文本复制检测报告单(全文标明引文)

ADBD2018R_2018030210331620180304175700200185955269　　检测时间：2018-03-04 17:57:00

检测文献:　12112918_张增辉_大流量服务器优化方案探究

作者:　张增辉

检测范围:

中国学术期刊网络出版总库
中国博士学位论文全文数据库/中国优秀硕士学位论文全文数据库
中国重要会议论文全文数据库
中国重要报纸全文数据库
中国专利全文数据库
互联网资源(包含贴吧等论坛资源)
英文数据库(涵盖期刊、博硕、会议的英文数据以及德国Springer、英国Taylor&Francis 期刊数据库等)
港澳台学术文献库
优先出版文献库
互联网文档资源
图书资源
CNKI大成编客-原创作品库
大学生论文联合比对库
个人比对库

时间范围:　1900-01-01至2018-03-04

检测结果

总文字复制比：0.2%　　跨语言检测结果：0%

去除引用文献复制比：0.2%　　去除本人已发表文献复制比：0.2%

单篇最大文字复制比：0.2%（浅谈JVM - 初学者 - CSDN博客）

重复字数：[33]	总字数：[15811]	单篇最大重复字数：[33]
总段落数：[10]	前部重合字数：[0]	疑似段落最大重合字数：[33]
疑似段落数：[1]	后部重合字数：[33]	疑似段落最小重合字数：[33]

指　标：□疑似剽窃观点　□疑似剽窃文字表述　□疑似自我剽窃　□疑似整体剽窃　□过度引用

表格：0　　脚注与尾注：0

0%（0）	12112918_张增辉_大流量服务器优化方案探究_第1部分（总2412字）
1.3%（33）	12112918_张增辉_大流量服务器优化方案探究_第2部分（总2599字）
0%（0）	12112918_张增辉_大流量服务器优化方案探究_第3部分（总584字）
0%（0）	12112918_张增辉_大流量服务器优化方案探究_第4部分（总607字）
0%（0）	12112918_张增辉_大流量服务器优化方案探究_第5部分（总1745字）
0%（0）	12112918_张增辉_大流量服务器优化方案探究_第6部分（总1140字）
0%（0）	12112918_张增辉_大流量服务器优化方案探究_第7部分（总441字）
0%（0）	12112918_张增辉_大流量服务器优化方案探究_第8部分（总741字）
0%（0）	12112918_张增辉_大流量服务器优化方案探究_第9部分（总602字）
0%（0）	12112918_张增辉_大流量服务器优化方案探究_第10部分（总4940字）

（注释：无问题部分　文字复制比部分　引用部分）

1. 12112918_张增辉_大流量服务器优化方案探究_第1部分

总字数：2412

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容

大流量服务器优化方案探究

摘要

为了应对网站拜访的量太大进一步引发的服务器性能冒出的问题，让我们需要的网络服务器也能够往每天服务几千万甚至上亿流量的大型互联网服务器靠拢，这要求我们必须把网络服务器的各个服务细节都做好、做精。那作为新手菜鸟该如何进行这一个实现过程的探索呢？很显然，当今大型互联网服务系统如高楼林立般屡见不鲜，他们是如何把一个个大数据服务系统做精的？既然有了前车之鉴，那作为新手去进行后车之师是最好不过的探寻方式了。

通过这一方式，我先后了解了各种分布式架构的原理与应用，而在当前的设计作品当中主要涉及了JVM性能调优、tomcat服务器bio与nio原理、基于Spring SpringMVC Mybatis的Maven多模块架构构建、zookeeper集群、zookeeper分布式锁、dubbo原理与应用、snowFlake算法的应用、MD5加密算法的了解与应用、三大连接池与druid的应用、redis的原理及数据设计、数据库交互设计与优化、Nginx的动静分离和负载均衡原理与应用。在这一探索实现的过程当中收获了很多宝贵的实践经验和丰富的理论知识，并且深刻认识到高等数学知识在社会生产生活中得到了广泛甚至需要更深刻地应用。

关键词：性能；架构；数据；算法；分布式

Research on the optimization scheme of large flow server

Author Haven

Abstract

In order to deal with that web site traffic is too large to cause server performance problems. The web servers we need are also able to close to large Internet servers that serve tens of millions or even billions of traffic a day. This requires that we have to perfect the details of every service on the web server. So how do you explore this implementation process as a novice? Obviously, Today's big Internet service systems which like a lot of tall buildings looks common. How do they make every big data service system perfect? Now that we have that the trajectory of the front traffic, Follow the track of the traffic if you're a beginner that was the best way to find out.

In this way,I learned all sorts of the principles and applications of various distributed architectures in order. In the current design work, it mainly involves: JVM performance tuning, The principle of Blocking IO and Non-blocking IO for the tomcat server, Construction of Maven multi-module architecture based on Spring SpringMVC Mybatis, Zookeeper cluster, Zookeeper distributed lock, Principle and application of dubbo, The application of snowFlake algorithm, Understanding and application of MD5 encryption algorithm, The application of the druid and the three connection pools, The principle and data design of redis database, MySQL interaction design and optimization, The principle and application of dynamic separation and load balancing of Nginx. In the process of this exploration, I have gained a lot of valuable practical experience and rich theoretical knowledge. And it is deeply realized that higher mathematics knowledge has been widely and even needs to be more deeply used in social production and life.

Key words: Performance; Architecture; Data; The algorithm; distributed

正文

第一章 JVM性能调优探索

Jvm的优化有多个不同的角度可以入手，比如系统、代码、算法等等。那么首先，砸门就必须先理解一下jvm的都有哪些不为人知的属性和故事[1]，如果把程序与人体的生命活动作比较的话，那么人体就是整个程序。

栈

如果把栈比作是器官，那么栈内的数据类型就是各种生命活动所需的各类养料。它寄放的数据对象在使用完它之后就会立刻被清理放空。线程私有。

栈分为两种：

本地虚拟机栈，为人体内器官（java方法method）的运作而服务的栈。

本地方法栈，为原系统的运行而提供的栈。

那么养料分为可变养料（摄食消化—间接养料）和不变养料（水、氧气之类的直接养料）。

不变养料

基本数据类型就是 ，像这样int 直接给定内容的buBianYangLiao = 6 ；而类似于基本数据类型直接赋予其值的也是直接以完

整属性存放在栈中：Object streq1="streq1";Object streq2="streq2";这时的字符串的对比直接通过streq1==streq2完成。这些类型的数据在被实例化之后是指向某一个内存空间，字符串的对比也是对比是否指向同一个内存空间。而StringBuffer和StringBuilder在实例化之后也是指向了某一个地址空间，相同的实例化不会再次创建，所以这两种也用“==”来对比值。

可变养料

比如对象类型的数据。一般通过实例化出来的对象:比如like这样Object-objeq1=new-String("streq1");Object-objeq2=new-String("streq2");的

这时objeq的值是一个堆中的地址，但是再次进行相同的实例化却会重新创建一个对象，所以相同的new得到的地址此时是不同的，因此使用objeq1.equals(objeq2)来进行对比。

堆

如果把堆比作是人体内环境，堆中的数据类型就是像一个个细胞一样成堆成堆的存在的，每一个细胞就是一个对象，每个细胞里面都包含了类、类型、值地址等等信息。如果把新陈代谢比作线程的话，那么每一个细胞都要进行新陈代谢，就是被所有活跃的线程所共同享有。如果细胞没有被用于生命活动，也就是没被引用，那么就被作为废物给清除掉。

方法区

如果把方法区比作是人体器官运行的方法，那么类、final常量和static变量等等就是具体某个器官的运行方法，每一个都是唯一存在的。器官的运行伴随着新陈代谢（被所有活跃的线程所共同享有）。

程序计数器

每一次新陈代谢（线程）都有阶段性标记（计数器计数）。比如说：所处阶段指示（行号指示器），哪些阶段还需要另外做什么事（标记程序运行所处位置）。

类加载

JVM的类加载过程粗略的可以分成3步

1、加载：类的加载当然是在方法区进行啦，类比作某个器官有自己运行的方法。

2、链接：这一阶段是把字符连接，转换成真正的地址指向

3、初始化：loader在加载的那个时候都是先给它所有对象给予固定的null值。

初始化才会改成name="honeycat"赋予自定义值。

类加载到这里完成，此时不会抛出类找不到异常。

类加载器的四级组成：

1.启动加载器 ； 2.扩展加载器 ； 3.应用程序加载器 ； 4.自定义加载器 。

从较低的层级的loader有接应到类开始就逐级往上传递给上级直到启动加载器，当然没有上级的话就返回给自己咯，然后最顶级加载，这是一种委托，可以防止执行loader本身的程序文件被用户的文件覆盖导致系统不能正常运行的作用。

垃圾回收（Garbage Collection[2]）

垃圾回收涉及“垃圾的定义”和“回收的方法”

垃圾的定义

判断内存中的某一个对象是否能够成为垃圾的方法有：

1.对象引用计数法：通过为内存中的每一个对象对应的都创立一个计数器，这样一来每个对象被多少的其他对象引用的数目就很清楚了。只要该对象被其他的给拿来用那么对应的这个计数器就加一，反而言之只要有对该对象的饮用关联一解除那么对应的这个计数器就减一。如果计数器跑完最后的结果=零那么就是没有被其他对象所引用，如此就可以回收啦。

2.可达性算法：从某个根节点作为开端向下搜寻（根节点可能是：1.类对象main开始；2.所有程序入口； ），只要节点有被引用，那么我们就把它捆绑到这颗树的枝相对应的某个位置上；正在处于该颗树上的节点对象说明此时正在被某某给引用了。

两个方法的区别：

第一种判断的方法实现起来较为简单，应用起来也比较高效；但是有缺点：1.对于已经没有被其他对象引用的垃圾对象不能够马上查出清理，检查垃圾的时效性不够好；2.这种方式貌似不能够应对程序对对象的循环的引用的情况。

第二种较为复杂，但可以比较彻底地GC。

回收的方法

标记-清除：在对数据对象进行是不是垃圾的判断的时候，一旦堆中的数据对象的值只要是‘null’的那么就把它清理删除掉。会比较易衍生出锁片级别的垃圾对象，这种垃圾不好处理，所以我们不建议去使用它。

复制：我们可以把内存一分为二，然后再把不是‘null’的数据对象进行复制并黏贴到后一部分内存中，原来那部分的数据对象就给它全部删除清空。但是缺点是：1.系统可用的内存空间在这个地方就会因此而减半；2.数据对象在两个部分内存空间之间交替的来去复制这种操作也是甚为繁琐了一些。

标记-整理：我们把内存中不是null的数据对象全部都集中到一块按照顺序来排列放置，然后把其他的数据对象值为null的都清空掉。

按分区对待分

增量收集：实时垃圾回收算法

分代收集：我们可以根据不同的对象所拥有的活跃性差异而区分为持久代的、老年代的或者是新生代的，然后根据时代的不同来区分对象再使用不同的GC算法来进行collection。

按系统线程分

串行收集：此时只用单线程来处理我们整个程序的GC；易实现、高效率；局限：只能用于单个核心系统的CPU处理器的机器。

并行收集：此时运用多线程技术来处理我们整个程序的GC；迅速高效；很适合多个核心的处理器的机器。

并发收集：串行和并行在执行GC时要暂停运行环境，即“停止和复制”[3]，堆越大暂停时间越长。采用并发的方式就会使两者同时进行而不需暂停原程序。

最后准备调优实践

流程与工具：

3. 12112918_张增辉_大流量服务器优化方案探究_第3部分

总字数：584

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容

第二章 tomcat服务器探索解析

Tomcat的网络IO实现方式[4]可在相关文献中查阅。

bio/阻塞io(blocking io)

Socket acceptForMyBlockTestNeed = socketServer.accept();

根据bio的处理流程来看，显然有多少个客户端连接，服务器的线程池就会对应开启多少个线程处理连接，而每一个连接都在占用着连接的资源。

nio/非阻塞io(no blocking io)

在tomcat的8.5版本更早以前处置连接默许使用的是bio的模式；nio被隐藏，当然我们可以按照实际的业务要求来自己定制nio；而在tomcat8.5以及之后则改为了默认nio的连接处理模式。

nio模式的处理流程分为两个阶段：

第一阶段：1.接受连接；2.处理连接（注册轮询监控和数据监听）；

同样是一个线程去对应处理一个来自客户的连接，为每个连接注册为监控选择器selector的对象，为每一个连接（线程）注册一个监听器OP_READ。

第二阶段：

轮询监控对每个线程进行轮询，在轮询时接受OP_READ数据监听器的事件通知；如果连接中包含数据交互的事件，则该连接需要被通知然后交由线程池处理，如果有数据传输就建立连接，如果没有就继续定时监听连接状态。

tomcat服务器nio原理示例：见附件一。

4. 12112918_张增辉_大流量服务器优化方案探究_第4部分

总字数：607

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容

第三章 ssm的多模块构建

一、maven的原理了解

通过相关资料的查阅了解，maven的运作主要靠pom文件的依赖性以及pom文件内部标志成员识别来建立整个项目的jar依赖树。通过树的建立和遍历识别可以使每个jar有自己的关系树，如果在建立识别中出现问题则会报出相关的异常问题。

二、整体架构介绍

1.base-lay项目

lay-parent模块

lay-biz模块

lay-client模块

lay-control模块

lay-web模块

2.druid-dubbo-server项目

server-parent模块
server-biz模块
server-client模块
server-client-impl模块
3.dubbo-customer项目
customer-parent模块
customer-control模块
customer-web模块
三、注解
我们可以发现每个项目当中都有一个parent的模块，这是作为一个maven资源库的功能，该模块内一般只配pom文件。这个资源库为所有子模块提供需要的jar包，这样做有个好处就是子模块运用的jar包都有了统一的管理；因为jar使用了统一的管理针对它们的version，子模块引用时不容易产生版本的冲突，子模块只在意引用哪一个jar不用在意版本号；由此在一定程度上简洁了代码和有效避免了jar冗余。

5. 12112918_张增辉_大流量服务器优化方案探究_第5部分

总字数：1745

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容

第四章探访zookeeper
拆分系统从水平拆分到垂直拆分演变了分布式，而今分布式已经可以有效地处理系统的伸缩性和性能的各种可能问题了。但是在进行系统的拆分的同时不可避免的带来了系统内部的复杂性的问题，因为拆分之后的每一个子系统的运作常常都不是孤立自主的。他们之间需要需要协作交互、协调配合，为此需要一种统一的服务机制来协调各个子系统的运作，本设计因为这样使用了集群选择的技术架构则是zookeeper。
zookeeper集群
通常由2n+1台服务器server组成。
1.leader，作为引领群众的公众人物，它会开启某一项决策的投票以及在投票之后对于决议结果的整体实施，从而更新内部的所有zk集群的服务status；
2.learner，这个role它涵盖了一个follower的角色和一个observer的角色；
3.follower这个role它主要做的是接受来自客户端相关的请求，然后再往客户端给予返回它对应请求的结果；它在选举谁成为master的过程中会成为候选人而被所有的吃瓜群众拉票；
4.observer作为第三方主要做的是把来自客户端的请求进行代为转告发给它们的首领leader；它作为群众的协调者是不参与到选master的投票环节中去的，它只需要做把首领leader的状态同步到基层群众去的本分工作；它所存在的目的是为了提高整个集群系统在读取应答环节的处理速度。
5. client(客户端)，请求发起方
zookeeper数据模型
znode既有文件所具有的特点那样有数据信息，也有目录所具有的特点那样拥有路径标识还可以有子级的znode。它需要被某一个用户单线程阻塞操作也就是原子性。
用一颗树状图来表示，树上的每一个分支节点或者叶子节点都称之为一个znode。树上的每一个zk节点之间都有各自不尽相同的信息，它们都有着共同的属性信息：
1.stat：这是用于存放zk节点的有关版本信息、有关访问权限信息等等的信息的寄存域。
2.data：与当前这个zk节点捆绑的数据.
3.children：处于当前这个zk节点的目录下面（也就是当前这个节点的子级节点）的znode节点.
分布式事务[5]。
Znode实体对象包含：
1.Watches：针对于每一个zk节点我们对应都可以设置一个监视器(watch)来监听节点的状态变化的事件。只要节点的状态发生了变化那么就会触发监视器事件（事件内容可以根据业务场景自主定义内容）定义内对应的方法事件并且执行，改变一次就触发一次。
2.数据访问：针对每一个zk节点的操作我们都需要遵循原子性的原则，也就是说每次我们对同一zk节点要做什么操作都是要一个个排队下去的。读写的相关运行动作都是会关系到一整个zk节点的全部数据的。因为每一个zk节点都有一个用于控制不同用户是否允许访问的列表，这个列表存放在stat中，这个列表详细规定每一个不同用户针对该目标zk节点可以提供执行的增删改查操作。

3.节点类型：每一个zk在其完成创建之后类型就确定了就不能再次改变了。--<1>.临时节点：客户端在与zk之间的会话一旦结束之后，那么这个zk节点就会马上被清轻删除掉；它的目录下面拥有子级的节点这件事情是不能被源规则答应的。-<2>.永久节点：该zk节点只有在用户手动触发删除相关的操作之后，那么这个znode节点才被删除。

4.顺序节点：用户在进行zk节点的创建时可以在zk节点的命名结尾增加赋予一个递增的计数命名尾数。这些计数处于每个子级zk节点的命名后缀使得这些子级zk相对于在同一父级别的节点内部相对来说都具有唯一性，它们的格式是10个位数的数字，如“0000 0000 69”。

基于zookeeper的分布式锁

在蜂拥的网络服务消费下，有一些服务或者程序是需要排队来一个一个地单独执行的，比如抢票、抢红包、抢单... ...面对这些类型的需求就催生了各种各样的分布式锁。对每一个来自客户的请求分别对应都创建业务所需要的线程来一个一个地处理，而分布式锁针对线程的生命周期做业务上的控制[6]。从线程的创建到死亡的周期内各环节都有各自的状态特点[7]。

具体实现示例：见附件二。

6. 12112918_张增辉_大流量服务器优化方案探究_第6部分

总字数：1140

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容

第五章 Dubbo探研及浅析

我们最好从dubbo的官方高清大图入手看整体架构

一、从垂直纵向看结构的层次

暂时抛开网络传输的层面来看分别有：

（1）config服务配置

（2）proxy服务代理

（3）register服务注册

（4）cluster服务调用方

（5）monitor服务监控

（6）protocol服务中心

Dubbo的服务各个执行流程的运作细节基本在这些层面的角色上协同配合进行。

1.网络服务层

服务层用作传输对象的实体是必须要要实现serializable而在传输之前做序列化工作的[8]。

2.基于RPC的dubbo网络协议层

官方对于RPC程序的实现，设计包括了5个组成部分：

客户端： <1>. User ； <2>. User – stub ；

交互层:<3>. RPCRuntime ；

服务端: <4>. Server – stub ； <5>. Server .

3.网络传输层(Dubbo的网络传输底层实现基于Netty[9])

在serialize层codec根据报文消息来源方向的不同，去进行序列化或是反序列化。

二、从水平横向看模块组成

从水平方向把图平均分为两半，右半边是服务的提供方，中间的分割线则是一系列的服务需要的信息--生产、注册等信息，左半边则是服务的调用方。

我们可以看出整个流程是：

从右上角的服务提供方开始-->（服务配置-->服务代理-->服务注册-->服务监控-->服务中心)；

然后再从：

左上方的服务调用方需要服务开始-->（消费配置-->消费代理-->消费注册-->服务监控-->服务中心)；最后从：

左上方的服务调用方进行消费开始-->从服务的一个个配置来进一步获取每个对应的一个个服务接口-->获取服务代理-->对于服务的规则过滤与对于服务的动作细节监控-->网络传输层-->服务过滤-->服务代理-->服务提供方实现服务.

基于spring[10]的具体服务配置示例：见附件三。

基于dubbo的文件上传

一种是用servlet方式，此方式需要在web.xml配置

其实来自官方的说明servlet的3.0版本的相关使用不仅仅可以在原本的web.Xml之中来配置，还可以使用spring的注解的实现方式来配置该文件上传在servlet之上的使用，但是对于这种方式的Part与那版本就是获取前端的文件；在dubbo之中我们需要的是从控制层获取，那么要怎么改为从控制层获取就是个问题了，显然协议或方式是需要改变的。

对于日志打印的优化，改用logback，我们利用代码的实现方式来启动logback的打印日志服务，相关的示例可以照见附件八。

7. 12112918_张增辉_大流量服务器优化方案探究_第7部分

总字数：441

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容

第六章 Druid的了解与使用

连接池介绍

系统在进行初始化的时候会把数据库与服务器之间的连接作为对象，然后将连接对象保持着连接的有效性放在某个内存空间中，每当有用户需要进行数据库访问的时候，系统就会把有效的空闲连接从池中拿出来给需要的用户访问开辟通道而不是一直反复无常地重新创建和结束关闭。用户使用完的连接就会被系统送回到连接池中并继续保持有效的空闲状态，以供后续的用户请求使用。连接的初始创建、后续创建、连接测试、断开处理都由连接池自身配置机制来进行管理。而且，还可通过设置数据库连接池的参数来优化数据库连接池的实际应用性能。

Druid各项性能配置：自己去查。

本设计利用了spring配置来对数据库的连接池进行了管理，并且是Druid的多个数据源可以根据业务的实际需求进行动态切换数据源的方式，按照实际应用项目内部对应为水平分库，项目间对应为垂直分库，在service层面利用注解标注该class所有服务对应的数据库。本设计的druid配置具体参照物：见附件九。

8. 12112918_张增辉_大流量服务器优化方案探究_第8部分

总字数：741

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容

第七章 Redis的应用与数据设计

我的redis数据设计小试

List

用户列表：user_唯一识别码（user_code）

{

user_id

user_name

user_nick

is_logIn

age

...

}

订单列表：order_唯一订单流水号（SerialOrderNum）

{

order_id

order_beloger

...

}

Hash

订单-用户HashSet：

哈希名为md5加密的sql语句(比如像这样的select * from data_order)

{

键为md5加密的sql语句（譬如像我这么写这么玩的select * from data_order where order_id=...）值为订单号

}

订单-总金额HashSet：哈希名为order_amount

{

键为订单号值为该订单总金额
}
ZSet
店内的单个商品成交量ZSet：可以实现热门商品的排序Hot_Ware
{
ware_amount1
ware_amount2
...
}
Set
某项活动的参加人员，人员是唯一的。比如：拼多多的拼单活动。
{
user_code1
user_code2
...
}
用redis创建订单号，原理是对于redis的操作具有单线程特性，由此可以用来创建分布式唯一id。详细的实现我们可以参照我的案例例：附件—之四。
将订单等数据信息保存到redis要建立唯一索引以便后面的查询等操作，由于redis不支持数据包含空格，所以保存sql语句要进行加密并且结果唯一，MD5的唯一性具有很强的保障。详细的实现我们可以参照我的案例例：附件—之五。

9. 12112918_张增辉_大流量服务器优化方案探究_第9部分

总字数：602

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容

第八章 Mysql数据与交互设计
数据库的分库分为水平分库和垂直分库两种，当前使用的垂直分库设计实现示例：见附件六。
分布式数据库[11]。
数据库的水平分库通常要做主从同步配置策略和数据库读写分离[12]等操作。
数据库连接池与servlet的结合使用
一般的数据库分库最好采用垂直分库，而分库的大小、库内内容等分配策略需要在预测将来的业务需求可能规模而制定；相对而言水平分库会显得较为复杂且更不好管理；例如：把order_manager分为order_manager1、order_manager2... ...
调度和锁定问题[13]。为了防止数据库正在更新的同时出现不同用户之间的冲突，采用版本号version作为乐观锁标志来限制用户对每张表的更新操作。
对于用户用到的注册、创建等操作需要唯一主键id时，使用雪花算法的分布式唯一主键生成策略，生成的id是32位的递增整型数据。根据实际情况，类似于订单表、费用单项表这样的可以考虑直接使用数据库自增。
当单张表的数据量不足以支撑业务数据量时当然是要使用数据库分表策略啦，既然每个用户的id是由32位的int组成的，那我们在执行表操作前就可以通过对32的int进行取余%操作（取余数根据需要分的表数量，被取余数就是用户id），得到的余数就是要进行操作的第几张表，比如：data_order1、data_order2... ...

10. 12112918_张增辉_大流量服务器优化方案探究_第10部分

总字数：4940

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容

第九章 Nginx动静分离和负载均衡
将nginx用在前端提供静态内容，并用他的代理模块作为负载均衡器，将Apache服务器用在后端提供动态内容[14]。
我们在这里假设使用了三台Nginx服务器做了负载均衡，其实在微服务情况下可以使用docker在一台服务器上搭建Nginx负载均衡服务集群。具体配置的可参照例子：见附件七可得。
需要注意的是如果你要进行负载均衡的话，那么你就需要把location ^~ /{}的相关配置注释掉，否则报冲突异常。

服务的路由和负载均衡中[15]也有详细的介绍。

平常常的情况下我们负载均衡本身自带的实现方式是轮询[15]；但是在服务器之间性能不均匀的情况下，需要根据实际情况来指定各个服务器的访问权重。但是这两种情况需要另外执行同步session到其他服务器的操作。

upstream tomcats {
server 192.168.44.161:8080 weight=10;
server 192.168.44.163:8080 weight=9;
server 127.0.0.1:8080 weight=11;
}
}

负载均衡策略：IP绑定ip_hash，这种情况下就可不用担心session跨域的问题了。

upstream tomcats {
ip_hash;
server 192.168.44.161:8080;
server 192.168.44.163:8080;
server 127.0.0.1:8080;
}
}

等等，还有其他的第三方的负载均衡策略。

上面的描述是针对Nginx配置的比较通用的各项语句法则[16]。

调试

Tomcat相关：

一、登录tomcat服务器的默认页面所报的access-denied的错误。

解决：目录conf下名为tomcat-user的xml文件配置错误。

二、tomcat启动闪退

解决：在对应的bat码文内码字end之前编写入码字pause；、ko掉其他运行的异样。

Log4j相关：

一、log4j部分警告。

解决：在conf文件夹下的名为catalina的文件后缀为properties的文件内部代码对应放入相干jar包的过滤。

Maven相关：

一、 编译Maven多模块需要根据模块间的依赖关系将jar和静态和资源文件编到指定编译目录。

解决：利用Maven构建build可以指定打包类型和打包目录等信息。

二、 利用Maven构建静态资源文件为jar，企图应用Spring3.0的REST-resource方式，让浏览器可以直接通过控制层代理访问jar内的静态资源文件从而实现渲染服务。

解决：Spring3.0的REST-resource方式实现的jar内资源提供服务需要通过java代码另外指定jar文件名称、路径实现代理通道。

SpringMVC相关：

一、 借助SpringMVC举行托管的对静态资源相关连接请求的处置的实现渠道有以下几种：1.<url-paten>/</url-paten>拒绝所有请求。2./*.do就是答应给它处理所有类型的文件连接要求；涌现js和css的连接会见404问题。3.每一类都给他配映射。

解决：改为第三种，对每一个需要处理的静态资源文件类型相应给上SpringMVC的servlet-mapping映射，结果每个类型文件正常访问；但依旧未确定/*.do的jsp正常连接会见，而js和css连接会见的404问题，个人观点是“对付jsp文件的连接会见方式区分于js、css、html等文件的资源连接会见方式;因为在第三种方式之下是不要配置SpringMVC对于jsp类型文件的映射的，反而给予其配置会引起报错；js、css、html等为文本类型静态文件，而jsp属于servlet层面经过java代码编译后才输出文本文件给浏览器的动态文件”，而问题应是*.do配置并未生效。

Zookeeper相关：

一、 cmd闪退。

解决：在对应的bat码文内码字end之前编写入码字pause；、ko掉其他运行的异样。

二、 Myid找不到。

解决：日志目录下的Myid去掉文件的后缀名、zookeeper配置文件后缀为conf内的相干片段dataDir=…路径的‘\’改为‘/’。

三、伪集群配置的Connection-refused-and-time-out错误：

解决：关闭防火墙、配置文件错误、三个cmd的服务与启动对应修改。

Dubbo相关：

一、dubbo配置文件的SAXParseException异常dubbo:application找不到错误。

解决：jar冲突、xml头部加载异常。

二、dubbo客户端报的l0ltec/zkclient/下的IZkStateListener文件找不到错误。

解决：只要在pom加入zk客户端的相关jar此处为ioi的客户端jar。

Logback相关：

一、代码手动加载logback日志失败。

解决：1.找文件；2.loggerContext加载失败。

二、类没有被包含异常，使用logback对Log4j日志工厂初始化的时候报的需要的包下找不到名为LoggerContext的类的问题原因。

解决：相关jar冲突。

三、logback在进行初始化的时候报的相关包下找不到名为LoggingEvent的类文件的相关错误。

解决：其实问题的原因出现在slf4J上，它的1.4.0版本对于log4j是有版本上的要求限制不然运行就会有问题，它要求版本是1.2.12的包括更高以上的。

MySQL相关：

一、卸载了 the old's version 的mysql之后，清除了注册表等信息，然后装了5.7版本的mysql，这个时候可以正常使用，但是重启电脑之后出现了mysql服务连接失败的异常报告，异常代码为10061的代号。

解决：虽然旧版本的mysql已经被卸载，但是服务在电脑的配置没有被移除，在两个版本mysql都为自启动时，按顺序第一个服务找不到启动文件报错。

Nginx相关：

一、Nginx启动失败，然后报出了worker_processes也就是工作的机器所提供的工作流程数的配置出现了数量错误导致了编译失败。

解决：该项worker_processes的配置需要根据服务器系统的内核配置情况来设置。

二、Nginx启动失败，然后报出了有关location^~/{}的代码出现问题，实际报的是代码在文件的配置出现冲突的异常。

解决：负载均衡情况下，需要把location ^~ /{}的相关配置注释掉

展望

虽然一直尽可能在工作之余赶时间去研究大流量服务器优化，使得作品在整体的架构设计上已有了一定的雏形，可是根据实际应用情况来看待，还是有很多后续工作需要长期去花时间完成的。

从实践固有缺陷来看：

首先，从吃土IT民工出身的实践人员身份可以知道，本人用的低配版笔记本而且仅可能有一台，所以就算了解了分布式集群策略，负载均衡策略等等，也没有实验的先提条件，虽然部分类型集群可作伪分布。所以在集群的实际运行情况、异常解决及原理等等都缺乏经验和知识累积。这个后面是需要继续进行实践的。

从实际应用层面来看：

服务器在实际环境下运行比测试、生产环境下运行是更为复杂、严苛的，更何况一个测试阶段都可能有不少问题的服务器，所以继续补充、继续测验的工作还是需要不断进行，这个关系到各种服务性能的可靠性、健壮性等。

从架构组件构成来看：

目前的技术架构在每一个层面都已经有了丰富、有效、健全… 的策略和产品，而本设计所用到的架构只是冰山一角；本设计在一些层面上还是缺乏热门架构研究的，比如：消息队列、前端缓存技术架构、前端架构、各架构层次的中间件…如何根据实际需求架设更完美的应用架构是每个技术忙人需要孜孜不倦持续探索的话题。

从茫茫架构之海眺望：

分布式架构在各种应用场景、业务需求、管理决策等等不同实际情况下都有着不尽相同的架设模式，本文的对分布式的应用仅仅是九牛一毛而已；小弟不才，在当前的大流量服务器的优化探索尚处于应用层往下探索的阶段，在对于底层技术原理与细节实现上还有待提高，所以展望后面的工作应是探索在不同的可能情况下如何实现实际需求以及更深层的实现原理理解等等。

显然，本设计在当前的章节分配上基本是以某一个架构作为章节题目，而如果把章节题目改为某一个架构层次就不同了，在相同层次里面将会是架构之间和策略之间的各种对比与较量。

总结

无论是做工作、学习、生存还是其他什么事，想要达到目的就一定离不开经验的积累。经验的积累离不开方法和实践，所以“有志者，事竟成”的‘志’往往是寻找对的方法和坚持实践。而在本设计的探索方法上遵循了探索轨迹，而不是从第九章到第一章的顺序；由浅及深由表入里（虽然还不够深，但这取决于目前能力），而不是天女散花遍地找茬。

不管在什么时候、什么地点都总会遇到困难。遇到问题时要尽量靠自己的知识、经验去解决，不能解决的可以通过查阅资料，从相关知识中获取灵感；实在不行就参考前人躺过坑后的经验之谈来理解去实践解决问题。相信没有什么事是过不去的坎的；如果有，那肯定是你自己的脑子有坑，你需要改变观点和思路。遵循前车的轨迹很好走，但当你变成行业内的佼佼者之后，你遇到的问题就有可能独特到没有人遇到过，这意味着你必须要自己去动手解决这些难题[17]。

致谢
大学，不论是堕落的、悲伤的、愉快的、勤奋的都终将过去。首先，我要感谢教导我们的那些一届又一届辛勤的教师们，你们的悉心传授使我们健康成长；然后，感谢我遇到的所有人，从发展哲学的角度，你们每个人都让我得到了不同领域、丰富多样的见识；最后，本文顺利得以完成，特以致谢导师张赛男讲师的关怀和指点。
参考文献
[1] 周志明.深入理解Java虚拟机:JVM高级特性与最佳实践. 北京:机械工业出版社,2011.06
[2] Richard Jones,Antony Hosking,Eliot Moss.The Garbage Collection Handbook: The Art of Automatic Memory Management. Chapman and Hall/CRC,2011.08
[3] [美]Bruce Eckel著,陈昊鹏译.Java编程思想.第4版. 北京:机械工业出版社,2008.09
[4] 曾宪杰.大型网站系统与Java中间件实践. 北京:电子工业出版社,2014.04
[5] 倪超.从Paxos到Zookeeper:分布式一致性原理与实践. 北京:电子工业出版社,2015.02
[6] Brian Goetz,Tim Peierls,Joshua Bloch,Joseph Bowbeer,David Holmes,Doug Lea,童云兰译.Java并发编程实战. 北京:机械工业出版社华章公司,2012.02
[7] 高洪岩.Java多线程编程核心技术. 北京:机械工业出版社,2015.06
[8] 许令波.深入分析Java Web技术内幕. 北京:电子工业出版社,2012.09
[9] Norman Maurer,Marvin Allen Wolfthal.Netty in Action. Manning Publications,2015.12
[10] 郝佳.Spring源码深度解析. 北京:人民邮电出版社,2013.09
[11] 杨传辉.大规模分布式存储系统:原理解析与架构实战. 北京:机械工业出版社,2013.09
[12] 李智慧.大型网站技术架构.核心原理与案例分析. 北京:电子工业出版社,2015.11
[13] [美]杜波依斯著,杨晓云王建桥杨涛译. MySQL技术内幕.第4版. 北京:人民邮电出版社, 2011.07
[14] [英]Peter Membrey[澳]David Hows[荷]Eelco Plugge著,武海峰陈晓亮译.实用负载均衡技术-网站性能优化攻略. 北京:人民邮电出版社,2013.05
[15] 陈康贤.大型分布式网站架构设计与实践. 北京:电子工业出版社,2014.09
[16] 陶辉.深入理解Nginx（第2版）. 北京:机械工业出版社,2016.02
[17] 子柳.淘宝技术这十年. 北京:电子工业出版社,2013.05

附件
附件一：
附件二：
附件三：
1.服务提供方
首先创建服务。
然后暴露服务。
2.服务调用方
调用dubbo服务进行消费。
附件四：
/**
*流水订单号生成器
*/
/**
*流水订单号生成器
*/
而需要注意的是nodeValueForIncr是需要每天定时更新重置的。
附件五：
/***保存订单信息到redis数据库*/
附件六：
1.数据库user_manager：
/*用户列表*/
2.数据库ware_manager：
/*商品家族表*/
/*商品类型表*/
/*单一商品表*/

3.数据库shop_manager：
/*商店列表*/
/*店内在售品类表*/
/*售类内在售单品表*/
4.数据库order_manager：
/*订单列表*/
/*费用列表*/
附件七：
第一台服务器的Nginx配置详情：
nserver.conf文件
nginx.conf文件
第二台服务器的Nginx配置文件
nserver.conf文件相同
nginx.conf文件
http{# 被代理的服务地址(负载均衡)
upstream tomcats {
#ip_hash;
server 192.168.44.161:8080;
server 127.0.0.1:8080;
server 192.168.44.163:8080;}
}
第三台服务器的Nginx配置文件
nserver.conf文件相同
nginx.conf文件
http{# 被代理的服务地址(负载均衡)
upstream tomcats {
#ip_hash;
server 192.168.44.161:8080;
server 192.168.44.163:8080;
server 127.0.0.1:8080;}
}
附件八：
代码手动开启logback日志：
附件九：
Spring增加配置：
某一个数据库的druid连接配置：