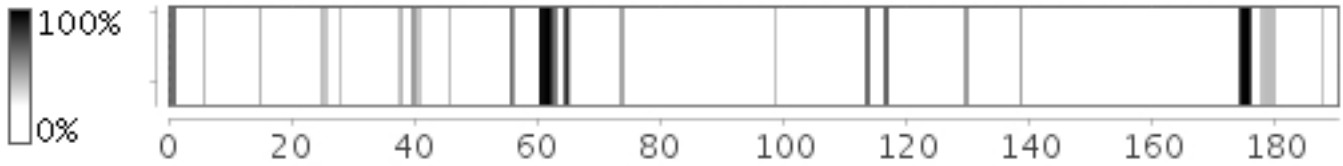
PaperPass专业版检测报告
简明打印版

比对结果（相似度）：
总　体：10 % *（总体相似度是指本地库、互联网的综合比对结果）*
本地库：7 % *（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：4 % *（期刊库相似度是指论文与学术期刊库的比对结果）*
学位库：5 % *（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：5 % *（互联网相似度是指论文与互联网资源的比对结果）*

编　号：58C0016410D28Q6SF
版　本：专业版
标　题：第二章
作　者：王江波
长　度：7877 字符(不计空格)
句子数：190句
时　间：2017-3-8 21:04:36
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:《基于非关系型数据库的大规模天文星表数据存储研究》**
   来源：学位论文 天津大学 2014 作者: 李连盟
3. **相似度：1 % 篇名:《基于Node.js高并发web系统的研究与应用》**
   来源：学位论文 电子科技大学 2014 作者: 陈瑶
4. **相似度：1 % 篇名:《百度视频泛需求检索数据处理子系统的设计与实现》**
   来源：学位论文 北京交通大学 2014 作者: 江涛
5. **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

6. 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

互联网相似资源列表：

1. 相似度：4％ 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
2. 相似度：3％ 标题：《Node-RED - 简书》
http://www.jianshu.com/p/e27ee86f22ea
3. 相似度：2％ 标题：《辛星浅析Redis中的pub/sub功能-布布扣-bubuko.com》
http://www.bubuko.com/infodetail-641726.html

全文简明报告：

{69％： 第二章 实时流数据处理的基础理论和技术}

2.1 node.js的事件驱动和非阻塞机制

node.js从2009年诞生至今，已近经过了八年的发展，目前node.js已经进入了青年时期，在各大中小型IT企业中的应用的十分广泛，尤其在web领域，不论是前端JS还是后端的web服务器，它都有用武之地。 {45％：node.js不仅仅是一种编程语言，更是一种工具和平台，为JavaScript提供运行环境。} 它封装了google的V8引擎，由于V8引擎解释执行JavaScript的速度快，效率高等特点，再加上node.js本身对其进行了优化，这使得node.js的性能也非常好。 而底层的代码执行模块使利用C++编写的，同时底层通过libuv库来实现了对事件循环队列的处理，并将耗时较长的I/O请求交给liveio来处理，以此来提高运行效率。 node.js的优秀性能主要体现在其优秀的系统架构上，图2-1所展示的就是node.js的架构图。

图2-1 node.js的系统架构图

Node.js底层的事件循环机制是利用libuv来实现的，libuv是一种高性能的事件驱动程序库，它屏蔽了因为平台不同而带来的差异。 在Windows平台中，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。 {45％：而在Linux平台上，node.js都是通过自行实现的线程池来完成异步非阻塞I/O的。} 而libuv就是起这样一个平台间的过渡角色，对外提供统一的API接口，图2-2所展示的就是事件驱动策略。

图2-2 node.js的事件驱动策略

Node.js采用的是事件驱动，异步编程的模式。 事件驱动这个词，对于程序员来说并不陌生，比如在网络套接字编程中，当socket有数据到来的时候，就会触发我们之前所注册的callback函数的执行，而node.js所提供的绝大多数API都是采用的这种编程模式。 下面就来详细阐述一下node.js的这种事件驱动编程模式。

我们可以与apache服务器的原理相比较一下，apache服务器采用的是单进程、多线程模型，一个用户请求对应一个线程，而node.js是单进程、单线程模型，它是通过事件驱动的方式来实现并发的，不会为每一个客户请求创建单独的线程，而是通过事件监听器来判断，最后触发callback函数的执行。 {43％：当node.js的主线程运行

的时候，就会创建一个事件队列（event queue），｝｛40％：在这个队列中几乎保存了程序所需要的每一个I/O操作，由于线程会循环地去处理事件队列中的I/O操作，｝该队列也被称为循环队列。｛42％：如果在程序的执行过程中，遇到了比如像文件的读写、数据库的查询等I/O操作来阻塞任务时，｝线程不会停下来等待这些操作，而是注册一个callback函数，转而继续执行队列中的下一个操作。而这里的callback函数，只有在这些阻塞任务执行结束之后通知主线程调用执行。在事件循环队列中，为了避免造成类似于递归调用的无限循环中，要求所有的callback函数都必须经过一个tick周期，在程序中的具体表现就是所有的callback函数都要执行process.nextTick()。图2-3所展示的就是node.js的事件驱动原理图。

图2-3 node.js的事件驱动原理图

从图2-3 node.js的事件驱动原理图可以看出，在node.js的应用中有读文件和查询数据看两种I/O操作，该应用的主线程会创建一个事件循环队列，在这个循环队列中有文件的打开操作、读文件、连接数据库、查询数据库等操作。举一个实际的例子来说明，假如node.js在执行下面这样一段代码：

程序的第二个参数就是一个回调函数，当程序运行到这里的时候，由于I/O操作会消耗大量时间而不会立刻返回查询的结果，｛46％：而是将该事件插入事件队列中，转而继续执行下面的代码。｝而当数据库查询操作返回后，就会将该事件发送到事件循环队列中，直到下一次循环监听到了该事件，就会触发回调函数的执行。｛53％：而只有当整个事件循环队列中的任务都执行结束后，node.js应用才会终止。｝

｛41％：对于node.js的异步I/O非阻塞机制也是建立在事件驱动机制之上的，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。｝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。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。也就是说，对于那些相对耗时比较长的I/O操作，比如读写文件等，还有一些网络通信，比如套接字，node.js会将这些操作交给一个称之为worker threads的线程池去执行，当这些操作执行结束后，｛43％：通过事件通知，并执行回调函数，这就是异步I/O非阻塞机制。｝

正因为node.js的这种事件驱动机制，使得那些十分耗时的I/O操作都可以异步执行，有效地解决了因为I/O操作而带来的性能和效率瓶颈问题。在许多轻量级、高实时、高流量的应用系统中，都能见到node.js的身影。本文中各个数据处理节点的设计和开发都是基于node.js的，同时前端可视化模块也是利用node.js的express框架进行开发的。

2.2 Node-red可视化流式处理框架

2.2.1 Node-red的概述

Node-red是IBM Emerging Technologies团队开发的一个可视化的数据流程编辑工具。程序员可以直接通过web浏览器就可以实现各种数据流程的编辑，同时可以实现对数据处理逻辑的编写，Node-red把这些数据流程称为一个flow，所编写的flow可以以json对象的形式保存为普通文件或者形成js库，方便用户分享、修改。｛68％：程序员在Node-red中可以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)。｛97％：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。｝｛68％：Node-red本身是基于node.js开发的，它的执行模型和node.js一样，也是通过事件驱动和非阻塞I/O机制来实现的，｝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)。}

本文所设计的实时流数据处理模型是通过利用Node-red来完成数据流程的管理以及处理数据的业务代码的编写两项工作。 接下来，详细阐述Node-red的编程模型以及它是如何管理数据流程。

2.2.2 Node-red的编程模型

本节我们通过介绍Node-red的一些关键概念和关键组件，并通过实际例子说明Node-red的编程模型。

（1）数据流程（flow），这是Node-red中最重要的一个概念，一个flow就是一个Node-red程序，它是多个节点连接在一起进行数据交换的集合。 在Node-red的底层，一个flow通常是由一系列的JavaScript对象和各个节点的配置信息组成，通过调用底层的node.js环境来执行JavaScript代码。

（2）节点（node），它是组成flow的最基本的元素，也是真正进行数据处理的载体。 当一个编写好的flow运行起来的时候，节点主要对从上游节点接收到的消息（简称message）进行处理，并产生新的消息传递给下游节点完成后续的处理工作。 {50％：一个Node-red的节点主要包括js文件这html文件，分别完成对节点功能的具体实现和节点UI设计。}

（3）消息（message），它是节点之间进行传输的对象，也是数据的真正载体。 本质上消息是一个JavaScript对象，包含了各种对数据描述的属性。 消息是Node-red处理的最基本的数据结构，只有在节点被激活时消息才会被处理，再加上节点是相互独立的，这就保证了所有的数据流式互不影响且无状态。

（4）连线（wire），它是节点与节点之间的连接桥梁，它们通常将节点的输出端点连接到下游节点的输入端，表示由一个节点生成的消息应该由下一个连接节点处理。

在了解了这些基本的Node-red组件之后，下面通过举例说明Node-red的编程模型。 假设要实时发送一个消息到 debug节点，来测试消息在节点之间的传输，用到了一个定时器 timestamp节点， 一个函数节点 function_ node以及一个 debug节点，如图2-4所示。

图2-4 Node-red中的数据流程图

在图2-4所展示的 flow中 timestamp节点每隔两秒去触发 test function节点，执行其中的代码， 而 msg. payload是一个 debug节点，用于在 debug面板展示 test function处理过的数据。 这里仅仅是为了说明Node-red的编程模型，因此test function节点并没有复杂的数据处理逻辑，仅仅是返回一个hello world的消息，其实现代码如下：

在function节点的内部可以编写任何JavaScript函数，用于处理上游节点发送过来的数据。

2.2.3 Node-red的基本配置

由于本文所设计的实时流数据处理模型，要对 Node- red的数据输入节点，输出节点以及数据处理节点进行重新设计， 同时新增 Redis数据的访问节点，因此，需要对 Node- red进行源码安装。 为了能够有效地利用Node-red进行流式数据处理和数据流程的管理，有必要阐述一下Node-red的基本配置。 经过源码安装后，Node-red的目录是十分清晰，各个模块的划分也是仅仅有条。 首先来了解一下Node-red的目录结构，图2-5展示了Node-red的目录结构。

图2-5 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；

{42%：（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。}

（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。

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

.....

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

{70%：Redis是一个开源的高性能key-value存储系统，它通过提供多种键值数据类型来适应不同应用场景下的存储需求，并借助许多高级的接口使其可以胜任如缓存、队列系统等不同的角色。}

Redis和 Memcached类似，数据都是缓存在内存的，而不同之处主要有三点，第一点就是 Redis之处存储的数据类型相对来说更加丰富， 比如像 string(字符串)、 list(链表)、 set(集合)、 zset(有序集合)以及 hash（哈希类型）。第二点 Redis还提供了数据持久化的功能，因为在内存中的数据有一个典型的问题，也就是当程序运行结束后，内存上的数据将会丢失，所以 Redis考虑到这点，提供了对数据持久化的支持，即将内存中的数据通过异步的方式写入到磁盘当中， 同时也不影响继续提供服务。 第三点就是在实现上， Memcached采用的是多线程技术，而 Redis采用的是单线程技术， 所以在多核处理器上， Memcached的性能和资源利用率上要高于 Redis，但是针对这一点目前也有很好的解决方案， 再加上 Redis的性能也已经足够优秀了，而且提供了许多 Memcached无法提供的高级功能，我们相信在不久的将来， Redis将会在很多领域完全替代 memcached。 表2-2 给出了Redis与Memcached的对比。

表2-2 Redis与Memcached的对比

RedisMemcached

{53%：数据库类型Key-value内存数据库Key-value内存数据库}

数据类型在定义value的时候

就要固定数据类型不需要固定，支持字符串，链表，集合，有序集合，hash

虚拟存储不支持支持

过期策略支持支持

分布式MagentMaster-slave主从同步

数据存储安全不支持备份到dump.rdb中

灾难恢复不支持AOF用于数据容灾

2.4.2 Redis数据库的实现原理

{51%：我们已经知道Redis提供了五种数据类型，分别是字符串、链表、集合、有序集合以及哈希表。} 开发Redis数据库的作者，为了让Redis支持者五种数据结构，首先对Redis的内存模型进行了设计，如图2-7 所示。

图2-7 Redis内存对象模型

从图2-7所展示的内存模型可以看出， Redis是通过一个叫做 redisObject核心对象来管理这些数据类型的， 在redisObject对象内部提供了一个 tpye字段，用于表示 value到底是属于那种数据类型， 而真正的值是通过一个数据指针来表示的。 这里的编码方式也就是表明了该类型的数据在Redis底层是使用的什么数据结构，比如在列表的底层是通过双端链表实现的，而有序集合是通过Skip List实现的。 另外，模型中还提供了一个虚拟内存字段，而只有在该字段的值为true的时候，才会真正地分配内存，在默认的情况下该字段值为false。

在认识了Redis的核心对象后，接下来简单阐述在redis数据库中这五种数据类型的底层实现原理。 Redis数据库底层提供了八种数据结构，在源码中都是通过宏来表示的，也就是之前提到的编码方式，如表2-3 所示。

表2-3 Redis底层提供的八种数据结构

编码常量对应的底层数据结构

REDIS_ENCODING_INTLong 类型的整数

REDIS_ENCODING_EMBSTREmbstr编码的动态字符串

REDIS_ENCODING_RAW简单的动态字符串

REDIS_ENCODING_HT字典

REDIS_ENCODING_LINKEDLIST双端链表

REDIS_ENCODING_ZIPLIST压缩列表

REDIS_ENCODING_INTSET整数集合

REDIS_ENCODING_SKIPLIST跳表和字典

对于字符串来说，其编码方式可以是long类型的整数，也可以是embstr编码的动态字符串，还可以是简单的动态字符串。 embstr编码的动态字符串是在redis3.0中新增的数据结构，它的好处在于只需要进行一次内存分配，而简单的动态字符串需要进行两次内存分配。 对字符串value的操作都是通过核心对象的数据指针进行的。

Redis中列表的底层数据结构可以利用双向链表实现，也可以利用压缩列表来实现，由于双向链表中的每个节点都具有直接前驱和直接后继， 因此 Redis的列表支持许多反向操作，但是有一个不足之处就是，每增加一个节点都会向系统申请一次内存， 这无疑带来了额外的内存开销。 但是对于压缩链表来说，它就要比双向链表更加节省空间，因为压缩链表只需要申请一次内存， 而且是一块连续的内存空间，但是为了保证内存的连续性，每次插入一个节点的时候都需要 realloc一次。

集合的内部实现是一个hashtable，只是其中的value永远是null，这实际上是通过计算hash的方式来实现快速重排，这也是集合之所以能够快速地判断一个元素是否在集合中的重要原因。 Redis中hash类型的底层数据结构可以是hashtable，也可以是压缩的列表，由于压缩列表是一段连续的内存空间，所以在压缩列表中的哈希对象是按照key1: value1，key2: value2这样的先后顺序来存放的，按这种方式实现的hash，在对象的数量不多且内容不大的情况下

，效率是非常高的。

Redis的有序集合的底层数据结构就是通过跳表来实现的，而没有采用hash和hashtable来实现，虽然hash可以实现快速的查找，但是无法保证有序。 关于有序集合的底层实现原理我们将在第三章有序集合的源码分析中做详细阐述。

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message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、底层实现原理以及发布/订阅机制。