哈尔滨工业大学

**计算机科学与技术学院/国家示范性软件学院**

**2019年春季学期**

**《软件架构与中间件》课程**

**实验报告**

**Lab 5：分布式系统整合实验**

|  |  |  |
| --- | --- | --- |
| **姓名** | **学号** | **联系方式** |
| 刘思琦 | 1163710228 | hit1163710228@163.com/18846183728 |
| 石昊 | 1163710226 | 2670041246@qq.com/15645077257 |

目 录

[1 实验概述 1](#_Toc11958985)

[1.1 实验目的 1](#_Toc11958986)

[1.2 实验要求 1](#_Toc11958987)

[2 实验内容与过程 1](#_Toc11958988)

[2.1 系统改造的宏观架构设计 1](#_Toc11958989)

[2.2 计算层上的系统改造——Nginx集群负载均衡的设计与实现 2](#_Toc11958990)

[2.3 计算层上的系统改造——Hadoop大数据分布式计算技术实现大规模用户访问日志统计 6](#_Toc11958991)

[2.4 数据层上的系统改造——web系统数据库分库分表的设计与实现 8](#_Toc11958992)

[2.5 数据层上的系统改造——系统查询的缓存结构的设计与实现 14](#_Toc11958993)

[2.6 表示层上的系统改造——React前端框架的重写 14](#_Toc11958994)

[2.7 改造前后的系统性能分析 16](#_Toc11958995)

[3 结对开发过程记录 17](#_Toc11958996)

[4 实验总结 18](#_Toc11958997)

[文档全部完成之后，请在上述区域点击右键，选择“更新域”，在打开的对话框中选择“更新整个目录”]

# 实验概述

## 实验目的

1）尝试综合运用计算层、数据层和表示层的架构技术

2）实现满足高性能、高可用、高可靠等质量属性的复杂软件系统

3）学会分析和构建高效的分布式软件系统

## 实验要求

1）组队：2人结对成组

2）题目：分布式系统整合

3）基于实验2-4在不同层面（计算、数据、表示）上的软件架构技术，对遗留进销存系统进行分布式改造。

4）按照用户需求，商业业务逻辑，将各部分的内容进行逐步整合，形成一个功能完整，且具有较高可用性、可靠性等的分布式软件系统。

5）分析改造前后系统的非功能指标。

6）应该出设计过程和实现细节

# 实验内容与过程

## 系统改造的宏观架构设计

1. **对原系统的简介**

（注：上学期的遗留进销存系统因为技术能力不完善，经验不足等原因，出现了架构腐化以及暗藏bug等情况，所以在咨询老师后选择了另外一个完全由我个人设计并实现的规模类似的web系统——学生信息管理系统，将会对此系统进行架构层面上的改造）

系统是基于mysql + servlet + jsp实现的一个Java web学生信息管理系统。后端采用的是dao – model – servlet三层架构，前端由Jsp + JavaScript整合EasyUI框架实现了友好的用户界面。

系统的功能性需求包括：

①添加、修改、查询、删除学院信息，学院信息包括学院编号，学院名称和学院描述；

②添加、修改、查询、删除班级信息，班级信息包括班号，所在学院和班级描述；

③添加、修改、查询、删除学生宿舍信息，学生宿舍信息包括宿舍编号，宿舍名称和宿舍地址；

④添加、修改、查询、删除学生信息，学生信息包括学生姓名，学号，所在班级，所在学院，性别，出生日期，联系方式和学生描述。

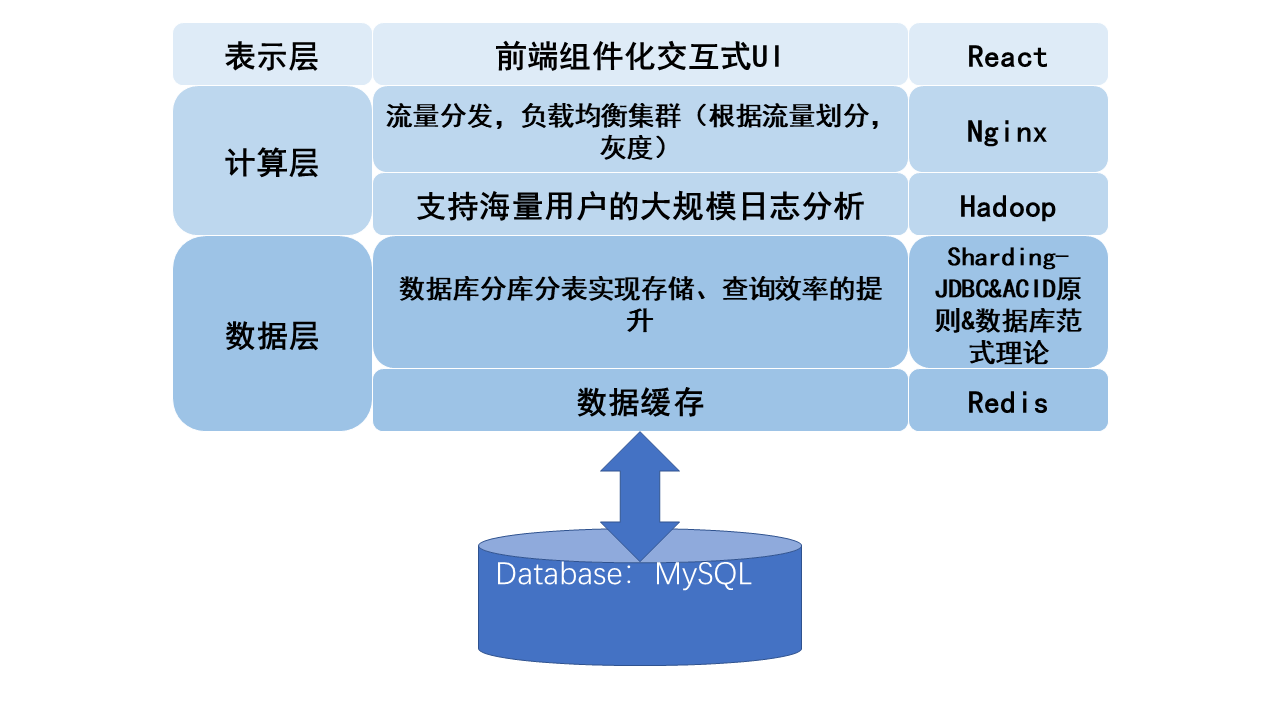
上述所有查询都支持模糊查询。

1. **系统改造的宏观架构设计**

在本次实验中，我计划通过数据层、计算层和表示层的三个层面上的改造，

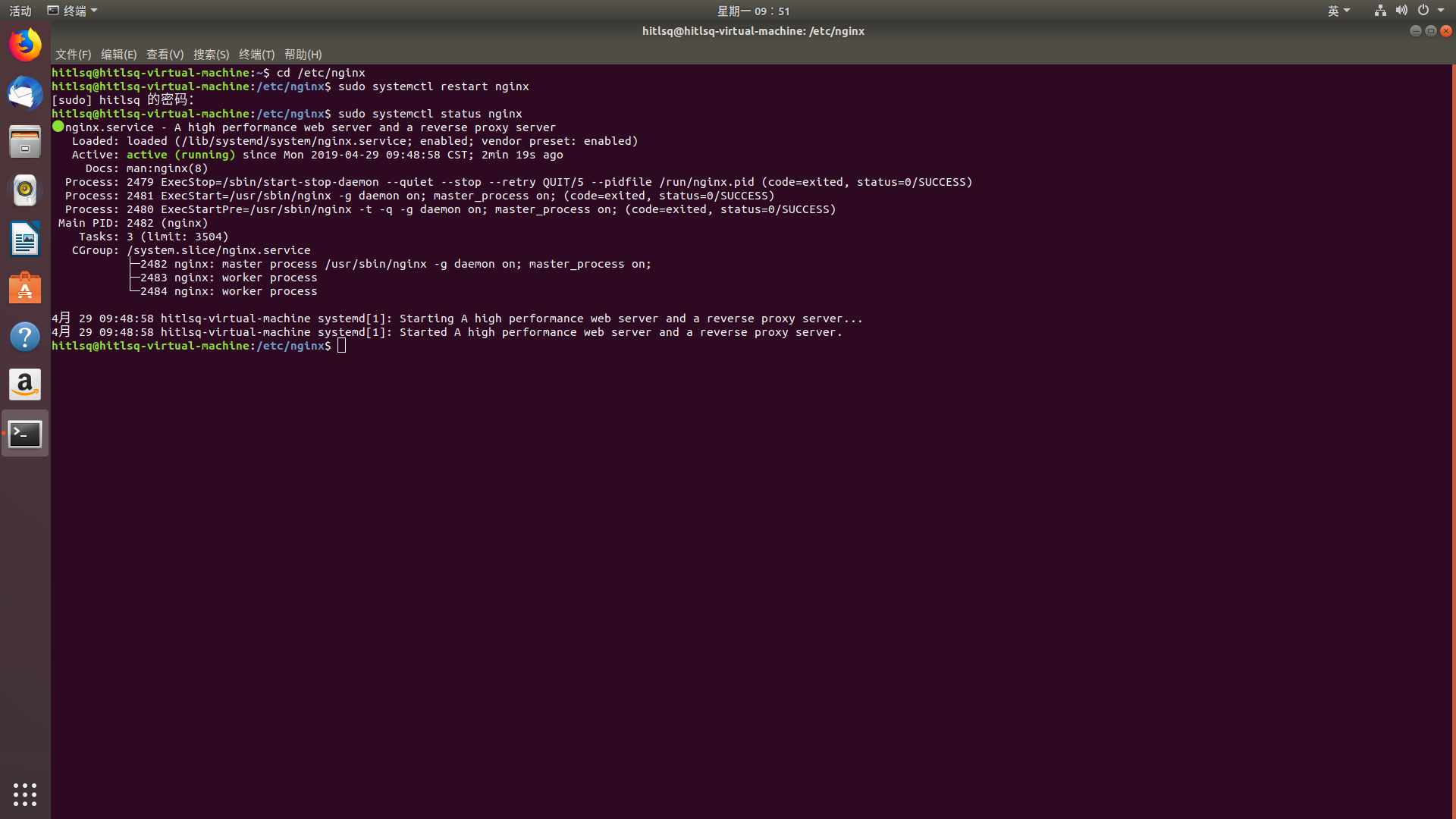
将原有的遗留Web系统改造为满足高性能、高可用、高可靠等质量属性的复杂软件系统。在数据层上，计划用React技术来重新设计Web系统的前端架构，并重写前端，使其具有React优秀的性能。在计算层上，计划使用Nginx负载均衡技术实现基于流量分发的完全分布式系统集群的负载均衡架构，同时使用Hadoop大数据计算技术实现海量用户访问请求的log日志统计。在数据层上，计划从系统的核心需求出发，重新设计数据库结构，对数据库结构在表层面上进行优化，所应用到的技术与思想包括Sharding-JDBC数据库分库分表技术，数据库关系模式范式理论以及数据库设计的ACID原则。同时还计划实现基于Redis的系统缓存架构，即所有的查询结果都放进了缓存，也就是把MySQL查询的结果放到了Redis中去，然后第二次发起该条查询时就可以从Redis中去读取查询的结果，从而不与MySQL交互，达到优化的效果，Redis的查询速度之于MySQL的查询速度相当于内存读写速度之于硬盘读写速度。

下图为系统改造的宏观设计架构图：



## 计算层上的系统改造——Nginx集群负载均衡的设计与实现

**1）搭建单虚拟机nginx**



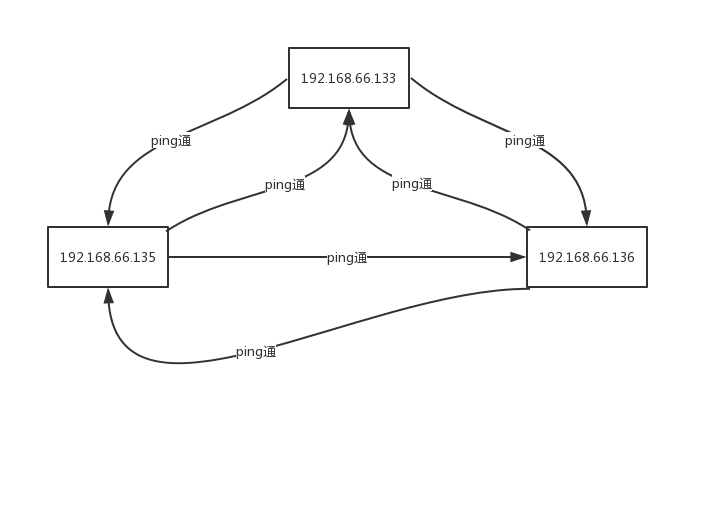
当出现如上图所示的结果时，表明nginx的配置文件修改成功，并且nginx已成功启动。

**2）搭建nginx虚拟集群环境，验证虚拟集群可以互相访问。**

给出三台虚拟机的性能配置：

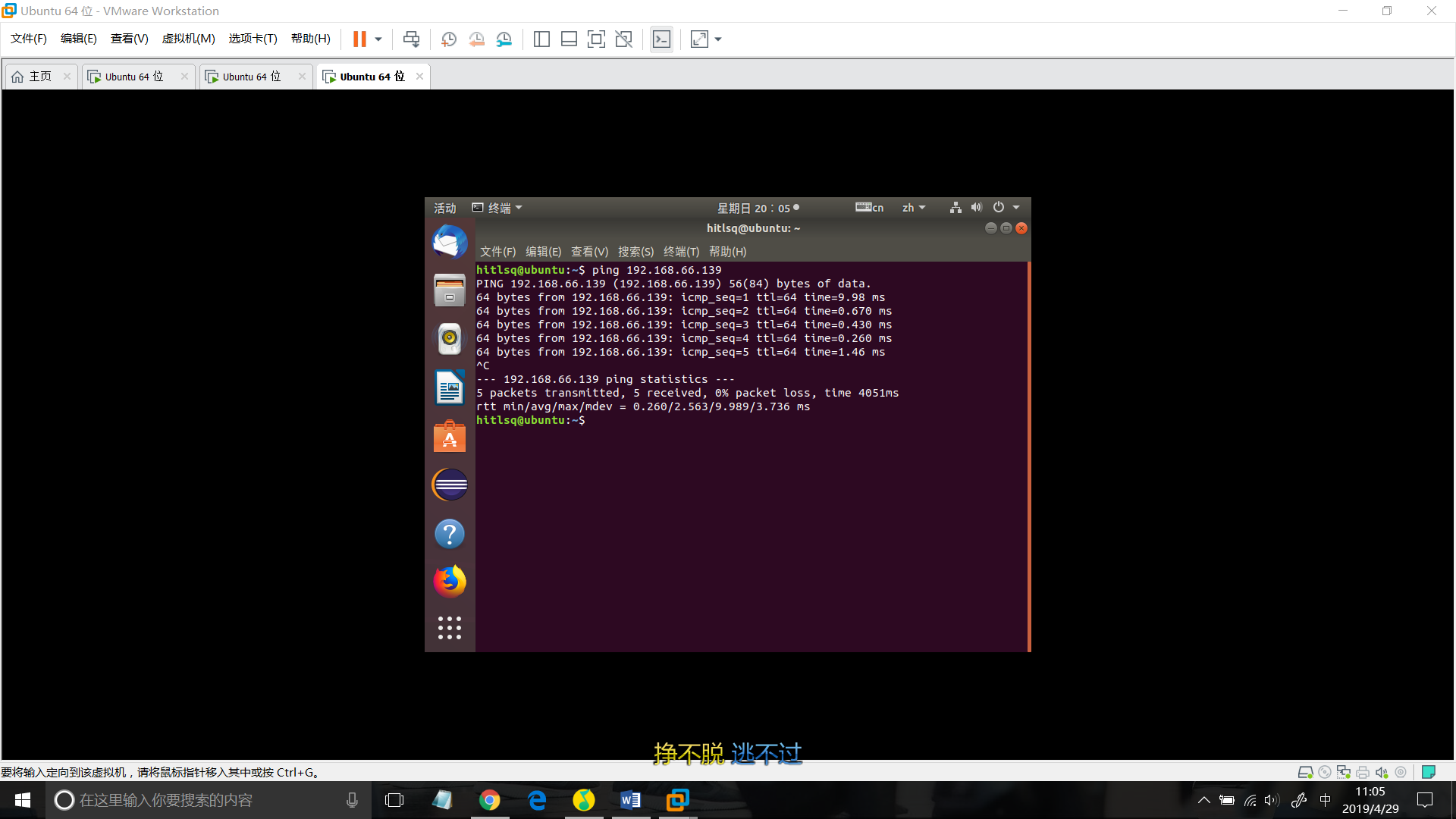
|  |  |  |  |
| --- | --- | --- | --- |
| 内存 | 处理器 | 硬盘 | 网络适配器 |
| 4GB | 4 | 60G | NAT |
| 3GB | 2 | 20G | NAT |
| 3GB | 2 | 20G | NAT |

验证虚拟机群可以互相访问的方式就是互ping其ip地址：



达到如上图所示的效果时才可以确保虚拟集群可以互相访问。

用192.168.66.133 ping 192.168.66.139为例：



这样说明192.168.66.133 可以访问192.168.66.139。最后依次验证之后，确定虚拟集群可以互相访问，搭建成功。

附：给出三台虚拟机的IP-功能及配置：

|  |  |
| --- | --- |
| IP | 功能 |
| 192.168.66.133 | Nginx负载均衡器 |
| 192.168.66.135 | Web服务器01 |
| 192.168.66.139 | Web服务器02 |

|  |  |
| --- | --- |
| IP | 配置 |
| 192.168.66.133 |  |
| 192.168.66.135 | 默认配置即可 |
| 192.168.66.139 | 默认配置即可 |

附：给出nginx负载均衡算法以及配置方法与结果

|  |  |  |
| --- | --- | --- |
| 负载均衡算法 | 配置方法 | 测试结果 |
| 默认算法（轮询调度） | 在upstream块中不添加额外的算法信息。 | 各个web服务器循环访问 |
| Least\_comm（最少连接） | 在upstream块中添加least\_conn | 优先访问not busy的服务器。 |
| Ip\_hash（ip哈希） | 在upstream块中添加ip\_hash | 在本次实验中，只有两个web服务器，所以在ip\_hash之后会根据结果只固定访问其中的一台。 |
| Weight balance（权重均衡） | 在upstream块中的服务器ip后添加权重值 | 假设为第一个服务器添加weight=5，为第二个服务器添加weight=2，那么访问的时候访问到两台服务器的概率比为5:2 |

**3） 应用负载均衡技术改造遗留的“进销存”系统，赋予支持海量用户的在线高并发请求的能力，请给出设计细节并分析负载均衡前后的区别。**

改造的步骤如下：

①在两台web服务器所在的ubuntu环境下配置mysql+eclipse jee+tomcat环境，具体的配置方法参考网上的博客即可，与本次实验主题关系不大，就不在这里陈述了。

②我用于测试的java web项目是一个学生信息管理系统，导入项目的sql文件到mysql中，然后在tomcat服务器上运行该系统（tomcat的端口号是8080）

③修改项目路径：更改tomcat服务器的server.xml配置文件，将配置文件中的path项设置为空，这样直接访问localhost:8080即可访问到项目，而无需在路径后添加项目名。这样做是便于后续步骤中在负载均衡器中配置upstream块。

④在负载均衡器中修改nginx.conf文件，将upstream块中的服务器的端口号从80修改为8080，负载均衡算法设置为least\_conn。

⑤重启nginx，在浏览器中访问负载均衡器的ip地址，即可看到学生信息管理系统的界面。

当真实面对海量用户的高并发请求的时候，负载均衡器可能会连接到很多台web服务器，将负载均衡策略设置为least\_conn，则请求会被优先发送到不那么“忙”的web服务器，这样就均衡了各个服务器的负载，不会出现一台服务器很忙，而另外的服务器闲置的情况。只有服务器足够多，性能足够，那么就可以承担高并发的访问请求。

负载均衡前后的区别：在负载均衡前，项目只在本地服务器上运行，如果服务器宕机或延迟过大，系统的访问效率就会降低甚至无法访问。但是利用nginx将对系统的访问请求分发到多台服务器，除非所有的服务器都宕机，否则不会出现项目无法访问的情况，这样提高了项目的稳定性。而且，least\_conn算法会使得请求优先发送到不那么“忙”、效率高的服务器，这样也提高了项目访问的效率。

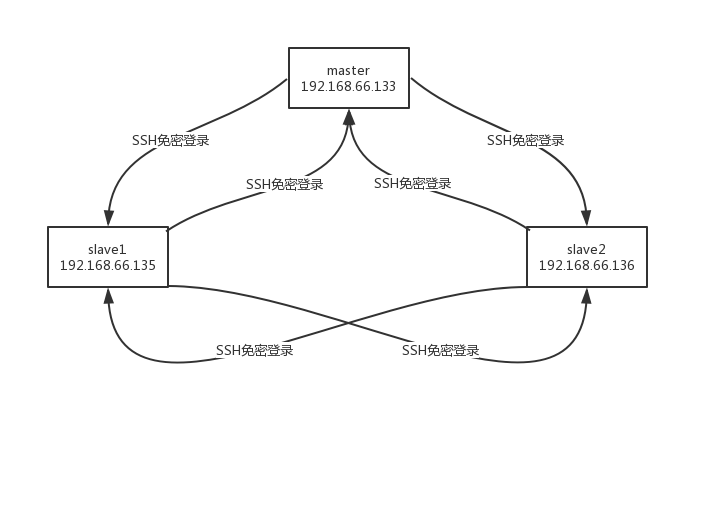
## 计算层上的系统改造——Hadoop大数据分布式计算技术实现大规模用户访问日志统计

1. **配置集群环境，设置主节点主机名为manager，从节点主机名为workerX(X为数字编号)**

附：给出虚拟机集群配置：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 虚拟机系统 | 节点机器名称 | IP地址 | 节点用户名 | 是否配置SSH |
| Ubuntu18.04 | Master | 192.168.66.133 | Hadoop | 是 |
| Ubuntu18.04 | Slave1 | 192.168.66.135 | Hadoop | 是 |
| Ubuntu18.04 | Slave2 | 192.168.66.139 | Hadoop | 是 |

然后配置集群SSH免密登录，具体步骤在前面的实验中已经给出。



1. **配置hadoop集群计算环境，数据块副本数为3。**

**配置hadoop集群计算环境需要如下几个步骤：**

①配置Java环境，安装JDK并且配置环境变量。

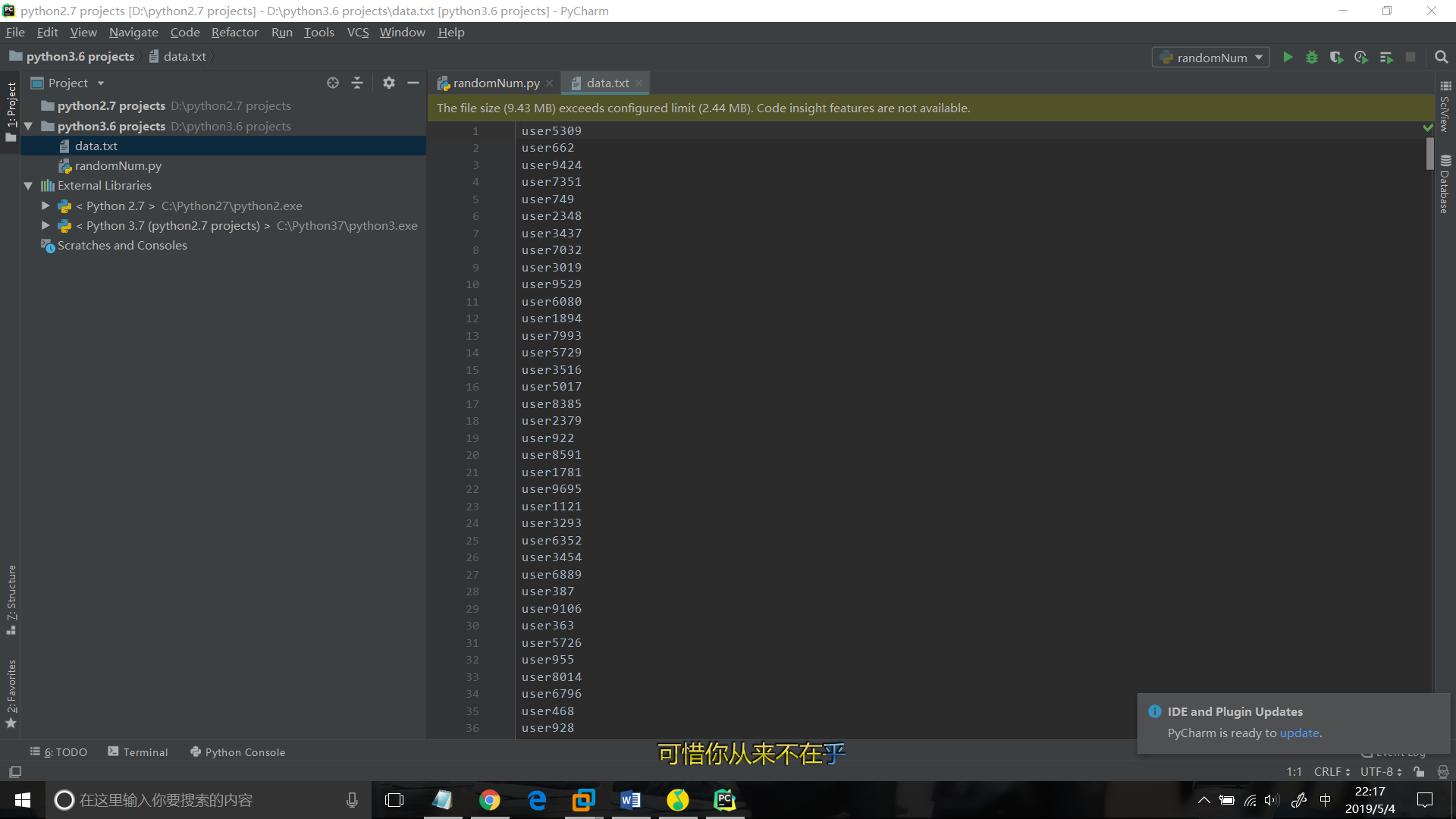
②安装hadoop

③配置hadoop（在/usr/local/hadoop/etc/hadoop路径下）

④将 master上配置好的hadoop所在文件夹/usr/local/hadoop复制到所有的Slave的/usr/local目录下

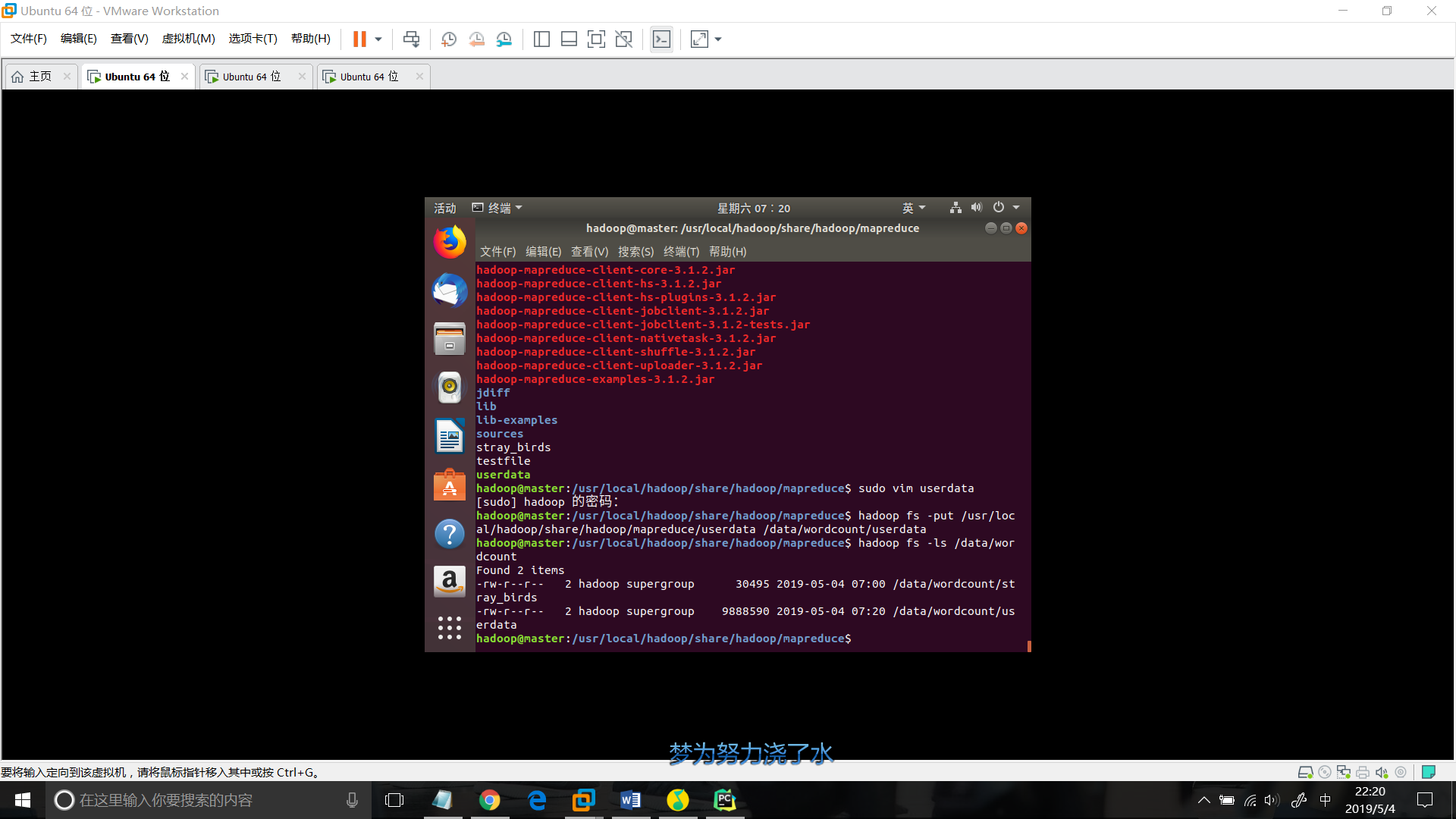
**3）使用hadoop集群环境计算遗留的“进销存”系统中海量用户的log日志中的访问统计。从1万个用户共100万次操作记录中计算每个用户的访问次数。**

① 获取并整理用户日志，user+编号为第N号user对系统进行了访问。下图为部分日志数据截图：

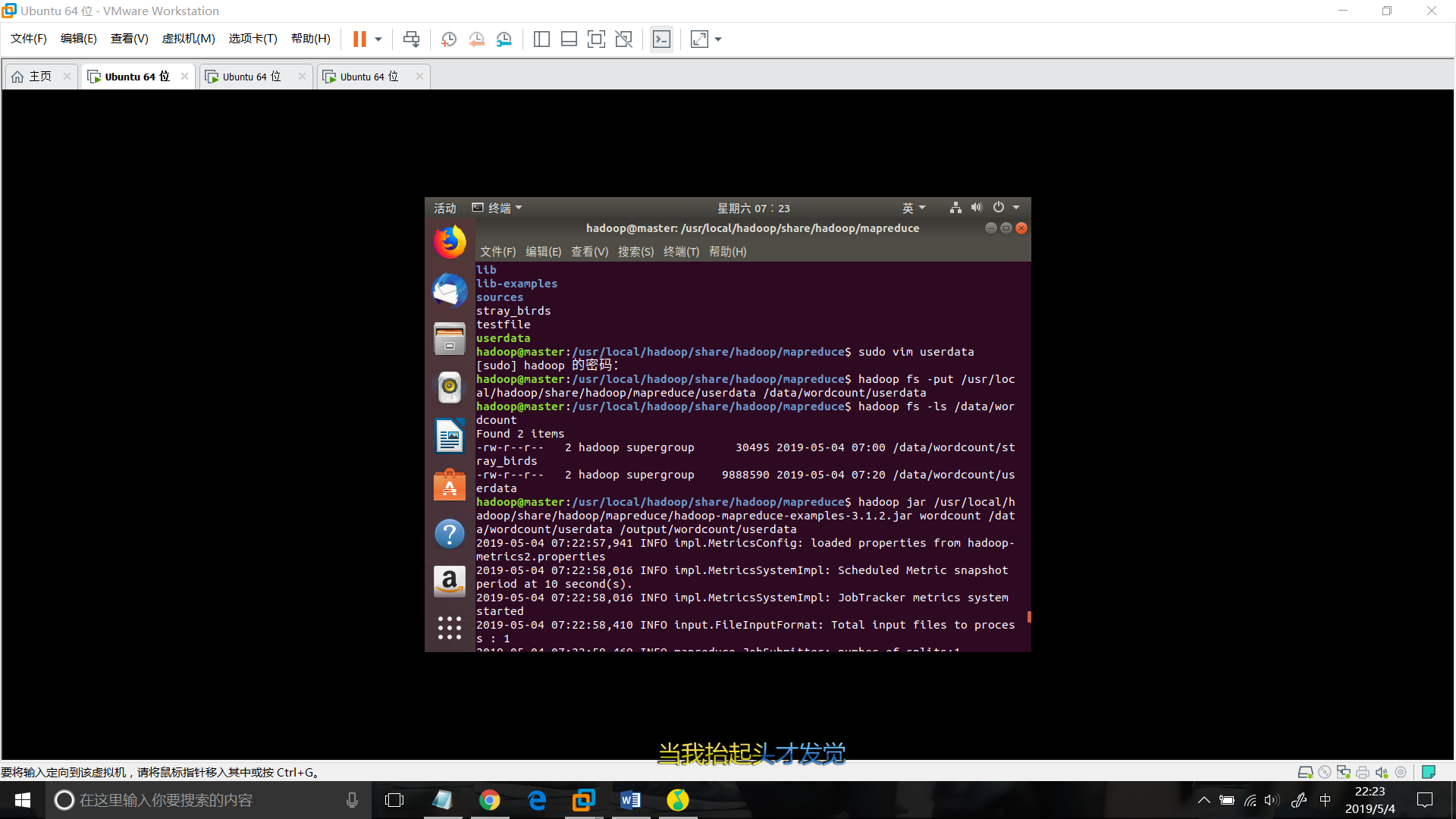


将整理后的日志文件保存到ubuntu虚拟机下的/usr/local/hadoop/share/hadoop/mapreduce目录中。

②将日志文件上传到HDFS中，然后查看上传后的文件情况：

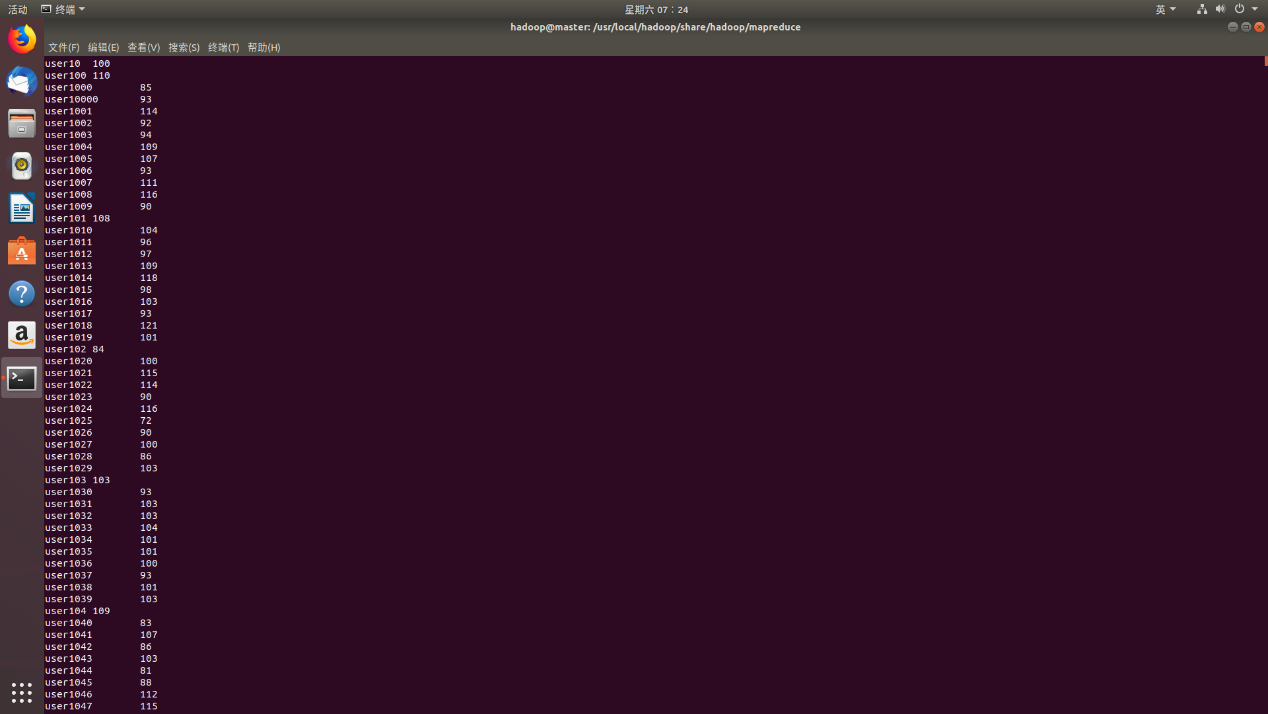


③执行以下命令运行wordcount：



④执行以下命令查看结果：

hadoop fs -text /output/wordcount/userdata/part-r-00000



这样就用wordcount方法计算出了每一个用户的访问次数，即完成了计算层面的大规模用户访问的日志统计。

## 数据层上的系统改造——web系统数据库分库分表的设计与实现

1. **基于数据库范式理论和ACID原则的数据库分表设计与实现**
2. **理论介绍：**
   1. **数据库范式理论：**首先要明白”范式（NF）”是什么意思。按照教材中的

定义，范式是“符合某一种级别的关系模式的集合，表示一个关系内部各属性之间的联系的合理化程度”。数据库范式分为1NF，2NF，3NF，BCNF，4NF，5NF。一般在我们设计关系型数据库的时候，最多考虑到BCNF就够。符合高一级范式的设计，必定符合低一级范式，例如符合2NF的关系模式，必定符合1NF。

接下来就对每一级范式进行一下解释，首先是第一范式（1NF）。

符合1NF的关系（可以理解为数据表。“关系模式”和“关系”的区别，类似于面向对象程序设计中”类“与”对象“的区别。”关系“是”关系模式“的一个实例，可以把”关系”理解为一张带数据的表，而“关系模式”是这张数据表的表结构。）1NF的定义为：符合1NF的关系中的每个属性都不可再分。第二范式（2NF）对1NF进行的改进是，2NF在1NF的基础之上，消除了非主属性对于码的部分函数依赖。第三范式（3NF） 3NF在2NF的基础之上，消除了非主属性对于码的传递函数依赖。BCNF在3NF的基础上消除主属性对于码的部分与传递函数依赖。

* 1. **ACID原则：**ACID原则是数据库事务正常执行的四个基本原则，分别指

原子性、一致性、独立性及持久性

事务的原子性(Atomicity)：是指一个事务要么全部执行，要么不执行，也就是说一个事务不可能只执行了一半就停止了。比如你从取款机取钱，这个事务可以分成两个步骤：1划卡，2出钱。不可能划了卡，而钱却没出来。这两步必须同时完成，要么就不完成。

事务的一致性(Consistency)：是指事务的运行并不改变数据库中数据的一致性。例如，完整性约束了a+b=10，一个事务改变了a，那么b也应该随之改变。

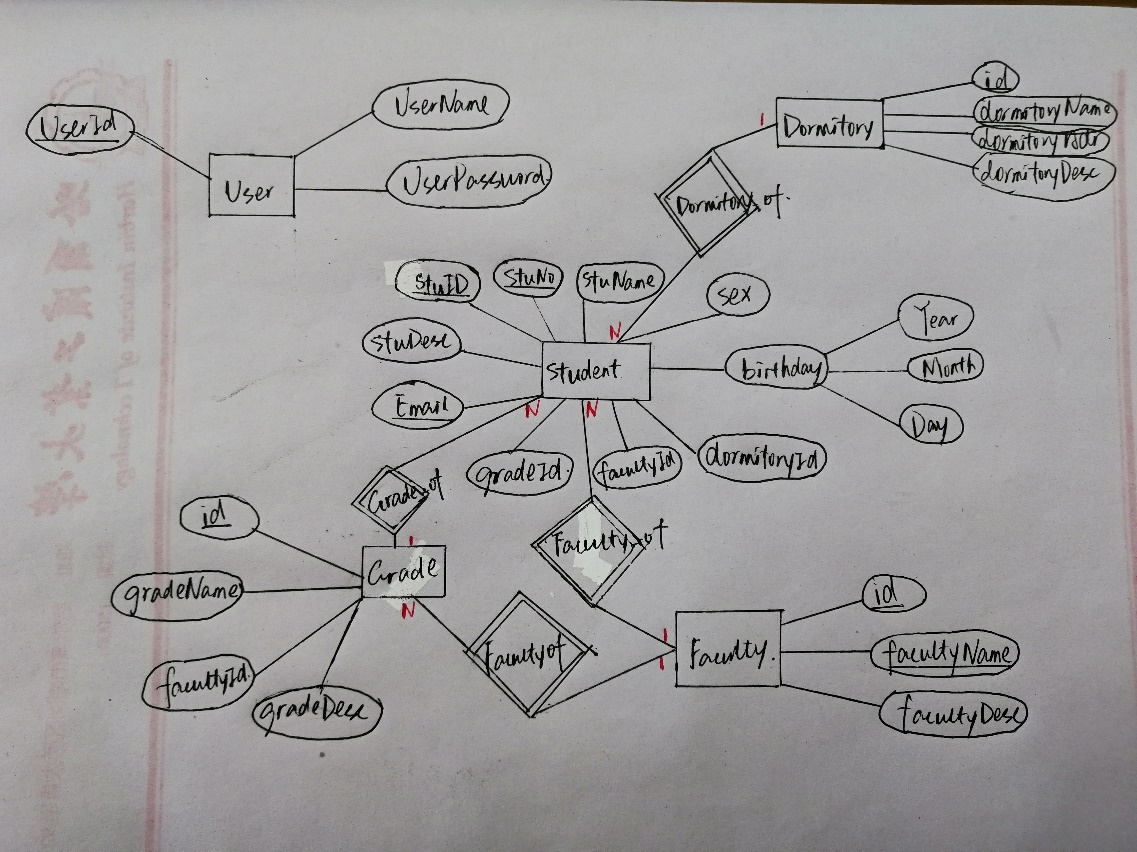
事务的独立性(Isolation）：事务的独立性也有称作隔离性，是指两个以上的事务不会出现交错执行的状态。因为这样可能会导致数据不一致。

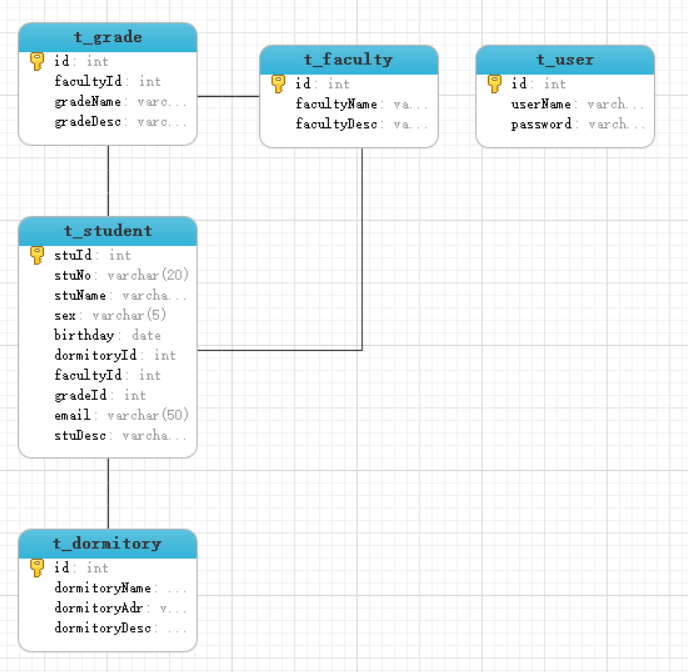
事务的持久性(Durability）：事务的持久性是指事务执行成功以后，该事务对数据库所作的更改便是持久的保存在数据库之中，不会无缘无故的回滚。

1. **实现手段**

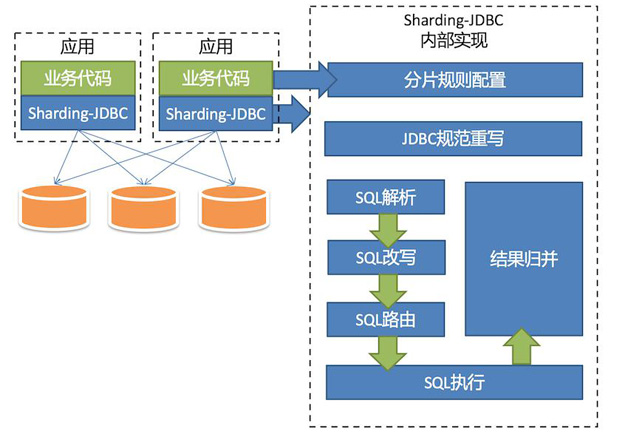
对数据库文件进行改写，将ER图中不同的实体用不同的表来存储，同时建立这些表之间的主外键联系，这样改写后的关系模式满足了第三范式的要求。

ER图如下：





1. **基于sharding-JDBC的数据库分库分表的设计与实现**
2. **理论介绍**
   1. **Sharding-JDBC的分库分表原理**



**分片规则配置**

Sharding-JDBC的分片逻辑非常灵活，支持分片策略自定义、复数分片键、多运算符分片等功能。如：根据用户ID分库，根据订单ID分表这种分库分表结合的分片策略；或根据年分库，月份+用户区域ID分表这样的多片键分片。

Sharding-JDBC除了支持等号运算符进行分片，还支持in/between运算符分片，提供了更加强大的分片功能。

Sharding-JDBC提供了spring命名空间用于简化配置，以及规则引擎用于简化策略编写。由于目前刚开源分片核心逻辑，这两个模块暂未开源，待核心稳定后将会开源其他模块。

**SQL改写**

SQL改写分为两部分，一部分是将分表的逻辑表名称替换为真实表名称。另一部分是根据SQL解析结果替换一些在分片环境中不正确的功能。这里具两个例子：

第1个例子是avg计算。在分片的环境中，以avg1 +avg2+avg3/3计算平均值并不正确，需要改写为（sum1+sum2+sum3）/（count1+count2+ count3）。这就需要将包含avg的SQL改写为sum和count，然后再结果归并时重新计算平均值。

第2个例子是分页。假设每10条数据为一页，取第2页数据。在分片环境下获取limit 10, 10，归并之后再根据排序条件取出前10条数据是不正确的结果。正确的做法是将分条件改写为limit 0, 20，取出所有前2页数据，再结合排序条件算出正确的数据。可以看到越是靠后的Limit分页效率就会越低，也越浪费内存。有很多方法可避免使用limit进行分页，比如构建记录行记录数和行偏移量的二级索引，或使用上次分页数据结尾ID作为下次查询条件的分页方式。

**SQL路由**

SQL路由是根据分片规则配置，将SQL定位至真正的数据源。主要分为单表路由、Binding表路由和笛卡尔积路由。

单表路由最为简单，但路由结果不一定落入唯一库（表），因为支持根据between和in这样的操作符进行分片，所以最终结果仍然可能落入多个库（表）。

Binding表可理解为分库分表规则完全一致的主从表。举例说明：订单表和订单详情表都根据订单ID作为分片键，任意时刻分片逻辑均相同。这样的关联查询和单表查询难度和性能相当。

笛卡尔积查询最为复杂，因为无法根据Binding关系定位分片规则的一致性，所以非Binding表的关联查询需要拆解为笛卡尔积组合执行。查询性能较低，而且数据库连接数较高，需谨慎使用。

**SQL执行**

路由至真实数据源后，Sharding-JDBC将采用多线程并发执行SQL，并完成对addBatch等批量方法的处理。

* 1. 考虑使用sharding-JDBC的原因：IDEA配置sharding-JDBC较为方便，

而且分片原理也与业务逻辑契合。

1. **实现手段**

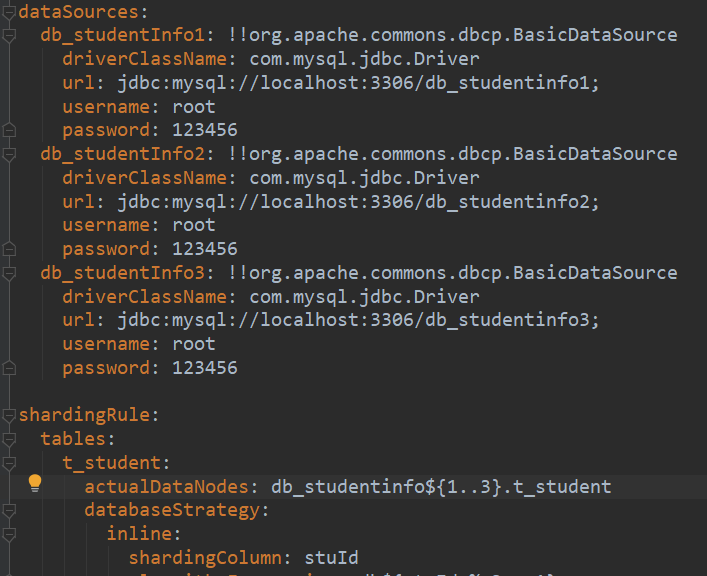
将数据库中的t\_student表切分为三个数据源db\_studentInfo${1..3}下的t\_student表，这样的切分方式可以满足单个数据库存储学生信息较多时候导致数据库容量不足等影响性能的情况。分片规则暂且采用根据学生的ID分片。这是在系统还不成熟的情况下，如果系统的功能很成熟，比如哈工大的教务系统，就可以采用根据学生学号的第4-6位的哈希值进行分片，将哈希值相同的学号分布到同一个数据源的数据表中，这样可以在根据ID切分的基础上实现把具有相同某属性（比如学院）的学生切分到同一数据源中，这样不仅仅可以释放单个数据库的存储空间，提高性能，并且可以优化查询的效率。

配置完sharding-JDBC之后需要根据配置改写代码后端的数据库存取逻辑。数据库实例需要建立三个数据库连接，分别映射到切分的三个数据源，三个数据源的数据关系模式是相同的。

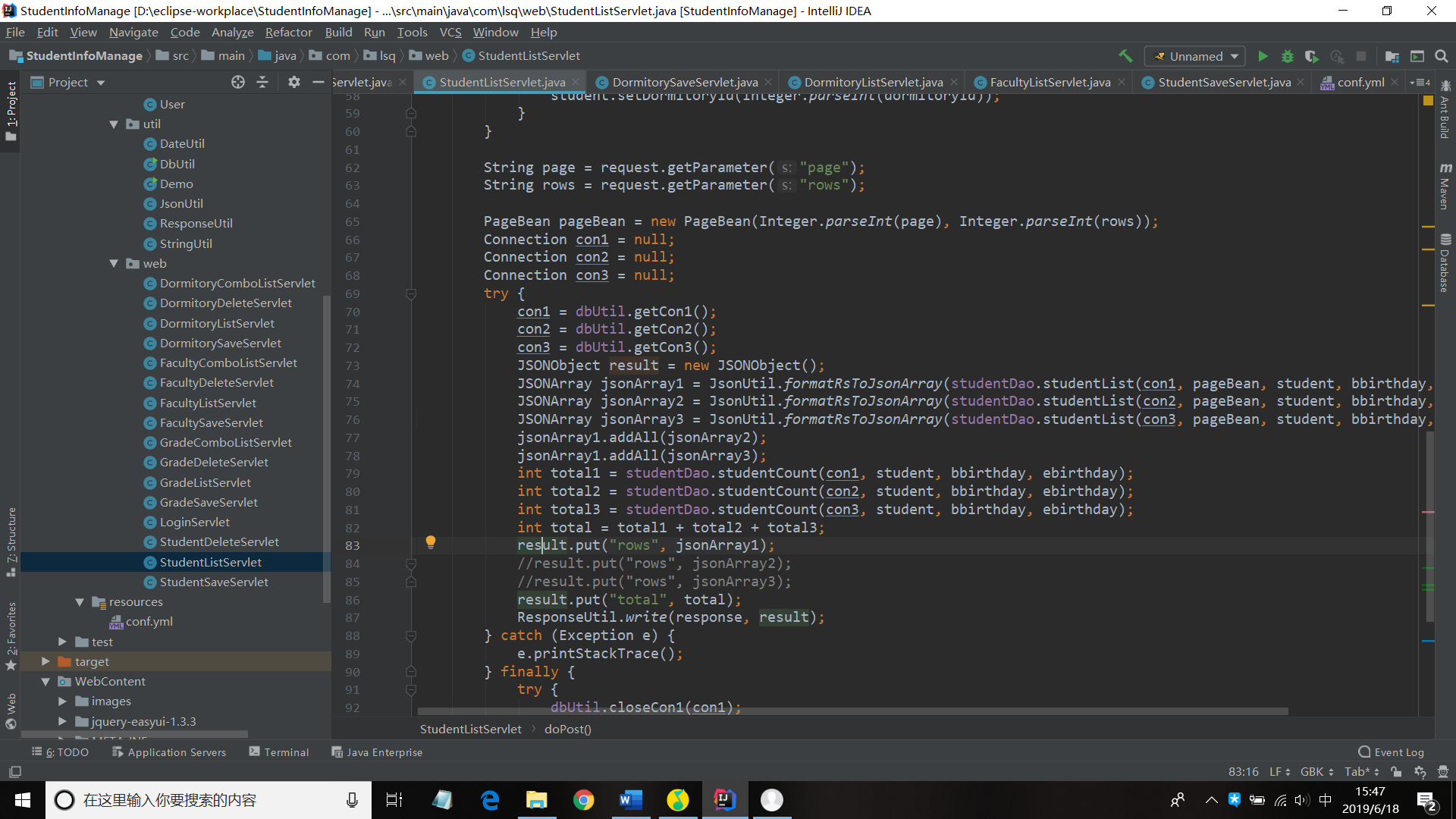
插入数据的时候根据学生的ID进行切分，学生的ID是不同于学生的学号的一个特定的t\_student表的主键，这个主键在现代数据库理论的专业术语也叫做代理键，具体的原理与本课程无关，就不在此详述了。采取的切分算法是学生的ID对3取余后加1，例如学生的ID是1，那么1 % 3 + 1 = 2，则将这个学生的信息存入db\_studentInfo2中，即第二个数据源中。

在查询数据和显示数据的时候需要更改系统后端逻辑，即分别在三个数据库上进行查询，将查询的结果整合到一起。统计的时候也需要统计每个数据库中的元组条目最后加到一起。

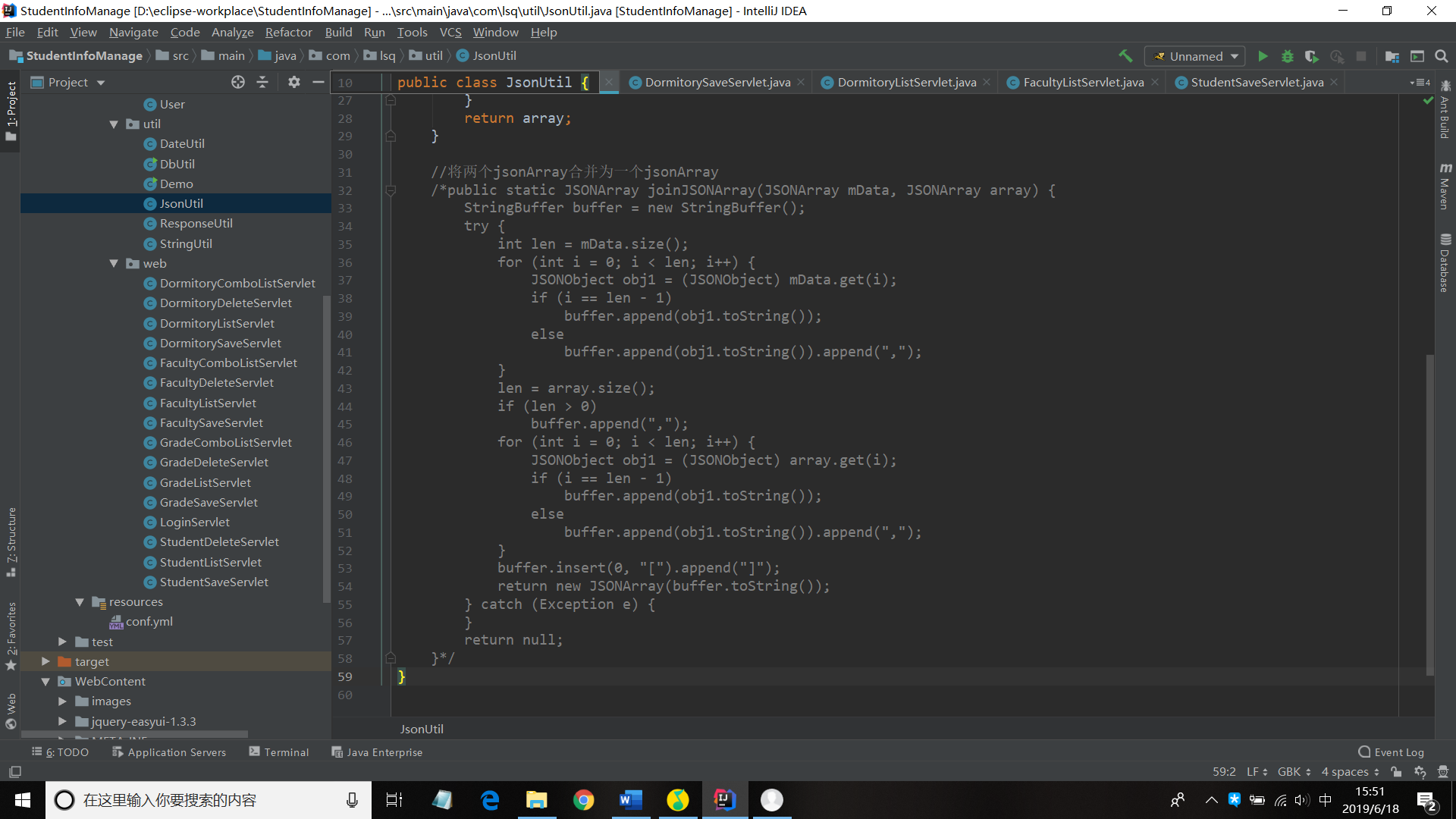
下图给出sharding-JDBC的关键配置config.yml，存放在项目目录中的resource目录下：



给出关键代码细节截图：



在显示学生信息列表时需要建立三个数据库的连接，将三个数据库中的学生信息分别提取到JSONArray中，并分别统计元组数量并加到一起，最后将三个JSONArray合并然后放入ResultSet中显示出来。在修改关键代码的时候遇到了很多的技术难题，其中一个就是多个JSONArray的合并，由于其内部结构的原因，单独编写函数对其进行合并很困难，起初我在JsonUtil.java中做出了如下的尝试：



但是在字符串转换的时候出现了问题。最后我选择采用阿里巴巴开源的JSON相关库fastjson来实现JSONArray的合并，导入fastjson.jar之后直接使用.addAll()方法即可完成这一功能。我们在<https://github.com/alibaba/fastjson>即可查看开源代码及下载相关jar包。

实现完成之后经测试，系统已经成功的根据插入学生的ID对学生信息进行了分片存储。

## 数据层上的系统改造——系统查询的缓存结构的设计与实现

## 表示层上的系统改造——React前端框架的重写

React 使创建交互式 UI 变得轻而易举。为应用的每一个状态设计简洁的视图，当数据改变时 React 能有效地更新并正确地渲染组件。

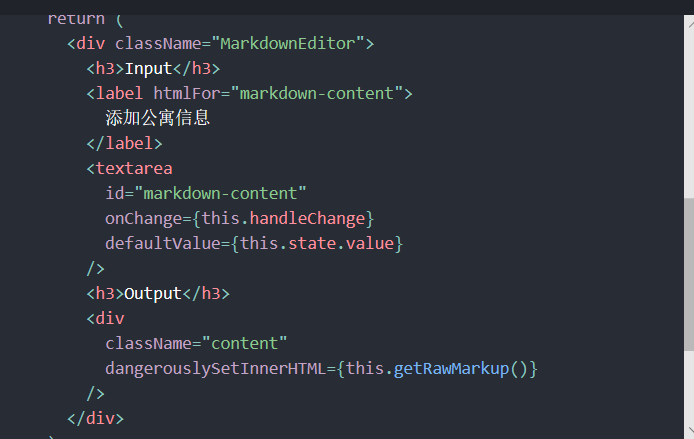


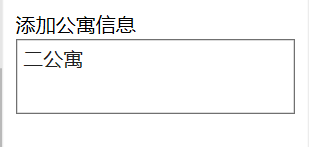
使用 props 和 state，创建一个简易的 Todo 应用





使用了一个名为 **remarkable** 的外部 Markdown 库。它可以实时转换 <textarea> 里的内容。





添加查询，搜索，修改功能

## 改造前后的系统性能分析

1. Nginx分布式负载均衡集群的性能分析

性能的瓶颈在于服务器的并发能力。

可以举一个很简单的例子，在遗留的进销存系统中，如果同时有3000000万个用户访问系统，而服务器的承受能力仅仅为同时支持1500000万个访问请求，那么就会发生服务器满载甚至过载导致TCP请求缓慢甚至宕机的情况，但是在Nginx负载均衡集群中，如果负载均衡器采用least\_conn算法将请求调度到三台服务器中，那么每台服务器平均只会承受1000000个并发访问量，每台服务器的负载量仅仅只达到了66.7%，而相同情况下原遗留系统的服务器负载量达到了200%，而Nginx负载均衡器的算法执行与代理转发的耗时完全可以忽略不计，显然，采用负载均衡的完全分布式集群可以大幅度降低每台服务器的负载，提高访问的速率。理论上，不考虑服务器的地理位置等因素的情况下，集群中的服务器越多，那么对访问请求的优化效果就会越明显。

1. Hadoop的性能分析

Hadoop集群是一种专门为存储和分析海量非结构化数据而设计的特定

类型的集群。本质上，它是一种计算集群，即将数据分析的工作分配到多个集群节点上，从而并行处理数据。

在大规模用户访问日志统计时，Hadoop有如下的优势：

一、适于大数据处理。大数据一般都是分布广泛并且是非结构化的。而Hadoop非常适合这类数据是因为Hadoop的工作原理在于将数据拆分成片，并将每个“分片”分配到特定的集群节点上进行分析。数据不必均匀分布，因为每个数据分片都是在独立的集群节点上进行单独处理的。

二、灵活的可扩展性。和其它任何类型的数据一样，大数据分析面临的一个重要问题也是数据量的不断增加。而且大数据最大的优势在于可以实时或接近实时地进行分析处理。而Hadoop集群的并行处理能力能明显提高分析速度，但随着要分析的数据量的增加，集群的处理能力可能会收到影响。但是，通过添加额外的集群节点可以有效的扩展集群。

三、低成本。Hadoop集群较为廉价有两个主要原因。它所需的软件是开源的，这样就可以降低成本。事实上，可以自由下载Apache Hadoop发行版。同时，Hadoop集群通过支持商用硬件控制了成本。不必购买服务器级硬件，便可以搭建一个强大的Hadoop集群。所以，事实证明，Hadoop集群的确是一个高性价比的解决方案。

四、故障容错能力。当一个数据分片发送到某个节点进行分析时，该数据在集群其它节点上会有副本。通过这种方式，即使一个节点发生故障，该节点数据的额外拷贝仍存在于集群内的其它地方，这样，数据仍可以进行分析处理。

3）数据库分库分表的性能分析

经过数据库的分库分表，数据库的压力被降为了原来的1/3，缓解了单表的大数据量给索引查询带来的压力，以及同一数据源内大规模的数据请求给系统带来的压力。

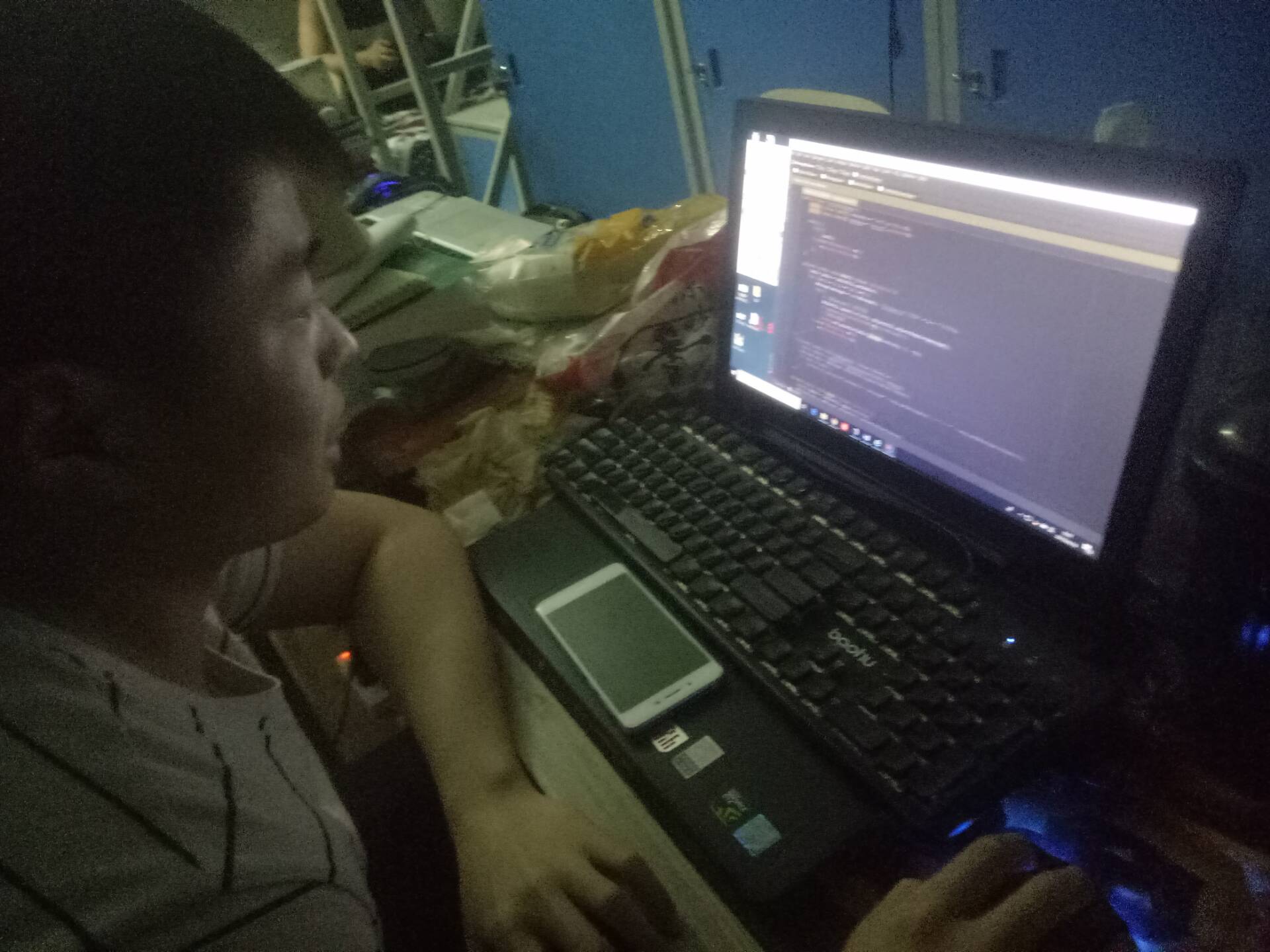
# 结对开发过程记录

**（1）角色切换与任务分工**

表1-1结对开发角色与任务分工

| 日期 | 时间(HH:MM - HH:MM) | 驾驶员角色 | 领航员角色 | 本段时间的任务 |
| --- | --- | --- | --- | --- |
| 2019/6/16 | 18:00-20:00 | 刘思琦 | 石昊 | 配置环境 |
| 2019/6/17 | 18:00-21:00 | 刘思琦 | 石昊 | 进行计算层改造 |
| 2019/6/18 | 18:00-21:00 | 刘思琦 | 石昊 | 进行数据层改造 |
| 2019/6/19 | 18:00-21:00 | 石昊 | 刘思琦 | 进行表示层改造 |

**（3）结对开发工作现场照片**



# 实验总结

在本次实验中克服了很多技术上的问题，将原有的遗留系统改造成了具有高性能、高可用、高可靠等质量属性的复杂软件系统，在改造系统的设计过程中体会到了架构的设计思想，以及在软件系统的开发过程中更加注意到架构对一个复杂的软件系统带来的性能上的影响。感受就是，软件系统的开发，远远不是采用某某框架，比如ssm，spring，spring-boot等等，而是从需求分析到架构设计到代码实现到评审测试到系统优化等一系列复杂的过程，如果视野仅仅停留在代码层面上，那么在以后的工作中，就只能作为一个低级的coder，低级的生产力，只有将视野上升到系统架构的粒度，掌握高并发等提高性能的技术，才可以在成为优秀程序员甚至架构师的路上更进一步。