哈尔滨工业大学

**计算学部**

**2024年秋季学期**

**《软件架构与中间件》课程**

**实验报告**

**Lab 2：计算层与数据层软件架构实验**

|  |  |  |
| --- | --- | --- |
| **姓名** | **学号** | **联系方式** |
| 蔡志宇 | 2022112831 | phantasia\_march@outlook.com |
| 杨昊岩 | 2022110553 | childking0909@gmail.com |

目 录

[1 实验概述 1](#_Toc180794331)

[1.1 实验目的 1](#_Toc180794332)

[1.2 实验要求 1](#_Toc180794333)

[2 实验内容与过程 1](#_Toc180794334)

[2.1 nginx集群负载均衡实验 1](#_Toc180794335)

[2.2 hadoop分布式计算实验 24](#_Toc180794336)

[2.3 Mycat数据库分库分表实验（我们选择了2.4进行实验内容） 24](#_Toc180794337)

[2.4 Sharding-JDBC数据库分库分表实验 25](#_Toc180794338)

[2.5 Redis数据缓存实验 28](#_Toc180794339)

[3 结对开发过程记录 32](#_Toc180794340)

[4 实验总结 33](#_Toc180794341)

[5 教师评语 33](#_Toc180794342)

[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”]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **学号：** | **2022112831** | **姓名：** | **蔡志宇** | |
| **学号：** | **2022110553** | **姓名** | **杨昊岩** | |
| 实验概述 | | | |  |
| 实验目的 1）学习构建集群的方法  2）学会使用典型的负载均衡框架  3）学会使用Hadoop 进行map-reduce编程  4）学会使用MyCat 或 Sharding-JDBC进行分库分表等数据架构技术  5）学会使用 Redis进行数据缓存优化  6）能够灵活应用计算/数据密集型中间件到实际系统 实验要求 1）2人结对成组  2）实验2.1必做，2.2选做，2.3和2.4两选一，2.5必做  3）结合前序课程中“进销存”管理类系统(或其他实际软件系统)进行计算层架构重构，支持海量用户的在线高并发请求场景  4）应给出关键过程的细节 | | | | |
| 实验内容与过程 | | | |  |
| nginx集群负载均衡实验  1. 请给出搭建单虚拟机nginx的关键过程，并修改nginx主机的首页内容从英文变为中文。   在Virtual Box 中安装Ubuntu 22.04虚拟机后，首先需要设置虚拟机为桥接网卡模式，以便虚拟机和宿主机存在于同一网段中，从而能够实现分流负载。    使用命令sudo apt install nginx即可安装nginx，通过ip addr获得当前虚拟机的ip地址，    在宿主机浏览器中输入此ip地址可以查看nginx页面    Nginx的默认页面位置存在于主机的/var/www/html/index.html中，通过修改文件中的内容，重启nginx即可发现页面内容改变。首先需要更改此html页面中的字符集为utf-8。    其次还需要修改nginx的配置文件中的字符集，nginx的配置文件位于/etc/nginx/site-enabled/default 中，添加charset utf-8.    输入虚拟机的ip，可以查看到当前的Nginx页面已经存在中文。     1. 请详析配置文件中涉及的upstream、location和server块。   在具体的配置文件/etc/nginx/nginx.conf中。为了整体的学习流程，我们完整的介绍所有的配置文件内容，不局限于server块中的upstream、location等内容。整个nginx配置文件的结构大致如下：    **首先是全局块的内容**。全局块是默认配置文件从开始到events块之间的一部分内容，主要设置一些影响Nginx服务器整体运行的配置指令，因此，这些指令的作用域是Nginx服务器全局。通常包括配置运行Nginx服务器的用户（组）、允许生成的worker process数、Nginx进程PID存放路径、日志的存放路径和类型以及配置文件引入等。    **然后是event块**。events块涉及的指令主要影响Nginx服务器与用户的网络连接。常用到的设置包括是否开启对多worker process下的网络连接进行序列化，是否允许同时接收多个网络连接，选取哪种事件驱动模型处理连接请求，每个worker process可以同时支持的最大连接数等。这一部分的指令对Nginx服务器的性能影响较大，在实际配置中应该根据实际情况灵活调整。    **然后是Http块**。http块是Nginx服务器配置中的重要部分，代理、缓存和日志定义等绝大多数的功能和第三方模块的配置都可以放在这个模块中。  前面已经提到，http块中可以包含自己的全局块，也可以包含server块，server块中又可以进一步包含location块，在本书中我们使用“http全局块”来表示http中自己的全局块，即http块中不包含在server块中的部分。  可以在http全局块中配置的指令包括文件引入、MIME-Type定义、日志自定义、是否使用sendfile传输文件、连接超时时间、单连接请求数上限等。    **接下来是对Server块的介绍。**  server 块定义了一个虚拟主机，用于处理特定的域名或 IP 请求。每个 server 块包含多个配置指令来指定该主机的行为，比如监听的端口、主机名和处理请求的规则。  常见的配置如下   1. server { 2. listen 80;  # 监听 80 端口 3. server\_name www.example.com;  # 匹配域名 5. root /var/www/html;  # 设置网站根目录 6. index index.html index.htm;  # 指定默认首页文件 8. location / { 9. try\_files $uri $uri/ =404;  # 文件存在则访问，否则返回404 10. } 12. error\_page 404 /404.html;  # 自定义 404 页面 13. }   **常用指令：**   * listen：指定服务器监听的端口或 IP 地址。 * server\_name：定义虚拟主机匹配的域名，支持通配符。 * root：指定静态文件的根目录。 * index：定义默认访问的首页文件名。 * error\_page：指定错误状态码的自定义页面。   **然后是关于Location块内容的介绍**。location 块用于定义 URL 路径匹配规则以及路径下请求的处理逻辑。它是 server 块中的子配置，允许为不同的路径或文件类型设置不同的处理方式。  示例配置：   1. server { 2. listen 80; 3. server\_name www.example.com; 5. location / { 6. root /var/www/html; 7. index index.html; 8. } 10. location /images/ { 11. root /data/images; 12. } 14. location ~\* \.(jpg|jpeg|png|gif|ico|bmp)$ { 15. expires 30d;  # 缓存 30 天 16. } 17. }   常见配置指令   * root：设置处理该路径的根目录。 * index：定义默认访问的文件。 * try\_files：用于检查多个文件是否存在并按顺序尝试。 * proxy\_pass：用于将请求转发到上游服务器。 * expires：设置缓存策略。   **最后是关于upstream块的介绍。**upstream 用于定义反向代理的上游服务器组。Nginx 通过反向代理将请求转发到后端的多个服务器，实现负载均衡和高可用。  Upstream块的语法格式如下   1. upstream backend { 2. server 192.168.1.101:8080;  # 上游服务器1 3. server 192.168.1.102:8080;  # 上游服务器2 4. server 192.168.1.103:8080;  # 上游服务器3 5. }   搭配上server块的完整示例如下：   1. upstream backend { 2. server 192.168.1.101:8080; 3. server 192.168.1.102:8080; 4. server 192.168.1.103:8080; 5. } 7. server { 8. listen 80; 9. server\_name www.example.com; 11. location / { 12. proxy\_pass http://backend;  # 请求转发到上游服务器组 13. proxy\_set\_header Host $host;  # 设置请求头 14. proxy\_set\_header X-Real-IP $remote\_addr; 15. proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for; 16. } 17. }   常见的指令：   * server：定义一个上游服务器，可以指定 IP 和端口。 * weight：为服务器设置权重，权重高的服务器会分配更多请求。 * max\_fails：设置最大失败次数，超过该次数后暂时不再请求该服务器。 * fail\_timeout：设置失败后的暂停时间。 * proxy\_pass：指定将请求转发到上游服务器或服务器组。   3）请描述所搭建的虚拟集群环境，验证虚拟集群可以互相访问。  使用VirtualBox 安装了一台Ubuntu22.04LTS系统的虚拟机，以及一台kali2024.2虚拟机和一台kali2024.3虚拟机。使用网口桥接模式从而每一台虚拟机获得了各自的与宿主机一致的网段下的ip地址。分别查看四台主机的地址：    我们分别让四台主机分别ping 另一台，形成一个回环，即可认为虚拟集群可以互相访问。由此我们可以得出结论，虚拟集群之间可以相互连接。    4）配置集群的负载均衡系统，给出关键过程，验证负载均衡系统是否工作。  首先配置nginx的配置文件，由于在nginx.conf中包含了所有conf.d目录下的.conf文件，所以我们只需要在conf.d目录下新建conf文件，并且写入均衡负载的upstream块以及server块代码即可。保存文件后重启nginx。    分别在另外两个虚拟机中部署好中间件，我们在publish接口出加入具体的注释，来标识当前的宿主机的http请求经过nginx均衡负载后究竟发往了哪一台服务器。    主机运行的客户端的发送的http请求的url更改为nginx监听的ip和端口：    当我们第一次发送消息的时候，server one中的中间件接受并处理了消息：    当我们第二次发送消息的时候，server two中的消息中间件接受并处理了消息：    由此可见，我们的分流系统已经正常工作。  5）配置并验证使用不同集群负载均衡算法（至少三种），给出过程、观测的情况，给出自己对负载均衡算法的理解和认识。  查阅资料得知，nginx内置了多种负载均衡策略。包括：轮询策略、加权轮询策略、ip\_hash策略、最少连接策略、url\_hash策略、fair策略和Sticky策略。   1. 轮询策略   该策略就是服务器将每个前端请求按顺序（时间顺序和排列次序）逐一分配到不同的后端服务器节点。如果后端服务器出现问题，即down掉，那么就会被自动剔除。  Nginx的默认策略就是轮询策略，就像我们在第四步中实现的那样，第一条消息被分配给了server one，第二条消息被分配给了server two。并不需要进行更多的配置。  **可以查看过程四中的具体的观测情况。这里不过多赘述。**   1. 加权轮询策略   该策略在基本的轮询策略基础上考虑各后端服务器节点接受请求的权重，指定各后端服务器节点被轮询到的机率，主要应用于后端服务器节点性能不均的情况。  例如：通过直接配置weight来设置访问机率，weight的大小和访问比率成正比。下面三个服务器（如果不配置weight，则默认配置为weight=1），第一个的权重是1，第二个的权重是3，第三个的权重是2，那么这三个后端服务器被访问的比率是1:3:2，即server172.31.3.82:9171被访问的机率最高，server172.31.3.82:9171次之，server172.31.3.82:9170访问的机率最小。示例配置：     1. Ip\_hash策略   该策略是将前端的访问IP进行hash操作，然后根据hash结果将请求分配到不同的后端服务器节点。这样会使得每个前端访问IP会固定访问一个后端服务器节点，好处是前端用户的session只在一个后端服务器节点上，不必考虑一个session存在多台服务器节点出现session贡献问题。  例如：因为weight是内置，所以可以直接和其他策略配合使用。本策略使用的是ip\_hash策略，需要在配置upstream中添加ip\_hash一行。示例配置：     1. 最少连接(least\_conn)策略   基于最少连接的负载平衡是另一种简单的方法。顾名思义，此方法将请求定向到当时具有最少活动连接的服务器。对于请求有时可能需要更长时间才能完成的应用程序，它比循环法更有效。   1. url\_hash策略   该策略将前端请求的url地址进行hash操作，根据hash结果将请求定向到同一后端服务器节点上，后台服务器为缓存是比较有效。一般url\_hash需要配合缓冲命中来使用。示例配置：     1. fair策略   该策略请求转发到负载最小的后端服务器节点上。Nginx通过后端服务器节点对响应时间来判断负载情况，响应时间最短的节点负载就相对较轻，Nginx就会将前端请求转发到此后端服务器节点上。示例配置：     1. Sticky策略   该策略在多台服务器的环境下，为了确保一个客户端只和一台服务器通讯，它会保持长连接，并在结束会话后再次选择一个服务器，保证了压力均衡。示例配置：    **具体的策略观测**   * + 1. 轮询策略：在不进行任何有关负载均衡的设置的时候。Nginx默认的配置就是轮询策略，会将请求轮流发送给不同的服务器。如下图所示。      * + 1. 加权轮询策略   首先我们修改nginx的覆盖均衡策略为加权轮询策略，为了效果更佳明显，我们将权重分别设置为1和5，然后重新启动nginx。    我们使用jmeter进行11次测试    在控制台中我们可以看见有9次请求被分流给了server two，两次请求分流给了server one     * + 1. 最少连接策略   设置好nginx服务器的配置文件，并重新启动nginx服务器：    使用jmeter发送50条请求给nginx服务器：    由于两台虚拟机的配置类似，并且实质上我们没有多余的测试用户，所以测出的结果基本上是两台服务器均衡的分配了所有的消息。    6） 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。  **整体的实现架构**     1. **对系统的改造**   我们的基于实验一中的智能体聊天系统，以及对应的日志存储系统完成了整个的实验流程。事实上我们在对整个实验完成过程中就是在改造原有的系统。包括对原有的客户端的发送的请求端口更改为nginx的监听端口。  请求端口：    Nginx监听端口，以及服务器的服务端口：    服务器的subscriber也要连接宿主机的docker中的数据库，更改ip：    总体来说关键的修改就是这部分内容。   1. **并发压力测试**   我们原有的系统成功在10秒中增长15000线程中达到了错误率0%的成果，在改进后系统中。由于电脑的性能原因，并且虚拟机的损耗也十分巨大，无法增长到15000线程，所以我们在同时减少线程和时间进行压力测试。  首先我们清空数据库，以便后续来查看测试后成功插入了多少条数据    我们是用如图的jmeter配置进行测试，设置Ramp uptime为0后，代表以服务器的最大能力来进行接受，由于ubuntu的虚拟机分配的物理资源有限，故实现的最大连接数大概率不及实验一本机运行的连接数：    Jmeter测试结果如图，吞吐量相比实验一有所下降，猜测是物理资源受限以及虚拟机资源损耗所致，实验目的达成：   hadoop分布式计算实验 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)，给出主要过程；同时给出配置SSH免密登录的关键步骤。  将下载好的jdk传入虚拟机并且解压到home/pha/jdk-11.0.11    修改/etc/profile 文件，配置JDK的路径    然后重载配置文件，我已经为用户pha配置过root权限。    修改 /etc/hostname 文件： master节点的主机设置为 master ，其他两个虚拟机分别设  置为slave1、slave2，并且修改/etc/hosts文件如下：    安装ssh    生成RSA密钥，并对集群内进行密钥的共享：    配置节点间的无密码访问，同理对slave2进行操作。    测试无密码ssh访问，成功了喵。    2）请配置hadoop集群计算环境，要求数据块副本数为3。给出主要过程，并验证正确性。  解压hadoop。    配置hadoop环境变量，并且更新配置文件source /etc/profile    Hadoop的Hadoop-env.sh配置文件中添加内容    修改core-site.xml中的主要内容    hdfs-site.xml中主要内容：value字段设为3，表示数据块副本数为3    mapred-site.xml中主要内容(MapReduce的配置文件)    配置好三台主机的Hadoop文件，也就是masters文件和workers文件，分别填入master和worker{$worker\_num} 这里就不多赘述。  在slave1和slave2中同样复制整个hadoop-3.2.2的目录    在master节点上执行hdfs namenode -format,在执行start-all.sh验证正确性    使用jps命令进行查看，成功开启hadoop     * 1. 请使用hadoop集群环境计算英文版[Stray Birds](https://en.wikisource.org/wiki/Stray_Birds)正文中每个单词出现的次数。   首先进入Hadoop目录，将飞鸟集stary\_birds文件上换到hdfs文件系统    调用word\_count函数来对飞鸟集文件进行处理，结果存储到output文件中    打开output文件可以查看飞鸟集中各个单词的出现频率     * 1. 请使用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。请模拟输入数据，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。   模拟1万个用户做了100万次操作记录，我们使用python写一个模拟数据的脚本，将生成的文件传入虚拟机。 Mycat数据库分库分表实验（我们选择了2.4进行实验内容） 1）请给出Mycat配置安装过程中遇到的问题和解决方案。  2）请详析Mycat的分库分表原理和操作方法。  3）请模拟具有复杂表结构和含有较大数据量的数据库表， 并基于此库表描述分库分表的结果，且验证分库分表的效果。 Sharding-JDBC数据库分库分表实验 1）请给出Sharding-JDBC配置安装过程中遇到的问题和解决方案。 1. 首先，打开IDEA，创建Springboot项目    然后，修改pom.xml，加载maven依赖    以docker部署MySQL服务    使用Springboot构建时，出现问题，需要在application.yaml中进一步修改配置项。因为Sharding-JDBC已经修改了配置方法，在yaml中定义分配方法。通过在新版Sharding-JDBC中查阅文档获取配置方案。  2）请详析Sharding-JDBC的分库分表原理和操作方法。  原理：  Sharding-JDBC在应用层实现水平拆分，无需改变业务代码。它基于用户设定的分片键(Sharding Key)，将数据分配到不同的数据节点上。具体是解析SQL请求，判断是否涉及到用户设定的分库分表逻辑，然后根据用户设定的路由策略，将SQL请求路由到合适的库和表。  **例如**：SELECT \* FROM orders WHERE order\_id = 123  假设订单号按order\_id % 4拆分到4个表，则 Sharding-JDBC 会根据123 % 4的结果确定该查询应路由到orders\_3表。  当涉及多库多表的查询时，Sharding-JDBC 会在不同的数据节点上并行执行查询，然后执行结果归并处理。对于聚合查询，会合并各节点的查询结果。如果是分页查询，按分页条件对合并后的结果再次排序截取。（即都需要先合并，再选择要不要截取）  **操作方法**：  现在的Sharding-JDBC支持Springboo框架，通过在application.yaml中设定分片策略来进行分库分表。常见的分片策略如下：   1. 范围分片（Range Sharding）：根据数据范围进行拆分，如用户 ID 小于 10000 的存储在数据库 1，大于 10000 的存储在数据库 2。 2. 哈希分片（Hash Sharding）：将分片键的哈希值取模，用于确定数据存储的位置，如order\_id % 4 分散到 4 张订单表。 3. 复合分片（Complex Sharding）：基于多个字段的组合拆分数据，如按用户 ID 和订单时间共同决定数据的存储位置。 4. 按日期分片（Date Sharding）：数据按日期区间拆分，如按月份生成一张表，orders\_2024\_10 表示 2024 年 10 月的订单数据。 5. 请模拟具有复杂表结构和含有较大数据量的数据库表， 并基于此库表描述分库分表的结果，且验证分库分表的效果。 6. 首先，创建多数据库服务器，模拟分布式数据库        1. 制定水平分库、分表规则。有两种分片算法可采用        1. 由于是分布式，需要考虑全局的id序列，不能简单采用主键自增，因此配置分布式序列算法      1. 编写Entity,Mapper        1. 编写测试类   首先是分库测试    看下结果：由于我也编写了分表逻辑，因此数据出现在了server-order0的order0和server-order1的order1表中      再测试一下分表：    看下结果：      可见数据实现了分库和分表插入，根据user\_id的inline类型的奇偶分库，再根据orderNo的哈希值模2分表 Redis数据缓存实验 1）请给出Redis配置安装过程中遇到的问题和解决方案。  1. 使用docker安装  docker run –name redis -p 6379:6379 redis:latest    使用redis-cli    2）请详析Redis的缓存清洗策略，数据迁移及扩容策略，面向缓存雪崩、穿透等问题的策略。   1. 缓存过期命令   EXPIRE key seconds [ NX | XX | GT | LT]   1. NX 当key没有过期时间时，更新key为seconds 2. XX 当key已过过期时间时，更新key为seconds 3. GT 当新设定的过期时间大于当前剩余时间时，更新key为seconds 4. LT 当新设定的过期时间小于当前剩余时间时，更新key为seconds 5. 过期缓存删除方式 1. 惰性删除：客户端请求查询key时，根据设置的过期时间检查，过期了就删除，重新从数据库获取。没过期就返回 2. 定期删除：设定定时任务，每次随机抽取一些设定过期时间的key，过期了就删除。如果key很多，那么就根据hash桶的维度去扫描key，扫到20个key为止，如果第一个桶不够，那就接着扫第二个桶。最后删除已选中的key，进行删除，如果过期的key超过25%，接着删 6. 内存淘汰策略  1. Volatile-random：随机删除，使用allkeys-random策略，适用于常规时段商品浏览等不具有明显业务特征的数据 2. Volatile-ttl：删除过期时间内剩余时间最短的key，根据过期时间与当前时间的差额进行短到长的排序，适用于不具有明显业务特征的数据 3. Volatile-lru：删除最近最少使用的key，也就是最近最不常用数据筛选出来，适用于具有冷热数据隔离需求的数据进行清理。比如电商的受季节影响的数据。 4. Volatile-lfu：删除访问次数最少的key。在LRU中，假如一个数据只是最近才被访问一次，就被作为热点保留，显然不合理。因此根据key最近的访问频率进行淘汰 5. 剩下的是针对全部的数据，也就是可以与不需要强制配置过期时间的key的数据分开。 7. 数据迁移策略 1. 主从架构：有一个主数据库实例（master）和多个从数据库实例（slave），可通过slaveof命令来使服务器成为新的从服务器。然后进行复制初始化，通过SYNC命令可以使master向slave发送快照文件，其中包含这段时间执行的写命令，slave执行快照后，master执行写操作都会同步给slave。如果需要将新从服务器变为主服务器，可以先对原master停止写操作，打开新redis的读写操作，修改业务为新的redis配置，然后断开数据同步，这样新redis就从slave升为master。或者使用aof，追加key到新redis服务器，不会删除原有的key。 2. 哨兵模式：基于主从模式，只不过引入哨兵来监控与自动处理故障，监控主从是否正常运行，master故障能自动将slave转换为master。数据迁移上同主从复制 3. 集群模式：通过分片进行数据管理，将哈希槽分配给节点，集群之间的信息通过Gossip协议进行交互。对于新插入的redis服务器，可以将插槽中的一些移动到新服务器，然后会根据CRC16算法得到结果，对结果进行16384取余数，得到插槽值，进而会有一部分请求分配到新服务器中。整个过程无需停止任何业务。 8. 数据扩容策略（上一部分提到的集群模式基础上）    1. 垂直扩容：增大redis服务器的硬件配置。对于更新的服务器，先按照主从模式，通过slaveof进行复制，作为从节点，然后转换业务配置升为主节点    2. 水平扩容：向集群中增加新的Redis实例，然后更新哈希槽，使其一部分由新的Redis进行管理，并将原节点属于该部分插槽的数据进行迁移，迁移后通知客户端新的节点信息。可采用aof或rdb进行迁移。 9. 缓存雪崩：缓存雪崩是指缓存中数据大批量到过期时间，而查询数据量巨大，请求直接落到数据库上，引起数据库压力过大甚至宕机。 解决方案： 10. 均匀过期：设置不同的过期时间，让缓存失效的时间点尽量均匀。可以为有效期增加随机值或者统一规划有效期。 11. 互斥锁：当需要回写时，只允许第一个线程获取缓存，访问数据库，然后加载到缓存中，剩下的线程需要等到该回写线程执行完，再获取缓存。 12. 数据永不过期：对于大量访问的数据，不设置过期时间，而采用异步的方式更新缓存中的数据。 13. 双层缓存策略：使用主备两层缓存，主缓存有效期正常设置，主缓存失效时从数据库加载最新值。而备份缓存中的数据有效期更长，当获取数据失败的时候读取该缓存，当主缓存更新时，同步通过主缓存更新缓存备份，防止失效时直接访问数据库。 14. 缓存穿透：缓存穿透指用户请求的数据不命中缓存，且数据库也没有，每次都需要查询，且返回空值。如果有攻击者不断请求不存在的数据，会导致数据库大量请求，甚至数据库宕机。 解决方案： 1. 布隆过滤器：是一种概率型数据结构，检测集合中是否存在一个特定的元素。通过k个哈希函数对元素计算产生k个哈希值，并以哈希值作为m位的位数组的下标（取余数），将所有k个对应的比特值由0设为1。当查询一个元素时，计算k个哈希值，并查询，如果有某个比特位为0，则该元素一定不在集合中。如果都为1，则可能存在集合中。对于判断不存在的元素，布隆过滤器时间复杂度低，节省空间不需要存储数据。但缺点加入比特位后是不能删除元素。最适用于不需要删除的场景，如黑名单和爬虫Url去重。 2. 返回空对象。如果缓存未命中，查询数据库、也为空，先将空对象写到该key中，请求不会落到数据库，并且给空对象设置过期时间。这样缺点是，在这段时间内可能会存在缓存和持久层数据不一致。 15. 缓存击穿：一个热点key，在失效的瞬间，并发穿破缓存请求数据库，导致数据库压力骤增。 解决方案： 1. 互斥锁：当需要回写时，只允许第一个线程获取缓存，访问数据库，然后加载到缓存中，剩下的线程需要等到该回写线程执行完，再获取缓存。 2. 热点数据永不过期：对于热点数据，不设置过期时间，而采用异步的方式更新缓存中的数据。   3）请模拟一个简单场景，实现缓存读写操作，缓存更新操作，给出缓存的效果，分析2问题中相关策略的效果。  1 设计数据库，插入一些数据    一开始，redis缓存为空     1. 编写python代码 2. 配置Redis缓存，LFU缓存，布隆过滤器和互斥锁     2．查询缓存，并通过布隆过滤器过滤    3. 如果有向数据库增删数据，同步更新redis，布隆过滤器（不能删除）    4. 模拟查询操作，先查询两次四个用户的订单，会触发缓存更新，然后增加删除订单，会同步更新缓存，查询不存在的用户被布隆过滤器过滤    redis缓存和数据库中都有数据变更      一段时间后，缓存分先后过期（不常用数据更早过期）  对于缓存穿透，布隆过滤器可有效防止数据库中从未存在的数据查询，来防止恶意攻击等。对于缓存击穿和雪崩，互斥锁（实现上为线程锁）实现了同时只有一个线程进行查询，然后更新缓存，防止了多线程多任务落在数据库服务器上，后续线程都在缓存中查询。同时LFU策略可有效淘汰最近使用次数最少的缓存。 | | | | |
| 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 | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |   【注意】该表格可自行增加更多的行。  **（2）工作日志**  由领航员负责记录，记录结对开发期间的遇到的问题、两人如何通过交流合作解决每个问题的。  表1-2 结对开发工作日志   | 日期/时间 | 问题描述 | 最终解决方法 | 交流过程 | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  |   【注意】该表格可自行增加更多的行  **（3）结对开发工作现场照片、或视频及文件沟通截图**  请其他同学帮助拍摄结对开发现场照片至少2张。  13071045_wLhN 1  结对开发现场照片1 结对开发现场照片2 | | | | |
| 实验总结 | | | |  |
| 【结合前期课程项目系统或其他实际软件系统，解决其中的消息订阅/分发问题等】 | | | | |
| 教师评语 | | | |  |
|  | | | | |