

## 文 后端的轮子（二）--- 数据库

数据库 吴yh坚 7月18日发布

本篇趟个雷，把数据库纳入到轮子中了，前面说到了 **数据库** 其实不算轮子，也说到了其实我写不出来数据库，这里所说的数据库严格来说是 **关系型数据库**，他比轮子复杂多了，是一个和操作系统差不多复杂度的东西，所以才能通过一个Oracle养活一家全球50强的公司，其次，数据库太复杂了，要写出来实在是力所不能及，但是后来有想了一下，如果我们从另外一个角度来审视数据库，那么也有比较简单的实现办法，那么，这一篇，我们来造个数据库吧，看吧，把 **关系型** 去掉了，因为，有了 **关系型** 几个字，数据库就变得复杂多了。

## 先来聊聊关系型数据库

关系型数据库（Relational Database）是一个伟大的发明，一般的数据库的理论，大概会分成以下三个部分。

- 首先，数据库是建立在关系模型基础上的，并且从理论上讲，是有完备的数学模型，也就是 **集合代数** 来做支撑的，他把我们真实世界中的联系和实体抽象成了 **关系模型**，并用这个发展出了数据库理论，这是数据库的理论基础。
- 其次，也有人通过这个关系模型，发明了SQL这种进行关系查询的编程语言，用来对这个关系型的数据集合进行操作。这个实际上给出了通过集合代数发展出来的关系型数据库怎么进行数据操作和检索的。
- 还有人，发展出了数据库设计的理论，也就是大家所熟悉的数据库三大范式【应该是5大范式】，用来教我们在实际场景中怎么设计一个数据库，几大范式实际上是把 **关系模型** 这个抽象的概念变成了几条规则，按照这几条规则去设计数据库，就能产生最少的数据冗余，最能体现出 **关系** 这个模型的核心。

我们发现，上面三个大的部分都是数据库的理论知识，其实并没有人告诉我们怎么来用代码实现一个数据库，因为科学家们认为实现它并不重要，那是工程师要考虑的事情，**Too Simple**，科学家只负责搞出理论，反正我们也不是科学家，那么我们就来做个工程师吧。

## 工程师眼中的数据库系统

既然是工程师，首先想到的就是如何实现一个数据库了，一个标准的数据库大概主要会包含以下几个大的模块。

- 底层的存储层，这个是必不可少的，他是整个数据库的核心数据结构，也就是数据是如何保存的，一般提供最简单的原子增删改查。
- 存储层上面就是引擎层了，这里会对底层的存储层进行各种组合型的操作来满足查询的需求之类的，而且数据库的事务支持也在这一层，我们熟悉的 **InnoDB** 就是一个数据库的存储引擎，他其实包含的就是这个引擎层和存储层了，引擎层提供对数据层的操作方法集合。
- 在引擎层之上还有个SQL的解析层，主要用来对SQL语句进行解析，分析，优化了，然后把SQL语句转化成引擎层的接口，进行具体的数据操作。
- 最上面就是对外的UI了，也就是用户交互层了，一般我们熟悉的就是网络交互了。

虽然看起来好像挺简单，就是这么三层，但是实际的数据库是非常非常复杂的，除了这些以外还有很多其他模块，比如 **用户权限管理**，**缓存模块**，**日志模块**，**备份模块**等等等等，大家可以仔细去看看 **InnoDB** 的书籍或者 **InnoDB** 的代码，光一个 **binlog** 就特别麻烦。

其实要保存数据，搜索系统也能保存数据，而且检索起来更快，并且两者的底层数据结构其实差别不是很大，但为什么用数据库呢？因为数据库的核心是 **可靠**，这个 **可靠** 就是靠数据库的引擎层来保证的，完整的binlog记录，崩溃后完整的重放机制，数据双写，内存数据定时刷新到磁盘，所有的这些都是为了保证数据的 **可靠**，不会丢失数据。

而上面说的每一个功能，都能单独的写一篇长文，所以说要实现一个数据库其实是很麻烦的，因为为了做到 **可靠**，必然会有很多冗余的数据或者冗余的操作来保证 **可靠**，但作为一个成熟的产品，还需要考虑到产品的 **性能**，所以，如何既 **可靠** 又 **性能** 优良，就变成了一个衡量数据库好坏的标准，当然，在这两点上，目前没人能干过Oracle了。

## 最小系统的数据库

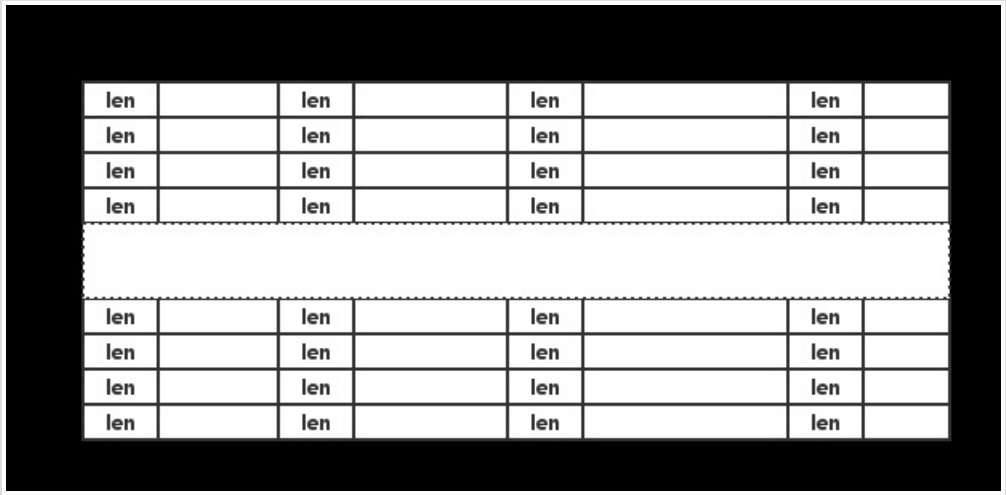
数据库如此之复杂，我们如何对他进行瘦身，来实现一个最小的数据库系统呢？我们可以从另外一个角度想想，就是我们拿数据库是干什么的？那就是 **存储和查询数据**，如果这么来想的话，就能简单不少。

首先，我们知道数据库最重要的功能就是存储数据，那么底层的存储部分是不能少的，其次，存储的数据要提供查询功能，不然存了就没意义了，这也是不能少的，第三，需要提供一个对外的接口可以 and 用户交互，不然就既不能存也不能查了。

所以，一个最最基本的数据库至少应该包含数据层，查询层（引擎层）和UI（用户接口）层三层，那么我们就用几个简单的文件来实现这三层，完成一个最小的数据库吧。

## 存储层

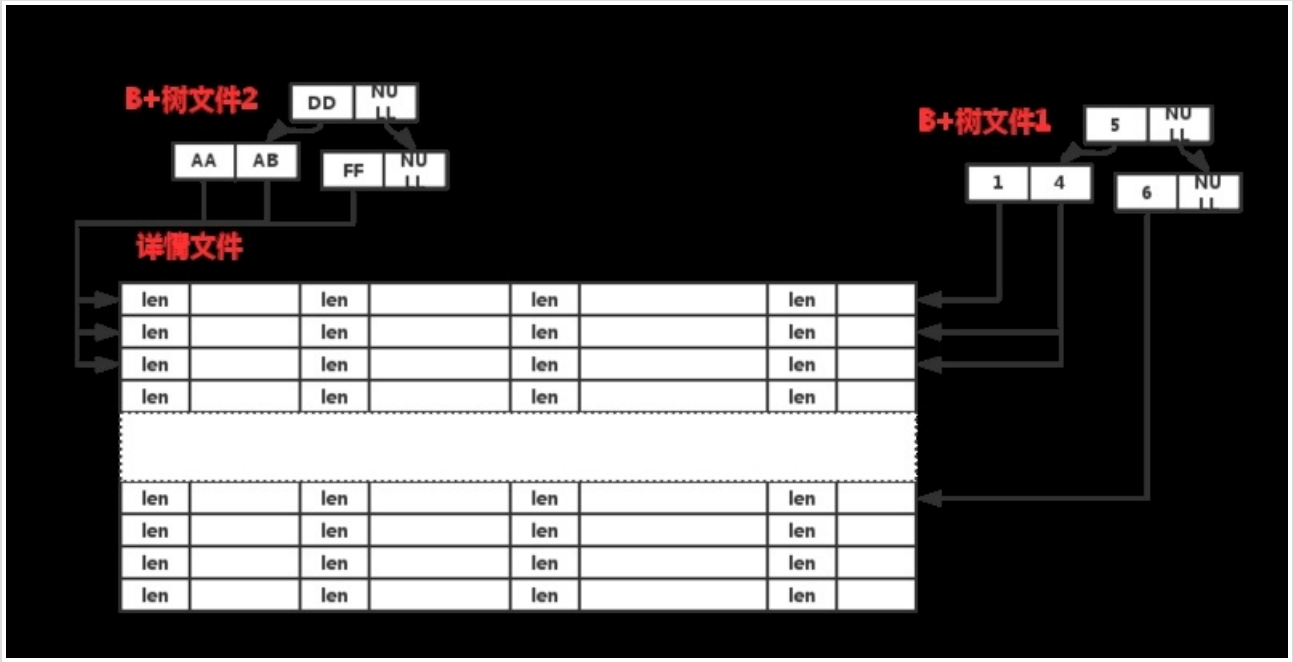
数据库的基本单位是 **列**，再上一级的基本单位就是 **表** 了，而且我们在建表的时候都会指定列的 **名称**，**类型**，**长度** 这三个最基本的属性，如果所有列都有这三个属性，那么其实我们是知道每一行数据最多有多少字节的，所以，我们可以设定没一行数据的长度都是定长的，那么整个表的长度也是定长的了，这样查询的时候可以根据行的长度进行快速定位数据，所以，**我们的最底层数据就是一个定长的表格了，每一列存储的时候就像下面这样，然后有个meta信息来存储列的属性**



这个看上去很简单吧？也容易实现吧，其实很多数据库也基本上确实是这么实现的，并不难理解吧？稍微注意一下的是每一列存储的时候，每个字段的前四个字节保存的是这个字段的实际长度，然后才是字段的实际内容，如果长度小于建表时的设定长度，那么有一部分空间是浪费掉的，虽然是浪费了，但还是值得的，因为可以让查询的时候省不少事。

这么下来，每行记录就是一个定长的，而一个数据库的表就是一个二进制文件了，但仅仅是这样还是不够的，因为这样结构，无论什么查询都需要扫描全表，依次进行判断，而我们在建表的时候都会建立索引，为了建立索引，我们还得实现一个 **B+树** 来存储索引，而 **B+树** 基本上是所有数据库的索引保存的数据结构，这里我们也有实现，如果对 **B+树** 感兴趣，可以看我之前的一篇文章，那篇有详细的B+树的实现方式，文章后有那篇文章的链接。

总之，数据底层我们就用了一个定长的二进制文件和几棵B+树，再加上一个meta信息文件来实现了一个数据库的底层数据层，很简单哈，但基本上包括了数据库真实的底层，虽然真正的数据库比这复杂多了，但也跑不掉这几个数据结构，整个看下来，数据层的数据结构大体上长这样子。



当然，数据层实现完了以后，还需要对上提供几个简单的接口，比如

- **建表接口** CreateTable( []FieldInfo ), 参数是每个字段的信息，包括字段的 **名称**，**长度**，**类型**

- **数据插入接口** AddData(map[string]string) , 参数是一个map, key是字段名称, value是字段内容
- **单字段查询接口** Find(fieldname,fieldvalue,op), 参数是字段名称, 字段值, 操作类型 (大于, 小于, 等于)
- **数据获取接口** GetData(docid), 参数是docid, 用来计算在文件中的偏移

## 查询层

底层已经有了, 接下来就是上面的查询层 (引擎层) 了, 这里我没用 **引擎** 两个字, 是因为最小数据库的实现上, 实在算不上一个引擎系统, 我们实现最简单的基本查询SQL ( **建表sql**, **插入数据sql**, **单表查询sql** ) 的解析, 在实际中, SQL的解析是一个异常复杂的工程, 涉及到语法分析, 预处理, 优化查询等几个大的部分, 因为SQL其实是一门编程语言, 要解析一门编程语言, 那么编译原理那一套基本上都会用得到。

这里我们换条路子, 因为只实现三种简单的SQL语句, 那么我们直接用正则和字符串的匹配来对SQL进行解析, 解析完成以后变成一个数据层的对外接口, 建表和插入数据都比较简单, 解析了SQL以后直接调用上面的 **第一** 和 **第二接口** 就行了。

数据查询的时候, 对查询SQL的 **WHERE** 之后的部分, 用了个小算法, 就是逆波兰表达式来对 **WHERE** 之后的语句进行解析, 变成一个栈结构来存储查询的内容, 然后通过弹栈的方式一个一个 **调用接口三**, 并且对结果进行求交和求并的操作, 最后得到结果以后, 再依次 **调用接口四** 获取最后的结果, 如果对逆波兰表达式不了解, 那么请自行百度一下, 很简单的, 主要用在对四则运算的优先级的解析中。

查询层的输入输出很简单, 他对外实际上只提供一个接口。 **ExecSqlSentence( Sql ) string**, 都是字符串, 输入是一条条的sql语句, 输出是数据。

## UI层 (用户接口层)

对于用户的接口层就更加简单了, 我们只需要提供一个TCP服务就行了, 用  **;分号** 来分割每次用户的输入, 也就是说, 我们telnet上我们这个数据库, 然后输入sql, 数据库就会返回数据了。

## 具体实现

我在github上建立了一个新的工程叫 **SparrowSys**, 麻雀工程, 意思很明显, 这是一个后端的麻雀, 是最简单的后端轮子, 目前我也已经提交了一部分代码, 数据库的还没有写完, 后面会补上的。

数据库的部分在 **src** 下的 **SparrowDB** 里面, 很明显的看到里面有 **DataLayer**, **EngineLayer**, **NetLayer**, 对应的就是上面的三层, 每层里面有一到两个文件, 都很简单, 目前DataLayer基本完成了, 后面会把EngineLayer和NetLayer补上, 后面的文章会说说使用, **utils** 文件夹中是一些公共的东西, 后面的其他轮子会用到的, 比如 **B+树** 就在 **utils** 里面。

目前这个工程里面东西不多, 不建议看, 后面我补全以后会说明, 欢迎大家提交你的实现来代替我的。接受任何 **pull request**。

十天没有更新了, 主要是代码没时间写, 所以没有测试结果可看, 本来准备等代码都写完了再来更新文章, 但最近实在是太忙了, 没时间写代码, 那先放出文章, 等代码补充完整了再说说测试效果吧。

代码地址: <https://github.com/wyh267/SparrowSys>

### B+树文章

如果你觉得不错, 欢迎转发给更多人看到, 也欢迎关注我的公众号, 主要聊聊搜索, 推荐, 广告技术, 还有瞎扯。。文章会在这里首先发出来: ) 扫描或者搜索微信号 **XJJ267** 或者搜索 **西加加语言** 就行



1 推荐

收藏

你可能感兴趣的文章

spring jdbcTemplate添加数据库乱码 752 浏览

Java大数据内存序列化浅谈（一） 1 收藏，1k 浏览

MongoDB 新建数据库用户例子 965 浏览



本文采用 [署名-相同方式共享 3.0 中国大陆许可协议](#)，分享、演绎需署名且使用相同方式共享。

讨论区

使用评论询问更多信息或提出修改意见，请不要在评论里回答问题

提交评论 ?

评论支持部分 Markdown 语法： **\*\*bold\*\*** *\_italic\_* [link] (http://example.com) > 引用 ``code`` - 列表 。 同时，被你 @ 的用户也会收到通知



本文隶属于专栏

吴说

一个老程序员说。。。。。



吴yh坚  
作者


关注专栏

系列文章

后端的轮子（一） 14 收藏，650 浏览

后端的轮子（三）--- 缓存 11 收藏，212 浏览

相关收藏夹



mongodb  
4 个条目 | 0 人关注



系统+构架+日志  
5 个条目 | 1 人关注

分享扩散：



