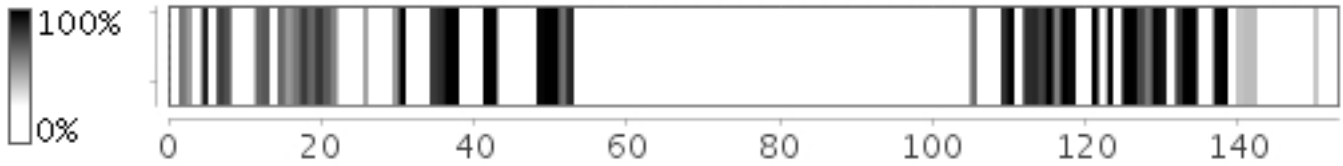
PaperPass专业版检测报告
简明打印版

比对结果（相似度）：
总　体：41 %　*（总体相似度是指本地库、互联网的综合比对结果）*
本地库：8 %　*（本地库相似度是指论文与学术期刊、学位论文、会议论文数据库的比对结果）*
期刊库：6 %　*（期刊库相似度是指论文与学术期刊库的比对结果）*
学位库：4 %　*（学位库相似度是指论文与学位论文库的比对结果）*
会议库： 1 %　*（会议库相似度是指论文与会议论文库的比对结果）*
互联网：39 %　*（互联网相似度是指论文与互联网资源的比对结果）*

编　号：58B6DC3DB3BD5TBYL
版　本：专业版
标　题：基于Node-red与Redis的实时流数据处理模型的设计与
作　者：王江波
长　度：5800 字符(不计空格)
句子数：153句
时　间：2017-3-1 22:35:41
比对库：学术期刊、学位论文（硕博库）、会议论文、互联网资源
查真伪：http://www.paperpass.com/check

句子相似度分布图：



本地库相似资源列表（学术期刊、学位论文、会议论文）：
1. 相似度：2％ 篇名:《基于事件的发布-订阅系统模型》
   来源：学术期刊 《计算机科学》 2006年1期 作者: 汪洋 谢江 王振宇
2. 相似度：1％ 篇名:《Memcached和Redis在高速缓存方面的应用》
   来源：学术期刊 《无线互联科技》 2012年9期 作者: 王心妍
3. 相似度：1％ 篇名:《基于redis的简易应用下载系统的设计与实现》
   来源：学术期刊 《信息通信》 2015年9期 作者: 武红宽 申敏 马怡伟
4. 相似度：1％ 篇名:《内存数据库在点击多方通话中应用》
   来源：学术期刊 《软件》 2012年12期 作者: 刘岩 王晶 王纯
5. 相似度：1％ 篇名:《NodeJS的异步非阻塞I/O研究》
   来源：学术期刊 《工业控制计算机》 2015年3期 作者: 许会元 何利力

6. 相似度：1％ 篇名:《Redis在外勤通系统高速缓存中的应用研究》
   来源：学术期刊 《无线互联科技》 2014年1期 作者: 李彬 朱亚兴
7. 相似度：1％ 篇名:《基于Redis的海量小文件分布式存储方法研究》
   来源：学术期刊 《计算机工程与科学》 2013年10期 作者: 刘高军 王帝澳
8. 相似度：1％ 篇名:《一种多存储引擎Key-Value分布式内存数据库的研究与实现》
   来源：学术期刊 《现代电信科技》 2013年1期 作者: 罗芸 张晓川 朱建中
9. 相似度：1％ 篇名:《基于函数指针的单片机程序模型》
   来源：学术期刊 《电脑知识与技术》 2009年23期 作者: 庞胜利
10. 相似度：1％ 篇名:《基于安全发布/订阅机制的房产信息集成平台实现》
    来源：学位论文 东南大学 2011 作者: 郑倩
11. 相似度：1％ 篇名:《基于P/S模式的分布对象中间件异步通信接口》
    来源：学术期刊 《计算机工程》 2009年6期 作者: 闫晓芬 郭银章
12. 相似度：1％ 篇名:《集群引擎MapReduce的中间数据存贮与传输优化的研究与实现》
    来源：学位论文 哈尔滨工业大学 2012 作者: 那钦
13. 相似度：1％ 篇名:《普适计算环境下异步事件交互技术研究及实现》
    来源：学位论文 国防科学技术大学 2007 作者: 李志凌
14. 相似度：1％ 篇名:《基于Node.js的JavaScript并发控制流框架》
    来源：学术期刊 《江汉大学学报（自然科学版）》 2015年2期 作者: 李轶

互联网相似资源列表：

1. 相似度：13％ 标题：《Redis 存储机制 - 任何技能都是从模仿开始，逐步升华。...》
   http://blog.csdn.net/zhu_tianwei/article/details/44892737
2. 相似度：9％ 标题：《小结Node.js中非阻塞IO和事件循环_node.js_脚本之家》
   http://www.jb51.net/article/55319.htm
3. 相似度：7％ 标题：《Node.js的事件驱动模型》
   http://www.edwardesire.com/2015/05/09/nodejs-event-model/
4. 相似度：6％ 标题：《IBM打造的最新物联网编程工具Node-Red基于Nodejs》
   http://www.zhongkerd.com/news/content-1565.html
5. 相似度：5％ 标题：《Redis 简介 - Redis 教程 - 自强学堂》
   http://www.ziqiangxuetang.com/redis/redis-intro.html
6. 相似度：3％ 标题：《Redis的使用 - 傲然青眼 - 博客园》
   http://www.cnblogs.com/dqqiu/p/usedOfRedis.html

全文简明报告：

2.1 node.js的事件驱动和非阻塞机制

{66％： 与nginx的服务原理类似，node.js采用事件驱动[14，15]的运行方式。} {54％： 众所周知， nginx采用的是多进程、单线程模型，但是node.js是通过事件驱动的方式，当它在处理客户请求的时候无需为每一个请求创建额外的线程，} 而是通过事件监听器来判断，触发callback函数的执行。 {92％： 在事件驱动的模型当中，每一个I/O工作都被添加到事件队列中，线程循环地处理队列上的工作任务，} 当在执行过程中遇到某一事件(如读取文件、查询数据库等操作)来阻塞任务时，线程不会停下来等待该事件， {83％： 而是留下一个处理结果的callback函数，转而继续执行队列中的下一个任务。} {77％： 这个传递到队列中的callback函数，只有在堵塞任务

运行结束后才被线程调用。} 图2-1为node.js的事件驱动原理图。 图2-1 node.js 的事件驱动原理图

从图2-1 node.js的事件驱动原理图可以看出，node.js开始启动的进程会创建一个事件循环队列。 {71%：循环没执行一次就是一个tick周期，在每个tick周期中，node.js会从事件循环队列中查看是否有事件需要处理，如果有就取出事件并执行相关的callback函数。} {74%：只有在事件队列事件所有都执行完毕，node.js应用就会终止。} node.js对堵塞I/O的处理[16]，其底层是通过线程池来确保工作的正常执行。 {69%：node.js从线程池中取得一个线程来执行复杂任务，而不必占用主循环线程，这样就防止堵塞I/O占用空闲资源而造成效率下降。} {55%：在堵塞任务执行完毕后，通过查找到事件队列中相应的callback函数来处理接下来为完成的工作。}

{66%：众所周知， I/O操作无疑是十分耗时的，当服务器端接收到大量的客户请求时，如果为每一个客户请求创建进程或线程，} {82%：这无疑会增加额外的内存开销，同时也可能浪费更多的时间资源。} {70%：正因为node.js是事件驱动的，于是它可以使用事件循环机制来解决I/O操作带来的性能和效率瓶颈问题。} {85%：在 node. js中，一个I/O操作通常会带有一个回调函数，当I/O操作完成并返回时，} {74%：就会调用这个callbacke函数，而主线程则继续执行接下来为完成的代码。} {62%：下面说明一下node.js异步非阻塞I/O的运行原理。}

在 Windows平台上， node. js是直接利用 Windows下的 IOCP（I/O Completion Port）通常称为I/O完成端口来实现的， 在IOCP的内部其实是利用了线程池的原理，这些线程是由 Windows系统内核自动管理， 不需要我们手动加以管理。 {52%：而在Linux平台上，node.js从v0.9.3版本以后，都是通过自行实现的线程池来完成异步非阻塞I/O的。} 由于这种跨平台的差异性，node.js通过构建一个平台层架构libuv，来屏蔽平台的差异性，所有平台的兼容性问题都是由这一层来完成。 图2-2 就是node.js的异步执行架构图。

图2-2 node.js的异步执行架构图

{55%：以上就异步非塞模型和事件驱动机制进行了总结。} {100%：而这个事件循环的机制并不仅仅是node. js所独有的，并且 node. js的代码是单线程执行的，} 在面对大量并发请求的时候， node. js是有着自己独特优势的，这种优势主要体现在 node. js的优秀的系统架构， 图2-3展示了 node. js的系统架构图。

图2-3 node.js的系统架构图

{83%：从该架构图中可以看出， node. js的底层有一个模块负责维护线程池，当一个I/O请求发出的时候，} {88%： node. js的底层线程创建模块将新建一个线程来处理请求，完成后再将结果交还给上层。} {100%：那么，当有多个请求的时候， node. js的底层模块将利用尽可能少的线程来完成最多的任务，} {100%：如果存在空闲的线程，它将继续被利用来做其他的事情，这对于前面说的针对每个请求开一个新的进程或线程而言，} 无疑“聪明”许多，也更加高效了。

2.2 Node-red可视化流式处理框架

2.2.1 Node-red的概述

{100%：Node-red是IBM公司开发的一个可视化的编程工具。} {100%：它允许程序员通过组合各部件来编写应用程序。} 这些部件可以是硬件设备(如： Arduino板子)、Web API(如： WebSocket in和WebSocket out)、功能函数(如： range)或者在线服务(如： twitter)。 {98%：Node-Red提供基于网页的编程环境，通过拖拽已定义node到工作区并用线连接node创建数据流来实现编程。} {100%：程序员通过点击‘Deploy’按钮实现一键保

存并执行。｝｛100％：程序以JSON字符串的格式保存，方便用户分享、修改。｝｛67％：Node-red基于node.js，它的执行模型和node.js一样，也是事件驱动非阻塞的，这一点在上一节关于node的事件驱动和非阻塞机制已经作了详尽的阐述。｝｛90％：理论上，node.js的所有模块都可以被封装成Node-red的一个或几个节点（node）。｝ 接下来，我们将详细地阐述如何搭建Node-red的环境，以及如何编写和管理自己的flow。

2.2.2 Node-red的编译与安装

由于本文要对Node-red的原始节点进行补充，增加新的节点使其适合流式数据的传输和计算。 因此我们在安装Node-red的时候选择了源码安装。 安装的平台选择了Ubuntu14.04长期支持版。 具体的安装流程如下：

依赖的安装：

由于Node-red是基于Node.js的，所以在安装Node-red前必须先安装Node.js。 这里推荐使用源码安装。

在github上获取Node.js的源码：

修改目录权限：

使用./configure创建编译文件，并按如下命令安装：

验证是否安装成功：

在按照上面三步安装之后，执行node -v命令之后，结果如图2-4 所示表示安装成功。

图2-4 node.js安装成功

Node-red的安装：

因为我们要对Node-red的原始节点进行补充，所以这里我们必须选择源码安装。 安装Node-red是利用npm（NodePackagedModule）来安装，新版的node.js里面已经集成了npm，不需要另外单独安装，通过执行npm -v命令就可查看npm是否安装成功。

在github上获取Node-red的源码：

安装Node-red：

安装grunt-cli：

为了能够利用Node-red顺利的创建应用，这里我们还需要安装grunt-cli，并且不需要安装成全局模块。

创建应用并运行Node-red：

验证Node-red是否运行成功：

按照上述步骤安装结束之后，如果执行node red命令之后，控制台出现如图2-5 所以表示安装成功：

图2-5 Node-red安装成功

2.2.3 Node-red的基本配置

安装好Node-red后，在Node-red的安装目录下有很多文件，从这些源码文件就可以看出Node-red的目录是十分清晰，各个模块的划分也是仅仅有条。 在这些文件当中，绝大部分文件用户都不需要关心，但是有几个重要文件需要尤其注意， 详细了解这些文件的作用和实用方法，对后面开发新节点有很重要的帮助，首先来了解一下Node- red的目录结构， 图2-6展示了 Node- red的目录结构。

图2-6 Node-red目录结构图

下面简单地介绍一下各个目录文件存储的内容和作用：

（1）在/public目录下是一些关于Node-red本身的静态文件，包括资源文件、css样式文件、以及前端页面的html文件；

（2）/node-modules目录下面是一些外部依赖库，也就是Node-red需要的一些Node.js模块。

（3）/red目录下面就是真正的Node-red代码，主要是一些核心api、事件驱动程序、服务器端主程序、系统设计程序以及Node-red的入口程序等。

（4）/test目录下面主要是放了一些用于测试的Node以及flow；

（5）/nodes目录是一个极其重要的目录，Node-red中所有的节点都是存放在这个目录下的，包括各个节点的html和js文件，本文中重新设计的数据节点也会放在这个目录下。

（6）settings.js文件是整个Node-red的系统配置文件，该文件描述了启动的参数细节、端口和ip设置以及各个启动目录的设置。

了解了各个目录文件的作用之后，接下来阐述如何配置Node-red。 Node-red几乎所有的配置信息都记录在setting.js文件中，首先要清楚各个配置选项的功能作用，表2-1 展示了Node-red的常用配置选项极其作用。

表2-1 Node-red常用配置选项说明

选项名默认值作用

uiPort1880指定Node-red网页的端口号

uiHost127.0.0.1指定Node-red网页的ip地址

debugMaxLength1000指定debug节点调试数据的最大显示长度

flowFilePrettytrue是否保存编写的flow

userDir安装目录指定flow保存的位置

functionGlobalContextundefined用于加载外部依赖，其值是json对象

.....

安装好Node-red后，按照表2-1 所示的配置选项说明进行配置，然后重新启动Node-red，就可以在浏览器中输入http: //127.0.0.1: {68 %： 1880，即可打开Node-red的可视化流程编辑界面。 }

2.4 基于内存计算的数据库Redis

2.4.1 Redis数据库的概述

Redis是一个key-value存储系统。 {87 %： 和memcached类似，但Redis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(有序集合)以及hash（哈希类型）。} {100 %： 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。 {87 %： 与memcached一样，为了保证效率，数据都是缓存在内存中[18]。} {88 %： 不同的是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同步，也就是主从同步。}

{80 %： 与其他key-value存储系统相比，Redis有着更为复杂的数据结构并且提供对它们的原子性操作，这是一个不同于其他数据库的一大重要突破。} {100 %： Redis的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。} {61 %： Redis运行在内存中，但是也可以持久化到本地磁盘中，所以在许多大数据应用场景下，当需要对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。} {100 %： 在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。} {98 %： 同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为它们并不需要进行随机访问。}

2.4.2 Redis数据库的储存原理

Redis存储机制分成两种Snapshot 和 AOF。 {100 %： 无论是那种机制，Redis都是将数据存储在内存中。}

Snapshot工作原理: {99 %： 是将数据先存储在内存，然后当数据累计达到某些设定的阈值的时候，就会触发一次DUMP操作，将变化的数据一次性写入数据文件（RDB文件）。}

AOF 工作原理: {100 %： 是将数据也是先存在内存，但是在存储的时候会使用调用fsync来完成对本次写操作的日志记录，这个日志揭露文件其实是一个基于Redis网络交互协议的文本文件。} {100 %： AOF调用fsync也不是说全部都是无阻塞的，在某些系统上可能出现fsync阻塞进程的情况，对于这种情况可以通过配置修改，但默认情况不要修改。} {79 %： AOF最关键的配置就是关于调用fsync追加日志文件的平率，有两种预设频率，每次记录进来都添加，每秒添加一次。}

{64 %： 这两种存储策略各有优点，从性能上讲， Snapshot方式的性能明显要高于 AOF方式，因为前者是采用

2进制方式存储数据，｝｛98％：数据文件比较小，加载快速，存储的时候是按照配置中的save策略来存储，每次都是聚合很多数据批量存储，｝｛94％：写入的效率很高，而AOF则一般都是工作在实时存储或者准实时模式下。｝ 相对来说存储的频率高，效率却偏低。 ｛84％：但是从数据安全性上来讲， AOF数据安全性高于Snapshot存储，因为Snapshot存储是基于累计批量的思想，也就是说在允许的情况下，｝｛100％：累计的数据越多那么写入效率也就越高，但数据的累计是靠时间的积累完成的，那么如果在长时间数据不写入RDB，｝｛100％：但Redis又遇到了崩溃，那么没有写入的数据就无法恢复了，但是AOF方式偏偏相反，根据AOF配置的存储频率的策略可以做到最少的数据丢失和较高的数据恢复能力。｝

2.4.3 Redis数据库的pub与sub机制

pub/sub功能即publish/subscribe功能[21，22]。 ｛99％：在基于事件的系统中，pub/sub是目前广泛使用的通信模型，它采用事件作为基本的通信机制，提供大规模系统所要求的松散耦合的交互模式：｝｛100％：订阅者比如客户端以事件订阅的方式表达出它有兴趣接收的一个事件或一类事件，发布者比如服务器可以将订阅者兴趣的事件随时通知相关订阅者。｝

Redis数据库也支持pub/ sub机制，本论文所设计的流式数据处理模型中，新引入了数据的输入节点也就是redis的subscribe节点， ｛42％：以及数据的输出节点publish节点，将这两个节点在采集数据的时候用。｝｛43％：订阅者可以订阅多个Channal[23]，而发布者可以通过Channel，向订阅者发送消息。｝｛44％：但是发布者所发的消息是异步的，不需要等待订阅者订阅，也不关心订阅者是否订阅，｝ 简单地说就是订阅者只能收到发布者后续所发送到该Channel上的消息，如果之前发送的消息没有接收， 那么也再也接收不到了，下面是Redis数据库的发布订阅命令。

PUBLISH： 向channel_test发布消息message。

SUBSCRIBE： 订阅channel_test消息，会收到发布者所发送的messag消息。

2.5 本章小结

｛40％：本章主要是对本论文的相关技术进行了介绍，首先对node.js的事件驱动和非阻塞机制进行了详细阐述，｝ 主要是为后面Node-red的节点设计打下理论基础，然后再是对Node-red进行了详细介绍， 最后还详细介绍了Redis数据库的基本概念、存储原理以及发布/订阅机制。