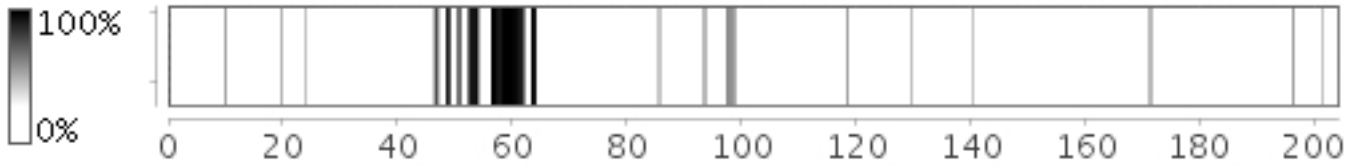
PaperPass专业版检测报告
简明打印版

比对结果（相似度）：
总　体：9 %　*（总体相似度是指本地库、互联网的综合比对结果）*
本地库：3 %　*（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：2 %　*（期刊库相似度是指论文与学术期刊库的比对结果）*
学位库：2 %　*（学位库相似度是指论文与学位论文库的比对结果）*
会议库： 0 %　*（会议库相似度是指论文与会议论文库的比对结果）*
互联网：7 %　*（互联网相似度是指论文与互联网资源的比对结果）*

编　号：58B90952565D34CSJ
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：9389 字符(不计空格)
句子数：204句
时　间：2017-3-3 14:12:34
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：1 % 篇名:《基于Java的数据库连接池技术在即时通信系统中的应用》
   来源：学术期刊 《电脑开发与应用》 2008年2期 作者: 褚媛 周家纪 卞腾 李刚
2. 相似度：1 % 篇名:《数据库连接池技术研究与应用》
   来源：学术期刊 《现代电子技术》 2007年5期 作者: 商杰 朱战立

互联网相似资源列表：
1. 相似度：8 % 标题：《Node-Red 开发套件_创客大爆炸　MakerCollider》
   http://www.makercollider.com/kit/detail?id=23

全文简明报告：

第三章 基于Node-red与Redis的实时流数据处理模型的设计

3.1 需求分析

本论文主要是研究和设计一种基于Node-red与Redis的实时流数据处理模型，应用场景为实际项目中的网站群的实时访问监控。 在本项目中旨在实时了解用户访问网站群的行为，捕捉用户请求并跟踪其所有响应，收集、处理并显示用户行为的细节数据，并可视化展示数据和挖掘数据背后的信息。 针对该流式计算模型在实际应用场景下提出如下的需求。

（1）高实时性； 在许多实时流数据处理的应用场景中，不论是数据的采集，还是数据的处理，都要求具有高实时性。 高实时性，要求模型在进行数据采集的时候满足不低于每秒钟50笔的采集速度，以免造成数据堆积，同时也要求具备高效的数据计算和处理能力。

（2）高性能； {51%：随着业务的不断扩展，数据量也不断的增大，对系统的性能要求也越来越严格。} 因此，从数据采集到数据处理再到数据可视化展示，各个环节都要求系统具有良好的性能。 最直观的表现就是在用户看到的可视化模块的数据更新延迟不能超过2秒钟。

（3）高可用； 系统可以通过集群等方式实现分布式部署，避免单点故障。

（4）可扩展； 数据量、计算量会随着业务的不断扩展而不断增大，这就要求模型需要有良好的扩展性。

（5）分布式； 为了提高数据的处理能力和计算效率，模型还需要具备分布式的处理能力；

（6）安全性； {46%：数据安全是任何系统的一个首要前提，流式数据处理模型也必须要保证数据的安全性。}

本论文在这些需求的基础之上，提出一种新的实时流数据处理模型，要在Node-red上设计出高效的数据接入和输出节点，同时也要有高效的数据处理节点。 结合redis的内存计算的优势，设计出redis数据库访问节点，用于统计中间结果集，以调高统计计算的效率。 同时，充分利用Redis的pub/sub机制来实现数据的流式异步传输。 {42%：最终将这套模型应用到实际系统中去加以验证。}

3.2 模型的总体架构

基于Node-red与Redis的实时流式数据处理模型的设计是搭建在Ubuntu环境下的，也可以部署在分布式环境上以提高流式数据的处理能力和计算效率。 该模型通过重新设计数据输入、输出以及数据计算节点，以完成对实时流式数据的处理。 整个模型的架构如图3-1 所示：

图3-1 流数据处理模型架构图

从该模型的架构图中可以看出，Redis数据库充当了数据交换的中心，而整个数据流的处理逻辑都交给计算节点群去完成。 数据首先通过 Redis的 channel（通道）进入 Redis server，然后 Node- red利用 redisSub节点去订阅相应通道（channel）的数据交给计算节点（function nodes）集群进行数据计算， 而计算节点集群所产生的中间结果

集，通过 redis_ in节点传给 redis server进行统计，最后产生的最终计算结果通过 redisPub节点发布到前端可视化模块中。

在原始的Node-red中是没有任何节点可以与Redis进行交互，为此，新增加了redisSub、redisPub、redis_in和redis_out节点。 为了，用户可以自定义数据的处理逻辑，引入了函数节点，多个函数节点构成了整个流式计算的计算节点群。 有了这些节点，就可以方便快捷地在Node-red上编写流式数据处理的业务代码，更为重要的是，这些业务代码可以实现一次编写多次使用，方便移植和维护。

3.3 节点处理模块的设计

节点是Node-red的重要组成元素，所有的flow都是通过一个一个的节点组成的，在Node-red中有三类基本的节点，数据输入节点、输出节点以及数据处理节点。 为了设计出适合流式数据处理的节点，这里必须对这三类节点进行补充设计， 在这一节中主要是对整个流式数据处理模型所需要的节点给出详细的设计方案。 Node-red的节点本身主要包括两份文件： js文件和html文件，js文件主要定义了组件具体做些什么事情，有什么样的功能；html文件主要定义了组件的属性，组件编辑框格式和帮助信息等，图3-2 为一个Node的设计方案：

图3-2 Node-red的节点设计方案图

将设计好的新节点重新安装部署到Node-red中，就可以在Node-red的前端编辑界面使用该节点进行数据处理。Node-red强大的扩展能力就是体现在用户可以设计Node-red没有提供的节点，来完成特定的任务。 由于Node-red本身在定义节点的时候有自己的要求和原则，所以，为了保证节点设计的正确性和有效性，节点设计的必须按照如下原则来进行：

{78％： （1）要求创建的节点要对各种数据类型的输入数据进行必要的处理，即使某些类型并不是这个节点所需要的。} 这样做有两个目的，一是为了便于对原始数据进行追加额外说明信息，二是为了便于节点的扩展。

{91％： （2）由于Node-Red在识别和处理节点的时候使用了大量的字符串匹配操作，所以在节点的定义中有一些名字的字符串是必须保持一致的，否则Node-Red在解析的时候就会出错。}

（3）.html文件分为3部分： {66％：节点的定义，节点的编辑模板和节点的帮助信息。} 节点的定义主要用于： {94％：确定节点的类型，可编辑的属性，在浏览器中显示的样式，是一段可执行的js代码，RED.nodes.registerType；} {95％：编辑模板主要是生成用户编辑该节点的实例时的界面(由data-template-name包括的一段HTML代码)，用户的输入最终会保存在node的定义中；}

（4）在.html文件中，data-template-name、node-input-xx、data-help-name都是Node-Red系统保留字。 data-template-name、data-help-name的取值必须和文件名字的name部分一致。 {100％： RED.nodes.registerType的第一个参数也必须和文件名字的name部分一致。}

{94％： （5）每个节点的可编辑的域在defaults中声明，data-template-name所包含的node-input-xx负责生成输入框。} {100％：defaults的每个域的名字必须和node-input-xx中的名字保持一致。} {100％：在.js文件中使用可编辑域的值的时候，直接访问defaults的域就可以，不必添加defaults前缀。}

{98％： （6）在.js文件中，RED.nodes.registerType用来注册一个node实例的生成函数，它的第一个参数必须和文件名字的name部分一致。} {90％：传给生成函数的参数是node可编辑域的值(已编辑完成)及节点共享域的值。

}

（7）input的callback是节点输入的处理函数。 {98%：需要注意的是，Node-Red节点之间数据传输使用的是名字为payload的域，这个也是Node-Red系统保留的。}

3.3.1 数据输入节点的设计

数据的输入节点（input node），主要是用于从外部设备或者其他外部接口获取数据到Node-red中进行数据分析。 在Node-red的一个flow中，输入节点是所有message的入口，为下一个Node产生新的message。 由于Node-red自带的输入节点很有限，而且不适合流式数据的输入，所以在这里必须补充设计数据的输入节点。

为了满足流式数据的输入需求，数据的输入节点的设计必须要满足一下几个原则：

（1）流式化数据，为了让成批到达的数据也能够在这样一个模型中得到计算，我们在设计数据输入节点的时候就要考虑到这点，也就是说让批量到达的数据逐条进入Node-red的flow。

（2）统一的数据格式，在一个数据处理模型中，数据格式的好与坏意味着后序进行数据计算的简与繁。

（3）高吞吐量，由于流式数据的产生是源源不断的，所以在设计输入节点的时候要充分考虑节点的数据吞吐量问题， 不然会造成大量数据的堆积，从而影响后续的数据分析与计算。

（4）高稳定性，输入节点是数据的入口，稳定性是必须考虑的一个因素。

（5）可移植性，为了能够将自己设计的数据输入节点共享给其他用户，节点的可移植性也十分重要。

为了设计出高效的适合流式数据传输的输入节点，考虑到流式数据的特点，结合redis数据库的sub机制，可以为Node-red新增一个redisSub节点。 从上一小节的总体架构图中我们可以看出，我们尽量让所有的数据通过 redis的发布订阅机制来进行收集， 把采集到的数据按类别放到不同的 redis通道（channel）中，然后在 Node- red中通过我们新增加的 redisSub节点去订阅相应 channel的数据， 这样就可以把数据引入 Node- red中，完成了数据的接入工作。 同样redisSub节点也包括两个文件，一个是编写具体功能的实现代码的文件js文件，另一个是用于界面设计和帮助文档描述的html文件。 由于Node-red原始节点的存在，所以在进行文件命名标号的时候从52号开始，因为文件名编号和节点的ID是紧密相关的，所以节点的标号必须唯一。 设计好新的节点后需要重新安装部署新节点到Node- red中，在利用 npm安装的时候， Node- red的节点注册模块会去检测 setting. js配置文件，依次加载配置文件中的其他外部模块。 图3-3 是整个redisSub节点的设计图。

图3-3 redisSub节点设计图

{41%：对于redisSub节点的界面ui设计，需要考虑有哪些信息需要用户输入该节点。} 因为每个节点都有自己的名字，所以首先需要的一个信息就是用户为该节点取一个名字，需要用户输入Name字段。 由于数据是存放在redis server上的，所以还需要redisSub节点的描述redis server的ip地址和端口号。 当redisSub节点连接上redis server后，不知道数据是位于redis的哪一个channel上，因此还必须给出通道名称，这些都是redisSub节点所需要的最基本的信息。 另外还有就是redisSub节点的帮助信息也必须给出一定的说明。 ui界面主要是定义在52_redisSub.html文件中。

而对于redisSub节点的具体功能，是在52_redisSub.js文件中实现的。 首先，要调用Node-red提供的节点创建函数createNode()创建一个节点，并把配置信息告诉节点。 {46%：节点接收到这些信息后，创建一个数据库连接池函数redisConnectionPool，将redis server的ip和port，和createNode函数内部所产生的uuid传递给连接池函数。} 数据库连接池函数主要是通过一个 connections数组的_ nodeCount来记录有多少 redisSub节点连接 redis server，当有一个新节点连接 redis时， 该值就会加一，同样当有一个节点断开了解的时候就会减一。 当有close请求到的时候首先要判断_nodeCount的值是否为0，来决定是否删除connections对象数组。 {59%：关于redis数据库连接池函数的执行流程如图3-4 所示：}

{55%：图3-4 Redis数据库连接池函数执行流程}

有了数据库连接池函数，就可以来实现redisSub的功能了。 redisSub节点的前端页面将用户输入的redis server的信息保存起来，然后通过参数传给连接池函数连接redis server。 连接数据库后调用 client. subscribe()方法去订阅指定的通道，如果订阅成功， 就让 client去监听一个 message事件，看通道是否有数据发送过来，如果有数据就封装在 msg. payload中， 让 node的 send()方法发送出来，供下游节点接收。 与此同时，client还要去监听redis的close事件，当redisSub节点断开与redis server的连接的时候，就要调用redisConnectionPool.close()方法去断开连接。

下面就是redisSub的功能函数的伪代码：

3.3.2 数据输出节点的设计

数据进入Node-red后，经过各个计算节点的数据计算、封装等工作，然后打包成系统规定的数据格式后，需要从Node-red中输出，进入后续的数据可视化展示。 数据的输出就用到了Node-red的输出节点。 Node-red的输出节点允许把数据输出到Node-red的flow以外的其他服务和应用上去，对内有一个数据输入的左断点，对外暴露一个公共接口。

在Node-red中有一个常用的输出节点就是debug节点，这个节点是在编写flow的时候调试的时候用的，主要显示打印出数据经过上一节点处理之后的具体信息。 Debug节点是一个具有开关的节点，允许程序员手动开启或者禁用该节点。 debug节点的使用也非常简单，只需要在 Node- red左侧的节点栏中找到该节点然后拖拽到相应节点的后面， 并用线连接起来就可以实现数据的传输，最后开启 debug的启动按钮，部署了所编写的 flow后， 就可以在 Node- red的最右侧的 debug面板中看到打印出来的具体数据。 值得注意的是， debug节点的只有一个数据的入口，而没有数据的输出端，在设计 debug的时候， 重新封装了 sendDebug()函数，用来发送消息，将消息直接发送到 Node- red的网页编辑器 debug视图上直接显示， 而不是交由下游节点做数据处理。 {50%：下面给出debug节点的设计逻辑的部分伪代码。}

有了debug节点，可以方便用户在编写自己的flow的时候，及时查看数据的处理情况。 本文在第四章中应用该模型来解决实际问题的时候，将大量应用到debug节点。

为了保证数据的实时地输出到Node-red的flow以外的其他服务和应用上，这里我们新引入了redisPub节点。 顾名思义， redisPub节点就是将 redis的 publis功能嵌入到 Node- red中，通过设计一个新的节点来将经过 Node- red处理和计算过的数据输出来， 这里之所以选择 redis的 publis发布数据，一方面保证了数据的异步传输，另一方面也保证了数据的隔离（原因是各个 redis的通道数据是相互隔离的， 互补干预）。 在坚持节点的设计原则的前提下，图3-5 给出了redisPub的设计方案。

图3-5 redisPub节点设计图

结合上一小节数据输入节点的设计可知， redisPub节点和 redisSub节点的设计恰好相反， redisPub节点只具有一个数据的输入接口， 也就是只有数据的输入端点，这一端是连接上一个数据处理节点的，在 redisPub节点中也必须定位 redis的位置， {43%：也就是 redis服务器的 ip，端口号，不管是在 redis集群还是在单点的 redis服务器中都必须要指定，} 同时还要指定数据输出到哪个 redis的 channel中。 所以redisPub节点的ui设计与redisSub节点的ui设计十分相识，不同的是他们的功能代码不一样，体现在js文件中。 图3-6 展示了redisPub节点的设计逻辑的具体流。

图3-6 RedisPub设计逻辑流程图

同样，在实现redisPub节点的时候，也用到了数据库连接池函数，关于这个函数的设计思想在上一小节redisSub的设计中已经做了详细阐述。 从上面流程图可以看出，当 redisPub节点成功连接 redis后，将去监听 input事件，当有数据输入该节点后， 就会调用 this. client. publish()发布函数，将封装好的数据（message对象）发布到指定的 channel上。

3.3.3 数据计算节点的设计

数据计算节点在Node-red中起着举足轻重的作用，几乎所有的flow中都会用到数据计算节点。 数据计算节点允许用户编写JavaScript函数来处理进入Node-red中的数据，编写自己的业务代码，将定义好的数据类型转化为在Node-red中流动的message对象。 {41%：在Node-red中的message实际上就是一个JavaScript对象，message对象至少要包含payload属性，用来保存具体的数据。} 就像下面这样一个最基本的Node-red的massage数据格式：

计算节点接收到message后，主要处理的也是payload字段中保存的信息，处理后的数据也会封装成一个message对象传到下一个节点。 然而，message对象不仅只具有payload字段，还可以扩展出更多的其他字段来补充说明message对象的属性。 比如下面这个message对象：

计算节点通常包含一个数据输入端点和一个或多个数据输出端点，在 Node- red中提供了部分具有特殊功能的数据处理节点， 比如 change_ node，可以用来增加或者删除 message的字段，再如 switch_ node，可以用来做开关节点使用， 它是通过判断 message对象的某一字段是否存在或者真假来决定最后输出什么样的 message对象。 为了能够进一步扩展Node-red的功能，方便利用JavaScript函数加载外部的js模块，这里引入function_node，也就是函数节点。 可以说function_node在Node-red中就像一把瑞士军刀，可以使用户不必依赖于现有的数量有限的几个节点来处理数据。 顾名思义，函数节点其实就是暴露出来的一个JavaScript函数，用户可用通过编写一个JavaScript函数来处理从上游节点流下来的message，并返回处理后的一个或多个massage。 函数节点是用来做数据处理和数据格式化的利器，引入函数节点使得Node-red的对流式数据进行处理变得简单容易。 图3-7 是function_node的设计图：

图3-7 Function_node的设计图

用户可以通过function_node内置的编辑器sandBox，编写用户自己的JavaScript函数来处理message。 在 function_node内编写的 JavaScript函数内部是调用本机上的 JavaScript运行环境来解释执行的，同时在函数节点中可以去调用外部的 js模块， 但是这首先会去配置文件 setting. js文件中找到要包含的模块。 所以function_node在执行每一个函数的时候首先会去检查这个配置文件，在这个文件中去查找全局的函数模块。 在setting.js中，通过functionGlobalContext支持全局模块：

对于自己编写的JavaScript函数要求每一个函数都有一个返回值，也就是一个message对象，即使没有显式地返回，每个函数都会默认返回一个payload字段为空字符串的message对象。

3.3.4 数据库访问节点的设计

由于在原始的Node-red中没有与Redis数据库进行交互的节点，但是本文所提出的模型中用到了Redis server来存储中间结果集，并在Redis server中进行去重统计，比如计算最大值、最小值、累计求和等。所以为了能够让Node-red与Redis进行交换，进行数据传输，所以必须设计出对Redis数据库的访问操作节点。在该模型中，主要需要的节点就是redis_in和redis_out节点，它们分别完成Redis读取数据和把数据存储到Redis两项任务。

在redis_in中封装了几乎所有的Redis操作命令，该节点提供一个命令选择器，指定用户命令进行Redis操作。另外，redis_in节点是一个具有数据输入端点的节点，它的数据同样来源于上游函数节点提供的message对象中的payload字段+(msg.payload)，用于指定命令的格式和所要操作的Redis集合。而对于redis_out节点恰好与redis_in节点相反，它没有数据的输入端，只有数据的输出端，因为在该节点内部已经将数据的输入端固化了，数据就是从Redis中来的，{44%：但是该节点具有一个数据的输出端，为下游节点提供数据源。}

根据以上对这两个节点功能的分析，接下来对这两个节点进行详细设计，首先是redis_in节点。该节点第一步工作就是要去连接Redis server，这里就会用到在3.3.1节中所提供的数据库连接池函数，连接成功后需要调用命令选择器，选择用户指定的命令，然后根据上游function节点提供的命令格式和指定的数据集，将这些信息组装成一条完整的Redis命令，最后调用Redis客户端去执行该命令。在图3-8中展示了redis_in节点的设计方案。

图3-8 redis_in设计方案

redis_in节点在向redis server存储数据的时候，主要的工作任务集中在命令选择器上。在命令选择器中保存了几乎所有的redis写入操作命令，是存放在一个数组对象中，首先要从这个数组中找到用户指定的命令，然后判断该命令是不是psubscribe或者subscribe命令，因为这两个命令在获取redis数据的时候还需要监听message事件，而其他命令没有该事件，所以必须单独处理。最后，将用户指定的命令与上游节点传输过来的数据集拼接成redis的命令交个redisClient执行。最终实现Node-red里的中间结果集存储到redis中，同时，通过上游节点指定的操作可以实现中间结果集在redis中的统计计算。

对于redis_out节点的设计与redis_in节点类似，不同的是，在redis_out节点中同样封装了redis命令，但是这些命令只是读取数据的命令，所以命令选择器中的命令与redis_in的不一样。另外，由于redis_out节点具有一个输出端，所以在input事件监听器中监听到的数据封装完成后，还要通过node.send()方法发送出去，供下一个节点接收。

3.4 节点的重新部署

节点的设计和实现完之后，一步重要的工作就是要将新设计的节点部署到Node-red中。节点可以作为模块打包或者发布到npm库中，这使得它们易于安装其所有依赖的模块。为了解决安装包的依赖关系，在打包节点的时候就要严格按照npm包管理规则来打包。图3-9是一个redisSub节点打包的目录结构：

图3-9 节点package目录结构

本文采取的是本地模块安装的方式，在本地安装节点模块，就用到了npm link命令。将节点在本地目录，链接到一个本地Node-red安装目录，这和npm安装是一样的。本地部署节点按照如下两个步骤即可完成部署。

1.在包含有package.json的目录下执行sudo npm link命令；

{52%： 2.在Node-red的运行运行目录下执行npm link [节点模块的名字]。 }

部署成功后，重新启动Node-red，然后浏览器中打开编辑界面，在最右侧的节点视图就可以看到新增加的节点，这样就完成了节点的设计和部署工作。 为接下来改模型的应用提供了技术支持。

3.5 本章小结

本章首先去基于 Node- red与 Redis的实时流数据处理模型及其应用进行了需求分析，同时也对整个模型的总体架构进行了设计， {41%： 简要阐述了各个模块的功能以及整个模型的数据处理流程。 } 然后对Node-red新引入的数据输入节点、输出节点、数据计算节点以及数据库访问节点给出详细设计方案。 最后，阐述将新节点安装部署到Node-red中，使其成为一个完整的流式数据处理框架。