Contents

[Pivotal Web Services 2](#_Toc441757369)

[Docker 4](#_Toc441757370)

[Monolithic vs micro-service 4](#_Toc441757371)

[monolithic application 4](#_Toc441757372)

[micro-services 4](#_Toc441757373)

[Eureka = Netflix Service Discovery Server + Client 4](#_Toc441757374)

[Communicate between front-end and back-end? 5](#_Toc441757375)

[分布式应用系统的架构演变 6](#_Toc441757376)

[Authentication and Authorization 8](#_Toc441757377)

[AD LDS Server 9](#_Toc441757378)

[邮件发送 11](#_Toc441757379)

[数据库 11](#_Toc441757380)

[MongoDB 11](#_Toc441757381)

[Redis (remote dictionary service) 13](#_Toc441757382)

[PostgreSQL 15](#_Toc441757383)

[Domain-specific language 15](#_Toc441757384)

[Drools 15](#_Toc441757385)

[分布式计算 16](#_Toc441757386)

[Hadoop 16](#_Toc441757387)

[测试 19](#_Toc441757388)

[压力测试 19](#_Toc441757389)

[Office Note 19](#_Toc441757390)

# Pivotal Web Services

1. your PWS Account -------Apps Manager
2. setup PWS Account
3. create ORG and space
4. CF Client -------deploy and manage applications
5. Install CF Client
6. Connect Client to PWS

// private

>cf login –a <https://api.run.pivotal.io> ------Cloud Foundry public API endpoint UserID：[qizhong.lin@philips.com](mailto:qizhong.lin@philips.com)

Password: same.to.google.email

Note: if connection fails, set HTTP\_PROXY=161.92.51.225:8080 in System Environment

1. Deploy an application to PWS

App> cf push

Manifest.yml

.war

1. Use application

Open route of application, it will return you web page(html+css+javascript), json or xml

Bind app to backing services (from Marketplace)?

（初步理解为：调用pivotal平台上已有的服务，如常见的有relational databases, MongoDB, Redis, RabbitMQ (message queue)）

技术点: Sprint Cloud



Scale app (supplied by platform)

1. Logs, Events, performance

How to monitor App Performance? (database call response time & throughout, performance data API access)

Bind app to Service (New Relic)

# Docker

Need to install MongoDB, RabbitMQ, or MySQL? Use Docker to simplify dev and test

Almost every interesting application uses at least one infrastructure service such as a database or a message broker. For example, if you tried to build and/or run the Spring Boot-based user registration service you would have discovered that it needs both MongoDB and RabbitMQ.

One option, of course, is to install both of those services on your machine. Unfortunately, installing a service is not always easy. Also, different projects might need different incompatible versions. Moreover, I’m not a fan of cluttering my machine with random services. Fortunately, there is a great way to solve this problem: Docker. You install Docker and use it to run the services that you need as containers.

Docker only directly runs on Linux so if you are using Mac OSX or Windows the first step is to install Boot2Docker (Mac OSX, Window). Boot2Docker installs the Docker command line locally but runs the Docker daemon in a Virtual Box VM

# Monolithic vs micro-service

monolithic application

(e.g. application = one huge WAR file)

micro-services

application as a collection of independently deployable services.

I have often introduced the idea as “SOA light” since you are building a service-oriented architecture. The trouble with the term SOA, however, is that it is associated with a lot of baggage: SOAP, ESBs, heavyweight ceremony, etc. Instead, I’ve talked about “Decomposing the monolith” or “Decomposing the WAR”. I’ve also used the term modular, polyglot architecture but that’s a bit of a mouthful.

Users expect a rich, interactive and dynamic user experience on a wide variety of clients including mobile devices. Applications must be highly scalable, highly available and run on cloud environments.

## Eureka = Netflix Service Discovery Server + Client

Eureka服务器没有存储功能，所有注册表里的每个服务（通过自己的application.yml的serviceUrl）每30秒发送一次心跳，表明自己活着。

Eureka客户端会缓存服务器注册表

// @EnableEurekaServer, @EnableDiscoveryClient运行Eureka服务器

Eureka服务器自带UI and HTTP API端口: localhost:8761 or localhost:8761/eureka/

默认情况下，每个Eureka服务器也是Eureka客户端，需要service URL访问peer

application.yml

server:

port: 8761

eureka:

client:

serviceUrl:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

instance:

hostname: localhost

//as standalone eureka server, 关闭客户端行为（因为服务器也是客户端）

client:

registerWithEureka: false

fetechRegistry: false

// @EnableEurekaClient （如下两个功能）

自动注册该服务于Eureka服务器中

application.yml

// eureka的基本信息会记录于服务器注册表中，从而别的服务可以访问到

eureka:

client:

serviceUrl:

defaultZone: <http://localhost:8761/eureka/> 定位Eureka服务器

instance:

hostname: localhost

nonSecurePort: 80

metadataMap:

instanceId: ${spring.application.name}:${spring.application.instance\_id:${random.value}}

通过Eureka服务器查询别的服务

## Communicate between front-end and back-end?

Emulator <-> web service directly

Device <-> web service, should in the same domain, fire-wall should in-bound port

1. Device browser access to web service

ip:port\ccs\patients\patient1

if can’t access, please do as follows:

control panel -> system and security -> windows firewall -> advance settings -> Inbound Rules

New Rules -> port -> TCP,UDP -> specific local ports: 9002, 9111

# 分布式应用系统的架构演变

Corba时代， 在1998年EJB1.0发布以后，就逐渐淡出历史舞台了

基于RMI/IIOP协议的EJB时代， 这个时代开始于1998年，到现在基本上已经划上了句号

不管是Corba，还是EJB，都有一些共同点：   
1) 通过专有的网络协议通讯   
2) 不能跨平台调用   
3) 通过分布式对象调用来实现分布式架构，换句话来说就是，分布式架构是绑定在面向对象的机制上的

基于SOAP协议的Web Services时代，个时代始于2001

web services有一些明显不同于Corba和EJB分布式对象架构的特征：   
1) 通过标准SOAP协议通讯，一般走HTTP通道   
2) 能够跨平台调用   
3) 通讯格式是xml文本，而不是二进制数据格式   
4) 通过RPC机制来实现分布式调用，而不是通过面向对象机制实现分布式调用

REST是专门为分布式调用设计的架构，在REST里面，分布式是通过对资源的操作来实现的，不是像EJB那样通过对象的方法调用来实现的。资源是一种抽象的概念，资源被映射到相应的一套URL规则上面了

分布式架构 协议 调用方式

-------------------------------------------------------

Corba架构 专有二进制协议 对象的CRUD操作

EJB架构 专有二进制协议 对象的CRUD操作

Web Services SOAP协议 RPC方式

REST HTTP协议 对资源的CRUD操作

--------------------------------------------------------

REST基础概念：

* 在REST中的一切都被认为是一种资源。
* 每个资源由URI标识。
* 使用统一的接口。处理资源使用POST，GET，PUT，DELETE操作类似创建，读取，更新和删除（CRUD）操作。
* 无状态。每个请求是一个独立的请求。从客户端到服务器的每个请求都必须包含所有必要的信息，以便于理解。
* 通信都是通过展现。例如XML，JSON

[SOA](http://www.cioage.com/soa/)的架构风格，它是基于企业业务架构的功能性分解，并且引入了两个高层次的抽象：企业业务服务和业务流程。企业业务服务代表的是现有IT能力（和企业的业务功能相一致）。业务流程编排业务服务，并定义业务的整体功能。

面向资源架构（ROA）的架构准则。ROA构建在资源这一概念之上；每个资源都是一个能够直接访问的分布式组件,可通过一个标准的、通用的接口来处理。所以，面向资源的架构（ROA）其最根本的还是一种基于资源的分解。

在最简单的情况下，服务可以被定义为一个自包含、独立开发、可部署、可管理和可维护的软件实现，它从整体上为企业提供特定的与业务相关的功能，并且在设计上是“可集成的”。

服务接口（尤其对某个给定的服务而言）定义服务功能，并且可由多种方式实现。存在两种基本的定义服务接口的方法——RPC风格和消息 （messaging）风格，RPC风格实现使用服务调用语义并且通过服务接口中的一组参数来定义。而消息风格的服务接口被有效地固定（本质上只需要进行 “执行”操作）使用XML文档作为输入和输出（这和GoF设计模式非常相似）。在这种情况下，服务语义是由输入和输出消息的语义来确定。

资源可以被定义为一个可直接访问的、独立开发的、可部署的、可管理的和可维护的软件构件，它支持特定的数据。一个资源就类似于一个对象，不过它是带有预定义（CRUD）接口语义的对象。实际上，一个资源就类似于一个对象，不过它是带有预定义（CRUD）接口语义的对象。

createResource——创建一个新的资源（以及相应的唯一标示）– PUT

getResourceRepresentation——获取资源信息– GET

deleteResource ——删除资源（可选地包括相关联的资源）– DELETE（只是引用的资源），POST（当需要删除相关联的资源时使用）

modifyResource——更改资源— POST

getMetaInforatmion——取得资源元数据信息—HEAD

资源通过两部分定义：资源URL和资源所提供的所有操作上定义的输入/输出参数。这和服务不同，服务的方法之间是完全独立，并且能够以独立端点（endpoints）的方式部署，而资源上的方法遵循OO语义，这意味着所有的方法（除createResource以外）都必须依附于底层的某个资源（同一个URL）。

如果把WS-\*比作是互联网世界的RPC，那么REST就是互联网世界的数据库管理系统（DBMS）……传统的基于[SOA](http://www.cioage.com/soa/" \t "_blank)的 集成表现了不同软件构件之间通过各种过程或方法进行交互。REST有效地将每个软件构件看作一组数据库表，而这些构件之间使用SELECT, INSERT, UPDATE和DELETE来通信。（或如你所想的使用GET, PUT, POST, DELETE）。那业务逻辑放在哪里呢？在存储过程中？不太对，其实在触发器中。

REST Web服务方法是指单纯使用REST技术作为通信手段来构建[SOA](http://www.cioage.com/soa/" \t "_blank)的一种方法

[SOA](http://www.cioage.com/soa/)P和REST最主要的区别在于REST是直接实现于HTTP协议之上，而[SOA](http://www.cioage.com/soa/" \t "_blank)P引人了一个抽象层（[SOA](http://www.cioage.com/soa/" \t "_blank)P消息传递）

REST是面向资源的，而每个资源之所以被称作为资源是因为他们能 够被识别和利用，这便是统一资源标识符（Universal Resource Identifier）被设计和如此命名的缘由。URI是网络上任何一个对象的名字，而它也仅仅是一个名字而已。URI就好比是资源的名字，它被用来标识、引用和查找资源，与现实世界中人名不同的是在网络上URI不会重复。

RPC是面向活动或方法的，无论这一方法是一个对象的成员方法还是全局 方法，我们习惯了通过客户端传递参数给他们然后获得返回值的模式，

对URI本身的“操作” 只存在两种：referencing和dereferencing，前者引用一个URI来指代一个资源，后者表示了通过URI来取回实际对象的命令，

REST定义了一个架构设计原则，其核心思想是服务的设计以系统资源为中心而不是以服务的功能为中心，包括资源的状态如何表示并如何通过HTTP传输给用不同语言编写的客户端

REST定义了一个架构设计原则，其核心思想是服务的设计以系统资源为中心而不是以服务的功能为中心，包括资源的状态如何表示并如何通过HTTP传输给用不同语言编写的客户端。

**当我们在设计Restful服务的时，应该遵循一下设计原则：**

**原则一： 使用HTTP的方法进行资源访问**

1) 使用HTTP POST方法去创建 资源

2) 使用HTTP GET方法去读取 资源

3) 使用HTTP PUT 方法去更新 资源

4) 使用HTTP DELETE方法去删除 资源

**原则二： 使用无状态/无会话的服务设计**

**原则三： 用目录结构风格的URL设计来表示资源**

用清晰的URL路径表示资源可以使客户端更容易理解和操作资源。URL可以被看作是一种自我解释的接口，不需要太多解释就可以让人明白该URL指向的是什么资源以及如何获得相关的资源。

**原则四： 使用XML或JSON来传输数据**

服务和请求的消息数据中包含了对于资源的属性的描述，服务应该采取结构良好并且易于阅读的方式来描述资源。资源可能是数据库中的某个记录集合或者是一个具体的记录，可以是文档，甚至可以是数据中心的服务器

代表性状态传输 (REST) 是一种用于分布式超媒体系统（如万维网）的体系结构样式。REST 风格的体系结构的核心概念是由统一资源标识符 (URI) 标识的资源。可以使用标准接口（如 HTTP）来处理这些资源以及使用这些资源的表示形式来交换信息。

Web Services 是一种基于组件的软件平台,是面向服务的Internet 应用。Web Services 是应用于Internet 的,而不是限于局域网或试验环境。这要求提出的Web Services 框架必须适用于现有的Internet 软件和硬件环境,即服务的提供者所提供的服务必须具有跨平台、跨语言的特性。其次,Web Services 所提供的服务不仅是向人,更需服务于其它应用系统。现有的Web网站也可以认为是面向服务的,但这种服务仅仅可以提供给人使用(只有人类才可以读懂浏览器 下载的页面) 。而新一代的Web Services 所提供的服务应能被机器所读懂,例如其它应用程序及移动设备中的软件系统。这样,我们可以看出,Web Services 的发展方向实际上是构造一个在现有Internet 技术上的分布计算系统。

　SOAP 是Web services 的通信协议。SOAP是一种简单的、轻量级的基于XML 的机制，用于在网络应用程序之间进行结构化数据交换。SOAP包括三部分:一个定义描述消息内容的框架的信封，一组表示应用程序定义的数据类型实例的编码 规则，以及表示远程过程调用和响应的约定。

WSDL表示WEB服务说明语言。WSDL文件是一个XML 文档，用于说明一组SOAP消息以及如何交换这些消息。

UDDI(统一描述发现和集成) 提供一种发布和查找服务描述的方法。UDDI 数据实体提供对定义业务和服务信息的支持。WSDL 中定义的服务描述信息是UDDI注册中心信息的补充。

Web Services 的技术主要建立在XML 的规范之上,这保证了这一体系结构的平台无关性、语言无关性和人机交互性能。

# Authentication and Authorization

Web 应用的安全性包括**用户认证（Authentication）和用户授权（Authorization）**两个部分。

用户认证指的是验证某个用户是否为系统中的合法主体，也就是说用户能否访问该系统。

用户授权指的是验证某个用户是否有权限执行某个操作。在一个系统中，不同用户所具有的权限是不同的。比如对一个资源来说，有的用户只能进行读取，而有的用户可以进行修改。一般来说，**系统会为不同的用户分配不同的角色，而每个角色则对应一系列的权限**

认证手段一般是保存到数据库表的用户名/密码，usb key，ldap等。一般情况下，我们都使用用户名/密码的

身份验证：

连同用户证书一起发送请求到指定URL，如果请求成功，浏览器就返回cookie,浏览器将稍后自动发送该cookie验证用户。

## AD LDS Server

目录服务其实也是一种数据库系统，只是这种数据库是一种树形结构，而不是通常使用的关系数据库。目录服务与关系数据库之间的主要区别在于：二者都允许对存储数据进行访问，只是目录主要用于读取，其查询的效率很高，而关系数据库则是为读写而设计的

访问数据库：

Start > Administrative Tools > ADSI Edit > Connection Settings:

Computer:

Select or Type a domain or server: qq0006cr450w3sw.code1.emi.philips.com:389

Connection Point:

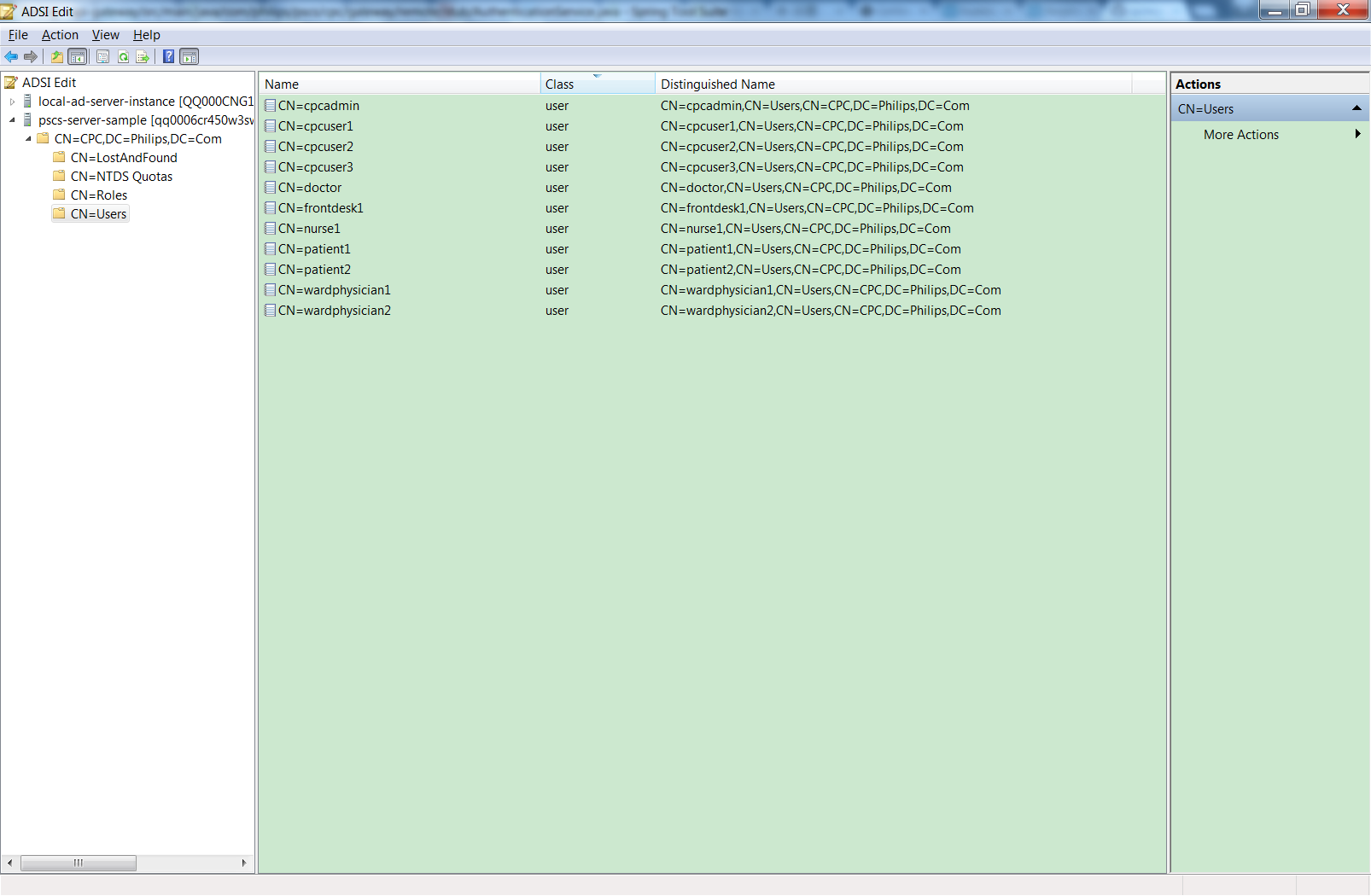
Select or type a Distinguished Name or Naming Context:

CN=CPC,DC=Philips,DC=Com

Name: name instance you want

则自动生成Path: LDAP://qq0006cr450w3sw.code1.emi.philips.com:389/CN=CPC,DC=Philips,DC=Com

Click ok -> ADSI Edit界面



1. 添加用户信息

Right click CN=CPC,DC=Philips,DC=Com -> new Objects… -> select container -> value: Users

Right click -> new Objects…-> select user -> value: username1

Double click CN= username1-> select one attribute -> click Edit -> edit value

Note: right click -> CN=username1 -> Reset Password -> pass@123

密码是不显示在Attribute Editor界面中，同样用代码也无法查询到。用户密码比对一般是在AD Server内部执行。发送密码到AD Server保存时，需要用md5进行加密

1. 添加角色信息

Right click CN=CPC,DC=Philips,DC=Com -> new Objects…-> select container -> value: Roles

Right click -> new Objects… -> select group -> value: WardPhysician

Double click CN=WardPhysician -> double click member -> Add DN… -> CN=wardphysician2,CN=Users,CN=CPC,DC=Philips,DC=Com

角色可以有很多用户，用户可以赋于多种角色，简单地说，角色和用户是多对多关系

## 邮件发送

SMTP（Simple Mail Transfer Protocol）即简单邮件传输协议,它是一组用于由源地址到目的地址传送邮件的规则，由它来控制信件的中转方式，属于TCP/IP协议簇。SMTP服务器则是遵循SMTP协议的发送邮件服务器，用来发送或中转发出的电子邮件。

跟大多数应用层协议一样，SMTP也存在两个 端：在发信人的邮件服务器上执行的客户端和在收信人的邮件服务器上执行的服务器端。SMTP的客户端和服务器端同时运行在每个邮件服务器上。当一个邮件服务器在向其他邮件服务器发送邮件消息时，它是作为SMTP客户在运行。

SMTP通常有两种工作模式：发送SMTP和接收SMTP。具体工作方式为：发送SMTP在接到用户的邮件请求后，判断此邮件是否为本地邮件，若是直接投送到用户的邮箱，否则向dns查询远端邮件服务器的MX纪录，并建立与远端接收SMTP之间的一个双向传送通道。具体过程如下：

（1）建立TCP连接。

（2）客户端向服务器发送HELO命令以标识发件人自己的身份，然后客户端发送MAIL命令。

（3）服务器端以OK作为响应，表示准备接收。

（4）客户端发送RCPT命令。

（5）服务器端表示是否愿意为收件人接收邮件。

（6）协商结束，发送邮件，用命令DATA发送输入内容。

（7）结束此次发送，用QUIT命令退出。

SMTP服务器基于DNS中的邮件交换（MX）记录路由电子邮件。电子邮件系统发邮件时是根据收信人的地址后缀来定位邮件服务器的。SMTP通过用户代理程序（UA）完成邮件的编辑、收取和阅读等功能；通过邮件传输代理程序（MTA）将邮件传送到目的地。

# 数据库

## MongoDB

Install and set env

>mongod –dbpath /data

>mongo

Mongo> show dbs //show all database

Mongo> use test // select database

Mongo> db.dropDatabase() // remove database

Mongo> show collections //show collections of selected database

Mongo> db.customer.drop() //delete collection

Mongo>db.customer.find() // read all documents of one collection of one database

Mongo>db.customer.find({‘firstname’:’Alice’}) // read

Mongo>db.customer.insert({‘firstname’:’QiZhong’, ‘lastname’: ‘Lin’}) // post

Mongo>db.customer.update({‘firstname’:’Alice’}, {‘firstname’:’QiZhong’, ‘lastname’:’Lin’}) // put

Mongo>db.customer.remove({‘lastname’:’Lin’}) //delete

## Redis (remote dictionary service)

也称为数据结构服务器，发布订阅系统，作为有用的数据结构算法和程序的工具包。作为栈，队列或优先队列， 对象存储系统（通过hashtable），集合；支持集合操作（并，交，差），提供原子命令，和多步命令的事务机制；键到期及缓存

1. Download and install [Redis-x64-3.0.500-rc2.msi](https://github.com/MSOpenTech/redis/releases/download/win-3.0.500-rc2/Redis-x64-3.0.500-rc2.msi) from <https://github.com/MSOpenTech/redis/releases>
2. Run Redis server:

C:\Program Files\redis > redis-server redis.windows.conf

1. Run Redis cli:

C:\Program Files\redis > redis-cli

127.0.0.1:6379> ping

// 键值

>set key value //mset key1 value1 key2 value2 …

> get key //mget key1 key2

//hash table (map) 嵌套的Redis对象

mset key1 value1 key2 value2

mget key1 key2

hmset key key1 value1 key2 value2 …

hvals key

hkeys key

for example:

mset user:qzlin:name “qizhong lin” user:qzlin:password changeit //使用:作为分离的键

hmset user:qzlin name “qizhong lin” password changeit //不使用分离的键

hvals user:qzlin

hkeys user:qzlin

// 列表

rpush qzlin:wishlist google yahoo baidu //建立列表

lrange qzlin:wishlist 0 -1 //取子集

lrem qzlin:wishlist 0 google //删除元素

rpoplpush qzlin:wishlist qzlin:visited //从左列表弹出元素， 在右列表压入元素

// set 集合

sad key value1 value2 … //创建集合

smembers key //获取集合

sinterstore key key1 key2 //求交集

sdiffstore key key1 key2 //差集

sunionstore key key1 key2 //并集key1, key2, 结果存入key中

for example:

sad news nytimes.com baidu.com

smembers news

// 有序集合 (随机存取的优先级队列)

zadd key score1 key1 score2 key2

zrange key idx0 idx 1

zrangebyscore key score0 score1

// 加权后求并集

zunionstore destination numkeys key1 key2 … [weights weight1 weight2 …] [aggregate sum|min|max]

for example:

zadd visits 500 google 9 yahoo 10 baidu

zunionstore importance 2 visits votes weights 1 2 aggregate sum

事务: multi … exec ; multi和exec内多命令按事务执行，在使用multi命令时，命令在实际上并不执行，而是排入队列，然后按顺序执行。

到期：有助于避免总的键集无限增长，做法是安排Redis经过一定的时间就删除一个键-值对

expire key seconds

setex key second value // 设置key value的生命

ttl key //获取key的生命

persist key //最近使用缓存算法(MRU, Most Recently Used)： 每当你检索一个值时，更新它的到期时间，确保你最近使用的键将继续保留在Redis中，而被忽视的键将正常到期

Redis命名空间称为数据库(database),以数字为键。默认的命名空间0（也称为数据库0）交互

// select 0

set greeting hello

get greeting

select 1 //通过select切换到数据库1

get greeting //返回(nil)

所有的数据库都运行在同一服务器实例内，Redis就允许用move命令，在不同命令空间之间移动键. 若单个Redis服务器运行不同的应用程序，又要允许这些应用程序相互之间交换数据

move greeting 2

select 2

get greeting

别的操作命令： rename, type, del, flushdb, flushall.

//发布-接收模式

127.0.0.1:6379> brpop comments 300 //某客户端监听 key=comments, 超时为5分钟,

127.0.0.1:6379> lpush comments “hello world!” //某客户端发布 key-values

## PostgreSQL

Download and install postgresql-9.5.0-rc1-windows-x64.exe

安装过程会要求设置管理员密码：qzlin （默认管理员:postgre）

数据库默认地址: C:\Program Files\PostgreSQL\9.5\data

Open pgAdmin III

连接服务器： right click PostgreSQL 9.5 (localhost:5432) -> 连接服务器

数据库 -> 创建数据库 //创建数据库

登录角色 -> 创建登录角色 //创建登录角色

# Domain-specific language

## Drools

规则引擎一般由3部分组成：规则库(Knowledge base)+Working Memory(Fact base)+推理机(规则引擎)。

使用 Drools 规则引擎实现业务逻辑

使用声明性编程方法编写程序的业务逻辑

实现 J2EE 或 J2SE 应用程序中业务逻辑最常见的方法是编写 Java 代码来实现需求文档的规则和逻辑。在大多数情况下，该代码的错综复杂性使得维护和更新应用程序的业务逻辑成为一项令人畏惧的任务，任何更改，不管多么简单，仍然会产生重编译和重部署成本。

将规则引擎看作实现复杂业务逻辑的框架。大多数规则引擎允许您使用声明性编程来表达对于某些给定信息或知识有效的结果

有多个规则引擎可供使用，其中包括商业和开放源码选择。商业规则引擎通常允许使用专用的类似英语的语言来表达规则。其他规则引擎允许使用脚本语言（比如 Groovy 或 Python）编写规则

Eclipse IDE（Versions 3.2 和 3.3）的一个 Drools 插件。我强烈建议您通过这个插件来使用 Drools。它可以简化使用 Drools 的项目开发，并且可以提高生产率。例如，该插件会检查规则文件是否有语法错误，并提供代码完成功能。它还使您可以调试规则文件，将调试时间从数小时减少到几分钟。您可以在规则文件中添加断点，以便在规则执行期间的特定时刻检查对象的状态。这使您可以获得关于规则引擎在特定时刻所处理的知识（knowledge）（在本文的后面您将熟悉这个术语）的信息。

何时使用规则引擎？

并非所有应用程序都应使用规则引擎。如果业务逻辑代码包括很多 if-else 语句，则应考虑使用一个规则引擎。维护复杂的 Boolean 逻辑可能是非常困难的任务，而规则引擎可以帮助您组织该逻辑。当您可以使用声明方法而非命令编程语言表达逻辑时，变化引入错误的可能性会大大降低。

如果代码变化可能导致大量的财政损失，则也应考虑规则引擎。许多组织在将已编译代码部署到托管环境中时具有严格的规则。例如，如果需要修改 Java 类中的逻辑，在更改进入生产环境之前，将会经历一个冗长乏味的过程：

必须重新编译应用程序代码。

在测试中转环境中删除代码。

由数据质量审核员检查代码。

由托管环境架构师批准更改。

计划代码部署。

即使对一行代码的简单更改也可能花费组织的几千美元。如果需要遵循这些严格规则并且发现您频繁更改业务逻辑代码，则非常有必要考虑使用规则引擎。

对客户的了解也是该决策的一个因素。尽管您使用的是一个简单的需求集合，只需 Java 代码中的简单实现，但是您可能从上一个项目得知，您的客户具有在开发周期期间甚至部署之后添加和更改业务逻辑需求的倾向（以及财政和政治资源）。如果从一开始就选择使用规则引擎，您可能会过得舒服一些。

# 分布式计算

## Hadoop

// install hadoop

添加JAVA\_HOME (查看java安装路径$ update-alternatives --config java)

安装ssh

$ sudo apt-get install ssh

$ sudo apt-get install rsync

下载并解压hadoop-x.y.z.tar.gz,然后放入到opt目录中

$sudo mv Documents/hadoop/tools/hadoop-2.7.1 /opt/

设置环境变量

$gedit ~/.bashrc

HADOOP\_HOME=/opt/hadoop-2.7.1

PATH=$PATH:$HADOOP\_HOME/bin

export HADOOP\_HOME

export PATH

$source ~/.bashrc

检查hadoop version

$ hadoop version

// install IDE

// install gradle

download and unzip gradle\*\*\*.zip

$sudo mv gradle\_path /opt

$ gedit ~/.bashrc

GRADLE\_HOME=<path to gradle bin file> (e.g. /opt/gradle-1.5 )

export GRADLE\_HOME

PATH=$PATH:$GRADLE\_HOME/bin

export PATH

$source ~/.bashrc

$gradle –version

// install spring tool suite

$ download and unzip sts\*\*\*.zip

$ sudo mv sts\_path /opt

//开发一个应用

MaxTemperatureDriver

build.gradle

build.gradle 文件内容如下：

buildscript {

repositories {

mavenCentral()

mavenLocal()

maven {

url "https://repository.cloudera.com/artifactory/cloudera-repos/"

}

dependencies {

classpath "org.springframework.boot:spring-boot-gradle-plugin:1.2.3.RELEASE"

}

}

}

repositories {

mavenCentral()

}

apply plugin: 'java'

jar {

baseName = 'MaxTemperatureDriver'

version = '0.1.0'

manifest {

attributes 'Main-Class': 'com.philips.v1.maxtemperaturedriver.MaxTemperatureDriver'

}

}

dependencies {

compile 'org.apache.hadoop:hadoop-core:1.2.1'

compile 'org.apache.hadoop:hadoop-common:2.7.1'

compile 'org.apache.hadoop:hadoop-client:2.7.1'

compile 'org.apache.hadoop:hadoop-hdfs:2.7.1'

compile 'org.apache.hadoop:hadoop-mapreduce-client-core:2.7.1'

compile 'org.apache.hadoop:hadoop-minicluster:2.7.1'

testCompile 'junit:junit:4.10'

}

task wrapper(type: Wrapper){

gradleVersion = '2.3'

}

sts导入MaxTemperatureDriver项目，创建包com.philips.v1.maxtemperaturedriver

创建map,reduce函数，再创建driver运行job (具体见Hadoop权威指南)

MaxTemperatureDriver$ gradle build

MaxTemperatureDriver$ Hadoop jar build/libs/MaxTemperatureDriver-0.1.0.jar –conf /opt/hadoop-2.7.1/etc/hadoop/hadoop-local.xml input/sample.txt output

# 测试

## 压力测试

JMeter 是 Apache 基金会 Jakarta 上的一个纯 Java 开源项目，起初用于基于 Web 的压力测试（pressure test），后来其应用范围逐渐扩展到对文件传输 FTP, 大型数据库（JDBC 方式），脚本程序（CGI, Perl 等），Web Services，Java 应用系统等方面的测试。JMeter 本身主要用于性能测试，如系统压力等。除此之外，JMeter 能够对应用系统做功能测试和回归测试，并且能够通过使用带有断言的脚本程序来验证系统然后返回用户期望的结果

消息传递协议：SOAP

SOAP（Simple Object Access Protocol）称为简单对象访问协议， 是 W3C 定义的一种标准消息传递协议，而它通常被认为是 Web Services 的事实标准。SOAP 是在去中心化（Decentralized）分布式（Distributed）环境中用来信息交换的一个轻量级协议。

SOAP 使用 RPC（远程过程调用）和消息传递来建立通信服务，SOAP RPC 定义了用于表示远程过程调用和应答的协议。SOAP 协议本身仅仅定义了消息的交换结构，它可以和许多现存因特网协议结合在一起使用，其中包括超文本传输协议（ HTTP），Java 消息服务（JMS）

HTTP（超文本传送协议）是属于应用层的面向对象的协议，是万维网 (WWW) 的基础，由于其简单快速、灵活、无连接、无状态的方式，适用于分布式网络信息系统。SOAP Over HTTP 应用就是指的是遵守 SOAP 编码规则的 HTTP 请求 / 响应，我们可以用简单的公式来对此作一个描述：HTTP + XML = SOAP。

JMeter 也同样提供了两种 Sampler 分别建立对这两种服务的调用：Web Services (SOAP) Request 和 JMS Point-to-Point

# Office Note

(modify the document with record)

Review -> Track Changes

Review -> New Comments

Review -> Accept