActivity），GPS活动（GPSActivity），订单详细信息活动（OrderInfoActiviry）。

其中登录活动完成对登录功能的实现及对是否连接数据网络进行验证，方便通过网络访问相关webservice。在此界面通过调用ActivitiesList.*exit*()方法实现了退出功能，如图7-3所示。



图7-3 智能移动终端系统架构图

GPS活动为此程序的中活动，主要功能均在此活动中实现。包括通过GPS获取位置信息，调用redis数据库webservice进行数据存储。通过绑定自定义服务，实现位置信息的后台存储。验证智能移动终端是否开启GPS功能，便于对位置的实时跟踪与监控。通过用户登录信息查询相关业务信息，利用多线程实习对webservice的异步调用，避免由于客户端等待服务时间过长，造成的程序异常和错误。通过相关数据源控件，对必要信息的列表展示。

订单详细信息活动就是对GPS活动选中的订单，具体信息的展示，和相关操作。

2）实体包就是依照数据表结构封装成的几个实体bean，以便于对数据的操作和信息的处理。

3）业务逻辑包中为对应实体bean的逻辑方法类，根据具体逻辑需要对公共方法包中的webservice方法进行调用，通过简单的分层架构，提高代码的清晰可读性和简易操作性。

4）公共方法包其中包括五种自定义方法类，两种远程数据访问方法类

（WebService\_Hbase，WebService\_Redis）实现配合webservice访问远程数据库。完成数据存储、获取与更改。一个活动集合（ActivitiesList）完成活动间跳转后

退出对活动资源的释放，避免内存出现不足或是泄露。GPS服务类（MyServiceForGPS）通过自定服务实现数据的实时存储。数据网络验证类（IsInternetAvaiable）判断数据网络是否可用，如图7-4所示。



图7-4手机客户端功能流程图

## 7.3实时信息查询

### 7.3.1实时信息采集

本系统采用android智能移动终端实现车载GPS的功能。本模块信息采集使用访问Redis上WebService来实现，方便快捷的实现了对信息的采集。

GPS格式

通过智能移动终端GPS芯片采集信息主要是当前位置的经纬度，两个坐标均为双精度浮点数据，存储在Redis（一个key-value存储系统）上为字符串。存储结构为有序集合(Set)，存储顺序即为位置信息随着时间递增。以发车日期为2013年10月31日，车辆牌号为苏DF0002为例。其存储格式为：苏DF0002\_GpsX\_2013年10月31日，车牌号\_GpsX\_年-月-日。X代表纬度，Y代表经度，经度与纬度分为两个有序集合。

客户端对数据的操作是通过Android提供的远程连接数据库Soap对象即简单对象访问协议（Simple Object Access Protocol，SOAP），是一种轻量的、简单的、基于XML的协议，它被设计成在WEB上交换结构化的和固化的信息。），来实现的。其主要过程就是现在了Soap和HTTP结合功能，主要实现的方法就是调用数据库已有的服务，调用服务时只需传入服务所需的参数即可。最后会获取到一个数据返回对象（SoapObject），进行数据解析，解析数据有两种数据类型，二维字符串数组和字符串。数组用来查询信息返回。字符串应用于数据更新信息返回。本系统主要采用hadoop环境搭载的Hbase及redisd的两种web服务，Hbase上的服务主要是进行主要配送流程的信息存储，主要使用QueryConditionNotKey（非主键查询）、QueryByConditionKey（主键查询）、UpdateData（更新数据）。Redis上的服务主要使用Zcard1(获取有序集合总数)、Zadd1（向有序集合中插入数据）。服务的返回数据集都是采用序列化Json数据，在本地将其转化为二维数组。对于更新操作返回的是字符串。Web服务组成数据操作层加上相应的业务逻辑组成完整的物流配送系统手机客户端。

通过web服务访问数据库结构逻辑图，简单对象访问协议，SOAP使用基于XML的数据结构和超文本传输协议(HTTP)的组合定义了一个标准的方法来使用Internet上各种不同操作环境中的分布式对象。其有特殊的语法规则。

1．SOAP消息必须由XML来编码

2．SOAP消息必须使用SOAP Envelope命名空间

3．SOAP消息不能包含DTD引用

4．SOAP消息不能包含DTD引用

5．SOAP消息不能包含XML处理指令

使用了android中为了提供的SOAP的工具类，通过相应的方法对请求与的数据对象绑定、和相应请求对象的序列化。定义了一种使用底层传输协议来完成在节点间交换SOAP封装的约定。通过访问相应的WSDL文件实现Web服务描述如何访问具体的接口，其中对数据库进行请求和数据库返回给客户端的相应都是一条SOAP消息，SOAP 消息就是一个普通的 XML 文档，包含下列元素：必需的 Envelope 元素，可把此 XML 文档标识为一条 SOAP 消息可选的 Header 元素，包含头部信息必需的 Body 元素，包含所有的调用和响应信息

可选的 Fault 元素，提供有关在处理此消息所发生错误的信息。客户端对返回的SOAP进行解析和处理，将其转化为可用信息提供给程序处理。解析主要操作为使用循环嵌套的方法，将消息解析为字符数组供调用者使用。整个架构流程如下图7-5所示：



图7-5 Web 服务访问数据库结构逻辑图

### 7.3.2实时地图技术

绘制地图技术采用的是ArcGIS API for Silverlight基本原理。地图绘制功能模块允许您创建丰富的强大制图功能、地理编码功能及地理处理功能的 Internet 应用程序和桌面应用程序,通过浏览器插件进行托管。利用的主要功能：

1．空间数据展示：加载地图服务或影像服务，展示服务数据。

2．图形绘制：在地图上交互式地绘制查询范围或地理标记等。

以及利用的主要特色:

3．充分利用微软.NET技术的优势，开发简单快捷。

4．地图支持弹簧动画（地图缩放、平移等操作具有连续性）。

5．支持将任何界面元素（控件）作为地图要素，随地图缩放。

所用到的缓存图层，通过预先将地图数据切片缓存在服务器硬盘，客户端访问时服务会根据其当前访问区域计算出显示的图片返回给客户端。客户端就可以看到地图了，其实是几张图片组合起来的。

为了画出美观的配送地图，及其配送路线，标记，实时监控地图回事使用了强大的地图功能，将世界地图与自定义的标记路线图层整合到同一界面内，即运用了两个图层。

1．缓存的地图服务图层，在这个图层中采用传统的地图服务，这个服务提供了所需要的世界地图，里面自带了放大和缩小，地理处理功能，只要装上插件，即可浏览出世界各处的地理情况

2．为了画出路线和起点终点的标记，定义了两种自定义的图层。一个的外观为标记图形的形状，自定义它的图片源，即可将标记定义为想要的标记，将其用经纬度的点定位在地图上；另一个的外观为线的形状，将其拐角的经纬度的点加入到一个集合，自定义它的颜色，粗细。并将这两个自定义的图层组成集合，放入图层集中中，显示出来。

还有将自定义的Graphic图层放入其中，就可以进行图形绘制。地图定义说明了构成地图的数据，包括所要显示的数据、存储位置、数据组织为图层的方式、数据的显示方式（如所用颜色、标注图元、地图缩放设置）。附带了多种涵盖全球地理和人口统计信息的示例数据。地图由地图对象的图层构成。这些地图对象可通过图元对象在地图中访问。此处有三种基本图元类型：

区域：特指覆盖给定区域的闭合对象。其中包括多边形、椭圆和矩形。区域包括国界、邮政编码边界和销售区域等。

点对象：表示数据的单一位置。其示例包括客户位置、饭店和停车计时器等。

线对象：覆盖给定距离的开放对象。包括直线、折线和弧线等。其示例有街道、河流和电力线路等。各种不同类型的对象置于单独的图层之中（最常见），或者也可以在同一图层中组合不同的对象。借助于本地图绘制功能模块，可以自定义和显示这些对象，令地图满足具体的需求。

样式：地图中的所有图元均显示特定的可见特征，例如颜色或符号类型。相应的特征也称为样式。绘制地图效果如图7-6所示：

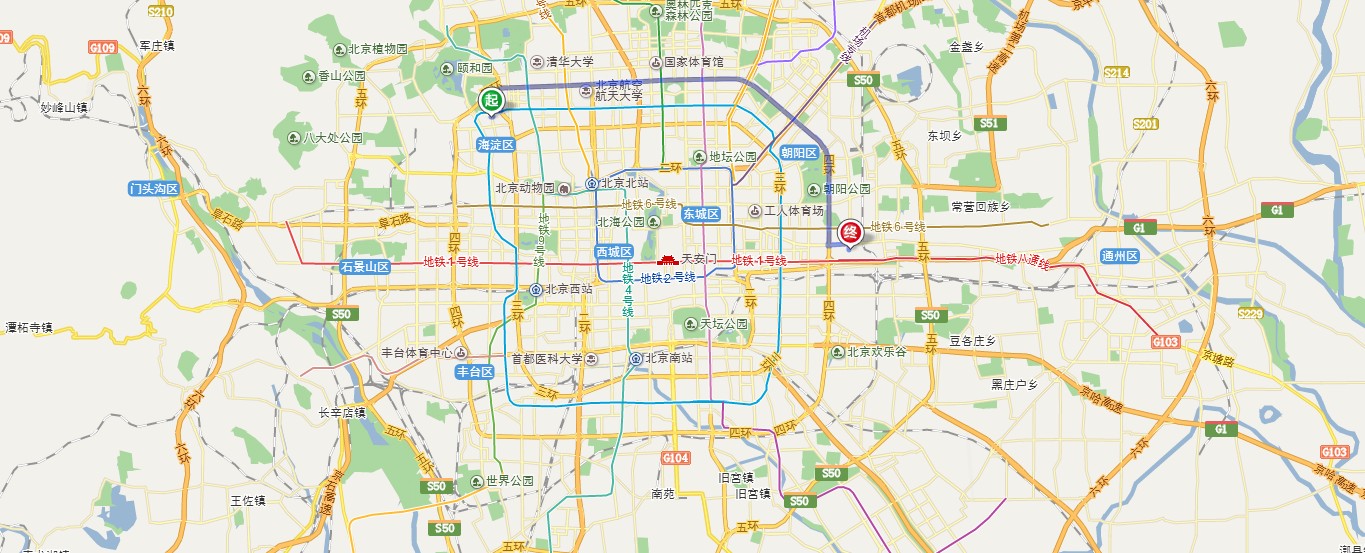


图7-6 地图效果图

# 8 海量数据分析处理

## 8.1 配送分析算法

### 8.1.1 算法理论

1．等级化

对于一个货物，设 它的重量是w,通过w的值对货物分成五级，设 级数为i，货物的重量就按等级做如下处理：



其中，wi 表示第i级的标准重量，W为处理后的重量。

对于一辆车，设 它的载重是c，通过c的值对车辆分成五级，设 级数为j，载重按等级做如下处理：



其中，cj 表示第j 级的标准载重，C为处理后的载重。

2．车辆匹配

利用贪心算法，使车辆资源充分得到利用，具体过程如下：

1）找到配送站里容量等级最小的车；

2）找到库存中重量等级最小的部分货物，使得这些货物尽可能的恰好能装下这些货物，如果无法满足，就找容量等级稍大的车，重复过程（2），直到满足；

3）对没有分配到车辆的货物，重复过程（2）直到所有货物都分配了有车。

以上所有关于货物重量、车辆载重的值都是使用等级划分以后的重量与载重。

### 8.1.2 MapReduce算法设计

由于这是一个一对多的数据匹配问题，需要对数据进行全局分析，所以这样就给MapReduce的实现带来了一些问题。为了解决这个问题，对一个job中的车辆进行了固定，让这个job针对某个车辆进行匹配。

在匹配之前，要对通过mapReduce算法，对车辆的载重进行排序，再依次对载重最小的未匹配车辆进行货物匹配。当货物或者车辆之一都被匹配完就停止匹配。

在一些排序过程中，如果不做处理“2”会排在“12”的后面，这会影响排序的正确性，为了解决这个问题，可以把“2”用“02”来表示。

1．车辆排序

知道，在mapReduce中有自己的排序过程，所以只要想办法利用这一点就可以了，那么，要怎样来利用呢？

在Map阶段把输出的Key设置为车辆的载重，在Map阶段和Reduce阶段之间系统会自动对Key进行排序，在Reduce阶段只需直接把输入的Key和Value作为Value输出即可。Map阶段的value值设置为车辆的ID即可。

排序过程如图8-1所示：



图8-1车辆排序

2．车辆匹配

要挑选出重量最小的货物，所以要想办法对货物重量进行排序，并且选出其中最小的部分货物时车辆刚好能尽可能多的装货物。

在Map阶段把输出的货物重量设置为Key，使其自动排序，其它部分数据设为Value,在Reduce阶段对当前车辆的容量与货物重量做匹配，输出为匹配的货物ID，如图8-2所示。



图8-2车辆匹配

## 8.2 路径分析算法

### 8.2.1 算法理论

路径分析算法描述：

1．设 G = ( P , E , Q) 为一具有7个节点的赋值有向图(图1) 。其网络图可用带权的邻接矩阵 Q来表示,Q[ i , j ] 表示弧( Pi , Pj) 的权值,如果 Pi 到 Pj 不连通,则 Q[ i , j ] = ∞。

又设辅助向量 D ,每个分量 D[ i ]表示从起始点到每个终点 Pi 的最短路径长度。假定起始点在有向图中的序号为 i1 ,并设定该向量的初始值为: D[ i ] = Q[ i1 , i ]。

令 S 为已经找到的从起点出发的最短路径的终点的集合。

2． 选择 Pj ,使得 D[ j ] = Min{ D[ i ] | Pi ∈P - S } Pi ∈P。

Pj 就是当前求得的一条从 Pi1 出发的最短路径的终点,令 S = S ∪{ Pj} 。

3．修改从 Pi1 出发到集合 P - S 中任意一顶点 Pk 的最短路径长度。

如果 D[ j ] + Q[ j , k ] < D[ k ] ,则修改 D[ k ] 为: D[ k ] = D[ j ] + Q[ j , k ]。

4．重复第2、 3步操作共6次,由此求得从 Pi1出发的到图上各个顶点的最短路径是依路径长度递增的序列。表 8-1 是图8-3根据上述路径算法计算的结果。表8- 1终点 从 P1 到其他各个节点的路径。

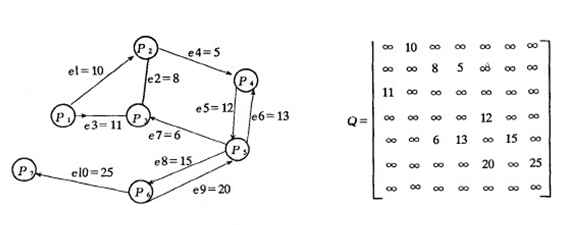
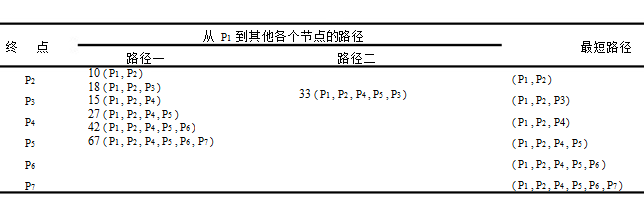


图8-3带权的有向图和邻接矩阵

表8-1



### 8.2.2 MapReduce算法设计：

用邻接表的方式来存储图，在文件中，每一行数据代表图的一个节点，行的格式可以采用一下格式：

|  |  |
| --- | --- |
| ID | Edges |

其中ID代表节点的标识，Edges代表从某一节点出发的所有的边（对于有向图）Edges又可以用从该节点出发的边的另一端的节点表示。例如图8-4就可以用邻接表表示。

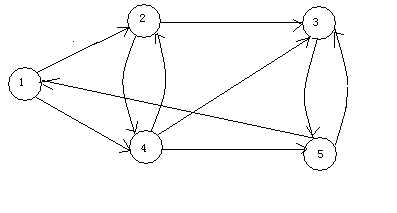


图8-4包含权值信息的有向图

在求解图的问题时，图中的节点和边往往含有更多的信息，在用标色法求解单元最短路径时，节点的信息还包括节点的颜色、节点到源点的距离等，边的信息包括边的权值等。对以上邻接表作稍微改进便可满足要求。更新后的图文件的每一行的格式如下：

|  |  |  |  |
| --- | --- | --- | --- |
| ID | distance | color | Edges（weight） |

图8-4所示有向图加上权值信息后如图8-5所示，用邻接表表示如下：

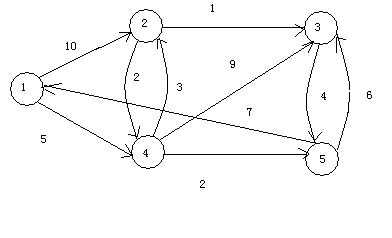
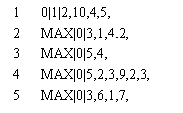


图8.2.4 包含权值信息的有向图

其中MAX表示无穷大，颜色域，0：白色；1：灰色；2：黑色

#### 8.2.2运行示例：

一个job得输出作为另一个job的输入，多个job组成job chain 来完成最短路径的求算，当某一个job的输出中所有的节点中没有灰色节点时时停止迭代。算法停止，如图8-6所示。

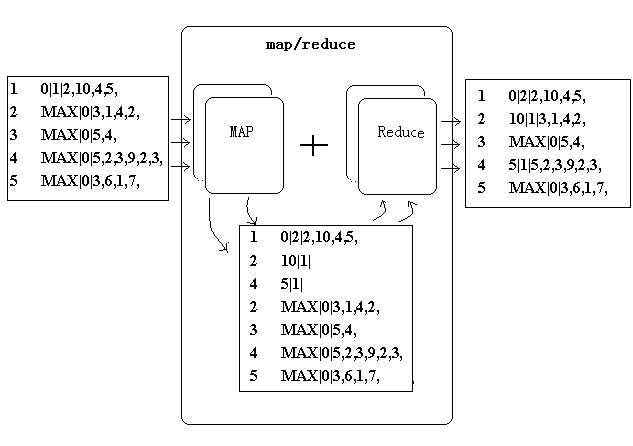
****

图8-6 求最短路径的一个job过程

# 附录

1．公司信息CompanyInfo(HBase)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段含义 | 主键 | 备注 |
| 1 | CompanyInfoID | 公司号 | 是 | 主键 |
| 2 | Name | 公司名称 |  |  |
| 3 | ServiceType | 业务说明 |  |  |
| 5 | PhoneNumber1 | 联系电话1 |  |  |
| 6 | PhoneNumber2 | 联系电话2 |  |  |
| 7 | PhoneNumber3 | 联系电话3 |  |  |
| 8 | Contacts | 联系人 |  |  |
| 9 | CreateTime | 创建时间 |  |  |
| 10 | EndTime | 结束时间 |  |  |
| 11 | Telautogram | 传真 |  |  |
| 12 | Psrovince | 所在地区（省） |  |  |
| 13 | City | 所在地区（市） |  |  |
| 14 | District | 所在地区（区） |  |  |
| 15 | Address | 详细地址 |  |  |
| 16 | CompanyLevel | 公司级别 |  | 总公司  公司  子公司  配送站 |
| 17 | TheCompanyID | 下属公司 |  | 公司级别为配送站是下属公司为无， |
| 18 | AdjacentCompany | 相邻公司 |  | 中间以逗号隔开 |
| 19 | CompanyLongitude | 经度 |  |  |
| 20 | CompanyLatitude | 纬度 |  |  |

2．车辆信息VehicleInfo (Hbase)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段含义 | 主键 | 备注 |
| 1 | VehicleInfoID | 车牌号 | 是 | 主键 |
| 2 | GpsNumber | Gps号 |  |  |
| 3 | AdministratorID | 管理员号 |  |  |
| 4 | RegisterTime | 登记时间 |  |  |
| 6 | CarState | 车辆状态 |  | 无配送  正在配送  维修 |
| 7 | CapacityLevel | 容量等级 |  | 1级  2级  3级  4级  5级 |
| 8 | SubordinateUits | 所属单位 |  |  |

3．配送单DistributionList(Hbase)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段含义 | 主键 | 备注 |
| 1 | DistributionListID | 配送单号 | 是 | 主键 |
| 2 | CourierTime | 配送时间 |  |  |
| 4 | CompletionStatus | 完成状态 |  | 未完成  完成 |
| 5 | CompletionTime | 完成时间 |  |  |
| 7 | StartStationID | 配送起始站ID |  | 公司/配送站id |
| 8 | EndStationID | 配送终点站ID |  | 公司/配送站id |
| 9 | PlateNumber | 车牌号 |  |  |
| 10 | OrderID | 订单ID |  | 所属订单用逗号隔开 |

4．用户User(Hbaes)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段含义 | 主键 | 备注 |
| 1 | UserID | 用户号 | 是 | 主键 |
| 2 | Name | 用户名 |  |  |
| 3 | Password | 密码 |  |  |
| 4 | RoleID | 角色号 |  |  |
| 5 | TheCompany | 所属公司 |  | 角色为用户时  所属公司为无 |
| 6 | Email | 邮箱 |  |  |
| 7 | Province | 所在地区（省） |  | 公司/配送站id |
| 8 | City | 所在地区（市） |  | 公司/配送站id |
| 9 | District | 所在地区（区） |  | 公司/配送站id |
| 10 | Address | 默认地址 |  |  |
| 11 | Postalcode | 邮政编码 |  |  |
| 12 | CellPhoneNumber | 手机号码 |  |  |
| 13 | TelePhoneNumber | 电话号码 |  |  |
| 14 | Gender | 性别 |  |  |
| 15 | CreateTime | 创建时间 |  |  |

5．订单信息Order

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段含义 | 主键 | 备注 |
| 1 | OrderID | 订单ID | 是 | 日期加随机数 |
| 2 | Name | 收货人姓名 |  |  |
| 3 | GenerationTime | 订单生成时间 |  |  |
| 5 | CompletionStatus | 订单完成状态 |  | 未完成  完成 |
| 6 | DeliveryStageIdentfy | 配送阶段唯一标识 |  |  |
| 7 | UserID | 用户ID |  |  |
| 8 | DestinationProvince | 目的地点(省) |  | 公司/配送站id |
| 9 | DestinationCity | 目的地点(市) |  | 公司/配送站id |
| 10 | DestinationDistrict | 目的地点(区) |  | 公司/配送站id |
| 11 | DestinationAddress | 目的地点地址 |  |  |
| 12 | StartProvince | 起始地点（省） |  | 公司/配送站id |
| 13 | StartCity | 起始地点（市） |  | 公司/配送站id |
| 14 | StartDistrict | 起始地点（区） |  | 公司/配送站id |
| 15 | StartAddress | 起始地点地址 |  |  |
| 16 | CellPhoneNumber | 手机号码 |  |  |
| 17 | TelePhoneNumber | 电话号码 |  |  |
| 18 | GoodsTypes | 货物说明 |  | 易碎  防潮  贵重物品  其他 |
| 19 | Notes | 备注 |  |  |
| 20 | Height | 重量 |  |  |
| 21 | CapacityLevel | 容量等级 |  | 1级  2级  3级  4级  5级 |

6．入库出库信息StorageInfo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段含义 | 主键 | 备注 |
| 1 | StorageInfoID | ID | 是 | 主键 |
| 2 | StorageTime | 入库时间 |  |  |
| 3 | OutboundTime | 出库时间 |  |  |
| 4 | SubordinateUits | 所属单位 |  | 公司/配送站id |
| 5 | OrderID | 订单号 |  |  |
| 6 | State | 状态 |  | 入库  出库 |

7．角色Role(HBase)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段含义 | 主键 | 备注 |
| 1 | RoleID | 角色号 | 是 | 主键  *用户*  *车牌号*  *车辆管理员*  *配送管理员*  *总公司管理员*  *公司管理员*  *子公司管理员*  *配送站管理员*  *仓库管理员* |
| 2 | RoleName | 角色名称 |  | 用户  车牌号  配送管理员  总公司管理员  公司管理员  子公司管理员  配送站管理员  仓库管理员 |
| 3 | CreateTime | 创建时间 |  |  |
| 4 | EndTime | 结束时间 |  |  |

8．权限分配PermissionAssignment(HBase)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段含义 | 主键 | 备注 |
| 1 | PermissionAssignmentID | 权限分配号 | 是 | 主键 |
| 2 | RoleID | 角色号 |  | 非空 |
| 3 | FunctionID | 功能号 |  | 非空 |
| 4 | CreateTime | 创建时间 |  |  |
| 5 | EndTime | 结束时间 |  |  |

9．功能权限FunctionalAuthority(HBase)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段含义 | 主键 | 备注 |
| 1 | FunctionalAuthorityID | 功能号 | 是 | 主键 |
| 2 | FunctionName | 功能名称 |  |  |
| 3 | CreateTime | 创建时间 |  |  |
| 4 | EndTime | 结束时间 |  |  |
| 5 | Description | 功能描述 |  |  |

10．配送阶段DeliveryStage

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段含义 | 主键 | 备注 |
| 1 | OrderID | 订单ID | 是 |  |
| 2 | StartStationID | 起始站ID |  | 公司/配送站id |
| 3 | EndStationID | 终点站ID |  | 公司/配送站id |
| 4 | State | 完成状态 |  | 完成  未完成 |
| 5 | DeliveryStageID | 配送阶段ID |  |  |
| 6 | DeliveryStageIdentfy | 配送阶段标识 |  |  |

11．付款单Payment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段名称 | 主键 | 备注 |
| 1 | PaymentID | 付款单ID | 是 | 主键 |
| 2 | UserID | 用户ID |  |  |
| 3 | OrderID | 订单ID |  |  |
| 4 | PaymentState | 付款状态 |  | 未付款  已付款 |

12．父订单表 FatherOrder

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段含义 | 主键 | 备注 |
| 1 | Id | 编号 | 是 | 全局唯一标识符 |
| 2 | OrderId | 订单号 |  |  |
| 3 | TotalValue | 金额 |  |  |
| 4 | GoodsId | 商品编号 |  |  |
| 5 | Customer\_id | 客户编号 |  |  |
| 6 | ContactName | 联系人姓名 |  |  |
| 7 | Phone | 联系人电话 |  |  |
| 8 | Mobile | 联系人手机 |  |  |
| 9 | Provinces | 省份 |  |  |
| 10 | City | 城市 |  |  |
| 11 | Area | 区 |  |  |
| 12 | Streets | 联系人地址(街道) |  |  |
| 13 | ZipCode | 邮编 |  |  |
| 14 | Status | 订单状态 |  | 0未处理1备货中2出货3投递4发货中5已签收6回访 |
| 15 | IsChecked | 订单审核 |  | 0未审核1已审核 |
| 16 | UpdateTime | 修改订单时间 |  |  |
| 17 | IsImportant | 订单是否重要 |  | 0不重要1重要 |
| 18 | CreateTime | 订单创建日期 |  |  |
| 19 | CreateOperator | 订单录入人 |  |  |
| 20 | IsCancel | 订单作废 |  | 0未作废1已作废 |
| 21 | CancelReason | 订单作废原因 |  | 0客户作废1客服作废 |

13．子订单表Order

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名称 | 字段含义 | 主键 | 备注 |
| 1 | Id | 编号 | 是 | 全局唯一标识符 |
| 2 | OrderId | 子订单编号 |  |  |
| 3 | FatherId | 父订单编号 |  |  |
| 4 | GoodsId | 商品编号 |  |  |
| 6 | ContactName | 联系人姓名 |  |  |
| 7 | Phone | 联系人电话 |  |  |
| 8 | Mobile | 联系人手机 |  |  |
| 9 | Provinces | 省份 |  |  |
| 10 | City | 城市 |  |  |
| 11 | Area | 区 |  |  |
| 12 | Streets | 联系人地址(街道) |  |  |
| 13 | ZipCode | 邮编 |  |  |
| 14 | Price | 单价 |  |  |
| 15 | Amount | 数量 |  |  |
| 16 | TotalMoney | 总金额 |  |  |
| 17 | Checker | 订单审核人 |  |  |
| 18 | PreparePerson | 订单备货人 |  |  |
| 19 | StockOutTime | 订单出库时间 |  |  |
| 20 | StockOutPerson | 出库人 |  |  |
| 21 | SendTime | 发货时间 |  |  |
| 22 | Sender | 发货人 |  |  |
| 23 | ServedTime | 订单送达日 |  |  |
| 24 | express\_code | 快递单号 |  |  |
| 25 | ConfirmPerson | 签收确认人 |  |  |
| 26 | CancelTime | 订单作废时间 |  |  |
| 27 | CancelOperator | 作废操作员 |  |  |
| 28 | CancelReason | 作废原因 |  |  |
| 29 | BackTime | 退回时间 |  |  |
| 30 | BackChecker | 退回确认人 |  |  |
| 31 | BackReason | 退回原因 |  |  |