小组成员(首字母排序)：

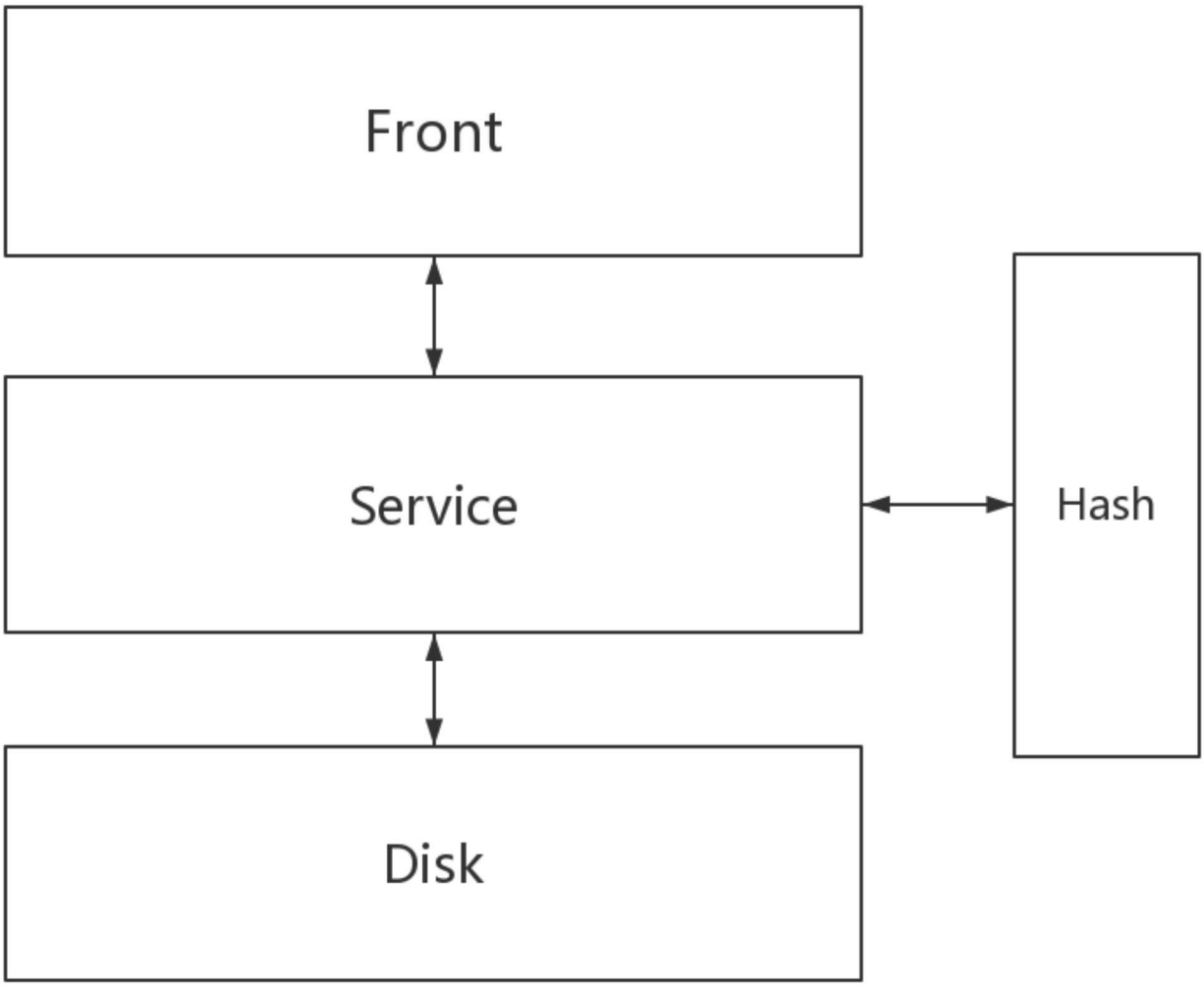
王培宇

王冠淞

谢俊

张尹嘉

程序架构



Front 是前端部分 Service 是框架结构部分 Disk 负责与磁盘中数据交互 Hash 是负责构建索引的Hash算法

一、开发环境

```
PHP：5.6

apache：2.4

JDK：1.8
```

二、使用方法

1，用户在首页输入需要查询的命令，点击“查看结果”查看命令执行结果；点击“清空输入”可清空输入栏。

2，用户可执行命令：select语句、insert语句、delete语句、update语句、count语句、help语句。

3，select语句格式：select name=张晓明,phone=13333333333 或者 select 姓名=张*,电话=13333333333

4，insert语句格式：insert name=张晓明,phone=13333333333

5，delete语句格式：delete name=张晓明,phone=13333333333

6，update语句格式：update name=张晓明,phone=13333333333 set name=张晓明,phone=14444444444

7，count语句格式：count name=张晓明

8，help：查看帮助

三、设计思路

前端设计：负责人：谢俊

1.前端使用php开发。index.php为主页。control.php作为控制器，分析主页输入的命令。若命令非法，则弹窗提示“命令错误”并返回主页。若为其他正确命令，则进行数据库操作并跳转到结果界面，并统计数据库操作事件，显示在结果界面。若查询失败，则弹窗提示。

2.后台使用java开发。php与java使用php-java-bridge通信。php实例化java编写的sql_analysis类，并将主页输入的命令作为参数传递给execute()函数，进行数据库操作。php接收sql_analysis的getResult()返回结果作为select语句的查询结果，用于展示于selectResult.php界面。

3.php-java-bridge使用方法：下载php-java-bridge 5.5.2，获得JavaBridge.jar和Java.inc。将JavaBridge.jar放到php的ext目录下，双击运行，并且根据选择的端口号相应修改java.inc中的#define ("JAVA_HOSTS", "127.0.0.1:8787")为define ("JAVA_HOSTS", "127.0.0.1:选择的端口号")，注意去掉前面的#号。编写java类sql_analysis，打包成jar包，将生成的jar包拷贝到apache的web应用根目录之下，同时将java.inc也拷到该目录下。在需要php与java通信的文件头引入Java.inc和相应jar包，即可创建java类定义的对象并使用方法。

Service 和 Hash： 负责人：王培宇 王冠淞

王培宇工作内容：

1.积极参与讨论设计数据存储机制。并对讨论出来的想法进行思考验证、尝试改进。

2.决定采用非定长的数据存储时，我们将整个实现分为4个部分。

> 1.前端
>
> 2.Map构造与更新和 对获取的数据进行筛选
>
> 3.hash思考与选择
>
> 4.disk数据底层存储实现。

其中一开始我负责hash 的思考与选择。完成后，接着实现Map的构造，在数据清洗的过程中完成Map的构造，并实现 将构造好的Map进行本地保存，当程序再次启动时，不需要再重新构造Map,而是直接将之前保存 的Map直接读取出来。

3.Map构造完成后，对select的查找，包括单属性查找，组合属性查找进行实现。并对姓+通配符实现姓的模糊查找

王冠淞的工作内容：

1.和其他队员讨论并决定数据库的实现方法（放弃了B+树，采用Map linenumber>的方式 存储列元素哈希值对应的行号，底层使用位移表和空闲行表维护磁盘上数据）。

2.完成项目框架的搭建，主要包括定义sql语句，前端调用的接口，Sql命令解析，调用hash函数和维护Map结构，最后调用disk层的 get,set,update和delete接口并返回数据给前端。

3.sql命令解析包括：select解析初级版本，后来加入对查询结果进行验证以解决hash冲突，但由于时间原因多筛选条件未使用多线程并行查找。 完成update语句，insert语句，delete语句，count 语句的解析。like语句只能全表搜索，故先没有做。

4.由于时间原因没有和前端实际连接，但流程已经走通，把我们最后的工程文件拷贝到apache服务器上，php就可以调用java接口。

Disk: 负责人：张尹嘉

Disk部分主要负责和磁盘上的文件进行交互， 为上层提供访问数据的基本接口，从而将Service和文件操作分离开。

Disk的实现是建立在两个文件的基础上的， 分别是 database 和 line_record。 将原始数据清洗后，一条数据的各个字段之间按照 “|-|”分开，然后用byte密集写入database文件， 并将这条数据在database文件中的数据条数（行号），起始偏移 以及 长度记录在line_record中。line_record中每条记录都是100byte，如果剩余部分用0填充。另外在内存中还维护了一个freetable， 用于记录之前删除后在database中剩余的磁盘空间。

get操作： 传入要获取的数据行号-- x， 用读取 x *100 到 x*100+100的数据，检查这条记录的行号是否和目标行号一致，如果不一致就根据两个行号的大小上移或者下移，直到取到目标行号的记录。然后根据记录在database中获取数据。 这样可以控制每次存取数据的平均复杂度在O（1）的级别上。

set操作： 把先检查freeTable看database中是否有空余的空间，如果有就把数据填充进空余空间并更新freeTable，没有就填充到database尾部。然后在line_record文件末尾添加一条记录。

delete操作： 先用和get一样的算法定位line_record中相应记录的位置，然后直接把记录对应的100bytes变为0， 并将database中相应的空间记录到freeTable中。 这里直接变为零是考虑到如果删除会造成后面的文件整体向前移动，耗费时间， 并且这样做也方便get时根据行号的模糊定位。 这样做会造成line_record越来越大， 因此需要定时进行维护，比如每一个小时stop-the-world一下然后维护下line_record或者一旦line_record大于多少空间以后进行维护。 但是由于时间限制我并没有实现这个功能。

update操作： update的操作就是delete和set的结合。但是每次会改变数据的行号。