股票行情数据推送系统SwiftMarket

的设计思想和实现

李华锋

2013/4/1

# 1 概述

## 1.1 解决思路

3月26日拿到问题后，晚上便开始阅读和理解题意，并形成粗略的解决思路。可简要描述为以下几点：

* 了解行业信息，翻查Level-2，安装大智慧客户端；
* 确定开发和运行环境，并搭建平台，包括Redhat Linux 6.1，GitHub等；
* 快速预研需使用的各个技术，留下印象作技术定位；
* 结合需求，思考各个系统的功能与接口，形成初步的实现思想；
* 首先开发股票行情模拟生成器，然后是数据推送中间件；
* 改进工作；
* 撰写文档，最终发布系统等；

## 1.2 系统简介

用户浏览器

高性能缓存

推送中间件

模拟生成器

http server

tcp server

cache

图 SwiftMarket系统基本架构

### 1.2.1 股票行情模拟生成器（DataSimulator）

持续地生成行情数据，并将数据发送给已连接的远端（推送中间件）。

### 1.2.2 数据推送中间件（DataPusher）

连接数据源（行情模拟生成器），并获取实时数据。同时，监听并接受Http连接，将实时数据推送至所有的http客户端。

## 1.3 使用技术

1) 行情模拟生成器

* Java多线程，通知/等待机制，读写锁；
* 基于TCP/IP的Socket；

2) 数据推送中间件

* node.js；
* 基于node.js的Web框架Express；
* socket.io(Web Socket的一个实现)；
* redis缓存服务；
* Html 5/javascript；
* JSON；
* CSS3

# 2 股票行情模拟生成器

## 2.1 行情动态数据

Level2包含的数据十分丰富。这里的行情模拟生成器只生产股票的信息有：股票代号，股票名称，实时价格，实时交易量和时间。

## 2.2 数据生成器

### 2.2.1 独立线程

数据生成器是一个独立运行的线程。该线程持续运行，不断产生数据并存放至共享数据区。每产生一次数据后，线程睡眠一定时间（2秒）。

### 2.2.2动态策略

* 按概率选择股票

线程的每次循环中，按概率选择股票，被选中的股票才可以更新数据。这样便可以模拟股票实时数据的更新频率不同的情况。

* 随机的价格变化幅度和交易量

每只股票原始价格为10元。每次更新中，按上一次价格随机增减。交易量为[0，1000]间的随机数。

价格公式： 新价格 = 旧价格×0.2×random，random为[0，1]间的小数。

## 2.3 TCP服务器与发送器

### 2.3.1 TCP服务器

TCP服务器是一个独立线程。它不断地监听指定的端口，当有连接来时，接受请求并建立连接，把连接（Socket）交给发送器处理，然后继续监听。

### 2.3.2 发送器

* 独立线程

发送器也是一个独立线程。它负责发送行情数据给连接的另一方。有两种情况：

1）刚刚建立连接时，发送器向远端发送所有的股票数据；

2）之后，向远端只发送更新的股票数据；

* 从共享区读数据

发送器从共享区读取数据，然后发送。

* JSON数据

每次发送数据前，都要把数据转换成JSON数据，再发送。

## 2.4 线程间数据共享

### 2.4.1 数据共享区

数据共享区存放的数据有两种：所有股票的数据，更新的股票数据。

共享区对象使用单例模式来实现。构造函数私有，同步地获取对象。

### 2.4.2 等待/通知

等待/通知机制由函数Wait()、notify()和notifyAll()实现。

* 更新数据的通知策略

1）只有一个写线程。可有多个读线程。

2）当发送器（读线程）要读数据时，它们会在wait()上等待（释放synchronized对象锁）。

3) 当生成器（写线程）更新数据后，它会使用notifyAll()通知所有读线程。这时，所有在wait()上等待的读线程开始读数据。

4）读线程下一次要读数据时，它们继续在wait（）上等待，直到下一次的更新数据的到来。

### 2.4.3 读写锁

只有等待/通知还不行。当读线程正在读数据时，写线程可能会更改数据。所以，给数据共享区加入了读写锁。多个读线程可以一起读，但读线程与写线程在同一时刻只有一方在共享区操作。

# 3 数据推送中间件

整个推送中间件用node.js写成。实现了事件机制和异步IO的node.js可提供高性能的并发处理。前后端javascript的同构与javascript的流行，使得node.js适合用于web后端。

WebSocket推送

(socket.io)

Redis缓存

用户浏览器

HTTP服务器

(express)

数据获取

(TCP客户端)

数据

生成器

图 数据推送中间件的框架

## 3.1 HTTP服务器

HTTP服务器监听某个端口，接受来自用户浏览器的HTTP请求，并回传响应数据。

* express

http服务器使用了express来做web框架。express是目前流行的基于node.js的框架。

* 返回html

HTTP连接刚建立成功时，服务器发送首页html文件给用户作为返回数据。使用了Response的sendfile()函数。

## 3.2 数据获取

* 连接行情模拟生成器，获取数据

使用TCP/IP连接行情模拟生成器。Socket对象监听“data”事件。该事件表示新数据到来。

* 立即推送

新数据的到达，触发推送数据给用户浏览器。推送工作由WebSocket（socket.io）完成。

* 更新缓存

获取数据后，更新redis缓存中的数据。

* JSON

获取数据后，按JSON格式处理。

## 3.3 数据推送

* WebSocket

数据推送依据WebSocket协议来完成。socket.io是该协议的一个实现。

从上面知道，HTTP服务器给用户浏览器返回html文件。浏览器执行文件中的脚本，与服务器建立WebSocket连接。那么，服务器可使用该连接推送数据。

* 推送情景

1) 刚建立WebSocket连接时，从redis缓存中获取所有的股票数据，然后推送给用户浏览器。**这样，便解决了用户刚建立连接时短时间无数据的问题。**

2) 新数据从模拟生成器到达时，立即触发推送。即，新到达的数据，马上被WebSocket发送到用户浏览器。

## 3.4 高性能缓存

* redis

高性能缓存用redis实现。redis是key-value存储系统，它可以存储string，还有list、集合等数据结构。

* 使用方法

使用node.js版的redis客户端连接redis服务器，进行存取。中间件与redis服务器可均在本地，也可以是异地。

* 缓存数据

缓存中有两类数据：所有股票的唯一标识符，所有股票的当前数据（下一次更新到来前）。

## 3.5 首页HTML

* HTML 5

基于html 5实现首页。股票数据按行显示在表格中。使用CSS3来修饰表格。

* 动态更新数据

数据更新由javascript脚本实现。浏览器与web服务器建立WebSocket连接，接收服务器推送过来的数据。两种情景：

1）若股票原先不存在，则添加新的数据行。

2）若股票原先存在，则更新相应行的数据。

# 4 系统发布与运行

## 4.1 发布在GitHub

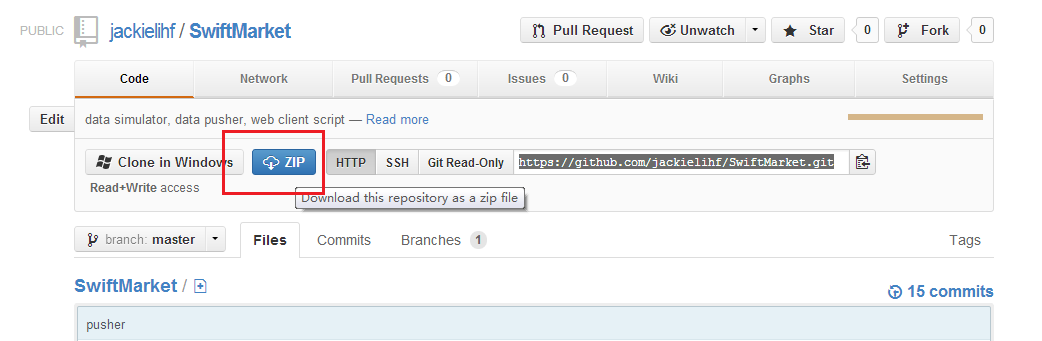
* 查看项目的GitHub地址

https://github.com/jackielihf/SwiftMarket

* 用于获取源代码的GitHub地址

https://github.com/jackielihf/SwiftMarket.git

* 直接从网站下载ZIP文件，如图：



## 4.2 部署过程

### 4.2.1 环境准备

* 操作系统

SwiftMarket的开发和运行测试的操作系统均为Redhat Enterprise Linux 6.1 x64。也可以使用其他Linux操作系统。

* 工具和版本

|  |  |  |
| --- | --- | --- |
| 工具 | 版本 | 用途 |
| JDK | SUN 1.7.0\_15 | Java运行环境 |
| node.js | 0.10.0-linux-x64 | Web服务器后端平台和语言 |
| npm | 1.2.14 | node.js包管理工具 |
| express | 3.1.0 | node.js的web框架 |
| redis for node.js | 0.8.2 | node.js的redis客户端 |
| socket.io | 0.9.14 | WebSocket协议的一个实现 |
| redis server | 2.6.12 | 高性能缓存 |

### 4.2.2 安装工具

* 安装JDK 1.7.0

到Sun公司的官方网站下载JDK1.7.0，安装完成后配置环境变量：

#vi /etc/profile

添加以下内容到文件末尾：

export JAVA\_HOME=/usr/java/jdk1.7.0\_15

export PATH=$PATH:$JAVA\_HOME/bin

export CLASSPATH=.:$JAVA\_HOME/lib

* 安装node.js

1) 从<http://nodejs.org/download/>下载 node-v0.10.0-linux-x64.tar.gz，或者是更新的版本。

2) 解压文件到目录/usr/local/node中

#tar xzvf node-v0.10.0-linux-x64.tar.gz

3) 设置环境变量

# vi /etc/profile

在文件末尾添加：

export NODE\_HOME=/usr/local/node/node-v0.10.0-linux-x64

export PATH=$PATH:$NODE\_HOME/bin

export NODE\_PATH=$NODE\_HOME/lib/node\_modules

保存文件后，source该配置使得它生效：

#source /etc/profile

4) 查看版本以确认是否安装成功：

#node -v

* 安装npm

Npm已经捆绑在node.js中，即安装node.js的同时安装了npm。查看npm版本的命令为：

# npm -v

* 使用npm全局安装express

# npm install -g express

安装完成后，可以在/usr/local/node/node-v0.10.0-linux-x64/lib/node\_modules目录中找到express目录。

* 使用npm全局安装redis客户端

#npm install -g redis

* 使用npm全局安装socket.io客户端

#npm install -g socket.io

* 安装redis server

1) 从官方网站<http://www.redis.io/> 下载稳定版本redis 2.6.12。

2) 解压后，编译和安装：

# tar xzvf redis-2.6.12.tar.gz

# cd redis-2.6.12

# make && make install

3) 查看版本

# redis-server -v

4) redis.conf是配置文件

5) 你可以只编译redis，而不安装它。这样需要在redis目录的src文件夹中找到可执行文件。

## 4.3 运行与测试

### 4.3.1 代码结构

SwiftMarket/

|--DataSimulator/

|--bin/

|--build/

|-src/

|--MANIFEST.MF

|--DataPusher/

|--pusher.js

|--index.html

|--doc/

|--some documents including Design Guide

|--redis/

|--redis.conf

|--README.md

|--startAll.sh

### 4.3.2 运行测试

* 编译DataSimulator生成可执行文件

DataSimulator用Java写成。在build目录下有编译该Java程序的脚本。

$cd SwiftMarket/DataSimulator/build

$sh build.sh

编译成功后会生成一个jar 文件dataSimulator.jar在build目录。

* 编译DataPusher

DataPusher用node.js写成，不需要编译。

* 启动运行DataSimulator

dataSimulator默认运行在端口9000上，也可以用参数项-p来指定特定端口。运行命令为：

$java -jar dataSimulator.jar [-p port]

* 启动redis server

redis.conf是 redis server的配置文件。在启动前，可以修改该配置文件。 Redis服务器默认运行在本地的6379端口。启动命令为：

$redis-server redis.conf

* 运行DataPusher

启动redis服务器成功后，才可以运行DataPusher。Web服务器默认在端口9090上监听来自用户浏览器的HTTP请求。

$cd DataPusher

$node pusher.js

你可以用更多的参数项来启动Web服务器：

1）-p指定http服务器监听端口；

2）-r指定redis服务器的IP地址；

3）-rp指定redis服务器的端口；

4）-s指定DataSimulator的IP地址；

5）-sp指定DataSimulator的端口；

* 脚本startAll.sh

你也可以执行脚本文件startAll.sh来启动DataSimulator，redis服务器和DataPusher。

$sh startAll.sh

* 从用户浏览器访问网站

当所有的服务器启动成功后，便可以从浏览器访问网站。本地的访问地址：

<http://localhost:9090>

或者使用web服务器的IP地址：

http://DataPusher's IP address:9090

* 在浏览器可成功看到SwiftMarket的首页。股票市场的实时数据不断的在更新。

### 4.3.3 效果图

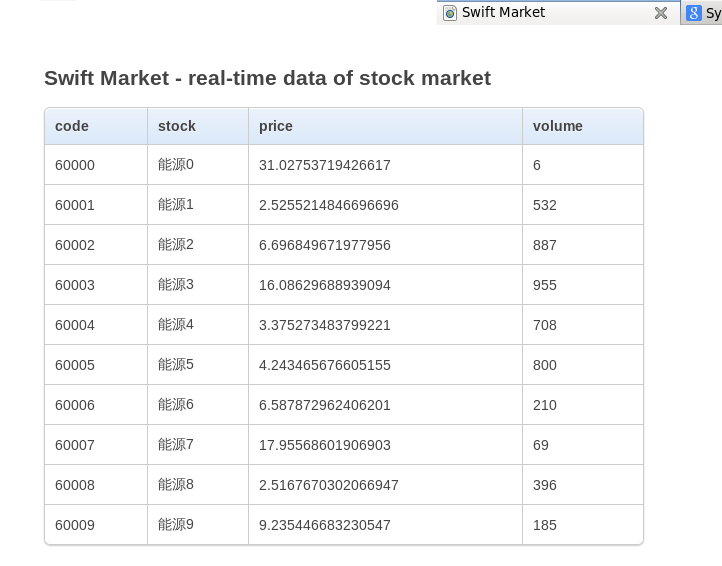


图 首页的股票数据表格（价格和成交量是动态变化的）

# 5 未来扩展与思考

## 5.1 Web框架

首先，可以做的是完善Web框架。比如，使用express时，可以实现URL路由，利用express提供的工具等。

给系统添加Mysql作为持久存储。

可以将模拟生成器、推送中间件、redis服务器部署在不同的物理机器上，测试性能。

## 5.2 用户体验

思考如何改善用户体验。增加网页的有益元素，比如Logo，装饰图片，页脚。

增加与用户的交互功能。增强数据安全。

## 5.3 性能与稳定

为了应对高并发和高访问量，须提高系统的性能和稳定性。可以考虑以下做法：

1）建设Apache或者Nginx服务器，负责静态数据的响应，如图片、静态网页等。

2）使用DNS负载均衡，结合NAT负载均衡或者直接路由，构建规模更大的服务群组。