**可配置化网络爬虫项目环境搭建说明书**

**小组名称：**可配置化网络平台小组

**小组口号：**我们爱学习

**指导教师：**陈博老师

**文档撰写人：**吴祥龙

**文档撰写时间：**2017年5月20日

团队分工记录表

|  |  |  |  |
| --- | --- | --- | --- |
| **项目名称** | **学号** | **姓名** | **分工** |
| 可配置化网络爬虫 | SA17225405 | 吴祥龙 | 环境搭建 |
| SA17225094 | 关玉婷 | 环境搭建 |
| SA17225362 | 王润 | 环境搭建 |
| SA17225362 | 唐冬鑫 | 环境搭建 |

# 

目录

[项目环境的搭建 2](#_Toc6046)

[1 WebMagic框架 2](#_Toc25482)

[1.1 总体架构 2](#_Toc20624)

[1.2 WebMagic四大组件 3](#_Toc21520)

[1.3 用于数据流转的对象 4](#_Toc29453)

[1.4 控制爬虫运转的引擎Spider 4](#_Toc15881)

[2 远程调用框架thrift 5](#_Toc14311)

[2.1 thrift的优点 5](#_Toc457)

[2.2 thrift架构 5](#_Toc26079)

[2.3 thrift 的协议，数据数据传输格式，传输方式和服务类型 6](#_Toc31707)

[3 ZooKeeper与集群的管理 7](#_Toc7388)

[3.1 Zookeeper技术 7](#_Toc21486)

[3.2 利用Zookeeper实现集群的管理 8](#_Toc22442)

[4 本章小结 9](#_Toc11274)

# 

# 项目环境的搭建

## 1 WebMagic框架

### 1.1 总体架构

WebMagic是一个用java开发的开源的垂直爬虫框架，用户可以使用WebMagic进行简单的定义开发一个功能强大的爬虫，在互联网上抓取想要的数据。

WebMagic的结构分为Downloader、PageProcessor、Scheduler、Pipeline四大组件，并由Spider将它们彼此组织起来。这四大组件对应爬虫生命周期中的下载、处理、管理和持久化等功能。WebMagic的设计参考了Scapy，但是实现方式更Java化一些。而Spider则将这几个组件组织起来，让它们可以互相交互，流程化的执行，可以认为Spider是一个大的容器，它也是WebMagic逻辑的核心。

WebMagic总体架构图如下：

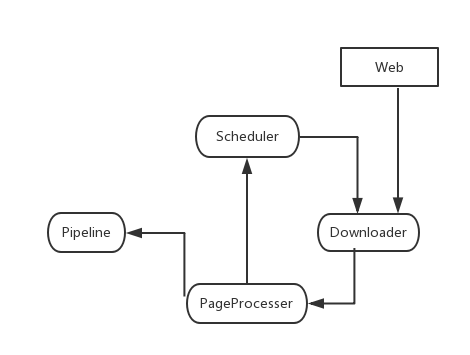


图 2-1 WebMagic总体架构图

### 1.2 WebMagic四大组件

1.Downloader

Downloader负责从互联网上下载页面，以便后续处理。WebMagic默认使用了HttpClient作为下载工具。

2.PageProcessor

PageProcessor负责解析页面，抽取有用信息，以及发现新的链接。WebMagic使用Jsoup作为HTML解析工具，并基于其开发了解析XPath的工具Xsoup。

在这四个组件中，PageProcessor对于每个站点每个页面都不一样，是需要使用者定制的部分。

3.Scheduler

Scheduler负责管理待抓取的URL，以及一些去重的工作。WebMagic默认提供了JDK的内存队列来管理URL，并用集合来进行去重。也支持使用Redis进行分布式管理。

除非项目有一些特殊的分布式需求，否则无需自己定制Scheduler。

4.Pipeline

Pipeline负责抽取结果的处理，包括计算、持久化到文件、数据库等。WebMagic默认提供了“输出到控制台”和“保存到文件”两种结果处理方案。

Pipeline定义了结果保存的方式，如果你要保存到指定数据库，则需要编写对应的Pipeline。对于一类需求一般只需编写一个Pipeline。

### 1.3 用于数据流转的对象

1. Request

Request是对URL地址的一层封装，一个Request对应一个URL地址。

它是PageProcessor与Downloader交互的载体，也是PageProcessor控制Downloader唯一方式。

除了URL本身外，它还包含一个Key-Value结构的字段extra。你可以在extra中保存一些特殊的属性，然后在其他地方读取，以完成不同的功能。例如附加上一个页面的一些信息等。

2. Page

Page代表了从Downloader下载到的一个页面——可能是HTML，也可能是JSON或者其他文本格式的内容。

Page是WebMagic抽取过程的核心对象，它提供一些方法可供抽取、结果保存等。在第四章的例子中，我们会详细介绍它的使用。

3. ResultItems

ResultItems相当于一个Map，它保存PageProcessor处理的结果，供Pipeline使用。它的API与Map很类似，值得注意的是它有一个字段skip，若设置为true，则不应被Pipeline处理。

### 1.4 控制爬虫运转的引擎Spider

Spider是WebMagic内部流程的核心。Downloader、PageProcessor、Scheduler、Pipeline都是Spider的一个属性，这些属性是可以自由设置的，通过设置这个属性可以实现不同的功能。Spider也是WebMagic操作的入口，它封装了爬虫的创建、启动、停止、多线程等功能。下面代码展示了一种Spider的创建和启动的过程。

public static void main(String[] args) {

Spider.create(new GithubRepoPageProcessor())

//从https://github.com/code4craft开始抓

.addUrl("https://github.com/code4craft")

//设置Scheduler，使用Redis来管理URL队列

.setScheduler(new RedisScheduler("localhost"))

//设置Pipeline，将结果以json方式保存到文件

.addPipeline(new JsonFilePipeline("D:\\data\\webmagic"))

//开启5个线程同时执行

.thread(5)

//启动爬虫

.run();

}

## 2 远程调用框架thrift

### 2.1 thrift的优点

在分布式服务框架中，一个最基础的问题就是远程服务是怎么通讯的，目前流行的服务调用方式有很多种，例如基于 SOAP 消息格式的 Web Service，基于 JSON 消息格式的 RESTful 服务等。其中所用到的数据传输方式包括 XML，JSON 等，然而 XML 相对体积太大，传输效率低，JSON 体积较小，新颖，但还不够完善。由 Facebook 开发的远程服务调用框架 Apache Thrift，它采用接口描述语言定义并创建服务，支持可扩展的跨语言服务开发，所包含的代码生成引擎可以在多种语言中，如 C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk 等创建高效的、无缝的服务，其传输数据采用二进制格式，相对 XML 和 JSON 体积更小，对于高并发、大数据量和多语言的环境更有优势。

### 2.2 thrift架构

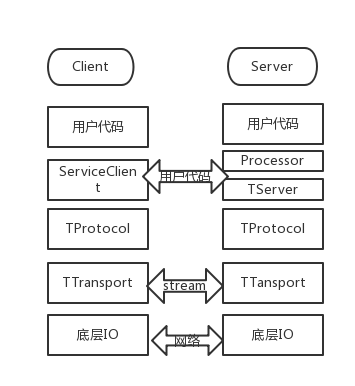


图 2-2 thrift的协议栈整体的架构图

图 2-2是thrift的协议栈整体的架构，thrift是一个客户端和服务器端的架构体系（c/s），在最上层是用户自行实现的业务逻辑代码。第二层是由thrift编译器自动生成的代码，主要用于结构化数据的解析，发送和接收。TServer主要任务是高效的接受客户端请求，并将请求转发给Processor处理。Processor负责对客户端的请求做出响应，包括RPC请求转发，调用参数解析和用户逻辑调用，返回值写回等处理。从TProtocol以下部分是thirft的传输协议和底层I/O通信。TProtocol是用于数据类型解析的，将结构化数据转化为字节流给TTransport进行传输。TTransport是与底层数据传输密切相关的传输层，负责以字节流方式接收和发送消息体，不关注是什么数据类型。底层IO负责实际的数据传输，包括socket、文件和压缩数据流等。

### 2.3 thrift 的协议，数据数据传输格式，传输方式和服务类型

协议层TProtocol：

在传输协议上总体上划分为文本(text)和二进制(binary)传输协议, 为节约带宽，提供传输效率，一般情况下使用二进制类型的传输协议为多数。

1.TBinaryProtocol  二进制编码格式进行数据传输。

 2.TCompactProtocol  高效的编码方式，使用类似于protobuffer的Variable-Length Quantity (VLQ) 编码(可以节省传输空间，使数据的传输效率更高)对数据进行压缩。

3.TJSONProtocol 使用JSON的数据编码协议进行数据传输。

4.TSimpleJSONProtocol 这种节约只提供JSON只写的协议，适用于通过脚本语言解析

5.TDebugProtocol 在开发的过程中帮助开发人员调试用的，以文本的形式展现方便阅读。

传输层Ttransport:

1.TSocket 使用阻塞式I/O进行传输，也是最常见的模式。

2.TFramedTransport 使用非阻塞方式，按块的大小，进行传输，类似于Java中的NIO。

3.TFileTransport 顾名思义按照文件的方式进程传输，虽然这种方式不提供Java的实现，但是实现起来非常简单。

4.TMemoryTransport 使用内存I/O，就好比Java中ByteArrayOutputStream实现。

5.TZlibTransport- 使用执行zlib压缩，不提供Java的实现。

6.TNonblockingTransport-使用非阻塞方式，用于构建异步客户端。

服务端类型 ：

1.TSimpleServer  单线程服务器端使用标准的阻塞式I/O。

2.TThreadPoolServer  多线程服务器端使用标准的阻塞式I/O。

3.TNonblockingServer 多线程服务器端使用非阻塞式I/O，并且实现了Java中的NIO通道。

## 3 ZooKeeper与集群的管理

### 3.1 Zookeeper技术

Zookeeper 作为一个分布式的服务框架，主要用来解决分布式集群中应用系统的一致性问题，它能提供基于类似于文件系统的目录节点树方式的数据存储，但是 Zookeeper 并不是用来专门存储数据的，它的作用主要是用来维护和监控你存储的数据的状态变化。通过监控这些数据状态的变化，从而可以达到基于数据的集群管理，Zookeeper 会维护一个具有层次关系的数据结构，它非常类似于一个标准的文件系统。

Zookeeper 这种数据结构有如下这些特点：

1.每个子目录项如 NameService 都被称作为 znode，这个 znode 是被它所在的路径唯一标识，如 Server1 这个 znode 的标识为 /NameService/Server1

2.znode 可以有子节点目录，并且每个 znode 可以存储数据，注意 EPHEMERAL 类型的目录节点不能有子节点目录

3.znode 是有版本的，每个 znode 中存储的数据可以有多个版本，也就是一个访问路径中可以存储多份数据

4.znode 可以是临时节点，一旦创建这个 znode 的客户端与服务器失去联系，这个 znode 也将自动删除，Zookeeper 的客户端和服务器通信采用长连接方式，每个客户端和服务器通过心跳来保持连接，这个连接状态称为 session，如果 znode 是临时节点，这个 session 失效，znode 也就删除了

5.znode 的目录名可以自动编号，如 App1 已经存在，再创建的话，将会自动命名为 App2

6.znode 可以被监控，包括这个目录节点中存储的数据的修改，子节点目录的变化等，一旦变化可以通知设置监控的客户端，这个是 Zookeeper 的核心特性，Zookeeper 的很多功能都是基于这个特性实现的。

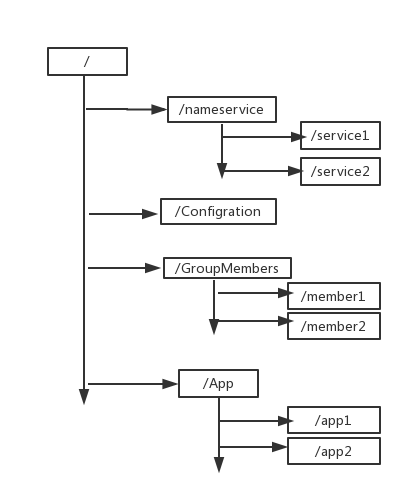


图 2-3 zookeeper目录树图

## 3.2 利用Zookeeper实现集群的管理

Zookeeper 能够很容易的实现集群管理的功能，如有多台 Server 组成一个服务集群，那么必须要一个主节点知道当前集群中每台机器的服务状态，一旦有机器不能提供服务，集群中其它集群必须知道，从而做出调整重新分配服务策略。同样当增加集群的服务能力时，就会增加一台或多台 Server，同样也必须让主节点知道。Zookeeper可以很容易的实现维护集群中机器的状态的功能。它的实现方式是在Zookeeper上创建一个EPHEMERAL 类型的目录节点，然后每个 Server 在它们创建目录节点的父目录节点上调用 getChildren(String path, boolean watch) 方法并设置 watch 为 true，由于是 EPHEMERAL 目录节点，当创建它的 Server 死去，这个目录节点也随之被删除，所以 Children 将会变化，这时 getChildren上的 Watch 将会被调用，所以其它 Server 就知道已经有某台 Server 死去了。新增 Server 也是同样的原理。

## 4 本章小结

本章分析了与这个课题相关的技术，包括WebMagic爬虫框架，thrift远程调用框架，Zookeeper集群管理框架。其中WebMagic框架是本文将要实现的分布式爬虫框架的基础，thrift框架是分布式爬虫框架的远程调用的工具，Zookeeper是分布式爬虫框架的集群的管理工具。