

微软面试 100 题系列

作者：July--结构之法算法之道 blog 之博主。

时间：2010 年 12 月 - 2012 年 9 月

出处：http://blog.csdn.net/v_JULY_v。

声明：本文档仅供学习之用，严禁用于任何商业用途。

前言

本微软面试 100 题系列，共计 11 篇文章，300 多道面试题，截取本 blog 索引性文章：程序员面试、算法研究、编程艺术、红黑树、数据挖掘 5 大系列集锦，中的第一部分编辑而成，如下图所示：

无私分享，造福天下

以下是本 blog 内的微软面试 100 题系列，经典算法研究系列，程序员编程艺术系列，红黑树系列，及数据挖掘十大算法等 5 大经典原创系列作品与一些重要文章的集锦：

一、微软面试 100 题系列

- 横空出世，席卷 Csdn--评微软等数据结构+算法面试 100 题（微软面试 100 题系列原题+答案索引）
- 微软 100 题（微软面试完整第 1-100 题）
- 微软面试 100 题 2010 年版全部答案集锦（含下载地址）
- 全新整理：微软、谷歌、百度等公司经典面试 100 题[第 101-160 题]
- 全新整理：微软、Google 等公司的面试题及解答[第 161-170 题]
- 十道海量数据处理面试题与十个方法大总结（十道海量数据处理面试题）
- 海量数据处理面试题集锦与 Bit-map 详解（十七道海量数据处理面试题）
- 教你如何迅速秒杀掉：99% 的海量数据处理面试题（解决海量数据处理问题之六把密匙）
- 九月腾讯，创新工场，淘宝等公司最新面试三十题（第 171-200 题）（2011 年度九月最新面试三十题）
- 十月百度，阿里巴巴，迅雷搜狗最新面试七十题（第 201-270 题）（2011 年度十月最新面试七十题）
- 十月下旬腾讯，网易游戏，百度最新校园招聘笔试题集锦（第 271-330 题）
- 最新九月百度人搜，阿里巴巴，腾讯华为京东 360 笔面试二十题（2012 年度最新九月笔试面试二十题）

本微软面试 100 题系列涵盖了数据结构、算法、海量数据处理等 3 大主题，相比于 [微软面试 100 题系列专栏](#)，去掉了那 3 篇关于答案永久勘误的文章（因为，自觉那些答案存在不少问题，当然，读者尽可以读读针对这 100 题一题一题写的程序员编程艺术系列）。

闲不多说，眼下九月正是校招，各种笔试，面试进行火热的时节，希望此份微软面试 100 题系列的 PDF 文档能给正在找工作的朋友助一臂之力！

如果读者发现了本系列任何一题的答案有问题，错误，bug，恳请随时不吝指正，你可以直接评论在原文之下，也可以通过邮件或私信联系我，我的联系方式如下：

- 邮箱：zhoulei0907@yahoo.cn
- 微博：<http://weibo.com/julyweibo>

祝诸君均能找到令自己满意的 offer 或工作，谢谢。July、二零一二年 9 月。

OK，以下是本 blog 内的微软面试 100 题系列文章的集锦（点击链接，即可跳转到相应页面）：

- 横空出世，席卷互联网--评微软等公司数据结构+算法面试 100 题..... 3
- 微软等公司数据结构+算法面试 100 题(第 1-100 题)首次完整亮相..... 9
- 微软等数据结构+算法面试 100 题全部答案集锦..... 36
- 全新整理：微软、谷歌、百度等公司经典面试 100 题[第 101-160 题]..... 111
- 全新整理：微软、谷歌等公司非常好的面试题及解答[第 161-170 题]..... 120
- 海量数据处理：十道面试题与十个海量数据处理方法总结..... 145
- 海量数据处理面试题与 Bit-map 详解 157
- 教你如何迅速秒杀掉：99%的海量数据处理面试题 167
- 九月腾讯，创新工场，淘宝等公司最新面试三十题（第 171-200 题） 183
- 十月百度，阿里巴巴，迅雷搜狗最新面试七十题（第 201-270 题） 192
- 十月下旬腾讯，网易游戏，百度最新校园招聘笔试题集锦(第 271-330 题)..... 212
- 最新九月百度人搜，阿里巴巴，腾讯华为京东笔试面试二十题..... 225
- 结语 231

横空出世，席卷互联网--评微软等公司数据结构+算法面试 100 题

横空出世，席卷互联网

---评微软数据结构+算法面试 100 题

作者：July。

时间：2010 年 10 月-11 月。版权所有，侵权必究。

出处：http://blog.csdn.net/v_JULY_v。

说明：本文原题为：“[横空出世，席卷 Csdn \[评微软等公司数据结构+算法面试 100 题\]](#)”，但后来此微软 100 题（加上后续的 80 道，共计 180 道面试题）已成一系列，被网络上大量疯狂转载，因此特改为上述题目。

入编程这一行之初，便常听人说，要多动手写代码。可要怎么写列？写些什么列？做些什么列？

c 语言程序设计 100 例，太过基础，入门之后，挑战性不够。直接做项目，初学者则需花费大量的时间与精力、且得有一定能力之后。

于是，这份精选微软等公司数据结构+算法面试 100 题的资料横空出世了：

[推荐] [整理] 算法面试：精选微软经典的算法面试 100 题[前 60 题]（帖子已结） 10.23
<http://topic.csdn.net/u/20101023/20/5652ccd7-d510-4c10-9671-307a56006e6d.html>。

上述帖子已结贴。如果，各位，对 100 题中任何一题、有任何问题，或想法，请把你的思路、或想法回复到这更新帖子上：

[推荐] 横空出世，席卷 Csdn：记微软等 100 题系列数次被荐[\[100 题永久维护地址\]](#)
11.26 日
<http://topic.csdn.net/u/20101126/10/b4f12a00-6280-492f-b785-cb6835a63dc9.html>。

仅仅一个月，此帖子 4 次上 csdn bbs 首页，3 次上 csdn 首页。总点击率已超过 10000（直至现在已被网络上大量疯狂转载，估计已被上十万人看过或见识到）。

在这份资料里，作者不仅大胆的罗列了微软等公司极具代表性的精彩 100 题，更为重要的是，作者在展示自己思考成果的同时，与一群志同道合的同志，一起思考每一道题，想办法怎样一步步去编写代码，并及时的整理自己的思路、和方案。

这 100 道题，不仅解决了大量初学者找不到编程素材、练习资料的尴尬，而且更是给你最直接的诱惑：作者随后直接亲自参与做这 100 题，或自个做，或引用他人方案，一步步带你思考，一步步挖代码给你看。

作者在展示自己和他人思考成果的同时，给他人带来了无比重要的分享，此举颇有开源精神。

不但授之以鱼，而且授之以渔。不但提供给你大量经典的编程素材，而且带给你思考的力量。此等幸运，非有心人莫属。在参与做这 100 道题的浩荡队伍中，有老师，有学生，有正在工作的上班族，有经验丰富的老者，前微软 SDET... 等等。如此无私奉献，享受帮助他人的乐趣，思考、分享、追根究底每一道题，此等境界，亦非每一人所有也。

编程就是享受思考。

一句话，盛宴已摆在桌前，敬请享用。

updated:

关于此一百道+后续 185 道（参见文末），近 300 道面试题的所有一切详情，请参见，如下：

原题

[珍藏版]微软等数据结构+算法面试全部 100 题全部出炉[100 题首次完整亮相] 1206

http://blog.csdn.net/v_JULY_v/archive/2010/12/06/6057286.aspx

//至此，第 1-100 题整理完成，如上所示。微软等 100 题系列 V0.1 版完成。2010 年 12 月 6 日。

[汇总 II]微软等公司数据结构+算法面试第 1-80 题[前 80 题首次集体亮相] 11.27

http://blog.csdn.net/v_JULY_v/archive/2010/11/27/6039896.aspx

帖子

1、2010 年 10 月 11 日，发表第一篇帖子：

算法面试：精选微软经典的算法面试 100 题[每周更新]（已结帖）

<http://topic.csdn.net/u/20101011/16/2befbfd9-f3e4-41c5-bb31-814e9615832e.html>;

2、2010 年 10 月 23 日，发表第二篇帖子：

[推荐] [整理]算法面试：精选微软经典的算法面试 100 题[前 40 题]（4 次被推荐，已结帖）

<http://topic.csdn.net/u/20101023/20/5652ccd7-d510-4c10-9671-307a56006e6d.html>;

3、2010 年 11 月 26 日，发表第三篇帖子，此微软等 100 题系列永久维护地址：

[推荐] 横空出世，席卷 Csdn：记微软等 100 题系列数次被荐[100 题维护地址]（帖子未结）

[http://topic.csdn.net/u/20101126/10/b4f12a00-6280-492f-b785-cb6835a63dc9.html。](http://topic.csdn.net/u/20101126/10/b4f12a00-6280-492f-b785-cb6835a63dc9.html)

资源

题目系列：

1.[珍藏版]微软等数据结构+算法面试 100 题全部出炉 [完整 100 题下载地址]:

<http://download.csdn.net/source/2885434>

2.[最新整理公布][汇总 II]微软等数据结构+算法面试 100 题[第 1-80 题] :

<http://download.csdn.net/source/2846055>

答案系列：

1.[最新答案 V0.4 版]微软等数据结构+算法面试 100 题[第 41-60 题答案] 2011、01、04: <http://download.csdn.net/source/2959162>

2.[答案 V0.3 版]微软等数据结构+算法面试 100 题[第 21-40 题答案] :

<http://download.csdn.net/source/2832862>

3.[答案 V0.2 版]精选微软数据结构+算法面试 100 题[前 20 题]-修正 :

<http://download.csdn.net/source/2813890>

//注：答案，仅仅只作为思路参考。

更多资源，下载地址：

- http://v_july_v.download.csdn.net/

谢谢。

本微软公司面试 100 题的全部答案目前已经上传资源，所有读者可到此处下载：

http://download.csdn.net/detail/v_JULY_v/3685306。2011.10.15。

维护

1. 关于本微软等公司数据结构+算法面试 100 题系列的郑重声明 1202:

http://blog.csdn.net/v_JULY_v/archive/2010/12/02/6050133.aspx

2. 各位,若关于这 100 题,有任何问题,可联系我,My e-mail: zhoulei0907@yahoo.cn
3. 各位, 若对这 100 题中任何一题, 有好的思路、或想法, 欢迎回复到下面的帖子上: 本微软等 100 题系列的永久维护, 帖子地址, [推荐]横空出世, 席卷 Csdn: 记微软等 100 题系列数次被荐[100 题永久维护地址] 11.26 日 :
<http://topic.csdn.net/u/20101126/10/b4f12a00-6280-492f-b785-cb6835a63dc9.html>

答案

为了更广泛的与读者就这微软等面试 100 题交流, 也为了更好的获取读者的反馈, 现在, 除了可以在我的帖子上, 发表思路回复, 和下载答案资源外, 我把此微软 100 题的全部答案直接放到了本博客上, 欢迎, 所有的广大读者批评指正。

答案 V0.2 版[第 1 题-20 题答案]

http://blog.csdn.net/v_JULY_v/archive/2011/01/10/6126406.aspx [博文 I]

答案 V0.3 版[第 21-40 题答案]

http://blog.csdn.net/v_JULY_v/archive/2011/01/10/6126444.aspx [博文 II]

答案 V0.4 版[第 41-60 题答案]

http://blog.csdn.net/v_JULY_v/archive/2011/02/01/6171539.aspx [博文 III]

有部分答案或参考或借鉴自此博客: <http://zhedahht.blog.163.com/>。特此声明, 十分感谢。

现今, 这 100 题的答案已经全部整理出来了, 微软面试 100 题 2010 年版全部答案集锦: http://blog.csdn.net/v_july_v/article/details/6870251。2011.10.13。

勘误

1.永久优化: 微软技术面试 100 题第 1-10 题答案修正与优化,

http://blog.csdn.net/v_JULY_v/archive/2011/03/25/6278484.aspx。

2.永久优化: 微软技术面试 100 题第 11-20 题答案修正与优化,

http://blog.csdn.net/v_JULY_v/archive/2011/04/04/6301244.aspx。

后续

- 微软面试 100 题 2010 年版全部答案集锦（含下载地址）
- 全新整理：微软、谷歌、百度等公司经典面试 100 题[第 101-160 题]
- 全新整理：微软、Google 等公司的面试题及解答[第 161-170 题]
- 十道海量数据处理面试题与十个方法大总结
- 海量数据处理面试题集锦与 Bit-map 详解
- 教你如何迅速秒杀掉：99% 的海量数据处理面试题(解决海量数据处理问题之六把密匙)
- 九月腾讯，创新工场，淘宝等公司最新面试三十题（第 171-200 题）
- 十月百度，阿里巴巴，迅雷搜狗最新面试七十题（第 201-270 题）
- 十月下旬腾讯，网易游戏，百度最新校园招聘笔试题集锦(第 271-330 题)

艺术

根据本 blog 里面的 180 道面试题为题材之一，我专门针对每一道编程题而创作了程序员编程艺术系列，力争将编程过程中所有能体现的到的有关选择合适的数据结构、寻找更高效的算法、编码规范等等内容无私分享，造福天下。详情，请参见：[程序员编程艺术系列](#)。目前已经写到了第十章，且将长期写下去。

本编程艺术系列分为三个部分，第一部分、程序设计，主要包括面试题目，ACM 题目等各类编程题目的设计与实现，第二部分、算法研究，主要以我之前写的[经典算法研究系列](#)为题材扩展深入，第三部分、编码规范，主要阐述有关编程中要注意的规范等问题。ok，一切的详情，请参见：[程序员编程艺术系列](#)。

加入

能在网上找到有意义的事情并不多，而如此能帮助到千千万万的初学者，和即将要找工作而参加面试的人的事情更是罕见。希望，你也能参与进我们之中来，一起来做这微软面试 187 题，一起享受无私分享，开源，思考，共同努力，彼此交流，探讨的诸多无限乐趣：

- [重启开源，分享无限—诚邀你加入微软面试 187 题的解题中](#)

有很多朋友跟我说，已毕业工作了的一般都不喜欢做面试编程题了。我觉不然，那得看你接受的是什么一种方式，如果抛开面试这个负担，纯粹为编程而编程，享受思考锻炼思维的乐趣，则也可以凝聚成一股开源军，且将声势浩大。如我去年 11 月发的微软面试贴，如今早已超过 1000 条回复：

<http://topic.csdn.net/u/20101126/10/b4f12a00-6280-492f-b785-cb6835a63dc9.html>。

版权声明:

- 1、本人对此微软面试 100 题系列，包括原题整理，上传资源，**帖子**，答案，勘误，修正与优化等系列的全部文章或内容，享有全部的版权。任何人转载或引用以上任何资料，一律必须以超链接形式注明出处。
- 2、未经本人书面许可，严禁任何出版社或个人出版本 **BLOG** 内任何内容。否则，永久追究法律责任，永不懈怠（July、二零一零年十月声明）。

微软等公司数据结构+算法面试 100 题(第 1-100 题)首次完整亮相

作者:July、2010 年 12 月 6 日。

1. 更新: 现今, 这 100 题的答案已经全部整理出来了, 微软面试 100 题 2010 年版全部答案集锦: http://blog.csdn.net/v_july_v/article/details/6870251。
 2. 关于此 100 道面试题的所有一切详情, 包括答案, 资源下载, 帖子维护, 答案更新, 都请参考此文: **横空出世, 席卷 Csdn [评微软等数据结构+算法面试 100 题]**。
 3. 以下 100 题中有部分题目整理自何海涛的博客 (<http://zhedahht.blog.163.com/>)。十分感谢。
-

微软等 100 题系列 V0.1 版终于结束了。

从 2010 年 10 月 11 日当天最初发表前 40 题以来, 直至此刻, 整理这 100 题, 已有近 2 个月。

2 个月, 因为要整理这 100 题, 很多很多其它的事都被我强迫性的搁置一旁, 如今, 要好好专心去做因这 100 题而被耽误的、其它的事了。

这微软等数据结构+算法面试 100 题系列(是的, 系列), 到底现在、或此刻、或未来, 对初学者有多大的意义,

在此, 我就不给予评说了。

由他们自己来认定。所谓, 公道自在人心, 我相信这句话。

任何人, 对以下任何资料、题目、或答案, 有任何问题, 欢迎联系我。

作者邮箱:

zhoulei0907@yahoo.cn

786165179@qq.com

作者声明:

转载或引用以下任何资料、或题目, 请注明作者本人 July 及出处。

向您的厚道致敬, 谢谢。

好了，请享受这完完整整的 100 题吧，这可是首次完整亮相哦。:D。

1. 把二元查找树转变成排序的双向链表（树）

题目：

输入一棵二元查找树，将该二元查找树转换成一个排序的双向链表。

要求不能创建任何新的结点，只调整指针的指向。

```
10
/
  \
6   14
/ \ / \
4 8 12 16
```

转换成双向链表

4=6=8=10=12=14=16。

首先我们定义的二元查找树 节点的数据结构如下：

```
struct BSTreeNode
{
    int m_nValue; // value of node
    BSTreeNode *m_pLeft; // left child of node
    BSTreeNode *m_pRight; // right child of node
};
```

2. 设计包含 min 函数的栈（栈）

定义栈的数据结构，要求添加一个 min 函数，能够得到栈的最小元素。

要求函数 min、push 以及 pop 的时间复杂度都是 O(1)。

3. 求子数组的最大和（数组）

题目：

输入一个整形数组，数组里有正数也有负数。

数组中连续的一个或多个整数组成一个子数组，每个子数组都有一个和。

求所有子数组的和的最大值。要求时间复杂度为 O(n)。

例如输入的数组为 1, -2, 3, 10, -4, 7, 2, -5， 和最大的子数组为 3, 10, -4, 7, 2,

因此输出为该子数组的和 18。

4. 在二元树中找出和为某一值的所有路径（树）

题目：输入一个整数和一棵二元树。

从树的根结点开始往下访问一直到叶结点所经过的所有结点形成一条路径。

打印出和与输入整数相等的所有路径。

例如 输入整数 22 和如下二元树

```
10
 / \
5  12
 /   \
4    7
```

则打印出两条路径：10, 12 和 10, 5, 7。

二元树节点的数据结构定义为：

```
struct BinaryTreeNode // a node in the binary tree
{
    int m_nValue; // value of node
    BinaryTreeNode *m_pLeft; // left child of node
    BinaryTreeNode *m_pRight; // right child of node
};
```

5. 查找最小的 k 个元素（数组）

题目：输入 n 个整数，输出其中最小的 k 个。

例如输入 1, 2, 3, 4, 5, 6, 7 和 8 这 8 个数字，则最小的 4 个数字为 1, 2, 3 和 4。

第 6 题（数组）

腾讯面试题：

给你 10 分钟时间，根据上排给出十个数，在其下排填出对应的十个数

要求下排每个数都是先前上排那十个数在下排出现的次数。

上排的十个数如下：

【0, 1, 2, 3, 4, 5, 6, 7, 8, 9】

举一个例子，

数值: 0,1,2,3,4,5,6,7,8,9

分配: 6,2,1,0,0,0,1,0,0,0

0 在下排出现了 6 次，1 在下排出现了 2 次，

2 在下排出现了 1 次，3 在下排出现了 0 次....

以此类推..

第 7 题（链表）

微软亚院之编程判断俩个链表是否相交

给出俩个单向链表的头指针，比如 `h1, h2`，判断这俩个链表是否相交。

为了简化问题，我们假设俩个链表均不带环。

问题扩展：

1.如果链表可能有环列？

2.如果需要求出俩个链表相交的第一个节点列？

第 8 题（算法）

此贴选一些 比较怪的题，，由于其中题目本身与算法关系不大，仅考考思维。特此并作一题。

1.有两个房间，一间房里有三盏灯，另一间房有控制着三盏灯的三个开关，

这两个房间是 分割开的，从一间里不能看到另一间的情况。

现在要求受训者分别进这两房间一次，然后判断出这三盏灯分别是由哪个开关控制的。

有什么办法呢？

2.你让一些人为你工作了七天，你要用一根金条作为报酬。金条被分成七小块，每天给出一块。

如果你只能将金条切割两次，你怎样分给这些工人？

3. ★用一种算法来颠倒一个链接表的顺序。现在在不用递归式的情况下做一遍。

★用一种算法在一个循环的链接表里插入一个节点，但不得穿越链接表。

★用一种算法整理一个数组。你为什么选择这种方法？

★用一种算法使通用字符串相匹配。

★颠倒一个字符串。优化速度。优化空间。

★颠倒一个句子中的词的顺序，比如将“我叫克丽丝”转换为“克丽丝叫我”，
实现速度最快，移动最少。

★找到一个子字符串。优化速度。优化空间。

★比较两个字符串，用 $O(n)$ 时间和恒量空间。

★假设你有一个用 1001 个整数组成的数组，这些整数是任意排列的，但是你知道所有的整数都在 1 到 1000(包括 1000)之间。此外，除一个数字出现两次外，其他所有数字只出现一次。假设你只能对这个数组做一次处理，用一种算法找出重复的那个数字。如果你在运算中使用了辅助的存储方式，那么你能找到不用这种方式的算法吗？

★不用乘法或加法增加 8 倍。现在用同样的方法增加 7 倍。

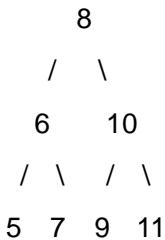
第 9 题 (树)

判断整数序列是不是二元查找树的后序遍历结果

题目：输入一个整数数组，判断该数组是不是某二元查找树的后序遍历的结果。

如果是返回 `true`，否则返回 `false`。

例如输入 5、7、6、9、11、10、8，由于这一整数序列是如下树的后序遍历结果：



因此返回 `true`。

如果输入 7、4、6、5，没有哪棵树的后序遍历的结果是这个序列，因此返回 `false`。

第 10 题 (字符串)

翻转句子中单词的顺序。

题目：输入一个英文句子，翻转句子中单词的顺序，但单词内字符的顺序不变。

句子中单词以空格符隔开。为简单起见，标点符号和普通字母一样处理。

例如输入 “I am a student.”，则输出 “student. a am I”。

第 11 题 (树)

求二叉树中节点的最大距离...

如果我们把二叉树看成一个图，父子节点之间的连线看成是双向的，

我们姑且定义“距离”为两节点之间边的个数。

写一个程序，

求一棵二叉树中相距最远的两个节点之间的距离。

第 12 题 (语法)

题目：求 $1+2+\dots+n$ ，

要求不能使用乘除法、`for`、`while`、`if`、`else`、`switch`、`case` 等关键字以及条件判断语句(`A?B:C`)。

第 13 题 (链表):

题目：输入一个单向链表，输出该链表中倒数第 k 个结点。链表的倒数第 0 个结点为链表的尾指针。

链表结点定义如下：

```
struct ListNode
```

```
{  
    int m_nKey;  
    ListNode* m_pNext;  
};
```

第 14 题 (数组):

题目：输入一个已经按升序排序过的数组和一个数字，

在数组中查找两个数，使得它们的和正好是输入的那个数字。

要求时间复杂度是 $O(n)$ 。如果有对数字的和等于输入的数字，输出任意一对即可。

例如输入数组 1、2、4、7、11、15 和数字 15。由于 $4+11=15$ ，因此输出 4 和 11。

第 15 题 (树):

题目：输入一颗二元查找树，将该树转换为它的镜像，

即在转换后的二元查找树中，左子树的结点都大于右子树的结点。

用递归和循环两种方法完成树的镜像转换。

例如输入：

```
8  
/\  
6 10  
/\ /\  
5 7 9 11
```

输出：

```
8  
/ \  
10 6  
/\ /\  
11 9 7 5
```

定义二元查找树的结点为：

```
struct BSTreeNode // a node in the binary search tree (BST)  
{  
    int m_nValue; // value of node  
    BSTreeNode *m_pLeft; // left child of node  
    BSTreeNode *m_pRight; // right child of node  
};
```

第 16 题 (树):

题目 (微软):

输入一颗二元树，从上往下按层打印树的每个结点，同一层中按照从左往右的顺序打印。

例如输入

```
8
 / \
6 10
/ \ \
5 7 9 11
```

输出 8 6 10 5 7 9 11。

第 17 题 (字符串):

题目：在一个字符串中找到第一个只出现一次的字符。如输入 abaccdeff，则输出 b。

分析：这道题是 2006 年 google 的一道笔试题。

第 18 题 (数组):

题目：n 个数字 (0,1,⋯,n-1) 形成一个圆圈，从数字 0 开始，

每次从这个圆圈中删除第 m 个数字 (第一个为当前数字本身，第二个为当前数字的下一个数字)。当一个数字删除后，从被删除数字的下一个继续删除第 m 个数字。

求出在这个圆圈中剩下的最后一个数字。

July：我想，这个题目，不少人已经见识过了。

第 19 题 (数组、递归):

题目：定义 Fibonacci 数列如下：

/ 0 n=0

f(n)=1 n=1

/ f(n-1)+f(n-2) n=2

输入 n，用最快的方法求该数列的第 n 项。

分析：在很多 C 语言教科书中讲到递归函数的时候，都会用 Fibonacci 作为例子。

因此很多程序员对这道题的递归解法非常熟悉，但....呵呵，你知道的。。

第 20 题 (字符串):

题目：输入一个表示整数的字符串，把该字符串转换成整数并输出。

例如输入字符串"345"，则输出整数 345。

第 21 题 (数组)

2010 年中兴面试题

编程求解：

输入两个整数 n 和 m , 从数列 1, 2, 3..... n 中 随意取几个数,
使其和等于 m , 要求将其中所有的可能组合列出来.

第 22 题 (推理):

有 4 张红色的牌和 4 张蓝色的牌, 主持人先拿任意两张, 再分别在 A、B、C 三人额头上贴任意两张牌, A、B、C 三人都可以看见其余两人额头上的牌, 看完后让他们猜自己额头上是什么颜色的牌, A 说不知道, B 说不知道, C 说不知道, 然后 A 说知道了。

请教如何推理, A 是怎么知道的。

如果用程序, 又怎么实现呢?

第 23 题 (算法):

用最简单, 最快速的方法计算出下面这个圆形是否和正方形相交。"

3D 坐标系 原点(0.0,0.0,0.0)

圆形:

半径 $r = 3.0$

圆心 $o = (*.* , 0.0, *.*)$

正方形:

4 个角坐标;

1:(*.* , 0.0, *.*)

2:(*.* , 0.0, *.*)

3:(*.* , 0.0, *.*)

4:(*.* , 0.0, *.*)

第 24 题 (链表):

链表操作, 单链表就地逆置,

第 25 题 (字符串):

写一个函数, 它的原形是 `int continuumax(char *outputstr,char *inputstr)`

功能:

在字符串中找出连续最长的数字串, 并把这个串的长度返回,

并把这个最长数字串付给其中一个函数参数 `outputstr` 所指内存。

例如: "abcd12345ed125ss123456789" 的首地址传给 `inputstr` 后, 函数将返回 9,

`outputstr` 所指的值为 123456789

26. 左旋转字符串（字符串）

题目：

定义字符串的左旋转操作：把字符串前面的若干个字符移动到字符串的尾部。

如把字符串 `abcdef` 左旋转 2 位得到字符串 `cdefab`。请实现字符串左旋转的函数。

要求时间对长度为 n 的字符串操作的复杂度为 $O(n)$ ，辅助内存为 $O(1)$ 。

27. 跳台阶问题（递归）

题目：一个台阶总共有 n 级，如果一次可以跳 1 级，也可以跳 2 级。

求总共有多少总跳法，并分析算法的时间复杂度。

这道题最近经常出现，包括 MicroStrategy 等比较重视算法的公司

都曾先后选用过这个这道题作为面试题或者笔试题。

28. 整数的二进制表示中 1 的个数（运算）

题目：输入一个整数，求该整数的二进制表达中有多少个 1。

例如输入 10，由于其二进制表示为 1010，有两个 1，因此输出 2。

分析：

这是一道很基本的考查位运算的面试题。

包括微软在内的很多公司都曾采用过这道题。

29. 栈的 push、pop 序列（栈）

题目：输入两个整数序列。其中一个序列表示栈的 push 顺序，

判断另一个序列有没有可能是对应的 pop 顺序。

为了简单起见，我们假设 push 序列的任意两个整数都是不相等的。

比如输入的 push 序列是 1、2、3、4、5，那么 4、5、3、2、1 就有可能是一个 pop 系列。

因为可以有如下的 push 和 pop 序列：

`push 1, push 2, push 3, push 4, pop, push 5, pop, pop, pop, pop,`

这样得到的 pop 序列就是 4、5、3、2、1。

但序列 4、3、5、1、2 就不可能是 push 序列 1、2、3、4、5 的 pop 序列。

30. 在从 1 到 n 的正数中 1 出现的次数（数组）

题目：输入一个整数 n ，求从 1 到 n 这 n 个整数的十进制表示中 1 出现的次数。

例如输入 12，从 1 到 12 这些整数中包含 1 的数字有 1, 10, 11 和 12，1 一共出现了 5 次。

分析：这是一道广为流传的 google 面试题。

31. 华为面试题（搜索）：

一类似于蜂窝的结构的图，进行搜索最短路径（要求 5 分钟）

32.（数组、规划）

有两个序列 a,b，大小都为 n,序列元素的值任意整数，无序；

要求：通过交换 a,b 中的元素，使[序列 a 元素的和]与[序列 b 元素的和]之间的差最小。

例如：

```
var a=[100,99,98,1,2,3];
var b=[1,2,3,4,5,40];
```

33.（字符串）

实现一个挺高级的字符匹配算法：

给一串很长字符串，要求找到符合要求的字符串，例如目的串：123

1*****3***2 ,12*****3 这些都要找出来

其实就是类似一些和谐系统。。。。

34.（队列）

实现一个队列。

队列的应用场景为：

一个生产者线程将 int 类型的数入列，一个消费者线程将 int 类型的数出列

35.（矩阵）

求一个矩阵中最大的二维矩阵(元素和最大).如：

1 2 0 3 4

2 3 4 5 1

1 1 5 3 0

中最大的是：

4 5

5 3

要求:(1)写出算法;(2)分析时间复杂度;(3)用 C 写出关键代码

第 36 题-40 题（有些题目搜集于 CSDN 上的网友，已标明）：

36.引用自网友：longzuo（运算）

谷歌笔试：

n 支队伍比赛，分别编号为 0, 1, 2... $n-1$ ，已知它们之间的实力对比关系，存储在一个二维数组 $w[n][n]$ 中， $w[i][j]$ 的值代表编号为 i , j 的队伍中更强的一支。所以 $w[i][j]=i$ 或者 j ，现在给出它们的出场顺序，并存储在数组 $order[n]$ 中，比如 $order[n]=\{4,3,5,8,1\dots\}$ ，那么第一轮比赛就是 4 对 3, 5 对 8。……胜者晋级，败者淘汰，同一轮淘汰的所有队伍排名不再细分，即可以随便排，下一轮由上一轮的胜者按照顺序，再依次两两比，比如可能是 4 对 5，直至出现第一名编程实现，给出二维数组 w ，一维数组 $order$ 和用于输出比赛名次的数组 $result[n]$ ，求出 $result$ 。

37. (字符串)

有 n 个长为 $m+1$ 的字符串，如果某个字符串的最后 m 个字符与某个字符串的前 m 个字符匹配，则两个字符串可以联接，问这 n 个字符串最多可以连成一个多长的字符串，如果出现循环，则返回错误。

38. (算法)

百度面试：

- 1.用天平（只能比较，不能称重）从一堆小球中找出其中唯一一个较轻的，使用 x 次天平，最多可以从 y 个小球中找出较轻的那个，求 y 与 x 的关系式。
- 2.有一个很大很大的输入流，大到没有存储器可以将其存储下来，而且只输入一次，如何从这个输入流中随机取得 m 个记录。
- 3.大量的 URL 字符串，如何从中去除重复的，优化时间空间复杂度

39. (树、图、算法)

网易有道笔试：

(1).

求一个二叉树中任意两个节点间的最大距离，两个节点的距离的定义是 这两个节点间边的个数，比如某个孩子节点和父节点间的距离是 1，和相邻兄弟节点间的距离是 2，优化时间空间复杂度。

(2).

求一个有向连通图的割点，割点的定义是，如果除去此节点和与其相关的边，有向图不再连通，描述算法。

40. 百度研发笔试题（栈、算法）

引用自：zp155334877

1)设计一个栈结构，满足一下条件：min, push, pop 操作的时间复杂度为 O(1)。

2)一串首尾相连的珠子(m 个)，有 N 种颜色(N<=10)，

设计一个算法，取出其中一段，要求包含所有 N 中颜色，并使长度最短。

并分析时间复杂度与空间复杂度。

3)设计一个系统处理词语搭配问题，比如说 中国 和人民可以搭配，

则中国人民 人民中国都有效。要求：

*系统每秒的查询数量可能上千次；

*词语的数量级为 10W；

*每个词至多可以与 1W 个词搭配

当用户输入中国人民的时候，要求返回与这个搭配词组相关的信息。

41.求固晶机的晶元查找程序（匹配、算法）

晶元盘由数目不详的大小一样的晶元组成，晶元并不一定全布满晶元盘，

照相机每次这能匹配一个晶元，如匹配过，则拾取该晶元，

若匹配不过，照相机则按测好的晶元间距移到下一个位置。

求遍历晶元盘的算法 求思路。

42.请修改 append 函数，利用这个函数实现（链表）：

两个非降序链表的并集，1->2->3 和 2->3->5 并为 1->2->3->5

另外只能输出结果，不能修改两个链表的数据。

43.递归和非递归俩种方法实现二叉树的前序遍历。

44.腾讯面试题（算法）：

1.设计一个魔方（六面）的程序。

2.有一千万条短信，有重复，以文本文件的形式保存，一行一条，有重复。

请用 5 分钟时间，找出重复出现最多的前 10 条。

3.收藏了 1 万条 url，现在给你一条 url，如何找出相似的 url。（面试官不解释何为相似）

45.雅虎（运算、矩阵）：

1.对于一个整数矩阵，存在一种运算，对矩阵中任意元素加一时，需要其相邻（上下左右）某一个元素也加一，现给出一正数矩阵，判断其是否能够由一个全零矩阵经过上述运算得到。

2.一个整数数组，长度为 n，将其分为 m 份，使各份的和相等，求 m 的最大值

比如{3, 2, 4, 3, 6} 可以分成{3, 2, 4, 3, 6} m=1;

{3,6}{2,4,3} m=2

{3,3}{2,4}{6} m=3 所以 m 的最大值为 3

46. 搜狐 (运算):

四对括号可以有多少种匹配排列方式？比如两对括号可以有两种：() () 和 (())

47. 创新工场 (算法):

求一个数组的最长递减子序列 比如{9, 4, 3, 2, 5, 4, 3, 2}的最长递减子序列为{9, 5, 4, 3, 2}

48. 微软 (运算):

一个数组是由一个递减数列左移若干位形成的，比如{4, 3, 2, 1, 6, 5}
是由{6, 5, 4, 3, 2, 1}左移两位形成的，在这种数组中查找某一个数。

49. 一道看上去很吓人的算法面试题 (排序、算法):

如何对 n 个数进行排序，要求时间复杂度 O(n)，空间复杂度 O(1)

50. 网易有道笔试 (sorry, 与第 39 题重复):

1. 求一个二叉树中任意两个节点间的最大距离，两个节点的距离的定义是 这两个节点间边的个数，

比如某个孩子节点和父节点间的距离是 1，和相邻兄弟节点间的距离是 2，优化时间空间复杂度。

2. 求一个有向连通图的割点，割点的定义是，

如果除去此节点和与其相关的边，有向图不再连通，描述算法。

51. 和为 n 连续正数序列 (数组)。

题目：输入一个正数 n，输出所有和为 n 连续正数序列。

例如输入 15，由于 $1+2+3+4+5=4+5+6=7+8=15$ ，所以输出 3 个连续序列 1-5、4-6 和 7-8。

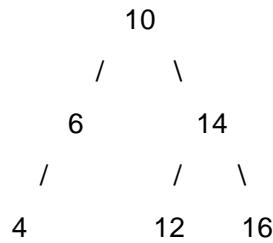
分析：这是网易的一道面试题。

52. 二元树的深度 (树)。

题目：输入一棵二元树的根结点，求该树的深度。

从根结点到叶结点依次经过的结点（含根、叶结点）形成树的一条路径，最长路径的长度为树的深度。

例如：输入二元树：



输出该树的深度 3。

二元树的结点定义如下：

```

struct SBinaryTreeNode // a node of the binary tree
{
    int m_nValue; // value of node
    SBinaryTreeNode *m_pLeft; // left child of node
    SBinaryTreeNode *m_pRight; // right child of node
};

```

分析：这道题本质上还是考查二元树的遍历。

53. 字符串的排列（字符串）。

题目：输入一个字符串，打印出该字符串中字符的所有排列。

例如输入字符串 abc，则输出由字符 a、b、c 所能排列出来的所有字符串
abc、acb、bac、bca、cab 和 cba。

分析：这是一道很好的考查对递归理解的编程题，

因此在过去一年中频繁出现在各大公司的面试、笔试题中。

54. 调整数组顺序使奇数位于偶数前面（数组）。

题目：输入一个整数数组，调整数组中数字的顺序，使得所有奇数位于数组的前半部分，所有偶数位于数组的后半部分。要求时间复杂度为 O(n)。

55.（语法）

题目：类 CMyString 的声明如下：

```

class CMyString
{
public:
    CMyString(char* pData = NULL);
    CMyString(const CMyString& str);
    ~CMyString(void);
    CMyString& operator = (const CMyString& str);
}

```

```
private:
    char* m_pData;
};
```

请实现其赋值运算符的重载函数，要求异常安全，即当对一个对象进行赋值时发生异常，对象的状态不能改变。

56.最长公共字串（算法、字符串）。

题目：如果字符串一的所有字符按其在字符串中的顺序出现在另外一个字符串二中，则字符串一称之为字符串二的子串。

注意，并不要求子串（字符串一）的字符必须连续出现在字符串二中。

请编写一个函数，输入两个字符串，求它们的最长公共子串，并打印出最长公共子串。

例如：输入两个字符串 BDCABA 和 ABCBDAB，字符串 BCBA 和 BDAB 都是它们的最长公共子串，则输出它们的长度 4，并打印任意一个子串。

分析：求最长公共子串（Longest Common Subsequence, LCS）是一道非常经典的动态规划题，因此一些重视算法的公司像 MicroStrategy 都把它当作面试题。

57.用俩个栈实现队列（栈、队列）。

题目：某队列的声明如下：

```
template<typename T> class CQueue
{
public:
    CQueue() {}
    ~CQueue() {}

    void appendTail(const T& node); // append a element to tail
    void deleteHead();           // remove a element from head

private:
    Stack<T> m_stack1;
    Stack<T> m_stack2;
};
```

分析：从上面的类的声明中，我们发现在队列中有两个栈。

因此这道题实质上是要求我们用两个栈来实现一个队列。

相信大家对栈和队列的基本性质都非常了解了：栈是一种后入先出的数据容器，因此对队列进行的插入和删除操作都是在栈顶上进行；队列是一种先入先出的数据容器，我们总是把新元素插入到队列的尾部，而从队列的头部删除元素。

58.从尾到头输出链表（链表）。

题目：输入一个链表的头结点，从尾到头反过来输出每个结点的值。链表结点定义如下：

```
struct ListNode
{
    int m_nKey;
    ListNode* m_pNext;
};
```

分析：这是一道很有意思的面试题。

该题以及它的变体经常出现在各大公司的面试、笔试题中。

59.不能被继承的类（语法）。

题目：用 C++ 设计一个不能被继承的类。

分析：这是 Adobe 公司 2007 年校园招聘的最新笔试题。

这道题除了考察应聘者的 C++ 基本功底外，还能考察反应能力，是一道很好的题目。

60.在 O(1) 时间内删除链表结点（链表、算法）。

题目：给定链表的头指针和一个结点指针，在 O(1) 时间删除该结点。链表结点的定义如下：

```
struct ListNode
{
    int m_nKey;
    ListNode* m_pNext;
};
```

函数的声明如下：

```
void DeleteNode(ListNode* pListHead, ListNode* pToBeDeleted);
```

分析：这是一道广为流传的 Google 面试题，能有效考察我们的编程基本功，还能考察我们的反应速度，更重要的是，还能考察我们对时间复杂度的理解。

61.找出数组中两个只出现一次的数字（数组）

题目：一个整型数组里除了两个数字之外，其他的数字都出现了两次。

请写程序找出这两个只出现一次的数字。要求时间复杂度是 O(n)，空间复杂度是 O(1)。

分析：这是一道很新颖的关于位运算的面试题。

62.找出链表的第一个公共结点（链表）。

题目：两个单向链表，找出它们的第一个公共结点。

链表的结点定义为：

```
struct ListNode
{
    int m_nKey;
    ListNode* m_pNext;
};
```

分析：这是一道微软的面试题。微软非常喜欢与链表相关的题目，因此在微软的面试题中，链表出现的概率相当高。

63. 在字符串中删除特定的字符（字符串）。

题目：输入两个字符串，从第一字符串中删除第二个字符串中所有的字符。

例如，输入”They are students.” 和”aeiou”，

则删除之后的第一个字符串变成”Thy r stdnts.”。

分析：这是一道微软面试题。在微软的常见面试题中，与字符串相关的题目占了很大的一部分，因为写程序操作字符串能很好的反映我们的编程基本功。

64. 寻找丑数（运算）。

题目：我们把只包含因子 2、3 和 5 的数称作丑数（Ugly Number）。例如 6、8 都是丑数，但 14 不是，因为它包含因子 7。习惯上我们把 1 当做是第一个丑数。

求按从小到大的顺序的第 1500 个丑数。

分析：这是一道在网络上广为流传的面试题，据说 google 曾经采用过这道题。

65. 输出 1 到最大的 N 位数（运算）

题目：输入数字 n，按顺序输出从 1 最大的 n 位 10 进制数。比如输入 3，

则输出 1、2、3 一直到最大的 3 位数即 999。

分析：这是一道很有意思的题目。看起来很简单，其实里面却有不少的玄机。

66. 颠倒栈（栈）。

题目：用递归颠倒一个栈。例如输入栈{1, 2, 3, 4, 5}，1 在栈顶。

颠倒之后的栈为{5, 4, 3, 2, 1}，5 处在栈顶。

67. 倆个闲玩娱乐（运算）。

1. 扑克牌的顺子

从扑克牌中随机抽 5 张牌，判断是不是一个顺子，即这 5 张牌是不是连续的。

2-10 为数字本身，A 为 1，J 为 11，Q 为 12，K 为 13，而大小王可以看成任意数字。

2.n 个骰子的点数。

把 n 个骰子扔在地上，所有骰子朝上一面的点数之和为 S。输入 n，
打印出 S 的所有可能的值出现的概率。

68. 把数组排成最小的数（数组、算法）。

题目：输入一个正整数数组，将它们连接起来排成一个数，输出能排出的所有数字中最小的一个。

例如输入数组{32, 321}，则输出这两个能排成的最小数字 32132。

请给出解决问题的算法，并证明该算法。

分析：这是 09 年 6 月份百度的一道面试题，

从这道题我们可以看出百度对应聘者在算法方面有很高的要求。

69. 旋转数组中的最小元素（数组、算法）。

题目：把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。输入一个排好序的数组的一个旋转，输出旋转数组的最小元素。

例如数组{3, 4, 5, 1, 2}为{1, 2, 3, 4, 5}的一个旋转，该数组的最小值为 1。

分析：这道题最直观的解法并不难。从头到尾遍历数组一次，就能找出最小的元素，时间复杂度显然是 O(N)。但这个思路没有利用输入数组的特性，我们应该能找到更好的解法。

70. 给出一个函数来输出一个字符串的所有排列（经典字符串问题）。

ANSWER 简单的回溯就可以实现了。当然排列的产生也有很多种算法，去看看组合数学，还有逆序生成排列和一些不需要递归生成排列的方法。

印象中 Knuth 的<TAOCP>第一卷里面深入讲了排列的生成。这些算法的理解需要一定的数学功底，也需要一定的灵感，有兴趣最好看看。

71. 数值的整数次方（数字、运算）。

题目：实现函数 double Power(double base, int exponent)，求 base 的 exponent 次方。

不需要考虑溢出。

分析：这是一道看起来很简单的问题。可能有不少的人在看到题目后 30 秒写出如下的代码：

```
double Power(double base, int exponent)
{
    double result = 1.0;
    for(int i = 1; i <= exponent; ++i)
        result *= base;
    return result;
```

```
}
```

72. (语法)

题目：设计一个类，我们只能生成该类的一个实例。

分析：只能生成一个实例的类是实现了 **Singleton** 模式的类型。

73. 对称字符串的最大长度（字符串）。

题目：输入一个字符串，输出该字符串中对称的子字符串的最大长度。

比如输入字符串“google”，由于该字符串里最长的对称子字符串是“goog”，因此输出 4。

分析：可能很多人都写过判断一个字符串是不是对称的函数，这个题目可以看成是该函数的加强版。

74. 数组中超过出现次数超过一半的数字（数组）

题目：数组中有一个数字出现的次数超过了数组长度的一半，找出这个数字。

分析：这是一道广为流传的面试题，包括百度、微软和 Google 在内的多家公司都曾经采用过这个题目。要几十分钟的时间里很好地解答这道题，除了较好的编程能力之外，还需要较快的反应和较强的逻辑思维能力。

75. 二叉树两个结点的最低共同父结点（树）

题目：二叉树的结点定义如下：

```
struct TreeNode
{
    int m_nvalue;
    TreeNode* m_pLeft;
    TreeNode* m_pRight;
};
```

输入二叉树中的两个结点，输出这两个结点在数中最低的共同父结点。

分析：求数中两个结点的最低共同结点是面试中经常出现的一个问题。这个问题至少有两个变种。

76. 复杂链表的复制（链表、算法）

题目：有一个复杂链表，其结点除了有一个 **m_pNext** 指针指向下一个结点外，

还有一个 **m_pSibling** 指向链表中的任一结点或者 **NULL**。其结点的 C++ 定义如下：

```
struct ComplexNode
{
```

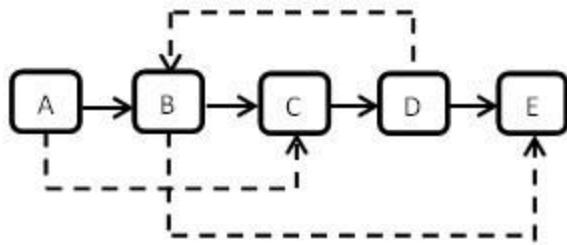
```

int m_nValue;
ComplexNode* m_pNext;
ComplexNode* m_pSibling;
};

```

下图是一个含有 5 个结点的该类型复杂链表。

图中实线箭头表示 `m_pNext` 指针，虚线箭头表示 `m_pSibling` 指针。为简单起见，指向 `NULL` 的指针没有画出。



请完成函数 `ComplexNode* Clone(ComplexNode* pHead)`，以复制一个复杂链表。

分析：在常见的数据结构上稍加变化，这是一种很新颖的面试题。

要在不到一个小时的时间里解决这种类型的题目，我们需要较快的反应能力，对数据结构透彻的理解以及扎实的编程功底。

77. 关于链表问题的面试题目如下（链表）：

1. 给定单链表，检测是否有环。

使用两个指针 `p1, p2` 从链表头开始遍历，`p1` 每次前进一步，`p2` 每次前进两步。如果 `p2` 到达链表尾部，说明无环，否则 `p1, p2` 必然会在某个时刻相遇(`p1==p2`)，从而检测到链表中有环。

2. 给定两个单链表(`head1, head2`)，检测两个链表是否有交点，如果有返回第一个交点。

如果 `head1==head2`，那么显然相交，直接返回 `head1`。

否则，分别从 `head1, head2` 开始遍历两个链表获得其长度 `len1` 与 `len2`，假设 `len1>=len2`，那么指针 `p1` 由 `head1` 开始向后移动 `len1-len2` 步，指针 `p2=head2`，

下面 `p1, p2` 每次向后前进一步并比较 `p1==p2` 是否相等，如果相等即返回该结点，否则说明两个链表没有交点。

3. 给定单链表(`head`)，如果有环的话请返回从头结点进入环的第一个节点。

运用题一，我们可以检查链表中是否有环。

如果有环，那么 `p1=p2` 重合点 `p` 必然在环中。从 `p` 点断开环，

方法为：`p1=p, p2=p->next, p->next=NULL`。此时，原单链表可以看作两条单链表，一条从 `head` 开始，另一条从 `p2` 开始，于是运用题二的方法，我们找到它们的第一个交点即为所求。

4. 只给定单链表中某个结点 `p`(并非最后一个结点，即 `p->next!=NULL`)指针，删除该结点。

办法很简单，首先是放 p 中数据,然后将 p->next 的数据 copy 入 p 中，接下来删除 p->next 即可。

5.只给定单链表中某个结点 p(非空结点)，在 p 前面插入一个结点。

办法与前者类似，首先分配一个结点 q，将 q 插入在 p 后，接下来将 p 中的数据 copy 入 q 中，然后再将要插入的数据记录在 p 中。

78.链表和数组的区别在哪里（链表、数组）？

分析：主要在基本概念上的理解。

但是最好能考虑的全面一点，现在公司招人的竞争可能就在细节上产生，谁比较仔细，谁获胜的机会就大。

79.（链表、字符串）

1.编写实现链表排序的一种算法。说明为什么你会选择用这样的方法？

2.编写实现数组排序的一种算法。说明为什么你会选择用这样的方法？

3.请编写能直接实现 strstr() 函数功能的代码。

80.阿里巴巴一道笔试题（运算、算法）

问题描述：

12 个高矮不同的人,排成两排,每排必须是从矮到高排列,而且第二排比对应的第一排的人高,问排列方式有多少种?

这个笔试题,很 YD,因为把某个递归关系隐藏得很深。

先来几组百度的面试题：

=====

81.第 1 组百度面试题

1.一个 int 数组，里面数据无任何限制，要求求出所有这样的数 a[i]，其左边的数都小于等于它，右边的数都大于等于它。

能否只用一个额外数组和少量其它空间实现。

2.一个文件，内含一千万行字符串，每个字符串在 1K 以内，

要求找出所有相反的串对，如 abc 和 cba。

3.STL 的 set 用什么实现？为什么不用 hash？

82.第 2 组百度面试题

1.给出两个集合 A 和 B，其中集合 A={name}，
集合 B={age、sex、scholarship、address、...}，

要求：

问题 1、根据集合 A 中的 name 查询出集合 B 中对应的属性信息；

问题 2、根据集合 B 中的属性信息(单个属性, 如 age<20 等), 查询出集合 A 中对应的 name。

2.给出一个文件，里面包含两个字段{url、size}，

即 url 为网址, size 为对应网址访问的次数,

要求：

问题 1、利用 Linux Shell 命令或自己设计算法，

查询出 url 字符串中包含 “baidu” 子字符串对应的 size 字段值；

问题 2、根据问题 1 的查询结果，对其按照 size 由大到小的排列。

(说明：url 数据量很大，100 亿级以上)

83.第 3 组百度面试题

1.今年百度的一道题目

百度笔试：给定一个存放整数的数组，重新排列数组使得数组左边为奇数，右边为偶数。

要求：空间复杂度 O(1)，时间复杂度为 O (n)。

2.百度笔试题

用 C 语言实现函数 void * memmove(void *dest, const void *src, size_t n)。

memmove 函数的功能是拷贝 src 所指的内存内容前 n 个字节到 dest 所指的地址上。

分析：

由于可以把任何类型的指针赋给 void 类型的指针

这个函数主要是实现各种数据类型的拷贝。

84.第 4 组百度面试题

2010 年 3 道百度面试题[相信，你懂其中的含金量]

1.a~z 包括大小写与 0~9 组成的 N 个数

用最快的方式把其中重复的元素挑出来。

2.已知一随机发生器，产生 0 的概率是 p，产生 1 的概率是 1-p，现在要你构造一个发生器，使得它构造 0 和 1 的概率均为 1/2；构造一个发生器，使得它构造 1、2、3 的概率均为 1/3；…，构造一个发生器，使得它构造 1、2、3、…n 的概率均为 1/n，要求复杂度最低。

3.有 10 个文件，每个文件 1G，

每个文件的每一行都存放的是用户的 query，每个文件的 query 都可能重复。

要求按照 query 的频度排序。

85.又见字符串的问题

1.给出一个函数来复制两个字符串 A 和 B。

字符串 A 的后几个字节和字符串 B 的前几个字节重叠。

分析：记住，这种题目往往就是考你对边界的考虑情况。

2. 已知一个字符串，比如 asderwsde, 寻找其中的一个子字符串比如 sde 的个数，如果没有返回 0，有的话返回子字符串的个数。

86.

怎样编写一个程序，把一个有序整数数组放到二叉树中？

分析：本题考察二叉搜索树的建树方法，简单的递归结构。

关于树的算法设计一定要联想到递归，因为树本身就是递归的定义。

而，学会把递归改称非递归也是一种必要的技术。

毕竟，递归会造成栈溢出，关于系统底层的程序中不到非不得以最好不要用。

但是对某些数学问题，就一定要学会用递归去解决。

87.

1. 大整数数相乘的问题。（这是 2002 年在一考研班上遇到的算法题）

2. 求最大连续递增数字串（如“ads3sl456789DF3456ld345AA”中的“456789”）

3. 实现 strstr 功能，即在父串中寻找子串首次出现的位置。

（笔试中常让面试者实现标准库中的一些函数）

88. 2005 年 11 月金山笔试题。编码完成下面的处理函数。

函数将字符串中的字符'*'移到串的前部分，

前面的非'*'字符后移，但不能改变非'*'字符的先后顺序，函数返回串中字符'*'的数量。

如原始串为：ab**cd**e*12，

处理后为*****abcde12，函数并返回值为 5。（要求使用尽量少的时间和辅助空间）

89. 神州数码、华为、东软笔试题

1. 2005 年 11 月 15 日华为软件研发笔试题。实现一单链表的逆转。

2. 编码实现字符串转整型的函数（实现函数 atoi 的功能），据说是神州数码笔试题。如将字

符串 "+123" 123, "-0123" -123, "123CS45" 123, "123.45CS" 123, "CS123.45"

0

3. 快速排序（东软喜欢考类似的算法填空题，又如堆排序的算法等）

4. 删除字符串中的数字并压缩字符串。

如字符串 "abc123de4fg56" 处理后变为 "abcdefg"。注意空间和效率。

（下面的算法只需要一次遍历，不需要开辟新空间，时间复杂度为 O(N)）

5. 求两个串中的第一个最长子串（神州数码以前试题）。

如"abRACTyeyt", "dgdsaeACTyey"的最大子串为"actyet"。

90.

1.不开辟用于交换数据的临时空间，如何完成字符串的逆序

(在技术一轮面试中，有些面试官会这样问)。

2.删除串中指定的字符

(做此题时，千万不要开辟新空间，否则面试官可能认为你不适合做嵌入式开发)

3.判断单链表中是否存在环。

91.

1.一道著名的毒酒问题

有 1000 桶酒，其中 1 桶有毒。而一旦吃了，毒性会在 1 周后发作。

现在我们用小老鼠做实验，要在 1 周内找出那桶毒酒，问最少需要多少老鼠。

2.有趣的石头问题

有一堆 1 万个石头和 1 万个木头，对于每个石头都有 1 个木头和它重量一样，
把配对的石头和木头找出来。

92.

1.多人排成一个队列,我们认为从低到高是正确的序列,但是总有部分人不遵守秩序。

如果说,前面的人比后面的人高(两人身高一样认为是合适的),

那么我们就认为这两个人是一对“捣乱分子”,比如说,现在存在一个序列:

176, 178, 180, 170, 171

这些捣乱分子对为

<176, 170>, <176, 171>, <178, 170>, <178, 171>, <180, 170>, <180, 171>,

那么,现在给出一个整型序列,请找出这些捣乱分子对的个数(仅给出捣乱分子对的数目即可,
不用具体的对)

要求:

输入:

为一个文件(**in**), 文件的每一行为一个序列。序列全为数字, 数字间用”,”分隔。

输出:

为一个文件(**out**), 每行为一个数字, 表示捣乱分子的对数。

详细说明自己的解题思路, 说明自己实现的一些关键点。

并给出实现的代码, 并分析时间复杂度。

限制:

输入每行的最大数字个数为 100000 个, 数字最长为 6 位。程序无内存使用限制。

93.在一个 int 数组里查找这样的数，它大于等于左侧所有数，小于等于右侧所有数。

直观想法是用两个数组 **a**、**b**。**a[i]**、**b[i]**分别保存从前到 *i* 的最大的数和从后到 *i* 的最小的数，一个解答：这需要两次遍历，然后再遍历一次原数组，

将所有 **data[i]>=a[i-1]&&data[i]<=b[i]** 的 **data[i]** 找出即可。

给出这个解答后，面试官有要求只能用一个辅助数组，且要求少遍历一次。

94.微软笔试题

求随机数构成的数组中找到长度大于=3 的最长的等差数列 9 d- x W) w9 ?" o3 b0 R

输出等差数列由小到大：

如果没有符合条件的就输出

格式：

输入[1,3,0,5,-1,6]

输出[-1,1,3,5]

要求时间复杂度，空间复杂度尽量小

95.华为面试题

1 判断一字符串是不是对称的，如： abccba

2.用递归的方法判断整数组 **a[N]** 是不是升序排列

96.08 年中兴校园招聘笔试题

1.编写 **strcpy** 函数

已知 **strcpy** 函数的原型是

char *strcpy(char *strDest, const char *strSrc);

其中 **strDest** 是目的字符串，**strSrc** 是源字符串。不调用 C++/C 的字符串库函数，请编写函数 **strcpy**

最后压轴之戏，终结此微软等 100 题系列 V0.1 版。

那就，

连续来几组微软公司的面试题，让你一次爽个够：

=====

97.第 1 组微软较简单的算法面试题

1.编写反转字符串的程序，要求优化速度、优化空间。

2.在链表里如何发现循环链接？

3.编写反转字符串的程序，要求优化速度、优化空间。

- 4.给出洗牌的一个算法，并将洗好的牌存储在一个整形数组里。
- 5.写一个函数，检查字符是否是整数，如果是，返回其整数值。
(或者：怎样只用 4 行代码编写出一个从字符串到长整形的函数？)

98.第 2 组微软面试题

- 1.给出一个函数来输出一个字符串的所有排列。
- 2.请编写实现 `malloc()` 内存分配函数功能一样的代码。
- 3.给出一个函数来复制两个字符串 A 和 B。字符串 A 的后几个字节和字符串 B 的前几个字节重叠。
- 4.怎样编写一个程序，把一个有序整数数组放到二叉树中？
- 5.怎样从顶部开始逐层打印二叉树结点数据？请编程。
- 6.怎样把一个链表掉个顺序（也就是反序，注意链表的边界条件并考虑空链表）？

99.第 3 组微软面试题

- 1.烧一根不均匀的绳，从头烧到尾总共需要 1 个小时。
现在有若干条材质相同的绳子，问如何用烧绳的方法来计时一个小时十五分钟呢？
- 2.你有一桶果冻，其中有黄色、绿色、红色三种，闭上眼睛抓取同种颜色的两个。
抓取多少个就可以确定你肯定有两个同一颜色的果冻？(5 秒-1 分钟)
- 3.如果你有无穷多的水，一个 3 公升的提桶，一个 5 公升的提桶，两只提桶形状上下都不均匀，问你如何才能准确称出 4 公升的水？(40 秒-3 分钟)
一个岔路口分别通向诚实国和说谎国。
来了两个人，已知一个是诚实国的，另一个是说谎国。
诚实国永远说实话，说谎国永远说谎话。现在你要去说谎国，
但不知道应该走哪条路，需要问这两个人。请问应该怎么问？(20 秒-2 分钟)

100.第 4 组微软面试题，挑战思维极限

- 1.12 个球一个天平，现知道只有一个和其它的重量不同，问怎样称才能用三次就找到那个球。13 个呢？(注意此题并未说明那个球的重量是轻是重，所以需要仔细考虑)(5 分钟-1 小时)
- 2.在 9 个点上画 10 条直线，要求每条直线上至少有三个点？(3 分钟-20 分钟)
- 3.在一天的 24 小时之中，时钟的时针、分针和秒针完全重合在一起的时候有几次？
都分别是什么时间？你怎样算出来的？(5 分钟-15 分钟)

终结附加题：

微软面试题，挑战你的智商

=====

说明：如果你是第一次看到这种题，并且以前从来没有见过类似的题型，
并且能够在半个小时之内做出答案，说明你的智力超常..)

1.第一题 . 五个海盗抢到了 100 颗宝石，每一颗都一样大小和价值连城。他们决定这么分：
抽签决定自己的号码（1、2、3、4、5）

首先，由 1 号提出分配方案，然后大家表决，当且仅当超过半数的人同意时，
按照他的方案进行分配，否则将被扔进大海喂鲨鱼

如果 1 号死后，再由 2 号提出分配方案，然后剩下的 4 人进行表决，
当且仅当超过半数的人同意时，按照他的方案进行分配，否则将被扔入大海喂鲨鱼。
依此类推

条件：每个海盗都是很聪明的人，都能很理智地做出判断，从而做出选择。

问题：第一个海盗提出怎样的分配方案才能使自己的收益最大化？

2.一道关于飞机加油的问题，已知：

每个飞机只有一个油箱，

飞机之间可以相互加油（注意是相互，没有加油机）

一箱油可供一架飞机绕地球飞半圈，

问题：

为使至少一架飞机绕地球一圈回到起飞时的飞机场，至少需要出动几架飞机？

（所有飞机从同一机场起飞，而且必须安全返回机场，不允许中途降落，中间没有飞机场）

//欢迎，关注另外不同的更精彩的 100 题 V0.2 版，和此 V0.1 版的答案等后续内容。

完。

此外，关于此 100 道面试题的所有一切详情，包括[答案](#)，资源下载，帖子维护，答案更新，
都请参考此文：[横空出世，席卷 Csdn \[评微软等数据结构+算法面试 100 题\]](#)。

作者声明：

本人 July 对以上所有任何内容和资料享有版权，转载请注明作者本人 July 及出处。
向您的厚道致敬。谢谢。二零一零年十二月六日。

微软等数据结构+算法面试 100 题全部答案集锦

作者：July、阿财。

时间：二零一一年十月十三日。

引言

无私分享造就开源的辉煌。

今是二零一一年十月十三日，明日 14 日即是本人刚好开博一周年。在一周年之际，特此分享出微软面试全部 100 题答案的完整版，以作为对本博客所有读者的回馈。

一年之前的 10 月 14 日，一个名叫 July（头像为手冢国光）的人在一个叫 csdn 的论坛上开帖分享微软等公司数据结构+算法面试 100 题，自此，与上千网友一起做，一起思考，一起解答这些面试题目，最终成就了一个名为：**结构之法算法之道的编程面试与算法研究并重的博客**，如今，此博客影响力逐步渗透到海外，及至到整个互联网。

在此之前，由于本人笨拙，这微软面试 100 题的答案只整理到了前 60 题（第 1-60 题答案可到本人资源下载处下载：http://v_july_v.download.csdn.net/），故此，常有朋友留言或来信询问后面 40 题的答案。只是因个人认为：一、答案只是作为一个参考，不可太过依赖；二、常常因一些事情耽搁（如在整理最新的今年九月、十月份的面试题：**九月腾讯，创新工场，淘宝等公司最新面试十三题、十月百度，阿里巴巴，迅雷搜狗最新面试十一题**）；三、个人正在针对那 100 题一题一题的写文章，多种思路，不断优化，即成**程序员编程艺术系列**（详情，参见文末）。自此，后面 40 题的答案迟迟未得整理。且个人已经整理的前 60 题的答案，在我看来，是有诸多问题与弊端的，甚至很多答案都是错误的。

(微软 10 题永久讨论地址：http://topic.csdn.net/u/20101126/10/b4f12a00-6280-492f-b785-cb6835a63dc9_9.html)

互联网总是能给人带来惊喜。前几日，一位现居美国加州的名叫阿财的朋友发来一封邮件，并把他自己做的全部 100 题的答案一并发予给我，自此，便似遇见了知己。十分感谢。

任何东西只有分享出来才更显其价值。本只需贴出后面 40 题的答案，因为前 60 题的答案本人早已整理上传至网上，但多一种思路多一种参考亦未尝不可。特此，把阿财的答案再稍加整理番，然后把全部 100 题的答案现今都贴出来。若有任何问题，欢迎不吝指正。谢谢。

上千上万的人都关注过此 100 题，且大都都各自贡献了自己的思路，或回复于**微软 100 题维护地址**上，或回复于本博客内，人数众多，无法一一标明，特此向他们诸位表示敬意和

感谢。谢谢大家，诸君的努力足以影响整个互联网，咱们已经迎来一个分享互利的新时代。

微软面试 100 题全部答案

最新整理的全部 100 题的答案参见如下（重复的，以及一些无关紧要的题目跳过。且因尊重阿财，未作过多修改。因此，有些答案是还有问题的，最靠谱的答案以[程序员编程艺术系列](#)为准，亦可参考个人之前整理的前 60 题的答案：

- 第 1-20 题答案：http://blog.csdn.net/v_JULY_v/archive/2011/01/10/6126406.aspx;
- 第 21-40 题答案：http://blog.csdn.net/v_JULY_v/archive/2011/01/10/6126444.aspx;
- 第 41-60 题答案：http://blog.csdn.net/v_JULY_v/archive/2011/02/01/6171539.aspx.

更新：有朋友反应，以下的答案中思路过于简略，还是这句话，一切以[程序员编程艺术系列](#)（多种思路，多种比较，细细读之自晓其理）为准（[我没怎么看阿财的这些答案，因为编程艺术系列已经说得足够清晰了。](#)之所以把阿财的这份答案分享出来，一者，编程艺术系列目前还只写到了第二十二章，即 100 题之中还只详细阐述了近 30 道题；二者，他给的答案全部是用英文写的，这恰好方便国外的一些朋友参考；三者是为了给那一些急功近利的、浮躁的人一份速成的答案罢了）。July、二零一一年十月二十四日更新。

当然，读者朋友有任何问题，你也可以跟阿财联系，他的邮箱地址是：kevinn9@gmail.com (把#改成@)。

1. 把二元查找树转变成排序的双向链表

题目：

输入一棵二元查找树，将该二元查找树转换成一个排序的双向链表。

要求不能创建任何新的结点，只调整指针的指向。

```
10
/
  \
6   14
/ \ / \
4 8 12 16
```

转换成双向链表

4=6=8=10=12=14=16。

首先我们定义的二元查找树节点的数据结构如下：

```
struct BSTreeNode
{
    int m_nValue; // value of node
    BSTreeNode *m_pLeft; // left child of node
```

```
    BSTreeNode *m_pRight; // right child of node  
};
```

ANSWER:

This is a traditional problem that can be solved using recursion.

For each node, connect the double linked lists created from left and right child node to form a full list.

```
/**  
 * @param root The root node of the tree  
 * @return The head node of the converted list.  
 */  
BSTreeNode * treeToLinkedList(BSTreeNode * root) {  
    BSTreeNode * head, * tail;  
    helper(head, tail, root);  
    return head;  
}  
  
void helper(BSTreeNode *& head, BSTreeNode *& tail, BSTreeNode *root) {  
    BSTreeNode *lt, *rh;  
    if (root == NULL) {  
        head = NULL, tail = NULL;  
        return;  
    }  
    helper(head, lt, root->m_pLeft);  
    helper(rh, tail, root->m_pRight);  
    if (lt!=NULL) {  
        lt->m_pRight = root;  
        root->m_pLeft = lt;  
    } else {  
        head = root;  
    }  
    if (rh!=NULL) {  
        root->m_pRight=rh;  
        rh->m_pLeft = root;  
    } else {  
        tail = root;  
    }  
}
```

2.设计包含 min 函数的栈。

定义栈的数据结构，要求添加一个 min 函数，能够得到栈的最小元素。

要求函数 min、push 以及 pop 的时间复杂度都是 O(1)。

ANSWER:

Stack is a LIFO data structure. When some element is popped from the stack, the status will recover to the original status as before that element was pushed. So we can recover the minimum element, too.

```
struct MinStackElement {
    int data;
    int min;
};

struct MinStack {
    MinStackElement * data;
    int size;
    int top;
}

MinStack MinStackInit(int maxSize) {
    MinStack stack;
    stack.size = maxSize;
    stack.data = malloc(sizeof(MinStackElement)*maxSize);
    stack.top = 0;
    return stack;
}

void MinStackFree(MinStack stack) {
    free(stack.data);
}

void MinStackPush(MinStack stack, int d) {
    if (stack.top == stack.size) error("out of stack space.");
    MinStackElement* p = stack.data[stack.top];
    p->data = d;
    p->min = (stack.top == 0 ? d : stack.data[top-1].min);
    if (p->min > d) p->min = d;
    top++;
}

int MinStackPop(MinStack stack) {
    if (stack.top == 0) error("stack is empty!");
    return stack.data[--stack.top].data;
}

int MinStackMin(MinStack stack) {
    if (stack.top == 0) error("stack is empty!");
    return stack.data[stack.top-1].min;
}
```

3.求子数组的最大和

题目：

输入一个整形数组，数组里有正数也有负数。

数组中连续的一个或多个整数组成一个子数组，每个子数组都有一个和。

求所有子数组的和的最大值。要求时间复杂度为 $O(n)$ 。

例如输入的数组为 1, -2, 3, 10, -4, 7, 2, -5， 和最大的子数组为 3, 10, -4, 7, 2,

因此输出为该子数组的和 18。

ANSWER:

A traditional greedy approach.

Keep current sum, slide from left to right, when sum < 0, reset sum to 0.

```
int maxSubarray(int a[], int size) {
    if (size<=0) error("error array size");
    int sum = 0;
    int max = - (1 << 31);
    int cur = 0;
    while (cur < size) {
        sum += a[cur++];
        if (sum > max) {
            max = sum;
        } else if (sum < 0) {
            sum = 0;
        }
    }
    return max;
}
```

4.在二元树中找出和为某一值的所有路径

题目：输入一个整数和一棵二元树。

从树的根结点开始往下访问一直到叶结点所经过的所有结点形成一条路径。

打印出和与输入整数相等的所有路径。

例如输入整数 22 和如下二元树

```
10
/
5 12
/
4 7
```

则打印出两条路径：10, 12 和 10, 5, 7。

二元树节点的数据结构定义为：

```
struct BinaryTreeNode // a node in the binary tree
{
    int m_nValue; // value of node
    BinaryTreeNode *m_pLeft; // left child of node
    BinaryTreeNode *m_pRight; // right child of node
};
```

ANSWER:

Use backtracking and recursion. We need a stack to help backtracking the path.

```
struct TreeNode {
    int data;
    TreeNode * left;
    TreeNode * right;
};

void printPaths(TreeNode * root, int sum) {
    int path[MAX_HEIGHT];
    helper(root, sum, path, 0);
}

void helper(TreeNode * root, int sum, int path[], int top) {
    path[top++] = root.data;
    sum -= root.data;
    if (root->left == NULL && root->right == NULL) {
        if (sum == 0) printPath(path, top);
    } else {
        if (root->left != NULL) helper(root->left, sum, path, top);
        if (root->right != NULL) helper(root->right, sum, path, top);
    }
    top--;
    sum += root.data;    //....
}
```

5. 查找最小的 k 个元素

题目：输入 n 个整数，输出其中最小的 k 个。

例如输入 1, 2, 3, 4, 5, 6, 7 和 8 这 8 个数字，则最小的 4 个数字为 1, 2, 3 和 4。

ANSWER:

This is a very traditional question...

O(nlogn): cat l_FILE | sort -n | head -n K

O(kn): do insertion sort until k elements are retrieved.

$O(n+k\log n)$: Take $O(n)$ time to bottom-up build a min-heap. Then sift-down $k-1$ times.

So traditional that I don't want to write the codes...

Only gives the siftup and siftdown function.

```
/**  
 * @param i the index of the element in heap a[0...n-1] to be sifted up  
 */  
void siftup(int a[], int i, int n) {  
    while (i>0) {  
        int j=(i&1==0 ? i-1 : i+1);  
        int p=(i-1)>>1;  
        if (j<n && a[j]<a[i]) i = j;  
        if (a[i] < a[p]) swap(a, i, p);  
        i = p;  
    }  
}  
void siftdown(int a[], int i, int n) {  
    while (2*i+1<n){  
        int l=2*i+1;  
        if (l+1<n && a[l+1] < a[l]) l++;  
        if (a[l] < a[i]) swap(a, i, l);  
        i=l;  
    }  
}
```

第 6 题

腾讯面试题：

给你 10 分钟时间，根据上排给出十个数，在其下排填出对应的十个数

要求下排每个数都是先前上排那十个数在下排出现的次数。

上排的十个数如下：

【0, 1, 2, 3, 4, 5, 6, 7, 8, 9】

举一个例子，

数值: 0,1,2,3,4,5,6,7,8,9

分配: 6,2,1,0,0,0,1,0,0,0

0 在下排出现了 6 次，1 在下排出现了 2 次，

2 在下排出现了 1 次，3 在下排出现了 0 次....

以此类推..

ANSWER:

I don't like brain teasers. Will skip most of them...

第 7 题

微软亚院之编程判断俩个链表是否相交

给出俩个单向链表的头指针，比如 `h1, h2`，判断这俩个链表是否相交。

为了简化问题，我们假设俩个链表均不带环。

问题扩展：

- 1.如果链表可能有环列？
- 2.如果需要求出俩个链表相交的第一个节点列？

ANSWER:

```
struct Node {  
    int data;  
    int Node *next;  
};  
  
// if there is no cycle.  
int isJoinedSimple(Node * h1, Node * h2) {  
    while (h1->next != NULL) {  
        h1 = h1->next;  
    }  
    while (h2->next != NULL) {  
        h2 = h2->next;  
    }  
    return h1 == h2;  
}  
  
// if there could exist cycle  
int isJoined(Node *h1, Node * h2) {  
    Node* cylic1 = testCylic(h1);  
    Node* cylic2 = testCylic(h2);  
    if (cylic1+cylic2==0) return isJoinedSimple(h1, h2);  
    if (cylic1==0 && cylic2!=0 || cylic1!=0 && cylic2==0) return 0;  
    Node *p = cylic1;  
    while (1) {  
        if (p==cylic2 || p->next == cylic2) return 1;  
        p=p->next->next;  
        cylic1 = cylic1->next;  
        if (p==cylic1) return 0;  
    }  
}  
  
Node* testCylic(Node * h1) {  
    Node * p1 = h1, *p2 = h1;  
    while (p2!=NULL && p2->next!=NULL) {
```

```

    p1 = p1->next;
    p2 = p2->next->next;
    if (p1 == p2) {
        return p1;
    }
}
return NULL;
}

```

第 8 题

此贴选一些比较怪的题，，由于其中题目本身与算法关系不大，仅考考思维。特此并作一题。

1.有两个房间，一间房里有三盏灯，另一间房有控制着三盏灯的三个开关，

这两个房间是分割开的，从一间里不能看到另一间的情况。

现在要求受训者分别进这两房间一次，然后判断出这三盏灯分别是由哪个开关控制的。

有什么办法呢？

ANSWER:

Skip.

2.你让一些人为你工作了七天，你要用一根金条作为报酬。金条被分成七小块，每天给出一块。

如果你只能将金条切割两次，你怎样分给这些工人？

ANSWER:

1+2+4;

3. ★用一种算法来颠倒一个链接表的顺序。现在在不用递归式的情况下做一遍。

ANSWER:

```

Node * reverse(Node * head) {
    if (head == NULL) return head;
    if (head->next == NULL) return head;
    Node * ph = reverse(head->next);
    head->next->next = head;
    head->next = NULL;
    return ph;
}
Node * reverseNonrecursve(Node * head) {
    if (head == NULL) return head;
    Node * p = head;
    Node * previous = NULL;
    while (p->next != NULL) {
        p->next = previous;
        previous = p;
        p = p->next;
    }
    p->next = previous;
    return p;
}

```

```

    previous = p;
    p = p->next;
}
p->next = previous;
return p;
}

```

★用一种算法在一个循环的链接表里插入一个节点，但不得穿越链接表。

ANSWER:

I don't understand what is "Chuanyue".

★用一种算法整理一个数组。你为什么选择这种方法？

ANSWER:

What is "Zhengli?"

★用一种算法使通用字符串相匹配。

ANSWER:

What is "Tongyongzifuchuan"... a string with "*" and "?"? If so, here is the code.

```

int match(char * str, char * ptn) {
    if (*ptn == '\0') return 1;
    if (*ptn == '*') {
        do {
            if (match(str++, ptn+1)) return 1;
        } while (*str != '\0');
        return 0;
    }
    if (*str == '\0') return 0;
    if (*str == *ptn || *ptn == '?') {
        return match(str+1, ptn+1);
    }
    return 0;
}

```

★颠倒一个字符串。优化速度。优化空间。

```

void reverse(char *str) {
    reverseFixlen(str, strlen(str));
}

void reverseFixlen(char *str, int n) {
    char* p = str+n-1;
    while (str < p) {
        char c = *str;
        *str = *p; *p=c;
    }
}

```

★颠倒一个句子中的词的顺序，比如将“我叫克丽丝”转换为“克丽丝叫我”，实现速度最快，移动最少。

ANSWER:

Reverse the whole string, then reverse each word. Using the reverseFixlen() above.

```
void reverseWordsInSentence(char * sen) {
    int len = strlen(sen);
    reverseFixlen(sen, len);
    char * p = str;
    while (*p != '\0') {
        while (*p == ' ' && *p != '\0') p++;
        str = p;
        while (p != ' ' && *p != '\0') p++;
        reverseFixlen(str, p-str);
    }
}
```

★找到一个子字符串。优化速度。优化空间。

ANSWER:

KMP? BM? Sunday? Using BM or sunday, if it's ASCII string, then it's easy to fast access the auxiliary array. Otherwise an hashmap or bst may be needed. Lets assume it's an ASCII string.

```
int bm strstr(char *str, char *sub) {
    int len = strlen(sub);
    int i;
    int aux[256];
    memset(aux, sizeof(int), 256, len+1);
    for (i=0; i<len; i++) {
        aux[sub[i]] = len - i;
    }
    int n = strlen(str);
    i=len-1;
    while (i<n) {
        int j=i, k=len-1;
        while (k>=0 && str[j--] == sub[k--]);
        if (k<0) return j+1;
        if (i+1<n)
            i+=aux[str[i+1]];
        else
            return -1;
    }
}
```

However, this algorithm, as well as BM, KMP algorithms use $O(|sub|)$ space. If this is not acceptable, Rabin-carp algorithm can do it. Using hashing to fast filter out most false

matchings.

```
#define HBASE 127
int rc strstr(char * str, char * sub) {
    int dest= 0;
    char * p = sub;
    int len = 0;
    int TO_REDUCE = 1;
    while (*p != '\0') {
        dest = HBASE * dest + (int)(*p);
        TO_REDUCE *= HBASE;
        len++;
    }
    int hash = 0;
    p = str;
    int i=0;
    while (*p != '\0') {
        if (i++<len) hash = HBASE * dest + (int)(*p);
        else hash = (hash - (TO_REDUCE * (int)(*(p-len))))*HBASE + (int)(*p);
        if (hash == dest && i>=len && strncmp(sub, p-len+1, len) == 0) return
i-len;
        p++;
    }
    return -1;
}
```

★比较两个字符串，用 $O(n)$ 时间和恒量空间。

ANSWER:

What is “comparing two strings”? Just normal string comparison? The natural way use $O(n)$ time and $O(1)$ space.

```
int strcmp(char * p1, char * p2) {
    while (*p1 != '\0' && *p2 != '\0' && *p1 == *p2) {
        p1++, p2++;
    }
    if (*p1 == '\0' && *p2 == '\0') return 0;
    if (*p1 == '\0') return -1;
    if (*p2 == '\0') return 1;
    return (*p1 - *p2); // it can be negotiated whether the above 3 if's are
necessary, I don't like to omit them.
}
```

★假设你有一个用 1001 个整数组成的数组，这些整数是任意排列的，但是你知道所有的整数都在 1 到 1000(包括 1000)之间。此外，除一个数字出现两次外，其他所有数字只出现一次。假设你只能对这个数组做一次处理，用一种算法找出重复的那个数字。如果你在运算中使用了辅助的存储方式，那么你能找到不用这种方式的算法吗？

ANSWER:

Sum up all the numbers, then subtract the sum from $1001*1002/2$.

Another way, use $A \text{ XOR } A \text{ XOR } B = B$:

```
int findX(int a[]) {  
    int k = a[0];  
    for (int i=1; i<=1000;i++)  
        k ^= a[i]^i;  
    }  
    return k;  
}
```

★不用乘法或加法增加 8 倍。现在用同样的方法增加 7 倍。

ANSWER:

```
n<<3;  
(n<<3)-n;
```

第 9 题

判断整数序列是不是二元查找树的后序遍历结果

题目：输入一个整数数组，判断该数组是不是某二元查找树的后序遍历的结果。

如果是返回 `true`，否则返回 `false`。

例如输入 5、7、6、9、11、10、8，由于这一整数序列是如下树的后序遍历结果：

```
      8  
     /   \  
    6     10  
   / \   / \  
  5  7  9  11
```

因此返回 `true`。

如果输入 7、4、6、5，没有哪棵树的后序遍历的结果是这个序列，因此返回 `false`。

ANSWER:

This is an interesting one. There is a traditional question that requires the binary tree to be re-constructed from mid/post/pre order results. This seems similar. For the problems related to (binary) trees, recursion is the first choice.

In this problem, we know in post-order results, the last number should be the root. So we have known the root of the BST is 8 in the example. So we can split the array by the root.

```
int isPostorderResult(int a[], int n) {  
    return helper(a, 0, n-1);  
}  
int helper(int a[], int s, int e) {
```

```

if (e==s) return 1;
int i=e-1;
while (a[e]>a[i] && i>=s) i--;
if (!helper(a, i+1, e-1))
    return 0;
int k = 1;
while (a[e]<a[i] && i>=s) i--;
return helper(a, s, l);
}

```

第 10 题

翻转句子中单词的顺序。

题目：输入一个英文句子，翻转句子中单词的顺序，但单词内字符的顺序不变。

句子中单词以空格符隔开。为简单起见，标点符号和普通字母一样处理。

例如输入 “I am a student.”，则输出 “student. a am I”。

Answer:

Already done this. Skipped.

第 11 题

求二叉树中节点的最大距离...

如果我们把二叉树看成一个图，父子节点之间的连线看成是双向的，

我们姑且定义"距离"为两节点之间边的个数。

写一个程序，

求一棵二叉树中相距最远的两个节点之间的距离。

ANSWER:

This is interesting... Also recursively, the longest distance between two nodes must be either from root to one leaf, or between two leafs. For the former case, it's the tree height. For the latter case, it should be the sum of the heights of left and right subtrees of the two leaves' most least ancestor.

The first case is also the sum the heights of subtrees, just the height + 0.

```

int maxDistance(Node * root) {
    int depth;
    return helper(root, depth);
}
int helper(Node * root, int &depth) {
    if (root == NULL) {
        depth = 0; return 0;
    }
}

```

```

    int ld, rd;
    int maxleft = helper(root->left, ld);
    int maxright = helper(root->right, rd);
    depth = max(ld, rd)+1;
    return max(maxleft, max(maxright, ld+rd));
}

```

第 12 题

题目：求 $1+2+\dots+n$,

要求不能使用乘除法、`for`、`while`、`if`、`else`、`switch`、`case` 等关键字以及条件判断语句

(A?B:C)。

ANSWER:

$$1+\dots+n=n*(n+1)/2=(n^2+n)/2$$

it is easy to get $n/2$, so the problem is to get n^2

though no if/else is allowed, we can easily go around using short-pass.

using macro to make it fancier:

```

#define T(X, Y, i) (Y & (1<<i)) && X+=(Y<<i)
int foo(int n){
    int r=n;
    T(r, n, 0); T(r, n, 1); T(r, n, 2); ... T(r, n, 31);
    return r >> 1;
}

```

第 13 题:

题目：输入一个单向链表，输出该链表中倒数第 k 个结点。链表的倒数第 0 个结点为链表的尾指针。

链表结点定义如下：

```

struct ListNode
{
    int m_nKey;
    ListNode* m_pNext;
};

```

Answer:

Two ways. 1: record the length of the linked list, then go $n-k$ steps. 2: use two cursors.

Time complexities are exactly the same.

```

Node * lastK(Node * head, int k) {
    if (k<0) error("k < 0");
    Node *p=head, *pk=head;
    for (;k>0;k--) {

```

```

    if (pk->next!=NULL) pk = pk->next;
    else return NULL;
}
while (pk->next!=NULL) {
    p=p->next, pk=pk->next;
}
return p;
}

```

第 14 题:

题目：输入一个已经按升序排序过的数组和一个数字，

在数组中查找两个数，使得它们的和正好是输入的那个数字。

要求时间复杂度是 $O(n)$ 。如果有多对数字的和等于输入的数字，输出任意一对即可。

例如输入数组 1、2、4、7、11、15 和数字 15。由于 $4+11=15$ ，因此输出 4 和 11。

ANSWER:

Use two cursors. One at front and the other at the end. Keep track of the sum by moving the cursors.

```

void find2Number(int a[], int n, int dest) {
    int *f = a, *e=a+n-1;
    int sum = *f + *e;
    while (sum != dest && f < e) {
        if (sum < dest) sum = *(++f);
        else sum = *(--e);
    }
    if (sum == dest) printf("%d, %d\n", *f, *e);
}

```

第 15 题:

题目：输入一颗二元查找树，将该树转换为它的镜像，

即在转换后的二元查找树中，左子树的结点都大于右子树的结点。

用递归和循环两种方法完成树的镜像转换。

例如输入：

```

8
/
6   10
/\   / \
5   7 9   11

```

输出：

```

8

```

```

    /   \
  10     6
  / \   / \
11  9  7  5

```

定义二元查找树的结点为:

```

struct BSTreeNode // a node in the binary search tree (BST)
{
    int m_nValue; // value of node
    BSTreeNode *m_pLeft; // left child of node
    BSTreeNode *m_pRight; // right child of node
};

```

ANSWER:

This is the basic application of recursion.

PS: I don't like the m_xx naming convention.

```

void swap(Node ** l, Node ** r) {
    Node * p = *l;
    *l = *r;
    *r = p;
}

void mirror(Node * root) {
    if (root == NULL) return;
    swap(&(root->left), &(root->right));
    mirror(root->left);
    mirror(root->right);
}

void mirrorIteratively(Node * root) {
    if (root == NULL) return;
    stack<Node*> buf;
    buf.push(root);
    while (!stack.empty()) {
        Node * n = stack.pop();
        swap(&(root->left), &(root->right));
        if (root->left != NULL) buf.push(root->left);
        if (root->right != NULL) buf.push(root->right);
    }
}

```

第16题:

题目 (微软):

输入一颗二元树，从上往下按层打印树的每个结点，同一层中按照从左往右的顺序打印。

例如输入

```
8
/
 \
6   10
/\   / \
5 7   9 11
```

输出 8 6 10 5 7 9 11。

ANSWER:

The nodes in the levels are printed in the similar manner their parents were printed. So it should be an FIFO queue to hold the level. I really don't remember the function name of the stl queue, so I will write it in Java...

```
void printByLevel(Node root) {
    Node sentinel = new Node();
    LinkedList<Node> q = new LinkedList<Node>();
    q.addFirst(root); q.addFirst(sentinel);
    while (!q.isEmpty()) {
        Node n = q.removeLast();
        if (n==sentinel) {
            System.out.println("\n");
            q.addFirst(sentinel);
        } else {
            System.out.println(n);
            if (n.left() != null) q.addFirst(n.left());
            if (n.right() != null) q.addFirst(n.right());
        }
    }
}
```

第 17 题：

题目：在一个字符串中找到第一个只出现一次的字符。如输入 abaccdeff，则输出 b。

分析：这道题是 2006 年 google 的一道笔试题。

ANSWER:

Again, this depends on what is "char". Let's assume it as ASCII.

```
char firstSingle(char * str) {
    int a[255];
    memset(a, 0, 255*sizeof(int));
    char *p=str;
    while (*p!='\0') {
        a[*p]++;
    }
}
```

```

        p++;
    }
    p = str;
    while (*p != '\0') {
        if (a[*p] == 1) return *p;
    }
    return '\0'; // this must be the one that occurs exact 1 time.
}

```

第 18 题:

题目: n 个数字 ($0, 1, \dots, n-1$) 形成一个圆圈, 从数字 0 开始,

每次从这个圆圈中删除第 m 个数字 (第一个为当前数字本身, 第二个为当前数字的下一个数字)。

当一个数字删除后, 从被删除数字的下一个继续删除第 m 个数字。

求出在这个圆圈中剩下的最后一个数字。

July: 我想, 这个题目, 不少人已经见识过了。

ANSWER:

Actually, although this is a so traditional problem, I was always too lazy to think about this or even to search for the answer.(What a shame...). Finally, by google I found the elegant solution for it.

The keys are:

1) if we shift the ids by k , namely, start from k instead of 0, we should add the result by $k \% n$

2) after the first round, we start from $k+1$ (possibly $\% n$) with $n-1$ elements, that is equal to an $(n-1)$ problem while start from $(k+1)$ th element instead of 0, so the answer is $(f(n-1, m)+k+1)\%n$

3) $k = m-1$, so $f(n, m) = (f(n-1, m)+m)\%n$.

finally, $f(1, m) = 0$;

Now this is a $O(n)$ solution.

```

int joseph(int n, int m) {
    int fn=0;
    for (int i=2; i<=n; i++) {
        fn = (fn+m)%i;
    }
    return fn;
}

```

hu...长出一口气。。。

第 19 题:

题目：定义 Fibonacci 数列如下：

/ 0 n=0

f(n)= 1 n=1

\ f(n-1)+f(n-2) n=2

输入 n，用最快的方法求该数列的第 n 项。

分析：在很多 C 语言教科书中讲到递归函数的时候，都会用 Fibonacci 作为例子。

因此很多程序员对这道题的递归解法非常熟悉，但....呵呵，你知道的。。

ANSWER:

This is the traditional problem of application of mathematics...

let A=

{1 1}

{1 0}

f(n) = A^(n-1)[0,0]

this gives a O(log n) solution.

```
int f(int n) {
    int A[4] = {1,1,1,0};
    int result[4];
    power(A, n, result);
    return result[0];
}

void multiply(int[] A, int[] B, int _r) {
    _r[0] = A[0]*B[0] + A[1]*B[2];
    _r[1] = A[0]*B[1] + A[1]*B[3];
    _r[2] = A[2]*B[0] + A[3]*B[2];
    _r[3] = A[2]*B[1] + A[3]*B[3];
}

void power(int[] A, int n, int _r) {
    if (n==1) { memcpy(A, _r, 4*sizeof(int)); return; }
    int tmp[4];
    power(A, n>>1, _r);
    multiply(_r, _r, tmp);
    if (n & 1 == 1) {
        multiply(tmp, A, _r);
    } else {
        memcpy(_r, tmp, 4*sizeof(int));
    }
}
```

第 20 题:

题目：输入一个表示整数的字符串，把该字符串转换成整数并输出。

例如输入字符串"345"，则输出整数 345。

ANSWER:

This question checks how the interviewee is familiar with C/C++? I'm so bad at C/C++...

```
int atoi(char * str) {
    int neg = 0;
    char * p = str;
    if (*p == '-') {
        p++; neg = 1;
    } else if (*p == '+') {
        p++;
    }
    int num = 0;
    while (*p != '\0') {
        if (*p>=0 && *p <= 9) {
            num = num * 10 + (*p-'0');
        } else {
            error("illegal number");
        }
        p++;
    }
    return num;
}
```

PS: I didn't figure out how to tell a overflow problem easily.

第 21 题

2010 年中兴面试题

编程求解：

输入两个整数 n 和 m，从数列 1, 2, 3.....n 中随意取几个数，
使其和等于 m，要求将其中所有的可能组合列出来。

ANSWER

This is a combination generation problem.

```
void findCombination(int n, int m) {
    if (n>m) findCombination(m, m);
    int aux[n];
    memset(aux, 0, n*sizeof(int));
    helper(m, 0, aux);
}
void helper(int dest, int idx, int aux[], int n) {
```

```

if (dest == 0)
    dump(aux, n);
if (dest <= 0 || idx==n) return;
helper(dest, idx+1, aux, n);
aux[idx] = 1;
helper(dest-idx-1, idx+1, aux, n);
aux[idx] = 0;
}
void dump(int aux[], int n) {
    for (int i=0; i<n; i++)
        if (aux[i]) printf("%3d", i+1);
    printf("\n");
}

```

PS: this is not an elegant implementation, however, it is not necessary to use gray code or other techniques for such a problem, right?

第 22 题:

有 4 张红色的牌和 4 张蓝色的牌，主持人先拿任意两张，再分别在 A、B、C 三人额头上贴任意两张牌，A、B、C 三人都可以看见其余两人额头上的牌，看完后让他们猜自己额头上是什么颜色的牌，A 说不知道，B 说不知道，C 说不知道，然后 A 说知道了。

请教如何推理，A 是怎么知道的。如果用程序，又怎么实现呢？

ANSWER

I don't like brain teaser. As an AI problem, it seems impossible to write the solution in 20 min...

It seems that a brute-force edge cutting strategy could do. Enumerate all possibilities, then for each guy delete the permutation that could be reduced if failed (for A, B, C at 1st round), Then there should be only one or one group of choices left.

But who uses this as an interview question?

第 23 题:

用最简单，最快速的方法计算出下面这个圆形是否和正方形相交。"

3D 坐标系原点(0.0,0.0,0.0)

圆形:

半径 r = 3.0

圆心 o = (*.*, 0.0, *.*)

正方形:

4 个角坐标;

```
1:(.*., 0.0, .*.)  
2:(.*., 0.0, .*.)  
3:(.*., 0.0, .*.)  
4:(.*., 0.0, .*.)
```

ANSWER

Crap... I totally cannot understand this problem... Does the `.*.` represent any possible number?

第 24 题:

链表操作,

- (1) .单链表就地逆置,
- (2) 合并链表

ANSWER

Reversing a linked list. Already done.

What do you mean by merge? Are the original lists sorted and need to be kept sorted? If not, are there any special requirements?

I will only do the sorted merging.

```
Node * merge(Node * h1, Node * h2) {  
    if (h1 == NULL) return h2;  
    if (h2 == NULL) return h1;  
    Node * head;  
    if (h1->data>h2->data) {  
        head = h2; h2=h2->next;  
    } else {  
        head = h1; h1=h1->next;  
    }  
    Node * current = head;  
    while (h1 != NULL && h2 != NULL) {  
        if (h1 == NULL || (h2!=NULL && h1->data>h2->data)) {  
            current->next = h2; h2=h2->next; current = current->next;  
        } else {  
            current->next = h1; h1=h1->next; current = current->next;  
        }  
    }  
    current->next = NULL;  
    return head;
```

```
}
```

第 25 题:

写一个函数,它的原形是 int continuumax(char *outputstr,char *intputstr)

功能:

在字符串中找出连续最长的数字串, 并把这个串的长度返回,

并把这个最长数字串付给其中一个函数参数 outputstr 所指内存。

例如: "abcd12345ed125ss123456789"的首地址传给 intputstr 后, 函数将返回 9,

outputstr 所指的值为 123456789

ANSWER:

```
int continuumax(char *outputstr, char *inputstr) {
    int len = 0;
    char * pstart = NULL;
    int max = 0;
    while (1) {
        if (*inputstr >= '0' && *inputstr <='9') {
            len++;
        } else {
            if (len > max) pstart = inputstr - len;
            len = 0;
        }
        if (*inputstr++ == '\0') break;
    }
    for (int i=0; i<len; i++)
        *outputstr++ = pstart++;
    *outputstr = '\0';
    return max;
}
```

26.左旋转字符串

题目:

定义字符串的左旋转操作: 把字符串前面的若干个字符移动到字符串的尾部。

如把字符串 abcdef 左旋转 2 位得到字符串 cdefab。请实现字符串左旋转的函数。

要求时间对长度为 n 的字符串操作的复杂度为 O(n), 辅助内存为 O(1)。

ANSWER

Have done it. Using reverse word function above.

27.跳台阶问题

题目：一个台阶总共有 n 级，如果一次可以跳 1 级，也可以跳 2 级。

求总共有多少总跳法，并分析算法的时间复杂度。

这道题最近经常出现，包括 MicroStrategy 等比较重视算法的公司都曾先后选用过这个这道题作为面试题或者笔试题。

ANSWER

$f(n) = f(n-1) + f(n-2)$, $f(1) = 1$, $f(2) = 2$, let $f(0) = 1$, then $f(n) = fibo(n-1)$;

28. 整数的二进制表示中 1 的个数

题目：输入一个整数，求该整数的二进制表达中有多少个 1。

例如输入 10，由于其二进制表示为 1010，有两个 1，因此输出 2。

分析：

这是一道很基本的考查位运算的面试题。

包括微软在内的很多公司都曾采用过这道题。

ANSWER

Traditional question. Use the equation $xxxxx10000 \& (xxxxx10000-1) = xxxxx00000$

Note: for negative numbers, this also hold, even with 100000000 where the “-1” leading to an underflow.

```
int countOf1(int n) {
    int c=0;
    while (n!=0) {
        n=n & (n-1);
        c++;
    }
    return c;
}
```

another solution is to lookup table. $O(k)$, k is $\text{sizeof}(\text{int})$;

```
int countOf1(int n) {
    int c = 0;
    if (n<0) { c++; n = n & (1<<((sizeof(int)*8)-1)); }
    while (n!=0) {
        c+=tab[n&0xff];
        n >>= 8;
    }
    return c;
}
```

29. 栈的 push、pop 序列

题目：输入两个整数序列。其中一个序列表示栈的 push 顺序，

判断另一个序列有没有可能是对应的 pop 顺序。

为了简单起见，我们假设 push 序列的任意两个整数都是不相等的。

比如输入的 push 序列是 1、2、3、4、5，那么 4、5、3、2、1 就有可能是一个 pop 系列。

因为可以有如下的 push 和 pop 序列：

push 1, push 2, push 3, push 4, pop, push 5, pop, pop, pop, pop,

这样得到的 pop 序列就是 4、5、3、2、1。

但序列 4、3、5、1、2 就不可能是 push 序列 1、2、3、4、5 的 pop 序列。

ANSWER

This seems interesting. However, a quite straightforward and promising way is to actually build the stack and check whether the pop action can be achieved.

```
int isPopSeries(int push[], int pop[], int n) {
    stack<int> helper;
    int i1=0, i2=0;
    while (i2 < n) {
        while (stack.empty() || stack.peek() != pop[i2]) {
            if (i1<n)
                stack.push(push[i1++]);
            else
                return 0;
        }
        while (!stack.empty() && stack.peek() == pop[i2])
            stack.pop(); i2++;
    }
    return 1;
}
```

30.在从 1 到 n 的正数中 1 出现的次数

题目：输入一个整数 n，求从 1 到 n 这 n 个整数的十进制表示中 1 出现的次数。

例如输入 12，从 1 到 12 这些整数中包含 1 的数字有 1, 10, 11 和 12, 1 一共出现了 5 次。

分析：这是一道广为流传的 google 面试题。

ANSWER

This is complicated... I hate it...

Suppose we have N=ABCDEFG.

if G<1, # of 1's in the units digits is ABCDEF, else ABCDEF+1

if F<1, # of 1's in the digit of tens is (ABCDE)*10, else if F==1: (ABCDE)*10+G+1, else (ABCDE+1)*10

if E<1, # of 1's in 3rd digit is (ABCD)*100, else if E==1: (ABCD)*100+FG+1, else

(ABCD+1)*100

… so on.

if A=1, # of 1 in this digit is BCDEFG+1, else it's 1*1000000;

so to fast access the digits and helper numbers, we need to build the fast access table of prefixes and suffixes.

```
int countOf1s(int n) {
    int prefix[10], suffix[10], digits[10]; //10 is enough for 32bit integers
    int i=0;
    int base = 1;
    while (base < n) {
        suffix[i] = n % base;
        digit[i] = (n % (base * 10)) - suffix[i];
        prefix[i] = (n - suffix[i] - digit[i]*base)/10;
        i++, base*=10;
    }
    int count = 0;
    base = 1;
    for (int j=0; j<i; j++) {
        if (digit[j] < 1) count += prefix;
        else if (digit[j]==1) count += prefix + suffix + 1;
        else count += prefix + base;
        base *= 10;
    }
    return count;
}
```

31. 华为面试题：

一类似于蜂窝的结构的图，进行搜索最短路径（要求 5 分钟）

ANSWER

Not clear problem. Skipped. Seems a Dijkstra could do.

```
int dij
```

32.

有两个序列 a,b，大小都为 n, 序列元素的值任意整数，无序；

要求：通过交换 a,b 中的元素，使[序列 a 元素的和]与[序列 b 元素的和]之间的差最小。

例如：

```
var a=[100,99,98,1,2, 3];
```

```
var b=[1, 2, 3, 4, 5, 40];
```

ANSWER

If only one swap can be taken, it is a $O(n^2)$ searching problem, which can be reduced to $O(n \log n)$ by sorting the arrays and doing binary search.

If any times of swaps can be performed, this is a double combinatorial problem.

In the book <<beauty of codes>>, a similar problem splits an array to halves as even as possible. It is possible to take binary search, when SUM of the array is not too high. Else this is a quite time consuming brute force problem. I cannot figure out a reasonable solution.

33.

实现一个挺高级的字符匹配算法：

给一串很长字符串，要求找到符合要求的字符串，例如目的串：123

1*****3***2 ,12*****3 这些都要找出来

其实就是类似一些和谐系统。。。

ANSWER

Not a clear problem. Seems a bitset can do.

34.

实现一个队列。

队列的应用场景为：

一个生产者线程将 int 类型的数入列，一个消费者线程将 int 类型的数出列

ANSWER

I don't know multithread programming at all....

35.

求一个矩阵中最大的二维矩阵(元素和最大).如:

1 2 0 3 4

2 3 4 5 1

1 1 5 3 0

中最大的是:

4 5

5 3

要求:(1)写出算法;(2)分析时间复杂度;(3)用 C 写出关键代码

ANSWER

This is the traditional problem in Programming Pearls. However, the best result is too complicated to achieve. So let's do the suboptimal one. $O(n^3)$ solution.

- 1) We have known that the similar problem for 1 dim array can be done in $O(n)$ time. However, this cannot be done in both directions in the same time. We can only calculate the accumulations for all the sublist from i to j , ($0 \leq i \leq j \leq n$) for each array in one dimension, which takes $O(n^2)$ time. Then in the other dimension, do the traditional greedy search.
- 3) To achieve $O(n^2)$ for accumulation for each column, accumulate 0 to i ($i=0, n-1$) first, then calculate the result by $acc(i, j) = acc(0, j) - acc(0, i-1)$

```
//acc[i*n+j] => acc(i,j)
void accumulate(int a[], int n, int acc[]) {
    int i=0;
    acc[i] = a[i];
    for (i=1; i<n; i++) {
        acc[i] = acc[i-1]+a[i];
    }
    for (i=1; i<n; i++) {
        for (j=i; j<n; j++) {
            acc[i*n+j] = acc[j] - acc[i-1];
        }
    }
}
```

第 36 题-40 题 (有些题目搜集于 CSDN 上的网友, 已标明):

36. 引用自网友: longzuo

谷歌笔试:

n 支队伍比赛, 分别编号为 0, 1, 2... $n-1$, 已知它们之间的实力对比关系,

存储在一个二维数组 $w[n][n]$ 中, $w[i][j]$ 的值代表编号为 i, j 的队伍中更强的一支。

所以 $w[i][j]=i$ 或者 j , 现在给出它们的出场顺序, 并存储在数组 $order[n]$ 中,

比如 $order[n]=\{4,3,5,8,1,\dots\}$, 那么第一轮比赛就是 4 对 3, 5 对 8。.....

胜者晋级, 败者淘汰, 同一轮淘汰的所有队伍排名不再细分, 即可以随便排,

下一轮由上一轮的胜者按照顺序, 再依次两两比, 比如可能是 4 对 5, 直至出现第一名

编程实现, 给出二维数组 w , 一维数组 $order$ 和用于输出比赛名次的数组 $result[n]$,

求出 $result$ 。

ANSWER

This question is like no-copying merge, or in place matrix rotation.

* No-copying merge: merge $order$ to $result$, then merge the first half from $order$, and so on.

* in place matrix rotation: rotate $01, 23, \dots, 2k/2k+1$ to $02\dots2k, 1, 3, \dots, 2k+1\dots$

The two approaches are both complicated. However, notice one special feature that the losers' order doesn't matter. Thus a half-way merge is much simpler and easier:

```
void knockOut(int **w, int order[], int result[], int n) {
    int round = n;
    memcpy(result, order, n*sizeof(int));
    while (round>1) {
        int i,j;
        for (i=0,j=0; i<round; i+=2) {
            int win= (i==round-1) ? i : w[i][i+1];
            swap(result, j, win);
            j++;
        }
    }
}
```

37.

有 n 个长为 $m+1$ 的字符串，

如果某个字符串的最后 m 个字符与某个字符串的前 m 个字符匹配，则两个字符串可以连接，

问这 n 个字符串最多可以连成一个多长的字符串，如果出现循环，则返回错误。

ANSWER

This is identical to the problem to find the longest acyclic path in a directed graph. If there is a cycle, return false.

Firstly, build the graph. Then search the graph for the longest path.

```
#define MAX_NUM 201
int inDegree[MAX_NUM];
int longestConcat(char ** strs, int m, int n) {
    int graph[MAX_NUM][MAX_NUM];
    int prefixHash[MAX_NUM];
    int suffixHash[MAX_NUM];
    int i,j;
    for (i=0; i<n; i++) {
        calcHash(strs[i], prefixHash[i], suffixHash[i]);
        graph[i][0] = 0;
    }
    memset(inDegree, 0, sizeof(int)*n);
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            if (suffixHash[i]==prefixHash[j] && strncmp(strs[i]+1, strs[j],
```

```

m) == 0) {
    if (i==j) return 0; // there is a self loop, return false.
    graph[i][0]++;
    graph[i][graph[i*n]] = j;
    inDegree[j]++;
}
}

}

return longestPath(graph, n);
}

/***
 * 1. do topological sort, record index[i] in topological order.
 * 2. for all 0-in-degree vertexes, set all path length to -1, do relaxation
in topological order to find single source shortest path.
*/
int visit[MAX_NUM];
int parent[MAX_NUM];
// -1 path weight, so 0 is enough.
#define MAX_PATH 0
int d[MAX_NUM];

int longestPath(int graph[], int n) {
    memset(visit, 0, n*sizeof(int));
    if (topSort(graph) == 0) return -1; //topological sort failed, there is
cycle.

    int min = 0;

    for (int i=0; i<n; i++) {
        if (inDegree[i] != 0) continue;
        memset(parent, -1, n*sizeof(int));
        memset(d, MAX_PATH, n*sizeof(int));
        d[i] = 0;
        for (int j=0; j<n; j++) {
            for (int k=1; k<=graph[top[j]][0]; k++) {
                if (d[top[j]] - 1 < d[graph[top[j]][k]]) { // relax with path
weight -1
                    d[graph[top[j]][k]] = d[top[j]] - 1;
                    parent[graph[top[j]][k]] = top[j];
                    if (d[graph[top[j]][k]] < min) min = d[graph[top[j]][k]];
                }
            }
        }
    }
}

```

```

        }
    }

    return -min;
}

int top[MAX_NUM];
int finished[MAX_NUM];
int cnt = 0;
int topSort(int graph[]){
    memset(visit, 0, n*sizeof(int));
    memset(finished, 0, n*sizeof(int));
    for (int i=0; i<n; i++) {
        if (topdfs(graph, i) == 0) return 0;
    }
    return 1;
}
int topdfs(int graph[], int s) {
    if (visited[s] != 0) return 1;
    for (int i=1; i<=graph[s][0]; i++) {
        if (visited[graph[s][i]]!=0 && finished[graph[s][i]]==0) {
            return 0; //gray node, a back edge;
        }
        if (visited[graph[s][i]] == 0) {
            visited[graph[s][i]] = 1;
            dfs(graph, graph[s][i]);
        }
    }
    finished[s] = 1;
    top[cnt++] = s;
    return 1;
}

```

Time complexity analysis:

Hash calculation: $O(nm)$

Graph construction: $O(n^2)$

Topological sort: as dfs, $O(V+E)$

All source longest path: $O(kE)$, k is 0-in-degree vertices number, E is edge number.

As a total, it's a $O(n^2+n^2m)$ solution.

A very good problem. But I really doubt it as a solve-in-20-min interview question.

百度面试：

1.用天平（只能比较，不能称重）从一堆小球中找出其中唯一一个较轻的，使用 x 次天平，最多可以从 y 个小球中找出较轻的那个，求 y 与 x 的关系式。

ANSWER:

$x=1, y=3$: if $a=b$, c is the lighter, else the lighter is the lighter...
do this recursively. so $y=3^x$;

2.有一个很大很大的输入流，大到没有存储器可以将其存储下来，而且只输入一次，如何从这个输入流中随机取得 m 个记录。

ANSWER

That is, keep total number count N . If $N \leq m$, just keep it.

For $N > m$, generate a random number $R = \text{rand}(N)$ in $[0, N]$, replace $a[R]$ with new number if R falls in $[0, m]$.

3.大量的 URL 字符串，如何从中去除重复的，优化时间空间复杂度

ANSWER

1. Use hash map if there is enough memory.
2. If there is no enough memory, use hash to put urls to bins, and do it until we can fit the bin into memory.

39.

网易有道笔试：

(1).

求一个二叉树中任意两个节点间的最大距离，
两个节点的距离的定义是这两个节点间边的个数，
比如某个孩子节点和父节点间的距离是 1，和相邻兄弟节点间的距离是 2，优化时间空间复杂度。

ANSWER

Have done this.

(2).

求一个有向连通图的割点，割点的定义是，如果除去此节点和与其相关的边，有向图不再连通，描述算法。

ANSWER

Do dfs, record $\text{low}[i]$ as the lowest vertex that can be reached from i and i 's successor

nodes. For each edge i , if $\text{low}[i] = i$ and i is not a leaf in dfs tree, then i is a cut point. The other case is the root of dfs, if root has two or more children ,it is a cut point.

```
/*
* g is defined as: g[i][] is the out edges, g[i][0] is the edge count,
g[i][1...g[i][0]] are the other end points.
*/
int cnt = 0;
int visited[MAX_NUM];
int lowest[MAX_NUM];
void getCutPoints(int *g[], int cuts[], int n) {
    memset(cuts, 0, sizeof(int)*n);
    memset(visited, 0, sizeof(int)*n);
    memset(lowest, 0, sizeof(int)*n);
    for (int i=0; i<n; i++) {
        if (visited[i] == 0) {
            visited[i] = ++cnt;
            dfs(g, cuts, n, i, i);
        }
    }
}

int dfs(int *g[], int cuts[], int n, int s, int root) {
    int out = 0;
    int low = visit[s];
    for (int i=1; i<=g[s][0]; i++) {
        if (visited[g[s][i]] == 0) {
            out++;
            visited[g[s][i]] = ++cnt;
            int clow = dfs(g, cuts, n, g[s][i], root);
            if (clow < low) low = clow;
        } else {
            if (low > visit[g[s][i]]) {
                low = visit[g[s][i]];
            }
        }
    }
    lowest[s] = low;
    if (s == root && out > 1) {
        cuts[s] = 1;
    }
    return low;
}
```

40.百度研发笔试题

引用自: zp155334877

1)设计一个栈结构，满足一下条件: min, push, pop 操作的时间复杂度为 O(1)。

ANSWER

Have done this.

2)一串首尾相连的珠子(m 个)，有 N 种颜色(N<=10)，

设计一个算法，取出其中一段，要求包含所有 N 中颜色，并使长度最短。

并分析时间复杂度与空间复杂度。

ANSWER

Use a sliding window and a counting array, plus a counter which monitors the num of zero slots in counting array. When there is still zero slot(s), advance the window head, until there is no zero slot. Then shrink the window until a slot comes zero. Then one candidate segment of (window_size + 1) is achieved. Repeat this. It is O(n) algorithm since each item is swallowed and left behind only once, and either operation is in constant time.

```
int shortestFullcolor(int a[], int n, int m) {
    int c[m], ctr = m;
    int h=0, t=0;
    int min=n;
    while (1) {
        while (ctr > 0 && h<n) {
            if (c[a[h]] == 0) ctr--;
            c[a[h]]++;
            h++;
        }
        if (h>=n) return min;
        while (1) {
            c[a[t]]--;
            if (c[a[t]] == 0) break;
            t++;
        }
        if (min > h-t) min = h-t;
        t++; ctr++;
    }
}
```

3)设计一个系统处理词语搭配问题，比如说中国和人民可以搭配，

则中国人民人民中国都有效。要求：

*系统每秒的查询数量可能上千次;

*词语的数量级为 10W;

*每个词至多可以与 1W 个词搭配

当用户输入中国人民的时候，要求返回与这个搭配词组相关的信息。

ANSWER

This problem can be solved in three steps:

1. identify the words
2. recognize the phrase
3. retrieve the information

Solution of 1: The most trivial way to efficiently identify the words is hash table or BST. A balanced BST with 100 words is about 17 levels high. Considering that 100k is not a big number, hashing is enough.

Solution of 2: Since the phrase in this problem consists of only 2 words, it is easy to split the words. There won't be a lot of candidates. To find a legal combination, we need the "matching" information. So for each word, we need some data structure to tell whether a word can co-occur with it. 100k is a bad number -- cannot fit into a 16bit digit. However, 10k*100k is not too big, so we can simply use array of sorted array to do this. 1G integers, or 4G bytes is not a big number, We can also use something like VInt to save a lot of space. To find an index in a 10k sorted array, 14 comparisons are enough.

Above operation can be done in any reasonable work-station's memory very fast, which should be the result of execution of about a few thousands of simple statements.

Solution of 3: The information could be too big to fit in the memory. So a B-tree may be adopted to index the contents. Caching techniques is also helpful. Considering there are at most 10^9 entries, a 3 or 4 level of B-tree is okay, so it will be at most 5 disk access. However, there are thousands of requests and we can only do hundreds of disk seeking per second. It could be necessary to dispatch the information to several workstations.

41.求固晶机的晶元查找程序

晶元盘由数目不详的大小一样的晶元组成，晶元并不一定全布满晶元盘，

照相机每次这能匹配一个晶元，如匹配过，则拾取该晶元，

若匹配不过，照相机则按测好的晶元间距移到下一个位置。

求遍历晶元盘的算法求思路。

ANSWER

Don't understand.

42.请修改 append 函数，利用这个函数实现：

两个非降序链表的并集，1->2->3 和 2->3->5 并为 1->2->3->5

另外只能输出结果，不能修改两个链表的数据。

ANSWER

I don't quite understand what it means by "not modifying linked list's data". If some nodes will be given up, it is weird for this requirement.

```
Node * head(Node *h1, Node * h2) {  
    if (h1==NULL) return h2;  
    if (h2==NULL) return h1;  
    Node * head;  
    if (h1->data < h2->data) {  
        head =h1; h1=h1->next;  
    } else {  
        head = h2; h2=h2->next;  
    }  
    Node * p = head;  
    while (h1!=NULL || h2!=NULL) {  
        Node * candi;  
        if (h1!=NULL && h2 != NULL && h1->data < h2->data || h2==NULL) {  
            candi = h1; h1=h1->next;  
        } else {  
            candi = h2; h2=h2->next;  
        }  
    }  
    if (candi->data == p->data) delete(candi);  
    else {  
        p->next = candi; p=candi;  
    }  
    return head;  
}
```

43.递归和非递归俩种方法实现二叉树的前序遍历。

ANSWER

```
void preorderRecursive(TreeNode * node) {  
    if (node == NULL) return;  
    visit(node);  
    preorderRecursive(node->left);  
    preorderRecursive(node->right);  
}
```

For non-recursive traversals, a stack must be adopted to replace the implicit program stack in recursive programs.

```
void preorderNonrecursive(TreeNode * node) {
    stack<TreeNode *> s;
    s.push(node);
    while (!s.empty()) {
        TreeNode * n = s.pop();
        visit(n);
        if (n->right!=NULL) s.push(n->right);
        if (n->left!=NULL) s.push(n->left);
    }
}

void inorderNonrecursive(TreeNode * node) {
    stack<TreeNode *> s;
    TreeNode * current = node;
    while (!s.empty() || current != NULL) {
        if (current != NULL) {
            s.push(current);
            current = current->left;
        } else {
            current = s.pop();
            visit(current);
            current = current->right;
        }
    }
}
```

Postorder nonrecursive traversal is the hardest one. However, a simple observation helps that the node first traversed is the node last visited. This recalls the feature of stack. So we could use a stack to store all the nodes then pop them out altogether.

This is a very elegant solution, while takes $O(n)$ space.

Other very smart methods also work, but this is the one I like the most.

```
void postorderNonrecursive(TreeNode * node) {
    // visiting occurs only when current has no right child or last visited
    // is his right child
    stack<TreeNode *> sTraverse, sVisit;
    sTraverse.push(node);
    while (!sTraverse.empty()) {
        TreeNode * p = sTraverse.pop();
```

```

    sVisit.push(p);
    if (p->left != NULL) sTraverse.push(p->left);
    if (p->right != NULL) sTraverse.push(p->right);
}
while (!sVisit.empty()) {
    visit(sVisit.pop());
}
}

```

44. 腾讯面试题:

1. 设计一个魔方（六面）的程序。

ANSWER

This is a problem to test OOP.

The object MagicCube must have following features

- 1) holds current status
- 2) easily doing transform
- 3) judge whether the final status is achieved
- 4) to test, it can be initialized
- 5) output current status

```

public class MagicCube {
    // 6 faces, 9 chips each face
    private byte chips[54];
    static final int X = 0;
    static final int Y = 1;
    static final int Z = 1;
    void transform(int direction, int level) {
        switch direction: {
            X : { transformX(level); break; }
            Y : { transformY(level); break; }
            Z : { transformZ(level); break; }
            default: throw new RuntimeException("what direction?");
        }
        void transformX(int level) { ... }
    }
}
// really tired of making this...
}

```

2. 有一千万条短信，有重复，以文本文件的形式保存，一行一条，有重复。

请用 5 分钟时间，找出重复出现最多的前 10 条。

ANSWER

10M msgs, each at most 140 chars, that's 1.4G, which can fit to memory.

So use hash map to accumulate occurrence counts.

Then use a heap to pick maximum 10.

3. 收藏了 1 万条 url, 现在给你一条 url, 如何找出相似的 url。(面试官不解释何为相似)

ANSWER

What a SB interviewer... The company name should be claimed and if I met such a interviewer, I will contest to HR. The purpose of interview is to see the ability of communication. This is kind of single side shutdown of information exchange.

My first answer will be doing edit distance to the url and every candidate. Then it depends on what interviewer will react. Other options includes: fingerprints, tries...

45. 雅虎:

1. 对于一个整数矩阵，存在一种运算，对矩阵中任意元素加一时，需要其相邻（上下左右）某一个元素也加一，现给出一正数矩阵，判断其是否能够由一个全零矩阵经过上述运算得到。

ANSWER

A assignment problem. Two ways to solve. 1: duplicate each cell to as many as its value, do Hungarian algorithm. Denote the sum of the matrix as M, the edge number is 2M, so the complexity is 2^*M^*M ; 2: standard maximum flow. If the size of matrix is NxN, then the algorithm using Ford Fulkerson algorithm is M^*N^*N .

too complex... I will do this when I have time...

2. 一个整数数组，长度为 n，将其分为 m 份，使各份的和相等，求 m 的最大值

比如{3, 2, 4, 3, 6} 可以分成{3, 2, 4, 3, 6} m=1;

{3,6}{2,4,3} m=2

{3,3}{2,4}{6} m=3 所以 m 的最大值为 3

ANSWER

Two restrictions on m, 1) $1 \leq m \leq n$; 2) $\text{Sum(array)} \bmod m = 0$

NOTE: no hint that $a[i] > 0$, so m could be larger than sum/max;

So firstly prepare the candidates, then do a brute force search on possible m's.

In the search , a DP is available, since if $f(\text{array}, m) = \text{OR}_i(f(\text{array-subset}(i), m))$, where $\text{Sum}(\text{subset}(i)) = m$.

```
int maxShares(int a[], int n) {  
    int sum = 0;
```

```

int i, m;
for (i=0; i<n; i++) sum += a[i];
for (m=n; m>=2; m--) {
    if (sum mod m != 0) continue;
    int aux[n]; for (i=0; i<n; i++) aux[i] = 0;
    if (testShares(a, n, m, sum, sum/m, aux, sum/m, 1)) return m;
}
return 1;
}

int testShares(int a[], int n, int m, int sum, int groupsum, int[] aux, int
goal, int groupId) {
    if (goal == 0) {
        groupId++;
        if (groupId == m+1) return 1;
    }
    for (int i=0; i<n; i++) {
        if (aux[i] != 0) continue;
        aux[i] = groupId;
        if (testShares(a, n, m, sum, groupsum, aux, goal-a[i], groupId)) {
            return 1;
        }
        aux[i] = 0;
    }
}

```

Please do edge cutting yourself, I'm quite enough of this...

46. 搜狐:

四对括号可以有多少种匹配排列方式? 比如两对括号可以有两种: ()() 和 (())

ANSWER:

Suppose k parenthesis has $f(k)$ permutations, k is large enough. Check the first parenthesis, if there are i parenthesis in it then, the number of permutations inside it and out of it are $f(i)$ and $f(k-i)$, respectively. That is

$$f(k) = \sum_{i=0}^{k-1} (f(i)*f(k-i-1));$$

which leads to the k 'th Catalan number.

47. 创新工场:

求一个数组的最长递减子序列比如{9, 4, 3, 2, 5, 4, 3, 2}的最长递减子序列为{9, 5, 4, 3, 2}

ANSWER:

Scan from left to right, maintain a decreasing sequence. For each number, binary search in the decreasing sequence to see whether it can be substituted.

```
int[] findDecreasing(int[] a) {
    int[] ds = new int[a.length];
    Arrays.fill(ds, 0);
    int dsl = 0;
    int lastdls = 0;
    for (int i=0; i<a.length; i++) {
        // binary search in ds to find the first element ds[j] smaller than
        a[i]. set ds[j] = a[i], or append a[i] at the end of ds
        int s=0, t=dsl-1;
        while (s<=t) {
            int m = s+(t-s)/2;
            if (ds[m] < a[i]) {
                t = m - 1;
            } else {
                s = m + 1;
            }
        }
        // now s must be at the first ds[j]<a[i], or at the end of ds[]
        ds[s] = a[i];
        if (s > dsl) { dsl = s; lastdls = i; }
    }
    // now trace back.
    for (int i=lastdls-1, j=dsl-1; i>=0 && j >= 0; i--) {
        if (a[i] == ds[j]) { j --; }
        else if (a[i] < ds[j]) { ds[j--] = a[i]; }
    }
    return Arrays.copyOfRange(ds, 0, dsl+1);
}
```

48.微软:

一个数组是由一个递减数列左移若干位形成的，比如{4, 3, 2, 1, 6, 5}

是由{6, 5, 4, 3, 2, 1}左移两位形成的，在这种数组中查找某一个数。

ANSWER:

The key is that, from the middle point of the array, half of the array is sorted, and the other half is a half-size shifted sorted array. So this can also be done recursively like a binary search.

```

int shiftedBinarySearch(int a[], int k) {
    return helper(a, k, 0, n-1);
}

int helper(int a[], int k, int s, int t) {
    if (s>t) return -1;
    int m = s + (t-s)/2;
    if (a[m] == k) return m;
    else if (a[s] >= k && k > a[m]) return helper(a, k, s, m-1);
    else return helper(a, k, m+1, e);
}

```

49.一道看上去很吓人的算法面试题:

如何对 n 个数进行排序，要求时间复杂度 $O(n)$ ，空间复杂度 $O(1)$

ANSWER:

So a comparison sort is not allowed. Counting sort's space complexity is $O(n)$.

More ideas must be exchanged to find more conditions, else this is a crap.

50.网易有道笔试:

1.求一个二叉树中任意两个节点间的最大距离，两个节点的距离的定义是这两个节点间边的个数，

比如某个孩子节点和父节点间的距离是 1，和相邻兄弟节点间的距离是 2，优化时间空间复杂度。

ANSWER:

Have done this before.

2.求一个有向连通图的割点，割点的定义是，

如果除去此节点和与其相关的边，有向图不再连通，描述算法。

ANSWER:

Have done this before.

51.和为 n 连续正数序列。

题目：输入一个正数 n ，输出所有和为 n 连续正数序列。

例如输入 15，由于 $1+2+3+4+5=4+5+6=7+8=15$ ，所以输出 3 个连续序列 1-5、4-6 和 7-8。

分析：这是网易的一道面试题。

ANSWER:

It seems that this can be solved by factorization. However, factorization of large n is impractical!

Suppose $n = i + (i+1) + \dots + (j-1) + j$, then $n = (i+j)(j-i+1)/2 = (j^2 - i^2 + i + j)/2$
 $\Rightarrow j^2 + j + (i^2 - i - 2n) = 0 \Rightarrow j = \sqrt{i^2 - i + 1/4 + 2n} - 1/2$

We know $1 \leq i < j \leq n/2 + 1$

So for each i in $[1, n/2]$, do this arithmetic to check if there is a integer answer.

```
int findConsecutiveSequence(int n) {
    int count = 0;
    for (int i=1; i<=n/2; i++) {
        int sqrt = calcSqrt(4*i*i+8*n-4*i+1);
        if (sqrt == -1) continue;
        if ((sqrt & 1) == 1) {
            System.out.println(i + " - " + ((sqrt-1)/2));
            count++;
        }
    }
    return count;
}
```

Use binary search to calculate sqrt, or just use math functions.

52.二元树的深度。

题目：输入一棵二元树的根结点，求该树的深度。

从根结点到叶结点依次经过的结点（含根、叶结点）形成树的一条路径，最长路径的长度为树的深度。

例如：输入二元树：

```
10
 / \
6 14
 / / \
4 12 16
```

输出该树的深度 3。

二元树的结点定义如下：

```
struct SBinaryTreeNode // a node of the binary tree
{
    int m_nValue; // value of node
    SBinaryTreeNode *m_pLeft; // left child of node
    SBinaryTreeNode *m_pRight; // right child of node
};
```

分析：这道题本质上还是考查二元树的遍历。

ANSWER:

Have done this.

53.字符串的排列。

题目：输入一个字符串，打印出该字符串中字符的所有排列。

例如输入字符串 abc，则输出由字符 a、b、c 所能排列出来的所有字符串
abc、acb、bac、bca、cab 和 cba。

分析：这是一道很好的考查对递归理解的编程题，

因此在过去一年中频繁出现在各大公司的面试、笔试题中。

ANSWER:

Full permutation generation. I will use another technique that swap two neighboring characters each time. It seems that all the characters are different. I need to think about how to do it when duplications is allowed. Maybe simple recursion is better for that.

```
void generatePermutation(char s[], int n) {
    if (n>20) { error("are you crazy?"); }
    byte d[n];
    int pos[n], dpos[n]; // pos[i], the position of i'th number, dpos[i]
the number in s[i] is the dpos[i]'th smallest
    qsort(s); // I cannot remember the form of qsort in C...
    memset(d, -1, sizeof(byte)*n);
    for (int i=0; i<n; i++) pos[i]=i, dpos[i]=i;

    int r;
    while (r = findFirstAvailable(s, d, pos, n)) {
        if (r== -1) return;
        swap(s, pos, dpos, d, r, r+d[r]);
        for (int i=n-1; i>dpos[r]; i--)
            d[i] = -d[i];
    }
}
int findFirstAvailable(char s[], byte d[], int pos[], int n) {
    for (int i=n-1; i>1; i--) {
        if (s[pos[i]] > s[pos[i]+d[pos[i]]]) return pos[i];
    }
    return -1;
}
```

```
#define aswap(ARR, X, Y) {int t=ARR[X]; ARR[X]=ARR[Y]; ARR[Y]=t;}
void swap(char s[], int pos[], int dpos[], byte d[], int r, int s) {
    aswap(s, r, s);
    aswap(d, r, s);
    aswap(pos, dpos[r], dpos[s]);
    aswap(dpos, r, s);
}
```

Maybe full of bugs. Please refer to algorithm manual for explanation.

Pros: Amotized O(1) time for each move. Only two characters change position for each move.

Cons: as you can see, very complicated. Extra space needed.

54. 调整数组顺序使奇数位于偶数前面。

题目：输入一个整数数组，调整数组中数字的顺序，使得所有奇数位于数组的前半部分，所有偶数位于数组的后半部分。要求时间复杂度为 O(n)。

ANSWER:

This problem makes me recall the process of partition in quick sort.

```
void partition(int a[], int n) {
    int i=j=0;
    while (i < n && (a[i] & 1)==0) i++;
    if (i==n) return;
    swap(a, i++, j++);
    while (i<n) {
        if ((a[i] & 1) == 1) {
            swap(a, i, j++);
        }
        i++;
    }
}
```

55. 题目：类 CMyString 的声明如下：

```
class CMyString
{
public:
    CMyString(char* pData = NULL);
    CMyString(const CMyString& str);
    ~CMyString(void);
    CMyString& operator = (const CMyString& str);
```

```

private:
    char* m_pData;
};

```

请实现其赋值运算符的重载函数，要求异常安全，即当对一个对象进行赋值时发生异常，对象的状态不能改变。

ANSWER

Pass...

56.最长公共字串。

题目：如果字符串一的所有字符按其在字符串中的顺序出现在另外一个字符串二中，则字符串一称之为字符串二的子串。

注意，并不要求子串（字符串一）的字符必须连续出现在字符串二中。

请编写一个函数，输入两个字符串，求它们的最长公共子串，并打印出最长公共子串。

例如：输入两个字符串 BDCABA 和 ABCBDAB，字符串 BCBA 和 BDAB 都是它们的最长公共子串，则输出它们的长度 4，并打印任意一个子串。

分析：求最长公共子串（Longest Common Subsequence, LCS）是一道非常经典的动态规划题，因此一些重视算法的公司像 MicroStrategy 都把它当作面试题。

ANSWER:

Standard DP...

$$\text{lcs}(ap1, bp2) = \max\{ \text{lcs}(p1, p2) + 1, \text{lcs}(p1, bp2), \text{lcs}(ap1, p2) \}$$

```

int LCS(char *p1, char *p2) {
    int l1= strlen(p1)+1, l2=strlen(p2)+1;
    int a[l1*l2];
    for (int i=0; i<l1; i++) a[i*l2] = 0;
    for (int i=0; i<l2; i++) a[i] = 0;
    for (int i=1; i<l1; i++) {
        for (int j=1; j<l2; j++) {
            int max = MAX(a[(i-1)*l2+l1], a[i*l2+l1-1]);
            if (p1[i-1] == p2[j-1]) {
                max = (max > 1 + a[(i-1)*l2+j-1]) ? max : 1+a[(i-1)*l2+j-1];
            }
        }
    }
    return a[l1*l2-1];
}

```

57.用俩个栈实现队列。

题目：某队列的声明如下：

```

template<typename T> class CQueue
{
public:
    CQueue() {}
    ~CQueue() {}
    void appendTail(const T& node); // append a element to tail
    void deleteHead(); // remove a element from head
private:
    Stack<T> m_stack1;
    Stack<T> m_stack2;
};

```

分析：从上面的类的声明中，我们发现在队列中有两个栈。

因此这道题实质上是要求我们用两个栈来实现一个队列。

相信大家对栈和队列的基本性质都非常了解了：栈是一种后入先出的数据容器，

因此对队列进行的插入和删除操作都是在栈顶上进行；队列是一种先入先出的数据容器，

我们总是把新元素插入到队列的尾部，而从队列的头部删除元素。

ANSWER

Traditional problem in CLRS.

```

void appendTail(const T& node) {
    m_stack1.push(node);
}

T getHead() {
    if (!m_stack2.isEmpty()) {
        return m_stack2.pop();
    }
    if (m_stack1.isEmpty()) error("delete from empty queue");
    while (!m_stack1.isEmpty()) {
        m_stack2.push(m_stack1.pop());
    }
    return m_stack2.pop();
}

```

58.从尾到头输出链表。

题目：输入一个链表的头结点，从尾到头反过来输出每个结点的值。链表结点定义如下：

```

struct ListNode
{
    int m_nKey;
    ListNode* m_pNext;
};

```

分析：这是一道很有意思的面试题。

该题以及它的变体经常出现在各大公司的面试、笔试题中。

ANSWER

Have answered this...

59.不能被继承的类。

题目：用 C++设计一个不能被继承的类。

分析：这是 Adobe 公司 2007 年校园招聘的最新笔试题。

这道题除了考察应聘者的 C++基本功底外，还能考察反应能力，是一道很好的题目。

ANSWER:

I don't know c++.

Maybe it can be done by implement an empty private default constructor.

60.在 O (1) 时间内删除链表结点。

题目：给定链表的头指针和一个结点指针，在 O(1)时间删除该结点。链表结点的定义如下：

```
struct ListNode
{
    int m_nKey;
    ListNode* m_pNext;
};
```

函数的声明如下：

```
void DeleteNode(ListNode* pListHead, ListNode* pToBeDeleted);
```

分析：这是一道广为流传的 Google 面试题，能有效考察我们的编程基本功，还能考察我们的反应速度，

更重要的是，还能考察我们对时间复杂度的理解。

ANSWER:

Copy the data from tobedeleted's next to tobedeleted. then delete tobedeleted. The special case is tobedelete is the tail, then we must iterate to find its predecessor.

The amortized time complexity is O(1).

61.找出数组中两个只出现一次的数字

题目：一个整型数组里除了两个数字之外，其他的数字都出现了两次。

请写程序找出这两个只出现一次的数字。要求时间复杂度是 O(n)，空间复杂度是 O(1)。

分析：这是一道很新颖的关于位运算的面试题。

ANSWER:

XOR.

62.找出链表的第一个公共结点。

题目：两个单向链表，找出它们的第一个公共结点。

链表的结点定义为：

```
struct ListNode
{
    int m_nKey;
    ListNode* m_pNext;
};
```

分析：这是一道微软的面试题。微软非常喜欢与链表相关的题目，因此在微软的面试题中，链表出现的概率相当高。

ANSWER:

Have done this.

63. 在字符串中删除特定的字符。

题目：输入两个字符串，从第一字符串中删除第二个字符串中所有的字符。例如，输入”They are students.” 和”aeiou”，则删除之后的第一个字符串变成”Thy r stdnts.”。

分析：这是一道微软面试题。在微软的常见面试题中，与字符串相关的题目占了很大的一部分，因为写程序操作字符串能很好的反映我们的编程基本功。

ANSWER:

Have done this? Use a byte array / character hash to record second string. then use two pointers to shrink the 1st string.

64. 寻找丑数。

题目：我们把只包含因子 2、3 和 5 的数称作丑数（Ugly Number）。例如 6、8 都是丑数，但 14 不是，因为它包含因子 7。习惯上我们把 1 当做是第一个丑数。求按从小到大的顺序的第 1500 个丑数。

分析：这是一道在网络上广为流传的面试题，据说 google 曾经采用过这道题。

ANSWER:

TRADITIONAL.

Use heap/priority queue.

```
int no1500() {
    int heap[4500];
    heap[0] = 2; heap[1] = 3; heap[2] = 5;
    int size = 3;
    for (int i=1; i<1500; i++) {
        int s = heap[0];
        heap[0] = s*2; siftDown(heap, 0, size);
        heap[size] = s*3; siftUp(heap, size, size+1);
```

```

        heap[size+1] = s*5; siftUp(heap, size+1, size+2);
        size+=2;
    }
}

void siftDown(int heap[], int from, int size) {
    int c = from * 2 + 1;
    while (c < size) {
        if (c+1<size && heap[c+1] < heap[c]) c++;
        if (heap[c] < heap[from]) swap(heap, c, from);
        from = c; c=from*2+1;
    }
}
void siftUp(int heap[], int from, int size) {
    while (from > 0) {
        int p = (from - 1)/ 2;
        if (heap[p] > heap[from]) swap(heap, p, from);
        from = p;
    }
}

```

65.输出 1 到最大的 N 位数

题目：输入数字 n，按顺序输出从 1 最大的 n 位 10 进制数。比如输入 3，则输出 1、2、3 一直到最大的 3 位数即 999。

分析：这是一道很有意思的题目。看起来很简单，其实里面却有不少的玄机。

ANSWER:

So maybe n could exceed i32? I cannot tell where is the trick...

Who will output 2×10^9 numbers...

66.颠倒栈。

题目：用递归颠倒一个栈。例如输入栈{1, 2, 3, 4, 5}，1 在栈顶。

颠倒之后的栈为{5, 4, 3, 2, 1}，5 处在栈顶。

ANSWER:

Interesting...

```

void reverse(Stack stack) {
    if (stack.size() == 1) return;
    Object o = stack.pop();
    reverse(stack);
    putToBottom(stack, o);
}

```

```

void putToBottom(Stack stack, Object o) {
    if (stack.isEmpty()) {
        stack.push(o);
        return;
    }
    Object o2 = stack.pop();
    putToBottom(stack, o);
    stack.push(o2);
}

```

67.俩个闲玩娱乐。

1.扑克牌的顺子

从扑克牌中随机抽 5 张牌，判断是不是一个顺子，即这 5 张牌是不是连续的。2-10 为数字本身，A 为 1，J 为 11，Q 为 12，K 为 13，而大小王可以看成任意数字。

ANSWER:

```

// make king = 0
boolean isStraight(int a[]) {
    Arrays.sort(a);
    if (a[0] > 0) return checkGaps(a, 0, 4, 0);
    if (a[0] == 0 && a[1] != 0) return checkGaps(a, 1, 4, 1);
    return checkGaps(a, 2, 4, 2);
}

boolean checkGaps(int []a, int s, int e, int allowGaps) {
    int i=s;
    while (i<e) {
        allowGaps -= a[i+1] - a[i] - 1;
        if (allowGaps < 0) return false;
        i++;
    }
    return true;
}

```

2.n 个骰子的点数。把 n 个骰子扔在地上，所有骰子朝上一面的点数之和为 S。输入 n，打印出 S 的所有可能的值出现的概率。

ANSWER:

All the possible values includes n to 6n. All the event number is 6^n .

For $n \leq S \leq 6n$, the number of events is $f(S, n)$

$$f(S, n) = f(S-6, n-1) + f(S-5, n-1) + \dots + f(S-1, n-1)$$

number of events that all dices are 1s is only 1, and thus $f(k, k) = 1$, $f(1-6, 1) = 1$, $f(x, 1) = 0$

where $x < 1$ or $x > 6$, $f(m, n) = 0$ where $m < n$

Can do it in DP.

```
void listAllProbabilities(int n) {
    int[][] f = new int[6*n+1][];
    for (int i=0; i<=6*n; i++) {
        f[i] = new int[n+1];
    }
    for (int i=1; i<=6; i++) {
        f[i][1] = 1;
    }
    for (int i=1; i<=n; i++) {
        f[i][i] = 1;
    }
    for (int i=2; i<=n; i++) {
        for (int j=i+1; j<=6*i; j++) {
            for (int k=(j-6<i-1)?i-1:j-6; k<j-1; k++)
                f[j][i] += f[k][i-1];
        }
    }
    double p6 = Math.power(6, n);
    for (int i=n; i<=6*n; i++) {
        System.out.println("P(S=" + i + ") = " + ((double)f[i][n] / p6));
    }
}
```

68. 把数组排成最小的数。

题目：输入一个正整数数组，将它们连接起来排成一个数，输出能排出的所有数字中最小的一个。

例如输入数组{32, 321}，则输出这两个能排成的最小数字 32132。

请给出解决问题的算法，并证明该算法。

分析：这是 09 年 6 月份百度的一道面试题，

从这道题我们可以看出百度对应聘者在算法方面有很高的要求。

ANSWER:

Actually this problem has little to do with algorithm...

The concern is, you must figure out how to arrange to achieve a smaller figure.

The answer is, if $ab < ba$, then $a < b$, and this is a total order.

```
String smallestDigit(int a[]) {
    Integer aux[] = new Integer[a.length];
```

```

for (int i=0; i<a.length; a++) aux[i] = a[i];
Arrays.sort(aux, new Comparator<Integer>(){
    int compareTo(Integer i1, Integer i2) {
        return (""+i1+i2).compareTo(""+i2+i1);
    }
});
StringBuffer sb = new StringBuffer();
for (int i=0; i<aux.length, i++) {
    sb.append(aux[i]);
}
return sb.toString();
}

```

69. 旋转数组中的最小元素。

题目：把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。输入一个排好序的数组的一个旋转，

输出旋转数组的最小元素。例如数组{3, 4, 5, 1, 2}为{1, 2, 3, 4, 5}的一个旋转，该数组的最小值为1。

分析：这道题最直观的解法并不难。从头到尾遍历数组一次，就能找出最小的元素，时间复杂度显然是 $O(N)$ 。但这个思路没有利用输入数组的特性，我们应该能找到更好的解法。

ANSWER

This is like the shifted array binary search problem. One blind point is that you may miss the part that the array is shifted by 0(or kN), that is not shifted.

```

int shiftedMinimum(int a[], int n) {
    return helper(a, 0, n-1);
}

int helper(int a[], int s, int t) {
    if (s == t || a[s] < a[t]) return a[s];
    int m = s + (t-s)/2;
    if (a[s]>a[m]) return helper(a, s, m);
    else return helper(a, m+1, t);
}

```

70. 给出一个函数来输出一个字符串的所有排列。

ANSWER 简单的回溯就可以实现了。当然排列的产生也有很多种算法，去看看组合数学，还有逆序生成排列和一些不需要递归生成排列的方法。

印象中 Knuth 的《TAOCP》第一卷里面深入讲了排列的生成。这些算法的理解需要一定的数学功底，也需要一定的灵感，有兴趣最好看看。

ANSWER:

Have done this.

71. 数值的整数次方。

题目：实现函数 double Power(double base, int exponent)，求 base 的 exponent 次方。

不需要考虑溢出。

分析：这是一道看起来很简单的问题。可能有不少的人在看到题目后 30 秒写出如下的代码：

```
double Power(double base, int exponent)
{
    double result = 1.0;
    for(int i = 1; i <= exponent; ++i)
        result *= base;
    return result;
}
```

ANSWER

...

```
double power(double base, int exp) {
    if (exp == 1) return base;
    double half = power(base, exp >> 1);
    return (((exp & 1) == 1) ? base : 1.0) * half * half;
}
```

72. 题目：设计一个类，我们只能生成该类的一个实例。

分析：只能生成一个实例的类是实现了 Singleton 模式的类型。

ANSWER

I'm not good at multithread programming... But if we set a lazy initialization, the "if" condition could be interrupted thus multiple constructor could be called, so we must add synchronized to the if judgements, which is a loss of efficiency. Putting it to the static initialization will guarantee that the constructor only be executed once by the java class loader.

```
public class Singleton {
    private static Singleton instance = new Singleton();
    private synchronized Singleton() {
    }
    public Singleton getInstance() {
        return instance();
    }
}
```

This may not be correct. I'm quite bad at this...

73. 对策字符串的最大长度。

题目：输入一个字符串，输出该字符串中对称的子字符串的最大长度。比如输入字符串“google”，由于该字符串里最长的对称子字符串是“goog”，因此输出 4。

分析：可能很多人都写过判断一个字符串是不是对称的函数，这个题目可以看成是该函数的加强版。

ANSWER

Build a suffix tree of x and $\text{inverse}(x)$, the longest anagram is naturally found.

Suffix tree can be built in $O(n)$ time so this is a linear time solution.

74. 数组中超过出现次数超过一半的数字

题目：数组中有一个数字出现的次数超过了数组长度的一半，找出这个数字。

分析：这是一道广为流传的面试题，包括百度、微软和 Google 在内的多家公司都曾经采用过这个题目。要几十分钟的时间里很好地解答这道题，除了较好的编程能力之外，还需要较快的反应和较强的逻辑思维能力。

ANSWER

Delete every two different digits. The last one that left is the one.

```
int getMajor(int a[], int n) {
    int x, cnt=0;
    for (int i=0; i<n; i++) {
        if (cnt == 0) {
            x = a[i]; cnt++;
        } else if (a[i]==x) {
            cnt++;
        } else {
            cnt--;
        }
    }
    return x;
}
```

75. 二叉树两个结点的最低共同父结点

题目：二叉树的结点定义如下：

```
struct TreeNode
{
    int m_nvalue;
    TreeNode* m_pLeft;
    TreeNode* m_pRight;
};
```

输入二叉树中的两个结点，输出这两个结点在数中最低的共同父结点。

分析：求数中两个结点的最低共同结点是面试中经常出现的一个问题。这个问题至少有两个变种。

ANSWER

Have done this. Do it again for memory...

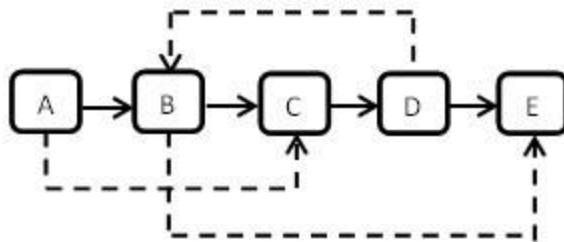
```
TreeNode* getLCA(TreeNode* root, TreeNode* X, TreeNode *Y) {  
    if (root == NULL) return NULL;  
    if (X == root || Y == root) return root;  
    TreeNode * left = getLCA(root->m_pLeft, X, Y);  
    TreeNode * right = getLCA(root->m_pRight, X, Y);  
    if (left == NULL) return right;  
    else if (right == NULL) return left;  
    else return root;  
}
```

76. 复杂链表的复制

题目：有一个复杂链表，其结点除了有一个 `m_pNext` 指针指向下一个结点外，还有一个 `m_pSibling` 指向链表中的任一结点或者 `NULL`。其结点的 C++ 定义如下：

```
struct ComplexNode  
{  
    int m_nValue;  
    ComplexNode* m_pNext;  
    ComplexNode* m_pSibling;  
};
```

下图是一个含有 5 个结点的该类型复杂链表。



图中实线箭头表示 `m_pNext` 指针，虚线箭头表示 `m_pSibling` 指针。为简单起见，指向 `NULL` 的指针没有画出。请完成函数 `ComplexNode* Clone(ComplexNode* pHead)`，以复制一个复杂链表。

分析：在常见的数据结构上稍加变化，这是一种很新颖的面试题。

要在不到一个小时的时间里解决这种类型的题目，我们需要较快的反应能力，对数据结构透彻的理解以及扎实的编程功底。

ANSWER

Have heard this before, never seriously thought it.

The trick is like this: take use of the old pSibling, make it points to the new created cloned node, while make the new cloned node's pNext backup the old pSibling.

```
ComplexNode * Clone(ComplexNode* pHead) {
    if (pHead == NULL) return NULL;
    preClone(pHead);
    inClone(pHead);
    return postClone(pHead);
}

void preClone(ComplexNode* pHead) {
    ComplexNode * p = new ComplexNode();
    p->m_pNext = pHead->m_pSibling;
    pHead->m_pSibling = p;
    if (pHead->m_pNext != NULL) preClone(pHead->m_pNext);
}

void inClone(ComplexNode * pHead) {
    ComplexNode * pSib = pHead->m_pNext;
    if (pSib == NULL) { pHead->m_pSibling = NULL; }
    else { pHead->m_pSibling = pSib->m_pSibling; }
    if (pHead->m_pNext != NULL) inClone(pHead->m_pNext);
}

ComplexNode * postClone(ComplexNode * pHead) {
    ComplexNode * pNew = pHead->m_pSibling;
    ComplexNode * pSib = pNew->m_pNext;
    if (pHead->m_pNext != NULL) {
        pNew->m_pNext = pHead->m_pNext->m_pSibling;
        pHead->m_pSibling = pSib;
        postClone(pHead->m_pNext);
    } else {
        pNew->pNext = NULL;
        pHead->m_pSibling = NULL;
    }
    return pNew;
}
```

77.关于链表问题的面试题目如下：

1.给定单链表，检测是否有环。

使用两个指针 p1,p2 从链表头开始遍历，p1 每次前进一步，p2 每次前进两步。如果 p2 到

达链表尾部，说明无环，否则 $p1$ 、 $p2$ 必然会在某个时刻相遇($p1==p2$)，从而检测到链表中有环。

2.给定两个单链表($head1$, $head2$)，检测两个链表是否有交点，如果有返回第一个交点。如果 $head1==head2$ ，那么显然相交，直接返回 $head1$ 。否则，分别从 $head1$, $head2$ 开始遍历两个链表获得其长度 $len1$ 与 $len2$ ，假设 $len1>=len2$ ，那么指针 $p1$ 由 $head1$ 开始向后移动 $len1-len2$ 步，指针 $p2=head2$ ，下面 $p1$ 、 $p2$ 每次向后前进一步并比较 $p1$ $p2$ 是否相等，如果相等即返回该结点，否则说明两个链表没有交点。

3.给定单链表($head$)，如果有环的话请返回从头结点进入环的第一个节点。

运用题一，我们可以检查链表中是否有环。如果有环，那么 $p1$ $p2$ 重合点 p 必然在环中。从 p 点断开环，方法为: $p1=p$, $p2=p->next$, $p->next=NULL$ 。此时，原单链表可以看作两条单链表，一条从 $head$ 开始，另一条从 $p2$ 开始，于是运用题二的方法，我们找到它们的第一个交点即为所求。

4.只给定单链表中某个结点 p (并非最后一个结点，即 $p->next!=NULL$)指针，删除该结点。

办法很简单，首先是放 p 中数据，然后将 $p->next$ 的数据 copy 入 p 中，接下来删除 $p->next$ 即可。

5.只给定单链表中某个结点 p (非空结点)，在 p 前面插入一个结点。办法与前者类似，首先分配一个结点 q ，将 q 插入在 p 后，接下来将 p 中的数据 copy 入 q 中，然后再将要插入的数据记录在 p 中。

78.链表和数组的区别在哪里？

分析：主要在基本概念上的理解。

但是最好能考虑的全面一点，现在公司招人的竞争可能就在细节上产生，谁比较仔细，谁获胜的机会就大。

ANSWER

1. Besides the common staff, linked list is more abstract and array is usually a basic real world object. When mentioning “linked list”, it doesn’t matter how it is implemented, that is, as long as it supports “get data” and “get next”, it is a linked list. But almost all programming languages provides array as a basic data structure.
2. So array is more basic. You can implement a linked list in an array, but cannot in the other direction.

79.

1.编写实现链表排序的一种算法。说明为什么你会选择用这样的方法？

ANSWER

For linked list sorting, usually mergesort is the best choice. Pros: O(1) auxiliary space,

compared to array merge sort. No node creation, just pointer operations.

```
Node * linkedListMergeSort(Node * pHead) {
    int len = getLen(pHead);
    return mergeSort(pHead, len);
}

Node * mergeSort(Node * p, int len) {
    if (len == 1) { p->next = NULL; return p; }
    Node * pmid = p;
    for (int i=0; i<len/2; i++) {
        pmid = pmid->next;
    }
    Node * p1 = mergeSort(p, len/2);
    Node * p2 = mergeSort(pmid, len - len/2);
    return merge(p1, p2);
}
Node * merge(Node * p1, Node * p2) {
    Node * p = NULL, * ph = NULL;
    while (p1!=NULL && p2!=NULL) {
        if (p1->data<p2->data) {
            if (ph == NULL) {ph = p = p1;}
            else { p->next = p1; p1 = p1->next; p = p->next;}
        } else {
            if (ph == NULL) {ph = p = p2;}
            else { p->next = p2; p2 = p2->next; p = p->next;}
        }
        p->next = (p1==NULL) ? p2 : p1;
    }
    return ph;
}
```

2.编写实现数组排序的一种算法。说明为什么你会选择用这样的方法？

ANSWER

Actually, it depends on the data. If arbitrary data is given in the array, I would choose quick sort. It is easy to implement, fast.

3.请编写能直接实现 strstr() 函数功能的代码。

ANSWER

Substring test? Have done this.

80.阿里巴巴一道笔试题

问题描述:

12 个高矮不同的人,排成两排,每排必须是从矮到高排列,而且第二排比对应的第一排的人高,问排列方式有多少种?

这个笔试题,很 YD,因为把某个递归关系隐藏得很深。

ANSWER

Must be

1 a b

c d e

c could be 2th to 7th (has to be smaller than d, e... those 5 numbers),

so $f(12) = 6 f(10) = 6 * 5 f(8) = 30 * 4 f(6) = 120 * 3 f(4) = 360 * 2 f(2) = 720$

81.第 1 组百度面试题

1.一个 int 数组, 里面数据无任何限制, 要求求出所有这样的数 $a[i]$, 其左边的数都小于等于它, 右边的数都大于等于它。能否只用一个额外数组和少量其它空间实现。

ANSWER

Sort the array to another array, compare it with the original array, all $a[i] = b[i]$ are answers.

2.一个文件, 内含一千万行字符串, 每个字符串在 1K 以内, 要求找出所有相反的串对, 如 abc 和 cba。

ANSWER

So we have ~10G data. It is unlikely to put them all into main memory. Anyway, calculate the hash of each line in the first round, at the second round calculate the hash of the reverse of the line and remembers only the line number pairs that the hashes of the two directions collides. The last round only test those lines.

3.STL 的 set 用什么实现的? 为什么不用 hash?

ANSWER

I don't quite know. Only heard of that map in stl is implemented with red-black tree. One good thing over hash is that you don't need to re-hash when data size grows.

82.第 2 组百度面试题

1.给出两个集合 A 和 B, 其中集合 A={name},

集合 B={age、sex、scholarship、address、...},

要求:

问题 1、根据集合 A 中的 name 查询出集合 B 中对应的属性信息;

问题 2、根据集合 B 中的属性信息（单个属性，如 age<20 等），查询出集合 A 中对应的 name。

ANSWER

SQL? Not a good defined question.

2.给出一个文件，里面包含两个字段{url、size}，即 url 为网址，size 为对应网址访问的次数

要求：

问题 1、利用 Linux Shell 命令或自己设计算法，查询出 url 字符串中包含“baidu”子字符串对应的 size 字段值；

问题 2、根据问题 1 的查询结果，对其按照 size 由大到小的排列。

（说明：url 数据量很大，100 亿级以上）

ANSWER

1. shell: gawk '/baidu/ { print \$2 }' FILE

2. shell: gawk '/baidu/ {print \$2}' FILE | sort -n -r

83.第 3 组百度面试题

1.今年百度的一道题目

百度笔试：给定一个存放整数的数组，重新排列数组使得数组左边为奇数，右边为偶数。

要求：空间复杂度 O(1)，时间复杂度为 O (n)。

ANSWER

Have done this.

2.百度笔试题

用 C 语言实现函数 void * memmove(void * dest, const void * src, size_t n)。memmove 函数的功能是拷贝 src 所指的内存内容前 n 个字节到 dest 所指的地址上。

分析：

由于可以把任何类型的指针赋给 void 类型的指针，这个函数主要是实现各种数据类型的拷贝。

ANSWER

```
//To my memory, usually memcpy doesn't check overlap, memmove do
void * memmove(void * dest, const void * src, size_t n) {
    if (dest==NULL || src == NULL) error("NULL pointers");
    byte * psrc = (byte*)src;
    byte * pdest = (byte*)dest;
    int step = 1;
    if (dest < src + n) {
        psrc = (byte*)(src+n-1);
```

```

    pdest = (byte*)(dest+n-1);
    step = -1;
}
for (int i=0; i<n; i++) {
    pdest = psrc;
    pdest += step; psrc += step;
}
}

```

84. 第 4 组百度面试题

2010 年 3 道百度面试题[相信，你懂其中的含金量]

1.a~z 包括大小写与 0~9 组成的 N 个数，用最快的方式把其中重复的元素挑出来。

ANSWER

By fastest, so memory is not the problem, hash is the first choice. Or trie will do.

Both run in O(Size) time, where size is the total size of the input.

2. 已知一随机发生器，产生 0 的概率是 p，产生 1 的概率是 1-p，现在要你构造一个发生器，使得它构造 0 和 1 的概率均为 1/2；构造一个发生器，使得它构造 1、2、3 的概率均为 1/3；…，构造一个发生器，使得它构造 1、2、3、…n 的概率均为 1/n，要求复杂度最低。

ANSWER

Run rand() twice, we got 00, 01, 10 or 11. If it's 00 or 11, discard it, else output 0 for 01, 1 for 10.

Similarly, assume $C(M, 2) \geq n$ and $C(M-1, 2) < n$. Do M rand()'s and get a binary string of M length. Assign 1100...0 to 1, 1010...0 to 2, ...

3. 有 10 个文件，每个文件 1G，

每个文件的每一行都存放的是用户的 query，每个文件的 query 都可能重复。

要求按照 query 的频度排序。

ANSWER

If there is no enough memory, do bucketing first. For each bucket calculate the frequency of each query and sort. Then combine all the frequencies with multiway mergesort.

85. 又见字符串的问题

1. 给出一个函数来复制两个字符串 A 和 B。字符串 A 的后几个字节和字符串 B 的前几个字节重叠。分析：记住，这种题目往往就是考你对边界的考虑情况。

ANSWER

Special case of memmove.

2. 已知一个字符串，比如 asderwsde, 寻找其中的一个子字符串比如 sde 的个数，如果没有返回 0，有的话返回子字符串的个数。

ANSWER

```
int count_of_substr(const char* str, const char * sub) {
    int count = 0;
    char * p = str;
    int n = strlen(sub);
    while (*p != '\0') {
        if (strcmp(p, sub, n) == 0) count++;
        p++;
    }
    return count;
}
```

Also recursive way works. Possible optimizations like Sunday algorithm or Rabin-Karp algorithm will do.

86.

怎样编写一个程序，把一个有序整数数组放到二叉树中？

分析：本题考察二叉搜索树的建树方法，简单的递归结构。关于树的算法设计一定要联想到递归，因为树本身就是递归的定义。而，学会把递归改称非递归也是一种必要的技术。毕竟，递归会造成栈溢出，关于系统底层的程序中不到非不得以最好不要用。但是对某些数学问题，就一定要学会用递归去解决。

ANSWER

This is the first question I'm given in a google interview.

```
Node * array2Tree(int[] array) {
    return helper(array, 0, n-1);
}

Node * helper(int[] array, int start, int end) {
    if (start > end) return NULL;
    int m = start + (end-start)/2;
    Node * root = new Node(array[m]);
    root->left = helper(array, start, m-1);
    root->right = helper(array, m+1, end);
    return root;
}
```

87.

1.大整数数相乘的问题。(这是 2002 年在一考研班上遇到的算法题)

ANSWER

Do overflow manually.

```
final static long mask = (1 << 31) - 1;
ArrayList<Integer> multiply(ArrayList <Integer> a, ArrayList<Integer> b)
{
    ArrayList<Integer> result = new ArrayList<Integer>(a.size()*b.size()
+1);
    for (int i=0; i<a.size(); i++) {
        multiply(b, a.get(i), i, result);
    }
    return result;
}
void multiply(ArrayList<Integer> x, int a, int base, ArrayList<Integer>
result) {
    if (a == 0) return;
    long overflow = 0;
    int i;
    for (i=0; i<x.size(); i++) {
        long tmp = x.get(i) * a + result.get(base+i) + overflow;
        result.set(base+i, (int)(mask & tmp));
        overflow = (tmp >> 31);
    }
    while (overflow != 0) {
        long tmp = result.get(base+i) + overflow;
        result.set(base+i, (int) (mask & tmp));
        overflow = (tmp >> 31);
    }
}
```

2.求最大连续递增数字串(如“ads3sl456789DF3456Id345AA”中的“456789”)

ANSWER

Have done this.

3.实现 strstr 功能，即在父串中寻找子串首次出现的位置。

(笔试中常让面试者实现标准库中的一些函数)

ANSWER

Have done this.

88.2005 年 11 月金山笔试题。编码完成下面的处理函数。

函数将字符串中的字符'*'移到串的前部分，前面的非'*'字符后移，但不能改变非'*'字符的先后顺序，函数返回串中字符'*'的数量。如原始串为：ab**cd**e*12，处理后为*****abcde12，函数并返回值为 5。（要求使用尽量少的时间和辅助空间）

ANSWER

It's like partition in quick sort. Just keep the non-* part stable.

```
int partitionStar(char a[]) {
    int count = 0;
    int i = a.length-1, j=a.length-1; // i for the cursor, j for the first
non-* char
    while (i >= 0) {
        if (a[i] != '*') {
            swap(a, i--, j--);
        } else {
            i--; count++;
        }
    }
    return count;
}
```

89.神州数码、华为、东软笔试题

1.2005 年 11 月 15 日华为软件研发笔试题。实现一单链表的逆转。

ANSWER

Have done this.

2.编码实现字符串转整型的函数（实现函数 atoi 的功能），据说是神州数码笔试题。如将字符串”+123” 123, ” -0123” -123, “123CS45” 123, “123.45CS” 123, “CS123.45” 0

ANSWER

```
int atoi(const char * a) {
    if (*a=='+') return atoi(a+1);
    else if (*a=='-') return - atoi(a+1);
    char *p = a;
    int c = 0;
    while (*p >= '0' && *p <= '9') {
        c = c*10 + (*p - '0');
    }
    return c;
}
```

3.快速排序（东软喜欢考类似的算法填空题，又如堆排序的算法等）

ANSWER

Standard solution. Skip.

4.删除字符串中的数字并压缩字符串。如字符串”abc123de4fg56”处理后变为”abcdefg”。

注意空间和效率。（下面的算法只需要一次遍历，不需要开辟新空间，时间复杂度为 $O(N)$ ）

ANSWER

Also partition, keep non-digit stable.

```
char * partition(const char * str) {
    char * i = str; // i for cursor, j for the first digit char;
    char * j = str;
    while (*i != '\0') {
        if (*i > '9' || *i < '0') {
            *j++ = *i++;
        } else {
            *i++;
        }
    }
    *j = '\0';
    return str;
}
```

5.求两个串中的第一个最长子串（神州数码以前试题）。

如"abRACTyeyt","dgdsaeACTyey"的最大子串为"ACTyet"。

ANSWER

Use suffix tree. The longest common substring is the longest prefix of the suffixes.

$O(n)$ to build suffix tree. $O(n)$ to find the lcs.

90.

1.不开辟用于交换数据的临时空间，如何完成字符串的逆序

(在技术一轮面试中，有些面试官会这样问)。

ANSWER

Two cursors.

2.删除串中指定的字符

(做此题时，千万不要开辟新空间，否则面试官可能认为你不适合做嵌入式开发)

ANSWER

Have done this.

3. 判断单链表中是否存在环。

ANSWER

Have done this.

91

1. 一道著名的毒酒问题

有 1000 桶酒，其中 1 桶有毒。而一旦吃了，毒性会在 1 周后发作。现在我们用小老鼠做实验，要在 1 周内找出那桶毒酒，问最少需要多少老鼠。

ANSWER

Have done this. 10 mices.

2. 有趣的石头问题

有一堆 1 万个石头和 1 万个木头，对于每个石头都有 1 个木头和它重量一样，把配对的石头和木头找出来。

ANSWER

Quick sort.

92.

1. 多人排成一个队列，我们认为从低到高是正确的序列，但是总有部分人不遵守秩序。如果说，前面的人比后面的人高(两人身高一样认为是合适的)，那么我们就认为这两个人是一对“捣乱分子”，比如说，现在存在一个序列：

176, 178, 180, 170, 171

这些捣乱分子对为

<176, 170>, <176, 171>, <178, 170>, <178, 171>, <180, 170>, <180, 171>,

那么，现在给出一个整型序列，请找出这些捣乱分子对的个数(仅给出捣乱分子对的数目即可，不用具体的对)

要求：

输入：

为一个文件(**in**)，文件的每一行为一个序列。序列全为数字，数字间用“,”分隔。

输出：

为一个文件(**out**)，每行为一个数字，表示捣乱分子的对数。

详细说明自己的解题思路，说明自己实现的一些关键点。

并给出实现的代码，并分析时间复杂度。

限制：

输入每行的最大数字个数为 100000 个，数字最长为 6 位。程序无内存使用限制。

ANSWER

The answer is the swap number of insertion sort. The straightforward method is to do insertion sort and accumulate the swap numbers, which is slow: $O(n^2)$

A sub-quadratic solution can be done by DP.

$$f(n) = f(n-1) + \text{Index}(n)$$

$\text{Index}(n)$, which is to determine how many numbers is smaller than $a[n]$ in $a[0..n-1]$, can be done in $\log(n)$ time using BST with subtree size.

93.在一个 int 数组里查找这样的数，它大于等于左侧所有数，小于等于右侧所有数。直观想法是用两个数组 a、b。 $a[i]$ 、 $b[i]$ 分别保存从前到 i 的最大的数和从后到 i 的最小的数，一个解答：这需要两次遍历，然后再遍历一次原数组，将所有 $\text{data}[i] \geq a[i-1] \& \& \text{data}[i] \leq b[i]$ 的 $\text{data}[i]$ 找出即可。给出这个解答后，面试官有要求只能用一个辅助数组，且要求少遍历一次。

ANSWER

It is natural to improve the hint... just during the second traversal, do the range minimum and picking together. There is no need to store the range minimums.

94.微软笔试题

求随机数构成的数组中找到长度大于=3 的最长的等差数列，输出等差数列由小到大：

如果没有符合条件的就输出

格式：

输入[1,3,0,5,-1,6]

输出[-1,1,3,5]

要求时间复杂度，空间复杂度尽量小

ANSWER

Firstly sort the array. Then do DP: for each $a[i]$, update the length of the arithmetic sequences. That's a $O(n^3)$ solution. Each arithmetic sequence can be determined by the last item and the step size.

95.华为面试题

1 判断一字符串是不是对称的，如： abccba

ANSWER

Two cursors.

2.用递归的方法判断整数组 a[N]是不是升序排列

ANSWER

```
boolean isAscending(int a[]) {  
    return isAscending(a, 0);  
}  
boolean isAscending(int a[], int start) {  
    return start == a.length - 1 || isAscending(a, start+1);  
}
```

96.08 年中兴校园招聘笔试题

1. 编写 strcpy 函数

已知 strcpy 函数的原型是

```
char *strcpy(char *strDest, const char *strSrc);
```

其中 strDest 是目的字符串，strSrc 是源字符串。不调用 C++/C 的字符串库函数，请编写函数 strcpy

ANSWER

```
char *strcpy(char *strDest, const char *strSrc) {  
    if (strSrc == NULL) return NULL;  
    char *i = strSrc, *j = strDest;  
    while (*i != '\0') {  
        *j++ = *i++;  
    }  
    *j = '\0';  
    return strDest;  
}
```

Maybe you need to check if src and dest overlaps, then decide whether to copy from tail to head.

最后压轴之戏，终结此微软等 100 题系列 V0.1 版。

那就，

连续来几组微软公司的面试题，让你一次爽个够：

=====

97.第 1 组微软较简单的算法面试题

1.编写反转字符串的程序，要求优化速度、优化空间。

ANSWER

Have done this.

2.在链表里如何发现循环链接?

ANSWER

Have done this.

3.编写反转字符串的程序，要求优化速度、优化空间。

ANSWER

Have done this.

4.给出洗牌的一个算法，并将洗好的牌存储在一个整形数组里。

ANSWER

Have done this.

5.写一个函数，检查字符是否是整数，如果是，返回其整数值。

(或者：怎样只用 4 行代码编写出一个从字符串到长整形的函数？)

ANSWER

Char or string?

have done atoi;

98.第 2 组微软面试题

1.给出一个函数来输出一个字符串的所有排列。

ANSWER

Have done this...

2.请编写实现 malloc() 内存分配函数功能一样的代码。

ANSWER

Way too hard as an interview question...

Please check wikipedia for solutions...

3.给出一个函数来复制两个字符串 A 和 B。字符串 A 的后几个字节和字符串 B 的前几个字节重叠。

ANSWER

Copy from tail to head.

4.怎样编写一个程序，把一个有序整数数组放到二叉树中？

ANSWER

Have done this.

5.怎样从顶部开始逐层打印二叉树结点数据？请编程。

ANSWER

Have done this...

6.怎样把一个链表掉个顺序（也就是反序，注意链表的边界条件并考虑空链表）？

ANSWER

Have done this...

99.第 3 组微软面试题

1.烧一根不均匀的绳，从头烧到尾总共需要 1 小时。现在有若干条材质相同的绳子，问如何用烧绳的方法来计时一个小时十五分钟呢？

ANSWER

May have done this... burn from both side gives $\frac{1}{2}$ hour.

2.你有一桶果冻，其中有黄色、绿色、红色三种，闭上眼睛抓取同种颜色的两个。抓取多少个就可以确定你肯定有两个同一颜色的果冻？(5 秒-1 分钟)

ANSWER

4.

3.如果你有无穷多的水，一个 3 公升的提捅，一个 5 公升的提捅，两只提捅形状上下都不均匀，问你如何才能准确称出 4 公升的水？(40 秒-3 分钟)

ANSWER

5 to 3 => 2

2 to 3, remaining 1

5 to remaining 1 => 4

一个岔路口分别通向诚实国和说谎国。

来了两个人，已知一个是诚实国的，另一个是说谎国的。

诚实国永远说实话，说谎国永远说谎话。现在你要去说谎国，

但不知道应该走哪条路，需要问这两个人。请问应该怎么问？(20 秒-2 分钟)

ANSWER

Seems there are too many answers.

I will pick anyone to ask: how to get to your country? Then pick the other way.

100. 第 4 组微软面试题，挑战思维极限

1.12 个球一个天平，现知道只有一个和其它的重量不同，问怎样称才能用三次就找到那个球。13 个呢？（注意此题并未说明那个球的重量是轻是重，所以需要仔细考虑）（5 分钟-1 小时）

ANSWER

Too complicated. Go find brain teaser answers by yourself.

2. 在 9 个点上画 10 条直线，要求每条直线上至少有三个点？（3 分钟-20 分钟）

3. 在一天的 24 小时之中，时钟的时针、分针和秒针完全重合在一起的时候有几次？都分别是什么时间？你怎样算出来的？（5 分钟-15 分钟）

30

终结附加题：

微软面试题，挑战你的智商

=====

说明：如果你是第一次看到这种题，并且以前从来没有见过类似的题型，
并且能够在半个小时之内做出答案，说明你的智力超常..)

1. 第一题. 五个海盗抢到了 100 颗宝石，每一颗都一样大小和价值连城。他们决定这么分：
抽签决定自己的号码（1、2、3、4、5）

首先，由 1 号提出分配方案，然后大家表决，当且仅当超过半数的人同意时，
按照他的方案进行分配，否则将被扔进大海喂鲨鱼

如果 1 号死后，再由 2 号提出分配方案，然后剩下的 4 人进行表决，
当且仅当超过半数的人同意时，按照他的方案进行分配，否则将被扔入大海喂鲨鱼。
依此类推

条件：每个海盗都是很聪明的人，都能很理智地做出判断，从而做出选择。

问题：第一个海盗提出怎样的分配方案才能使自己的收益最大化？

Answer:

A traditional brain teaser.

Consider #5, whatever #4 proposes, he won't agree, so #4 must agree whatever #3 proposes. So if there are only #3-5, #3 should propose (100, 0, 0). So the expected income of #3 is 100, and #4 and #5 is 0 for 3 guy problem. So whatever #2 proposes, #3

won't agree, but if #2 give #4 and #5 \$1, they can get more than 3-guy subproblem. So #2 will propose (98, 0, 1, 1). So for #1, if give #2 less than \$98, #2 won't agree. But he can give #3 \$1 and #4 or #5 \$2, so this is a (97, 0, 1, 2, 0) solution.

2.一道关于飞机加油的问题，已知：

每个飞机只有一个油箱，

飞机之间可以相互加油（注意是相互，没有加油机）

一箱油可供一架飞机绕地球飞半圈，

问题：

为使至少一架飞机绕地球一圈回到起飞时的飞机场，至少需要出动几架飞机？

（所有飞机从同一机场起飞，而且必须安全返回机场，不允许中途降落，中间没有飞机场）

Pass。ok，微软面试全部 100 题答案至此完。

后记

2010 已过，如今个人早已在整理 2011 最新的面试题，参见如下：

- 微软、谷歌、百度等公司经典面试 100 题[第 1-60 题]（微软 100 题第二版前 60 题）
- 微软、Google 等公司非常好的面试题及解答[第 61-70 题]（微软 100 题第二版第 61-70 题）
- 十道海量数据处理面试题与十个方法大总结（十道海量数据处理面试题）
- 海量数据处理面试题集锦与 Bit-map 详解（十七道海量数据处理面试题）
- 九月腾讯，创新工场，淘宝等公司最新面试十三题（2011 年度 9 月最新面试 30 题）
- 十月百度，阿里巴巴，迅雷搜狗最新面试十一题（2011 年度十月最新面试题集锦）

一切的详情，可看此文：[横空出世，席卷 Csdn—评微软等数据结构+算法面试 100 题](#)（在此文中，你能找到与微软 100 题所有一切相关的东西）。资源下载和维护地址分别如下所示：

- 所有的资源下载（题目+答案）地址：http://v_july_v.download.csdn.net/。
- 本微软等 100 题系列 V0.1 版，永久维护地址：<http://topic.csdn.net/u/20101126/10/b4f12a00-6280-492f-b785-cb6835a63dc9.html>。

欢迎，任何人，就以上任何内容，题目与答案思路，或其它任何问题、与我联系。本人邮箱：zhoulei0907@yahoo.cn。

更新：本微软公司面试 100 题的全部答案日前已经上传资源，所有读者可到此处下载：
http://download.csdn.net/detail/v_JULY_v/3685306。2011.10.15。

程序员编程艺术第一~二十七章集锦与总结

- 第一章、左旋转字符串
- 第二章、字符串是否包含问题
- 第三章、寻找最小的 k 个数
- 第三章续、Top K 算法问题的实现
- 第三章再续：快速选择 SELECT 算法的深入分析与实现
- 三之三续、求数组中给定下标区间内的第 K 小（大）元素
- 第四章、现场编写类似 strstr/strcpy/strpbrk 的函数
- 第五章、寻找满足条件的两个或多个数
- 第六章、求解 500 万以内的亲和数
- 第七章、求连续子数组的最大和
- 第八章、从头至尾漫谈虚函数
- 第九章、闲话链表追赶问题
- 第十章、如何给 10^7 个数据量的磁盘文件排序
- 第十一章、最长公共子序列（LCS）问题
- 第十二~十五章：数的判断，中签概率，IP 访问次数，回文问题（初稿）
- 第十六~第二十章：全排列，跳台阶，奇偶排序，第一个只出现一次等问题
- 第二十一~二十二章：出现次数超过一半的数字，最短摘要的生成
- 第二十三、四章：杨氏矩阵查找，倒排索引关键词 Hash 不重复编码实践
- 第二十五章：Jon Bentley：90%无法正确实现二分查找
- 第二十六章：基于给定的文档生成倒排索引的编码与实践
- 第二十七章：不改变正负数之间相对顺序重新排列数组

作者声明：本人 July 对以上所有内容和资料享有版权，转载请注明作者本人 July 及出处。向你的厚道致敬。谢谢。二零一一年十月十三日、以诸君为傲。

全新整理：微软、谷歌、百度等公司经典面试 100 题[第 101-160 题]

整理:July、二零一一年三月九日。

应网友承诺与要求，全新整理。转载，请注明出处。

博主说明：

此 100 题 V0.2 版，本人不再保证，还会提供答案。

因为之前整理的 [微软 100 题](#)，已经基本上，把题目都出尽了。见谅。

微软十五道面试题

1、有一个整数数组，请求出两两之差绝对值最小的值，

记住，只要得出最小值即可，不要求求出是哪两个数。

2、写一个函数，检查字符是否是整数，如果是，返回其整数值。

(或者：怎样只用 4 行代码编写出一个从字符串到长整形的函数？)

3、给出一个函数来输出一个字符串的所有排列。

4、(a)请编写实现 `malloc()` 内存分配函数功能一样的代码。

(b)给出一个函数来复制两个字符串 A 和 B。字符串 A 的后几个字节和字符串 B 的前几个字节重叠。

5、怎样编写一个程序，把一个有序整数数组放到二叉树中？

6、怎样从顶部开始逐层打印二叉树结点数据？请编程。

7、怎样把一个链表掉个顺序（也就是反序，注意链表的边界条件并考虑空链表）？

8、请编写能直接实现 `int atoi(const char * pstr)` 函数功能的代码。

9、编程实现两个正整数的除法

编程实现两个正整数的除法，当然不能用除法操作符。

```
// return x/y.  
int div(const int x, const int y)  
{  
    ....  
}
```

10、在排序数组中，找出给定数字的出现次数

比如 [1, 2, 2, 2, 3] 中 2 的出现次数是 3 次。

11、平面上 N 个点，每两个点都确定一条直线，

求出斜率最大的那条直线所通过的两个点（斜率不存在的情况不考虑）。时间效率越高越好。

12、一个整数数列，元素取值可能是 0~65535 中的任意一个数，相同数值不会重复出现。

0 是例外，可以反复出现。

请设计一个算法，当你从该数列中随意选取 5 个数值，判断这 5 个数值是否连续相邻。

注意：

- 5 个数值允许是乱序的。比如： 8 7 5 0 6

- 0 可以通配任意数值。比如： 8 7 5 0 6 中的 0 可以通配成 9 或者 4

- 0 可以多次出现。

- 复杂度如果是 O(n2) 则不得分。

13、设计一个算法，找出二叉树上任意两个结点的最近共同父结点。

复杂度如果是 O(n2) 则不得分。

14、一棵排序二叉树，令 $f=(\text{最大值}+\text{最小值})/2$ ，

设计一个算法，找出距离 f 值最近、大于 f 值的结点。

复杂度如果是 O(n2) 则不得分。

15、一个整数数列，元素取值可能是 1~N (N 是一个较大的正整数) 中的任意一个数，相同数值不会重复出现。

设计一个算法，找出数列中符合条件的数对的个数，满足数对中两数的和等于 N+1。

复杂度最好是 O(n)，如果是 O(n2) 则不得分。

谷歌八道面试题

16、正整数序列 Q 中的每个元素都至少能被正整数 a 和 b 中的一个整除，现给定 a 和 b，需要计算出 Q 中的前几项，例如，当 a=3, b=5, N=6 时，序列为 3, 5, 6, 9, 10, 12

(1)、设计一个函数 void generate (int a,int b,int N,int * Q) 计算 Q 的前几项

(2)、设计测试数据来验证函数程序在各种输入下的正确性。

17、有一个由大小写组成的字符串，现在需要对他进行修改，将其中的所有小写字母排在大写字母的前面（大写或小写字母之间不要求保持原来次序），如有可能尽量选择时间和空间效率高的算法 c 语言函数原型 void proc (char *str) 也可以采用你自己熟悉的语言

18、如何随机选取 1000 个关键字

给定一个数据流，其中包含无穷尽的搜索关键字（比如，人们在谷歌搜索时不断输入的关键字）。如何才能从这个无穷尽的流中随机的选取 1000 个关键字？

19、判断一个自然数是否是某个数的平方

说明：当然不能使用开方运算。

20、给定能随机生成整数 1 到 5 的函数，写出能随机生成整数 1 到 7 的函数。

21、 $1024!$ 末尾有多少个 0？

22、有 5 个海盗，按照等级从 5 到 1 排列，最大的海盗有权提议他们如何分享 100 枚金币。

但其他人要对此表决，如果多数反对，那他就会被杀死。

他应该提出怎样的方案，既让自己拿到尽可能多的金币又不会被杀死？

（提示：有一个海盗能拿到 98% 的金币）

23、Google2009 华南地区笔试题

给定一个集合 $A=[0,1,3,8]$ （该集合中的元素都是在 0, 9 之间的数字，但未必全部包含），

指定任意一个正整数 K ，请用 A 中的元素组成一个大于 K 的最小正整数。

比如， $A=[1,0]$ $K=21$ 那么输出结构应该为 100。

百度三道面试题

24、用 C 语言实现一个 revert 函数，它的功能是将输入的字符串在原串上倒序后返回。

25、用 C 语言实现函数 `void * memmove(void *dest, const void *src, size_t n)`。`memmove` 函数的功能是拷贝 `src` 所指的内存内容前 n 个字节到 `dest` 所指的地址上。

分析：由于可以把任何类型的指针赋给 `void` 类型的指针，这个函数主要是实现各种数据类型的拷贝。

26、有一根 27 厘米的细木杆，在第 3 厘米、7 厘米、11 厘米、17 厘米、23 厘米这五个位置上各有一只蚂蚁。

木杆很细，不能同时通过一只蚂蚁。开始时，蚂蚁的头朝左还是朝右是任意的，它们只会朝前走或调头，但不会后退。

当任意两只蚂蚁碰头时，两只蚂蚁会同时调头朝反方向走。假设蚂蚁们每秒钟可以走一厘米的距离。

编写程序，求所有蚂蚁都离开木杆的最短时间和最长时间。

腾讯七道面试题

- 27、**请定义一个宏，比较两个数 a、b 的大小，不能使用大于、小于、if 语句
- 28、**两个数相乘，小数点后位数没有限制，请写一个高精度算法
- 29、**有 A、B、C、D 四个人，要在夜里过一座桥。他们通过这座桥分别需要耗时 1、2、5、10 分钟，只有一支手电，并且同时最多只能两个人一起过桥。请问，如何安排，能够在 17 分钟内这四个人都过桥？
- 30、**有 12 个小球，外形相同，其中一个小球的质量与其他 11 个不同，给一个天平，问如何用 3 次把这个小球找出来，并且求出这个小球是比其他的轻还是重
- 31、**在一个文件中有 10G 个整数，乱序排列，要求找出中位数。内存限制为 2G。只写出思路即可。
- 32、**一个文件中有 40 亿个整数，每个整数为四个字节，内存为 1GB，写出一个算法：求出这个文件里的整数里不包含的一个整数
- 33、**腾讯服务器每秒有 2w 个 QQ 号同时上线，找出 5min 内重新登入的 qq 号并打印出来。

雅虎三道面试题

- 34、**编程实现：把十进制数(long 型)分别以二进制和十六进制形式输出，不能使用 printf 系列
- 35、**编程实现：找出两个字符串中最大公共子字符串，如"abccade", "dgcadde"的最大子串为 "cad"
- 36、**有双向循环链表结点定义：

```
struct node
{
    int data;
    struct node *front,*next;
};
```

有两个双向循环链表 A, B，知道其头指针为： pHeadA,pHeadB， 请写一函数将两链表中 data 值相同的结点删除。

联想五道笔试题

- 37、1)**、设计函数 int atoi(char *s)。

- 2)、`int i=(j=4,k=8,l=16,m=32); printf("%d",i);` 输出是多少?
- 3)、解释局部变量、全局变量和静态变量的含义。
- 4)、解释堆和栈的区别。
- 5)、论述含参数的宏与函数的优缺点。

38、顺时针打印矩阵

题目：输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字。

例如：如果输入如下矩阵：

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

则依次打印出数字 1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10。

分析：包括 Autodesk、EMC 在内的多家公司在面试或者笔试里采用过这道题。

39、对称子字符串的最大长度

题目：输入一个字符串，输出该字符串中对称的子字符串的最大长度。

比如输入字符串 “google”，由于该字符串里最长的对称子字符串是 “goog”，因此输出 4。

分析：可能很多人都写过判断一个字符串是不是对称的函数，这个题目可以看成是该函数的加强版。

40、用 1、2、2、3、4、5 这六个数字，写一个 main 函数，打印出所有不同的排列，

如：512234、412345 等，要求：“4”不能在第三位，“3”与“5”不能相连。

41、微软面试题

一个有序数列，序列中的每一个值都能够被 2 或者 3 或者 5 所整除，1 是这个序列的第一个元素。求第 1500 个值是多少？

网易五道游戏笔试题

42、两个圆相交，交点是 A1, A2。现在过 A1 点做一直线与两个圆分别相交另外一点 B1, B2。

B1B2 可以绕着 A1 点旋转。问在什么情况下，B1B2 最长

43、Smith 夫妇召开宴会，并邀请其他 4 对夫妇参加宴会。在宴会上，他们彼此握手，并且满足没有一个人同自己握手，没有两个人握手一次以上，并且夫妻之间不握手。

然后 Mr. Smith 问其它客人握手的次数，每个人的答案是不一样的。

求 Mrs Smith 握手的次数

44、有 6 种不同颜色的球，分别记为 1,2,3,4,5,6，每种球有无数个。现在取 5 个球，求在一下

的条件下：

- 1、5 种不同颜色，
- 2、4 种不同颜色的球，
- 3、3 种不同颜色的球，
- 4、2 种不同颜色的球，

它们的概率。

45、有一次数学比赛，共有 A, B 和 C 三道题目。所有人都至少解答出一道题目，总共有 25 人。在没有答出 A 的人中，答出 B 的人数是答出 C 的人数的两倍；单单答出 A 的人，比其他答出 A 的人总数多 1；在所有只有答出一道题目的人当中，答出 B 和 C 的人数刚好是一半。求只答出 B 的人数。

46、从尾到头输出链表

题目：输入一个链表的头结点，从尾到头反过来输出每个结点的值。链表结点定义如下：

```
struct ListNode
{
    int m_nKey;
    ListNode* m_pNext;
};
```

分析：这是一道很有意思的面试题。该题以及它的变体经常出现在各大公司的面试、笔试题中。

47、金币概率问题（威盛笔试题）

题目：10 个房间里放着随机数量的金币。每个房间只能进入一次，并只能在一个房间中拿金币。一个人采取如下策略：前四个房间只看不拿。随后的房间只要看到比前四个房间都多的金币数，就拿。否则就拿最后一个房间的金币。

编程计算这种策略拿到最多金币的概率。

48、找出数组中唯一的重复元素

1-1000 放在含有 1001 个元素的数组中，只有唯一的一个元素值重复，其它均只出现一次。每个数组元素只能访问一次，设计一个算法，将它找出来；不用辅助存储空间，能否设计一个算法实现？

49、08 百度校园招聘的一道笔试题

题目大意如下：

一排 N (最大 1M) 个正整数+1 递增，乱序排列，第一个不是最小的，把它换成-1，
最小数为 a 且未知求第一个被-1 替换掉的数原来的值，并分析算法复杂度。

50、一道 SPSS 笔试题求解

题目：输入四个点的坐标，求证四个点是不是一个矩形

关键点：

1. 相邻两边斜率之积等于-1，
2. 矩形边与坐标系平行的情况下，斜率无穷大不能用积判断。
3. 输入四点可能不按顺序，需要对四点排序。

51、矩阵式螺旋输出

```
The Array matrix[5][5] is :  
    1   16   15   14   13  
    2   17   24   23   12  
    3   18   25   22   11  
    4   19   20   21   10  
    5   6    7    8    9  
  
Press any key to continue . . .
```

52、求两个或 N 个数的最大公约数和最小公倍数。

53、最长递增子序列

题目描述：设 $L = \langle a_1, a_2, \dots, a_n \rangle$ 是 n 个不同的实数的序列， L 的递增子序列是这样一个子序列

$L_{in} = \langle a_{k1}, a_{k2}, \dots, a_{km} \rangle$ ，其中 $k_1 < k_2 < \dots < k_m$ 且 $a_{k1} < a_{k2} < \dots < a_{km}$ 。

求最大的 m 值。

54、字符串原地压缩

题目描述：“eeeeaaaff” 压缩为 “e5a3f2”，请编程实现。

55、字符串匹配实现

请以两种方法，回溯与不回溯算法实现。

56、一个含 n 个元素的整数数组至少存在一个重复数，

请编程实现，在 $O(n)$ 时间内找出其中任意一个重复数。

57、求最大重叠区间大小

题目描述：请编写程序，找出下面“输入数据及格式”中所描述的输入数据文件中最大重叠区间的大小。对一个正整数 n ，如果 n 在数据文件中某行的两个正整数（假设为 A 和 B ）之间，即 $A \leq n \leq B$ 或 $A \geq n \geq B$ ，则 n 属于该行；

如果 n 同时属于行 i 和 j ，则 i 和 j 有重叠区间；重叠区间的大小是同时属于行 i 和 j 的整数个数。

例如，行 (10 20) 和 (12 25) 的重叠区间为 [12 20]，其大小为 9，行(20 10)和(20 30)的重叠区间大小为 1。

58、整数的素数和分解问题

歌德巴赫猜想说任何一个不小于 6 的偶数都可以分解为两个奇素数之和。

对此问题扩展，如果一个整数能够表示成两个或多个素数之和，则得到一个素数和分解式。

对于一个给定的整数，输出所有这种素数和分解式。

注意，对于同构的分解只输出一次（比如 5 只有一个分解 $2 + 3$ ，而 $3 + 2$ 是 $2 + 3$ 的同构分解式）。

例如，对于整数 8，可以作为如下三种分解：

$$(1) 8 = 2 + 2 + 2 + 2$$

$$(2) 8 = 2 + 3 + 3$$

$$(3) 8 = 3 + 5$$

59、google 的一道面试题

题目：

输入 $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n$ ，

在 $O(n)$ 的时间, $O(1)$ 的空间将这个序列顺序改为 $a_1, b_1, a_2, b_2, a_3, b_3, \dots, a_n, b_n$ ，

且不需要移动，通过交换完成，只需一个交换空间。

例如， $N=9$ 时，第 2 步执行后，实际上中间位置的两边对称的 4 个元素基本配对，只需交换中间的两个元素即可，如下表所示。颜色表示每次要交换的元素，左边向右交换，右边向左交换。

交换过程如下表所示

	1	2	3	4	5	6	7	8	9	n+ 1	n+ 2	n+ 3	n+ 4	n+ 5	n+ 6	n+ 7	n+ 8	n+ 9			
n- 8	n- 7	n- 6	n- 5	n- 4	n- 3	n- 2	n- 1	N	2n- 8	2n- 7	2n- 6	2n- 5	2n- 4	2n- 3	2n- 2	2n- 1	2n	交换开始位置	交换个数		
a1	a2	a3	a4	a5	a6	a7	a8	a9	b1	b2	b3	b4	b5	b6	b7	b8	b9	2↔n+1	2n- 1↔n	1	
1	a1	b1	a3	a4	a5	a6	a7	a8	b8	a2	b2	b3	b4	b5	b6	b7	a9	b9	3↔n+1	2n- 2↔n	2
2	a1	B1	a2	b2	a5	a6	a7	b6	b7	a3	a4	b3	b4	b5	a8	b8	a9	b9	5↔n+1	2n- 4↔n	4
3	a1	B1	a2	b2	X 1	Y1=(a6 a7 b7)		X 2	X3	Y2=(a4 b3 b4)			X4	a8	b8	a9	b9	对称交 换			
	a1	B1	a2	b2	X 3	Y2		X4	X1	Y1			X2	a8	b8	a9	b9				
	a1	B1	a2	b2	X 3	Y2		X1	X4	Y1			X2	a8	b8	a9	b9				
4	a1	B1	a2	b2	A 3	A 4	B3	B4	A 5	B5	A6	A7	B6	B7	a8	b8	a9	b9			
5	a1	B1	a2	b2	A 3	B3	A 4	B4	A 5	B5	A6	B6	A7	B7	a8	b8	a9	b9			

交换 x1,x3；交换 x2,x4；再交换中间的 x1,x4；交换 y1,y2。

60、百度笔试题

给定一个存放整数的数组，重新排列数组使得数组左边为奇数，右边为偶数。

要求：空间复杂度 O(1)，时间复杂度为 O (n)。

版权声明：

- 1、以上全部题目的知识产权，归原公司微软、谷歌、百度等公司所有。
- 2、本人对本 BLOG 内所有任何文章和资料享有版权，转载，请注明作者本人，并以链接形式注明出处。
- 3、侵犯本人版权相关利益者，个人会在[腾讯微博](#)、[CSDN 迷你博客](#)中永久追踪，给予谴责。
同时，保留追究法律责任的权利。向您的厚道致敬，谢谢。

July、二零一一年三月十日。

全新整理：微软、谷歌等公司非常好的面试题及解答[第 161-170 题]

整理：July。

时间：二零一一年四月十日。

微博：<http://weibo.com/julyweibo>。

出处：http://blog.csdn.net/v_JULY_v。

引言

此微软 100 题 V0.2 版的前 60 题，请见这：[微软、谷歌、百度等公司经典面试 100 题\[第 1-60 题\]](#)。关于本人整理[微软 100 题](#)的一切详情，请参见这：[横空出世，席卷 Csdn \[评微软等数据结构+算法面试 100 题\]](#)。

声明

1、下面的题目来不及一一细看，答案大部是摘自网友，且个人认为比较好一点的思路，对这些思路和答案本人未经细细验证，仅保留意见。

2、为尊重作者劳动成果，凡是引用了网友提供的面试题、思路，或答案，都一一注明了网友的昵称。若对以下任何一题的思路，不是很懂的，欢迎留言或评论中提出，我可再做详细阐述。

3、以下的每一题，都是自个平时一一搜集整理的，转载请务必注明出处。任何人，有任何问题，欢迎不吝指正。谢谢。

微软、Google 等公司一些非常好的面试题、第 61-70 题

61、腾讯现场招聘问题

liuchen1206

今天参加了腾讯的现场招聘会，碰到这个一个题目：

在一篇英文文章中查找指定的人名，人名使用二十六个英文字母（可以是大写或小写）、空格以及两个通配符组成（*、？），通配符“*”表示零个或多个任意字母，通配符“？”表示一个任意字母。

如：“J* Smi??” 可以匹配 “John Smith” .

请用 C 语言实现如下函数:

```
void scan(const char* pszText, const char* pszName);
```

注: `pszText` 为整个文章字符, `pszName` 为要求匹配的英文名。

请完成些函数实现输出所有匹配的英文名, 除了 `printf` 外, 不能用第三方的库函数等。

代码一 (此段代码已经多个网友指出, bug 不少, 但暂没想到解决办法):

```
//copyright@ falcomavin && July

//updated:
//多谢 Yingmg 网友指出, 由于之前这代码是从编译器->记事本->本博客, 辗转三次而来
//的, 所以, 之前的代码不符合缩进规范, 特此再把它搬到编译器上, 调整好缩进后, 不
//再放到记事本上, 而是直接从编译器上贴到这里来。

//July, 说明。2011.04.17。
#include <iostream>
using namespace std;

int scan(const char* text, const char* pattern)
{
    const char *p = pattern;      // 记录初始位置, 以便 pattern 匹配一半失败可返
    if (*pattern == 0) return 1;    // 匹配成功条件
    if (*text == 0) return 0;      // 匹配失败条件

    if (*pattern != '*' && *pattern != '?')
    {
        if (*text != *pattern)    //如果匹配不成功
            return scan(text+1, pattern); //text++, 寻找下一个匹配
    }

    if (*pattern == '?')
    {
        if (!isalpha(*text))    // 通配符'?'匹配失败
        {
            pattern = p;        // 还原 pattern 初始位置
            return scan(text+1, pattern); //text++, 寻找下一个匹配
        }
        else                    // 通配符'?'匹配成功
        {
            return scan(text+1, pattern + 1); //双双后移, ++
        }
    }
}
```

```

    return scan(text, pattern+1); // 能走到这里，一定是在匹配通配符'*'了
}

int main()
{
    char *i, *j;
    i = new char[100];
    j = new char[100];
    cin>>i>>j;
    cout<<scan(i,j);
    return 0;
}

```

代码二：

```

//qq120848369:
#include <iostream>
using namespace std;
const char *pEnd=NULL;

bool match(const char *pszText,const char *pszName)
{
    if(*pszName == '/0') // 匹配完成
    {
        pEnd=pszText;
        return true;
    }

    if(*pszText == '/0') // 未匹配完成
    {
        if(*pszName == '*')
        {
            pEnd=pszText;
            return true;
        }

        return false;
    }

    if(*pszName!= '*' && *pszName!= '?')
    {
        if(*pszText == *pszName)
        {

```

```

        return match(pszText+1,pszName+1);
    }

    return false;
}
else
{
    if(*pszName == '*')
    {
        return match(pszText,pszName+1)|match(pszText+1,pszName);
        //匹配0个,或者继续*匹配下去
    }
    else
    {
        return match(pszText+1,pszName+1);
    }
}

void scan(const char *pszText, const char *pszName)
{
    while(*pszText!= '/0')
    {
        if(match(pszText,pszName))
        {
            while(pszText!=pEnd)
            {
                cout<<*pszText++;
            }

            cout<<endl;
        }
        return;
    }
}

int main()
{
    char pszText[100],pszName[100];

    fgets(pszText,100,stdin);
    fgets(pszName,100,stdin);
    scan(pszText,pszName);
}

```

```
    return 0;
}
```

wangxugangzy05:

这个是 kmp 子串搜索（匹配），稍加改造，如 `abcabd*?abe***?de` 这个串，我们可以分成 `abcabd,?,abe,?,?`，并按顺序先匹配 `abcabd`，当匹配后，将匹配的文章中地址及匹配的是何子串放到栈里记录下来，这样，每次匹配都入栈保存当前子串匹配信息，当一次完整的全部子串都匹配完后，就输出一个匹配结果，然后回溯一下，开始对栈顶的子串的进行文中下一个起始位置的匹配。

62、微软三道面试题

yeardoublehua

1. 给一个有 N 个整数的数组 S 和另一个整数 X ，判断 S 里有没有 2 个数的和为 X ，请设计成 $O(n \log 2(n))$ 的算法。
2. 有 2 个数组..里面有 N 个整数。
设计一个算法 $O(n \log 2(n))$ ，看是否两个数组里存在一个同样的数。
3. 让你排序 N 个比 $N^{\sqrt{2}}$ 小的数，要求的算法是 $O(n)$ （给了提示..说往 N 进制那方面想）

qq120848369:

1,快排,头尾夹逼.

```
#include <iostream>
#include <algorithm>
#include <utility>
using namespace std;
typedef pair<int,int> Pair;

Pair findSum(int *s,int n,int x)
{
    sort(s,s+n); //引用了库函数

    int *begin=s;
    int *end=s+n-1;

    while(begin<end) //俩头夹逼，很经典的方法
    {
        if(*begin+*end>x)
        {
            --end;
        }
    }
}
```

```

        else if(*begin+*end<x)
        {
            ++begin;
        }
        else
        {
            return Pair(*begin,*end);
        }
    }

    return Pair(-1,-1);
}

int main()
{
    int arr[100]=
    {
        3, -4, 7, 8, 12, -5, 0, 9
    };

    int n=8,x;

    while(cin>>x)
    {
        Pair ret=findSum(arr,n,x);
        cout<<ret.first<<","<<ret.second<<endl;
    }

    return 0;
}

```

2,快排,线性扫描

```

#include <iostream>
#include <algorithm>
using namespace std;

bool findSame(const int *a,int len1,const int *b,int len2,int *result)
{
    int i,j;
    i=j=0;

    while(i<len1 && j<len2)
    {
        if(a[i]<b[j])

```

```

    {
        ++i;
    }
    else if(a[i]>b[j])
    {
        ++j;
    }
    else
    {
        *result=a[i];
        return true;
    }
}
return false;
}

int main()
{
    int a[100],b[100],len1,len2,result;
    cin>>len1;

    for(int i=0;i<len1;++i)
    {
        cin>>a[i];
    }

    cin>>len2;
    for(int i=0;i<len2;++i)
    {
        cin>>b[i];
    }

    if( findSame(a,len1,b,len2,&result) )
    {
        cout<<result<<endl;
    }
    return 0;
}

```

3,基数排序已经可以 $O(n)$ 了,准备 10 个 $\text{vector}<\text{int}>$,从最低位数字开始,放入相应的桶里,然后再顺序取出来,然后再从次低位放入相应桶里,在顺序取出来.比如: $N=5$, 分别是:

4,10,7,123,33

0 : 10

1

```
2
3 : 123,33
4 : 4
5
6
7 : 7
8
9
```

顺次取出来: 10,123,33,,4,7

```
0 : 4,7
1 : 10
2 : 123
3 : 33
4
5
6
7
8
9
```

依次取出来: 4,7,10,123,33

```
0 : 4,7, 10, 33
1 : 123
2
3
4
5
6
7
8
9
```

依次取出来: 4,7,10,33,123

完毕。

代码, 如下:

```
#include <iostream>
```

```
#include <string>
#include <queue>
#include <vector>

using namespace std;

size_t n;      //n 个数
size_t maxLen=0;    //最大的数字位数
vector< queue<string> > vec(10);    //10 个桶
vector<string> result;

void sort()
{
    for(size_t i=0;i<maxLen;++i)
    {
        for(size_t j=0;j<result.size();++j)
        {
            if( i>=result[j].length() )
            {
                vec[0].push(result[j]);
            }
            else
            {
                vec[ result[j][result[j].length()-1-i]-'0' ].push(result[j]);
            }
        }
        result.clear();
    }

    for(size_t k=0;k<vec.size();++k)
    {
        while(!vec[k].empty())
        {
            result.push_back(vec[k].front());
            vec[k].pop();
        }
    }
}

int main()
{
    cin>>n;
```

```

string input;

for(size_t i=0;i<n;++i)
{
    cin>>input;
    result.push_back(input);

    if(maxLen == 0 || input.length()>maxLen)
    {
        maxLen=input.length();
    }
}

sort();

for(size_t i=0;i<n;++i)
{
    cout<<result[i]<<" ";
}

cout<<endl;

return 0;
}

```

xiaoboalex:

第一题,1. 给一个有 N 个整数的数组 S..和另一个整数 X, 判断 S 里有没有 2 个数的和为 X, 请设计成 $O(n \log n)$ 的算法。

如果限定最坏复杂度 $n \lg n$ 的话就不能用快排。

可以用归并排序, 然后 $Y=X-E$, 用两分搜索依次查找每一个 Y 是否存在, 保证最坏复杂度为 $n \lg n$.

63、微软亚洲研究院

hinyunsin

假设有一颗二叉树, 已知这棵树的节点上不均匀的分布了若干石头, 石头数跟这棵二叉树的节点数相同, 石头只可以在子节点和父节点之间进行搬运, 每次只能搬运一颗石头。请问如何以最少的步骤将石头搬运均匀, 使得每个节点上的石头上刚好为 1。

个人, 暂时还没看到清晰的, 更好的思路, 以下是网友 mathe、casahama、Torey 等人给

的思路：

mathe:

我们对于任意一个节点，可以查看其本身和左子树，右子树的几个信息：

- i) 本身上面石子数目
- ii) 左子树中石子数目和节点数目的差值
- iii) 右子树中石子数目和节点数目的差值
- iv) 通过 i), ii), iii) 可以计算出除掉这三部份其余节点中石子和节点数目的差值。

如果上面信息都已经计算出来，那么对于这个节点，我们就可以计算出同其关联三条边石子运送最小数目。比如，如果左子树石子数目和节点数目差值为 $a < 0$, 那么表示比如通过这个节点通向左之数的边至少运送 a 个石子；反之如果 $a > 0$, 那么表示必须通过这个节点通向左子树的边反向运送 a 个石子。同样可以计算出同父节点之间的最小运送数目。

然后对所有节点，将这些数目 (ii, iii, iv 中) 绝对值相加就可以得出下界。

而证明这个下界可以达到也很简单。每次找出一个石子数目大于 1 的点，那么它至少有一条边需要向外运送，操作之即可。每次操作以后，必然上面这些绝对值数目减 1。所以有限步操作后必然达到均衡。所以现在唯一余下的问题就是如何计算这些数值问题。这个我们只要按照拓扑排序，从叶节点开始向根节点计算即可。

casaahama:

节点上的石头数不能小于 0。所以当子节点石头数 == 0 并且 父节点石头数 == 0 的时候，是需要继续向上回溯的。基于这一点，想在一次遍历中解决这个问题是不可能的。

这一点考虑进去的话，看来应该再多加一个栈保存这样类似的结点才行。

Torey:

后序遍历

证明：

在一棵只有三个节点的子二叉树里，石头在子树里搬运的步数肯定小于等于子树外面节点搬运的步数。

石头由一个子树移到另一个子树可归结为两步，一为石头移到父节点，二为石头由父节点移到子树结点，所以无论哪颗石头移到哪个节点，总步数总是一定。

关于树的遍历，在面试题中已出现过太多次了，特此稍稍整理以下：

二叉树结点存储的数据结构：

```
typedef char datatype;
typedef struct node
{
    datatype data;
    struct node* lchild,*rchild;
} bintnode;
typedef bintnode* bintree;
```

```
bintree root;
```

1.树的前序遍历即：

按根 左 右 的顺序，依次

前序遍历根结点->前序遍历左子树->前序遍历右子树

前序遍历，递归算法

```
void preorder(bintree t)
//注, bintree 为一指向二叉树根结点的指针
{
    if(t)
    {
        printf("%c",t->data);
        preorder(t->lchild);
        preorder(t->rchild);
    }
}
```

2.中序遍历，递归算法

```
void preorder(bintree t)
{
    if(t)
    {
        inorder(t->lchild);
        printf("%c",t->data);
        inorder(t->rchild);
    }
}
```

3.后序遍历，递归算法

```
void preorder(bintree t)
{
    if(t)
    {
        postorder(t->lchild);
        postorder(t->rchild);
        printf("%c",t->data);
    }
}
```

关于实现二叉树的前序、中序、后序遍历的递归与非递归实现的更多，请参考这（微软100题第43题答案）：

[http://blog.csdn.net/v_JULY_v/archive/2011/02/01/6171539.aspx。](http://blog.csdn.net/v_JULY_v/archive/2011/02/01/6171539.aspx)

64、淘宝校园笔试题

goengine

N 个鸡蛋放到 M 个篮子中，篮子不能为空，要满足：对任意不大于 N 的数量，能用若干个篮子中鸡蛋的和表示。

写出函数，对输入整数 N 和 M ，输出所有可能的鸡蛋的放法。

比如对于 9 个鸡蛋 5 个篮子

解至少有三组：

1 2 4 1 1

1 2 2 2 2

1 2 3 2 1

思路一、

Sorehead 在我的微软 100 题，维护地址上，已经对此题有了详细的思路与阐释，以下是他的个人思路+代码：

Sorehead

思路：

1、由于每个篮子都不能为空，可以转换成每个篮子先都有 1 个鸡蛋，再对剩下的 $N-M$ 个鸡蛋进行分配，这样就可以先求和为 $N-M$ 的所有排列可能性。

2、假设 $N-M=10$ ，求解所有排列可能性可以从一个比较简单的递规来实现，转变为下列数组：(10,0)、(9,1)、(8,2)、(7,3)、(6,4)、(5,5)、(4,6)、(3,7)、(2,8)、(1,9)

这里对其中第一个元素进行循环递减，对第二个元素进行上述递规重复求解，

例如(5,5)转变成：(5,0)、(4,1)、(3,2)、(2,3)、(1,4)

由于是求所有排列可能性，不允许有重复记录，因此结果就只能是非递增或者非递减队列，这里我采用的非递增队列来处理。

3、上面的递规过程中对于像(4,6)这样的不符合条件就可以跳过不输出，但递规不能直接跳出，必须继续进行下去，因为(4,6)的结果集中还是有不少能符合条件的。

我写的是非递规程序，因此(4,6)这样的组合我就直接转换成 4,4,2，然后再继续做处理。

4、 $N-M$ 的所有排列可能性已经求出来了，里面的元素全部加 1，如果 $N-M < M$ ，剩下的元素就全部是 1，这样 N 个鸡蛋放入 M 个篮子的所有可能性就全部求出来了。注意排列中可能元素数量会超过篮子数量 M ，去除这样的排列即可。

5、接下来的结果就是取出上述结果集中不满足“对于任意一个不超过 N 的正整数，都能由某几个篮子内蛋的数量相加得到”条件的记录了。

首先是根据这个条件去除不可能有结果的情况：如果 $M > N$ ，显而易见这是不可能有结果的；那对于给定的 N 值， M 是否不能小于某个值呢，答案是肯定的。

6、对于给定的 N 值， M 值最小的组合应该是 1,2,4,8,16,32... 这样的序列，这样我们就

可以计算出 M 的最小值可能了，如果 M 小于该值，也是不可能有结果的。

7、接下来，对于给定的结果集，由于有个篮子的鸡蛋数量必须为 1，可以先去掉最小值大于 1 的记录；同样，篮子中鸡蛋最大数量也应该不能超过某值，该值应该在 $N/2$ 左右，具体值要看 N 是奇数还是偶数了，原因是因为超过这个值，其它篮子的鸡蛋数量全部相加都无法得到比该值小 1 的数。

8、最后如何保证剩下的结果中都是符合要求的，这是个难题。当然有个简单方法就是对结果中的每个数挨个进行判断。

```
//下面是他写的代码:  
void malloc_egg(int m, int n)  
{  
    int *stack, top;  
    int count, max, flag, i;  
  
    if (m < 1 || n < 1 || m > n)  
        return;  
  
    //得到 m 的最小可能值，去除不可能情况  
    i = n / 2;  
    count = 1;  
    while (i > 0)  
    {  
        i /= 2;  
        count++;  
    }  
    if (m < count)  
        return;  
  
    //对 m=n 或 m=n-1 进行特殊处理  
    if (m >= n - 1)  
    {  
        if (m == n)  
            printf("1,");  
        else  
            printf("2,");  
        for (i = 0; i < m; i++)  
            printf("1,");  
        printf("/n");  
        return;  
    }  
  
    if ((stack = malloc(sizeof(int) * (n - m))) == NULL)  
        return;
```

```

stack[0] = n - m;
top = 0;

//得到篮子中鸡蛋最大数量值
max = n % 2 ? n / 2 : n / 2 - 1;
if (stack[0] <= max)
{
    printf("%d,", n - m + 1);
    for (i = 1; i < m; i++)
        printf("1,");
    printf("\n");
}

do
{
    count = 0;
    for (i = top; i >= 0 && stack[i] == 1; i--)
        count++;

    if (count > 0)
    {
        top -= count;
        stack[top]--;
        count++;
        //保证是个非递增数列
        while (stack[top] < count)
        {
            stack[top + 1] = stack[top];
            count -= stack[top];
            top++;
        }
        stack[++top] = count;
    }
    else
    {
        stack[top]--;
        stack[++top] = 1;
    }
}

//去除元素数量会超过篮子数量、超过鸡蛋最大数量值的记录
if (top >= m - 1)
    continue;
if (stack[0] > max)

```

```

continue;

//对记录中的每个数挨个进行判断，保证符合条件二
flag = 0;
count = m - top;
for (i = top; i >= 0; i--)
{
    if (stack[i] >= count)
    {
        flag = 1;
        break;
    }
    count += stack[i] + 1;
}
if (flag)
    continue;

//输出记录结果值
for (i = 0; i < m; i++)
{
    if (i <= top)
        printf("%d,", stack[i] + 1);
    else
        printf("1,");
}
printf("\n");
}

while (stack[0] > 1);

free(stack);
}

```

存在的问题：

1、程序我没有进行严格的测试，所以不能保证中间没有问题，而且不少地方都可以再优化，中间有些部分处理得不是很好，有时间我再好好改进一下。

2、有些情况还可以特殊处理一下，例如 $M>N/2$ 时，似乎满足条件一的所有组合都是满足条件二的；当 $N=(2 \text{ 的 } n \text{ 次方}-1)$, $M=n$ 时，结果只有一个，就是 1、2、4、...(2 的 $n-1$ 次方)，应该可以根据这个对其它结果进行推导。

3、这种方法是先根据条件一得到所有可能性，然后在这个结果集中去除不符合条件二的，感觉效率不是很好。个人觉得应该有办法可以直接把两个条件一起考虑，靠某种方式主动推出结果，而不是我现在采用的被动筛选方式。其实我刚开始就是想采用这种方式，但得到的结果集中总是缺少一些了排列可能。

思路二、以下是晖的个人思路：

qq675927952

N个鸡蛋分到M个篮子里($N > M$),不能有空篮子,对于任意不大于N的数,保证有几个篮子的鸡蛋数和等于此数,编程实现输入N,M两个数,输出所有鸡蛋的方法。

1、全输出的话本质就是搜索+剪枝。

2、 (n, m, min) 表示当前状态,按照篮子里蛋的数目从小到大搜索。搜到了第m个篮子,
1..m个篮子面共放了n个蛋,当前的篮子放了min个蛋。下一个扩展 $(n+t, m+1, t)$,for
 $t=min...n+1$ 。当 $n+(M-m)*min>N$ (鸡蛋不够时)或者 $2^M(M-m)*n+2^M(M-m)-1 < N$ (鸡蛋太多)
时 把这个枝剪掉…… ;

3、太多时的情况如下: n,n+1,2n+2,4n+4,8n+8....。代码如下:

```
//copyright@晖
//updated:
//听从网友 yingmg 的建议,再放到编译器上,调整下了缩进。
#include <iostream>
using namespace std;
long pow2[20];
int N,M;
int ans[1000];
void solve( int n , int m , int Min )
{
    if(n == N && m == M)
    {
        for(int i=1;i<=M;i++)
        {
            cout<<ans[i]<<" ";
        }
        cout<<endl;
        return ;
    }
    else if( n + (M-m)*Min > N || N > pow2[M-m]*n + pow2[M-m]-1)
        return ;
    else
    {
        for(int i = Min; i <= n+1; i++)
        {
            ans[m+1] = i;
            solve(n+i,m+1,i);
        }
    }
}
```

```

int main()
{
    pow2[0] = 1;
    for(int i=1;i<20;i++)
    {
        pow2[i] = pow2[i-1]<<1;
    }
    cin>>N>>M;
    if( M > N || pow2[M]-1 < N)
    {
        cout<<"没有这样的组合"<<endl;
    }
    solve( 0 , 0 , 1 );
    system("pause");
    return 0;
}

```

此思路二来自：<http://blog.csdn.net/qq675927952/archive/2011/03/30/6290131.aspx>。

65、华为面试

qq5823996

char *str = "AbcABca";

写出一个函数，查找出每个字符的个数，区分大小写，要求时间复杂度是 n （提示用 ASCII 码）

很基础，也比较简单的一题，看下这位网友给的代码吧：

nlqllove:

```

#include <stdio.h>

int main(int argc, char** argv)
{
    char *str = "AbcABca";
    int count[256] = {0};

    for (char *p=str; *p; p++)
    {
        count[*p]++;
    }

    // print
    for (int i=0; i<256; i++)

```

```

{
    if (count[i] > 0) //有个数大于零的，就打印出来
    {
        printf("The count of %c is: %d\n", i, count[i]);
    }
}
return 0;
}

```

66、微软面试题

yaoha2003

请把一个整形数组中重复的数字去掉。例如：

1, 2, 0, 2, -1, 999, 3, 999, 88

答案应该是：

1, 2, 0, -1, 999, 3, 88

67、请编程实现全排列算法。

全排列算法有两个比较常见的实现：递归排列和字典序排列。

yysdsyl:

(1) 递归实现

从集合中依次选出每一个元素，作为排列的第一个元素，然后对剩余的元素进行全排列，如此递归处理，从而

得到所有元素的全排列。算法实现如下：

```

#include <iostream>
#include <algorithm>
using namespace std;

template <typename T>
void CalcAllPermutation_R(T perm[], int first, int num)
{
    if (num <= 1) {
        return;
    }

    for (int i = first; i < first + num; ++i) {
        swap(perm[i], perm[first]);
        CalcAllPermutation_R(perm, first + 1, num - 1);
    }
}

```

```

        swap(perm[i], perm[first]);
    }
}

int main()
{
    const int NUM = 12;
    char perm[NUM];

    for (int i = 0; i < NUM; ++i)
        perm[i] = 'a' + i;

    CalcAllPermutation_R(perm, 0, NUM);
}

```

程序运行结果（优化）：

```
-bash-3.2$ g++ test.cpp -O3 -o ttt
```

```
-bash-3.2$ time ./ttt
```

```
real    0m10.556s
```

```
user    0m10.551s
```

```
sys     0m0.000s
```

程序运行结果（不优化）：

```
-bash-3.2$ g++ test.cpp -o ttt
```

```
-bash-3.2$ time ./ttt
```

```
real    0m21.355s
```

```
user    0m21.332s
```

```
sys     0m0.004s
```

（2）字典序排列

把升序的排列（当然，也可以实现为降序）作为当前排列开始，然后依次计算当前排列的下一个字典序排列。

对当前排列从后向前扫描，找到一对为升序的相邻元素，记为 i 和 j ($i < j$)。如果不存在这样一对为升序的相邻元素，则所有排列均已找到，算法结束；否则，重新对当前排列从后向前扫描，找到第一个大于 i 的元素 k ，交换 i 和 k ，然后对从 j 开始到结束的子序列反转，则此时得到的新排列就为下一个字典序排列。这种方式实现得到的所有排列是按字典序有序的，这也是 C++ STL 算法 `next_permutation` 的思想。算法实现如下：

```
#include <iostream>
```

```

#include <algorithm>
using namespace std;

template <typename T>
void CalcAllPermutation(T perm[], int num)
{
    if (num < 1)
        return;

    while (true) {
        int i;
        for (i = num - 2; i >= 0; --i) {
            if (perm[i] < perm[i + 1])
                break;
        }

        if (i < 0)
            break; // 已经找到所有排列

        int k;
        for (k = num - 1; k > i; --k) {
            if (perm[k] > perm[i])
                break;
        }

        swap(perm[i], perm[k]);
        reverse(perm + i + 1, perm + num);
    }
}

int main()
{
    const int NUM = 12;
    char perm[NUM];

    for (int i = 0; i < NUM; ++i)
        perm[i] = 'a' + i;

    CalcAllPermutation(perm, NUM);
}

```

程序运行结果（优化）：

```

-bash-3.2$ g++ test.cpp -O3 -o ttt
-bash-3.2$ time ./ttt

```

```
real    0m3.055s
user    0m3.044s
sys     0m0.001s
```

程序运行结果（不优化）：

```
-bash-3.2$ g++ test.cpp -o ttt
-bash-3.2$ time ./ttt
```

```
real    0m24.367s
user    0m24.321s
sys     0m0.006s
```

使用 `std::next_permutation` 来进行对比测试，代码如下：

```
#include <iostream>
#include <algorithm>
using namespace std;

template <typename T>
size_t CalcAllPermutation(T perm[], int num)
{
    if (num < 1)
        return 0;

    size_t count = 0;
    while (next_permutation(perm, perm + num)) {
        ++count;
    }

    return count;
}

int main()
{
    const int NUM = 12;
    char perm[NUM];

    for (int i = 0; i < NUM; ++i)
        perm[i] = 'a' + i;

    size_t count = CalcAllPermutation(perm, NUM);

    return count;
}
```

```
}
```

程序运行结果（优化）：

```
-bash-3.2$ g++ test.cpp -O3 -o ttt
```

```
-bash-3.2$ time ./ttt
```

```
real    0m3.606s  
user    0m3.601s  
sys     0m0.002s
```

程序运行结果（不优化）：

```
-bash-3.2$ g++ test.cpp -o ttt
```

```
-bash-3.2$ time ./ttt
```

```
real    0m33.850s  
user    0m33.821s  
sys     0m0.006s
```

测试结果汇总一（优化）：

（1）递归实现：0m10.556s

（2-1）字典序实现：0m3.055s

（2-2）字典序 next_permutation：0m3.606s

测试结果汇总二（不优化）：

（1）递归实现：0m21.355s

（2-1）字典序实现：0m24.367s

（2-2）字典序 next_permutation：0m33.850s

由测试结果可知，自己实现的字典序比 `next_permutation` 稍微快点，原因可能是 `next_permutation` 版本有额外的函数调用开销；而归实现版本在优化情况下要慢很多，主要原因可能在于太多的函数调用开销，但在不优化情况下执行比其它二个版本要快，原因可能在于程序结构更简单，执行的语句较少。

比较而言，递归算法结构简单，适用于全部计算出所有的排列（因此排列规模不能太大，计算机资源会成为限制）；而字典序排列逐个产生、处理排列，能够适用于大的排列空间，并且它产生的排列的规律性很强。

68、阿里巴巴三道面试题

fenglibing

- 1、澳大利亚的父母喜欢女孩，如果生出来的第一个女孩，就不再生了，如果是男孩就继续生，直到生到第一个女孩为止，问若干年后，男女的比例是多少？
- 2、3点15的时针和分针的夹角是多少度
- 3、有8瓶水，其中有一瓶有毒，最少尝试几次可以找出来

69、阿里巴巴 2011 实习生笔试题

给一篇文章，里面是由一个个单词组成，单词中间空格隔开，再给一个字符串指针数组，比如 `char *str[]={"hello","world","good"};`
求文章中包含这个字符串指针数组的最小子串。注意，只要包含即可，没有顺序要求。

分析：文章也可以理解为一个大的字符串数组，单词之前只有空格，没有标点符号。

我最开始想到的思路，是：

维护一个队列+KMP 算法

让字符的全部序列入队，比较完一个就出队，保持长度

至于字符串的六种序列，实现排列预处理，

最后，时间复杂度为： O （字符事先排列） $+O$ （KMP 比较）。

后来，本 BLOG 算法交流群内有人提出：

Sur 鱼：

这个用 kmp 算法的话，明显不如用 trie 好；

将 str 中的成员建一棵 trie 树，这样的话字符事先不需要排序，复杂度应该低些。

梦想天窗：

我觉得这个应该用 DFA（即有限状态自动机）。

70、Google 算法笔试题

有一台机器，上面有 m 个储存空间。然后有 n 个请求，第 i 个请求计算时需要占 $R[i]$ 个空间，储存计算结果则需要占据 $O[i]$ 个空间（据 $O[i]$ 个空间（其中 $O[i] < R[i]$ ）。问怎么安排这 n 个请求的顺序，使得所有请求都能完成。你的算法也应该能够判断出无论如何都不能处理完的情况。比方说， $m=14$, $n=2$, $R[1]=10$, $O[1]=5$, $R[2]=8$, $O[2]=6$ 。在这个例子中，我们可以先运行第一个任务，剩余 9 个单位的空间足够执行第二个任务；但如果先走第二个任务，第一个任务执行时空间就不够了，因为 $10 > 14 - 6$ 。

matrix67：

当时花了全部的时间去想各种网络流、费用流、图的分层思想等等，最后写了一个铁定错误的贪心上去。直到考试结束 4 个小时以后我才想到了正确的算法——只需要按照 R 值和 O

值之差（即释放空间的大小）从大到小排序，然后依次做就是了……

Z.Hao:

此算法题曾是交大 09 年招保研生的复试题。Matrix67 给出的算法是不完整的。

某日阳光明媚下午曾和 petercai 共同商讨过，应该是先对驻留内存进行排序，选择驻留内存最小的里面可以在当前内存中运行且（运行内存-驻留内存）最小的进行调度。但是这种算法显然仍然仅仅不够..此题目前还有容考虑。

若各位想到更好的思路，或者以上任何一题的思路或答案有任何问题，欢迎不吝指正。完。

updated:

本文评论中，[qiquanchang](#)、[hellorld](#) 倆位网友指出：此第七十题是死锁检测算法，银行家算法。

非常感谢，俩位的指导。多谢。

update again:

如果你对以上任何一代的思路，有任何问题，欢迎在留言或评论中告知。如果您对以上任何一题，有更好的代码或思路，欢迎发到我的第二个邮箱，786165179@qq.com。若经采纳，将更新到本文中，非常感谢。

July、2011..4.17。

版权声明：转载本 BLOG 内任何一篇文章，必须以超链接形式注明出处。

海量数据处理：十道面试题与十个海量数据处理方法总结

作者：July、youwang、yanxionglu。

时间：二零一一年三月二十六日

本文之总结：教你如何迅速秒杀掉：99%的海量数据处理面试题。有任何问题，欢迎随时交流、指正。

出处：http://blog.csdn.net/v_JULY_v。

第一部分、十道海量数据处理面试题

1、海量日志数据，提取出某日访问百度次数最多的那个 IP。

首先是这一天，并且是访问百度的日志中的 IP 取出来，逐个写入到一个大文件中。注意到 IP 是 32 位的，最多有个 2^{32} 个 IP。同样可以采用映射的方法，比如模 1000，把整个大文件映射为 1000 个小文件，再找出每个小文中出现频率最大的 IP(可以采用 hash_map 进行频率统计，然后再找出频率最大的几个) 及相应的频率。然后再在这 1000 个最大的 IP 中，找出那个频率最大的 IP，即为所求。

或者如下阐述（雪域之鹰）：

算法思想：分而治之 +Hash

1. IP 地址最多有 $2^{32}=4G$ 种取值情况，所以不能完全加载到内存中处理；
2. 可以考虑采用“分而治之”的思想，按照 IP 地址的 $\text{Hash(IP)} \% 1024$ 值，把海量 IP 日志分别存储到 1024 个小文件中。这样，每个小文件最多包含 4MB 个 IP 地址；
3. 对于每一个小文件，可以构建一个 IP 为 key，出现次数为 value 的 Hash map，同时记录当前出现次数最多的那个 IP 地址；
4. 可以得到 1024 个小文件中的出现次数最多的 IP，再依据常规的排序算法得到总体上出现次数最多的 IP；

2、搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为 1-255 字节。

假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门。），请你统计最热门的 10 个查询串，要求使用的内存不能超过 1G。

典型的 Top K 算法，还是在这篇文章里头有所阐述，详情请参见：[十一、从头到尾彻底解析 Hash 表算法](#)。

文中，给出的最终算法是：

第一步、先对这批海量数据预处理，在 $O(N)$ 的时间内用 Hash 表完成统计（之前写成了排序，特此订正。July、2011.04.27）；

第二步、借助堆这个数据结构，找出 Top K，时间复杂度为 $N \log K$ 。

即，借助堆结构，我们可以在 \log 量级的时间内查找和调整/移动。因此，维护一个 K （该题目中是 10）大小的小根堆，然后遍历 300 万的 Query，分别和根元素进行对比所以，我们最终的时间复杂度是： $O(N) + N * O(\log K)$ ， $(N$ 为 1000 万， N' 为 300 万)。ok，更多，详情，请参考原文。

或者：采用 trie 树，关键字域存该查询串出现的次数，没有出现为 0。最后用 10 个元素的最小堆来对出现频率进行排序。

3、有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 字节，内存限制大小是 1M。返回频数最高的 100 个词。

方案：顺序读文件中，对于每个词 x ，取 $\text{hash}(x) \% 5000$ ，然后按照该值存到 5000 个小文件（记为 $x_0, x_1, \dots, x_{4999}$ ）中。这样每个文件大概是 200k 左右。

如果其中的有的文件超过了 1M 大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过 1M。

对每个小文件，统计每个文件中出现的词以及相应的频率（可以采用 trie 树/hash_map 等），并取出出现频率最大的 100 个词（可以用含 100 个结点的最小堆），并把 100 个词及相应的频率存入文件，这样又得到了 5000 个文件。下一步就是把这 5000 个文件进行归并（类似与归并排序）的过程了。

4、有 10 个文件，每个文件 1G，每个文件的每一行存放的都是用户的 query，每个文件的 query 都可能重复。要求你按照 query 的频度排序。

还是典型的 TOP K 算法，解决方案如下：

方案 1：

顺序读取 10 个文件，按照 $\text{hash(query)} \% 10$ 的结果将 query 写入到另外 10 个文件（记为）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。

找一台内存 2G 左右的机器，依次对用 $\text{hash_map(query, query_count)}$ 来统计每个 query 出现的次数。利用快速/堆/归并排序按照出现次数进行排序。将排序好的 query 和对

应的 `query_cout` 输出到文件中。这样得到了 10 个排好序的文件（记为）。

对这 10 个文件进行归并排序（内排序与外排序相结合）。

方案 2：

一般 `query` 的总量是有限的，只是重复的次数比较多而已，可能对于所有的 `query`，一次性就可以加入到内存了。这样，我们就可以采用 `trie` 树/`hash_map` 等直接来统计每个 `query` 出现的次数，然后按出现次数做快速/堆/归并排序就可以了。

方案 3：

与方案 1 类似，但在做完 `hash`，分成多个文件后，可以交给多个文件来处理，采用分布式的架构来处理（比如 `MapReduce`），最后再进行合并。

5、给定 `a`、`b` 两个文件，各存放 50 亿个 `url`，每个 `url` 各占 64 字节，内存限制是 4G，让你找出 `a`、`b` 文件共同的 `url`？

方案 1：可以估计每个文件的大小为 $5G \times 64 = 320G$ ，远远大于内存限制的 4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

遍历文件 `a`，对每个 `url` 求取 `hash(url) % 1000`，然后根据所取得的值将 `url` 分别存储到 1000 个小文件（记为 `a0, a1, ..., a999`）中。这样每个小文件的大约 300M。

遍历文件 `b`，采取和 `a` 相同的方式将 `url` 分别存储到 1000 小文件（记为 `b0, b1, ..., b999`）。这样处理后，所有可能相同的 `url` 都在对应的小文件（`a0vsb0, a1vsb1, ..., a999vsb999`）中，不对应的小文件不可能有相同的 `url`。然后我们只要求出 1000 对小文件中相同的 `url` 即可。

求每对小文件中相同的 `url` 时，可以把其中一个小文件的 `url` 存储到 `hash_set` 中。然后遍历另一个小文件的每个 `url`，看其是否在刚才构建的 `hash_set` 中，如果是，那么就是共同的 `url`，存到文件里面就可以了。

方案 2：如果允许有一定的错误率，可以使用 `Bloom filter`，4G 内存大概可以表示 340 亿 `bit`。将其中一个文件中的 `url` 使用 `Bloom filter` 映射为这 340 亿 `bit`，然后挨个读取另外一个文件的 `url`，检查是否与 `Bloom filter`，如果是，那么该 `url` 应该是共同的 `url`（注意会有一定的错误率）。

`Bloom filter` 日后会在本 BLOG 内详细阐述。

6、在 2.5 亿个整数中找出不重复的整数，注，内存不足以容纳这 2.5 亿个整数。

方案 1：采用 2-Bitmap（每个数分配 2bit，00 表示不存在，01 表示出现一次，10 表示多次，11 无意义）进行，共需内存 $2^{32} \times 2 \text{ bit} = 1 \text{ GB}$ 内存，还可以接受。然后扫描这 2.5

亿个整数，查看 Bitmap 中相对应位，如果是 00 变 01，01 变 10，10 保持不变。所描完事
后，查看 bitmap，把对应位是 01 的整数输出即可。

方案 2：也可采用与第 1 题类似的方法，进行划分小文件的方法。然后在小文件中找出
不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

**7、腾讯面试题：给 40 亿个不重复的 unsigned int 的整数，没排过序的，然后再给一个数，
如何快速判断这个数是否在那 40 亿个数当中？**

与上第 6 题类似，我的第一反应时快速排序+二分查找。以下是其它更好的方法：

方案 1：oo，申请 512M 的内存，一个 bit 位代表一个 unsigned int 值。读入 40 亿个数，
设置相应的 bit 位，读入要查询的数，查看相应 bit 位是否为 1，为 1 表示存在，为 0 表示不
存在。

dizengrong：

方案 2：这个问题在《编程珠玑》里有很好的描述，大家可以参考下面的思路，探讨一
下：

又因为 2^{32} 为 40 亿多，所以给定一个数可能在，也可能不在其中；

这里我们把 40 亿个数中的每一个用 32 位的二进制来表示

假设这 40 亿个数开始放在一个文件中。

然后将这 40 亿个数分成两类：

1. 最高位为 0
2. 最高位为 1

并将这两类分别写入到两个文件中，其中一个文件中数的个数 ≤ 20 亿，而另一个 ≥ 20
亿（这相当于折半了）；

与要查找的数的最高位比较并接着进入相应的文件再查找

再然后把这个文件又分成两类：

1. 次最高位为 0
2. 次最高位为 1

并将这两类分别写入到两个文件中，其中一个文件中数的个数 ≤ 10 亿，而另一个 ≥ 10
亿（这相当于折半了）；

与要查找的数的次最高位比较并接着进入相应的文件再查找。

.....

以此类推，就可以找到了，而且时间复杂度为 $O(\log n)$ ，方案 2 完。

附：这里，再简单介绍下，位图方法：

使用位图法判断整形数组是否存在重复

判断集合中存在重复是常见编程任务之一，当集合中数据量比较大时我们通常希望少进行几次扫描，这时双重循环法就不可取了。

位图法比较适合于这种情况，它的做法是按照集合中最大元素 `max` 创建一个长度为 `max+1` 的新数组，然后再次扫描原数组，遇到几就给新数组的第几位置上 1，如遇到 5 就给新数组的第六个元素置 1，这样下次再遇到 5 想置位时发现新数组的第六个元素已经是 1 了，这说明这次的数据肯定和以前的数据存在着重复。这种给新数组初始化时置零其后置一的做法类似于位图的处理方法故称位图法。它的运算次数最坏的情况为 $2N$ 。如果已知数组的最大值即能事先给新数组定长的话效率还能提高一倍。

欢迎，有更好的思路，或方法，共同交流。

8、怎么在海量数据中找出重复次数最多的一个？

方案 1：先做 `hash`，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）。

9、上千万或上亿数据（有重复），统计其中出现次数最多的 N 个数据。

方案 1：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用 `hash_map`/搜索二叉树/红黑树等来进行统计次数。然后就是取出前 N 个出现次数最多的数据了，可以用第 2 题提到的堆机制完成。

10、一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词，请给出思想，给出时间复杂度分析。

方案 1：这题是考虑时间效率。用 `trie` 树统计每个词出现的次数，时间复杂度是 $O(n*le)$ (le 表示单词的平均长度)。然后是找出出现最频繁的前 10 个词，可以用堆来实现，前面的题中已经讲到了，时间复杂度是 $O(n*lg10)$ 。所以总的时间复杂度，是 $O(n*le)$ 与 $O(n*lg10)$ 中较大的哪一个。

附、100w 个数中找出最大的 100 个数。

方案 1：在前面的题中，我们已经提到了，用一个含 100 个元素的最小堆完成。复杂度为 $O(100w*lg100)$ 。

方案 2：采用快速排序的思想，每次分割之后只考虑比轴大的一部分，知道比轴大的一部分在比 100 多的时候，采用传统排序算法排序，取前 100 个。复杂度为 $O(100w*100)$ 。

方案 3：采用局部淘汰法。选取前 100 个元素，并排序，记为序列 L。然后一次扫描剩余的元素 x，与排好序的 100 个元素中最小的元素比，如果比这个最小的要大，那么把这个最小的元素删除，并把 x 利用插入排序的思想，插入到序列 L 中。依次循环，知道扫描了所有的元素。复杂度为 $O(100w^*100)$ 。

致谢：<http://www.cnblogs.com/youwang/>。

第二部分、十个海量数据处理方法大总结

ok，看了上面这么多的面试题，是否有点头昏。是的，需要一个总结。接下来，本文将简单总结下一些处理海量数据问题的常见方法，**而日后，本 BLOG 内会具体阐述这些方法。**

下面的方法全部来自 <http://hi.baidu.com/yanxionglu/blog>/博客，对海量数据的处理方法进行了一个一般性的总结，当然这些方法可能并不能完全覆盖所有的问题，但是这样的一些方法也基本可以处理绝大多数遇到的问题。下面的一些问题基本直接来源于公司的面试笔试题目，方法不一定最优，如果你有更好的处理方法，欢迎讨论。

一、Bloom filter

适用范围：可以用来实现数据字典，进行数据的判重，或者集合求交集

基本原理及要点：

对于原理来说很简单，位数组+k 个独立 hash 函数。将 hash 函数对应的值的位数组置 1，查找时如果发现所有 hash 函数对应位都是 1 说明存在，很明显这个过程并不保证查找的结果是 100% 正确的。同时也不支持删除一个已经插入的关键字，因为该关键字对应的位会牵动到其他的关键字。所以一个简单的改进就是 counting Bloom filter，用一个 counter 数组代替位数组，就可以支持删除了。

还有一个比较重要的问题，如何根据输入元素个数 n，确定位数组 m 的大小及 hash 函数个数。当 hash 函数个数 $k=(\ln 2)^*(m/n)$ 时错误率最小。在错误率不大于 E 的情况下，m 至少要等于 $n \cdot \lg(1/E)$ 才能表示任意 n 个元素的集合。但 m 还应该更大些，因为还要保证 bit 数组里至少一半为 0，则 m 应该 $\geq n \cdot \lg(1/E) \cdot \lg e$ 大概就是 $n \cdot \lg(1/E) \cdot 1.44$ 倍 (\lg 表示以 2 为底的对数)。

举个例子我们假设错误率为 0.01，则此时 m 应大概是 n 的 13 倍。这样 k 大概是 8 个。

注意这里 m 与 n 的单位不同，m 是 bit 为单位，而 n 则是以元素个数为单位(准确的是不同元素的个数)。通常单个元素的长度都是有很多 bit 的。所以使用 bloom filter 内存上通常都是节省的。

扩展：

Bloom filter 将集合中的元素映射到位数组中，用 k (k 为哈希函数个数) 个映射位是否全 1 表示元素在不在这个集合中。**Counting bloom filter (CBF)** 将位数组中的每一位扩展为一个 **counter**，从而支持了元素的删除操作。**Spectral Bloom Filter (SBF)** 将其与集合元素的出现次数关联。**SBF** 采用 **counter** 中的最小值来近似表示元素的出现频率。

问题实例：给你 A,B 两个文件，各存放 50 亿条 URL，每条 URL 占用 64 字节，内存限制是 4G，让你找出 A,B 文件共同的 URL。如果是三个乃至 n 个文件呢？

根据这个问题我们来计算下内存的占用， $4G=2^{32}$ 大概是 40 亿 * 8 大概是 340 亿， $n=50$ 亿，如果按出错率 0.01 算需要的大概是 650 亿个 bit。现在可用的是 340 亿，相差并不多，这样可能会使出错率上升些。另外如果这些 urlip 是一一对应的，就可以转换成 ip，则大大简单了。

二、Hashing

适用范围：快速查找，删除的基本数据结构，通常需要总数据量可以放入内存

基本原理及要点：

hash 函数选择，针对字符串，整数，排列，具体相应的 **hash** 方法。

碰撞处理，一种是 **open hashing**，也称为拉链法；另一种就是 **closed hashing**，也称开地址法，**opened addressing**。

扩展：

d-left hashing 中的 d 是多个的意思，我们先简化这个问题，看一看 **2-left hashing**。**2-left hashing** 指的是将一个哈希表分成长度相等的两半，分别叫做 T_1 和 T_2 ，给 T_1 和 T_2 分别配备一个哈希函数， h_1 和 h_2 。在存储一个新的 **key** 时，同时用两个哈希函数进行计算，得出两个地址 $h_1[key]$ 和 $h_2[key]$ 。这时需要检查 T_1 中的 $h_1[key]$ 位置和 T_2 中的 $h_2[key]$ 位置，哪一个位置已经存储的（有碰撞的）**key** 比较多，然后将新 **key** 存储在负载少的位置。如果两边一样多，比如两个位置都为空或者都存储了一个 **key**，就把新 **key** 存储在左边的 T_1 子表中，**2-left** 也由此而来。在查找一个 **key** 时，必须进行两次 **hash**，同时查找两个位置。

问题实例：

1). 海量日志数据，提取出某日访问百度次数最多的那个 IP。

IP 的数目还是有限的，最多 2^{32} 个，所以可以考虑使用 **hash** 将 ip 直接存入内存，然后进行统计。

三、bit-map

适用范围：可进行数据的快速查找，判断，删除，一般来说数据范围是 int 的 10 倍以下

基本原理及要点：使用 bit 数组来表示某些元素是否存在，比如 8 位电话号码

扩展：bloom filter 可以看做是对 bit-map 的扩展

问题实例：

1)已知某个文件内包含一些电话号码，每个号码为 8 位数字，统计不同号码的个数。

8 位最多 99 999 999，大概需要 $99m$ 个 bit，大概 10 几 m 字节的内存即可。

2)2.5 亿个整数中找出不重复的整数的个数，内存空间不足以容纳这 2.5 亿个整数。

将 bit-map 扩展一下，用 2bit 表示一个数即可，0 表示未出现，1 表示出现一次，2 表示出现 2 次及以上。或者我们不用 2bit 来进行表示，我们用两个 bit-map 即可模拟实现这个 2bit-map。

四、堆

适用范围：海量数据前 n 大，并且 n 比较小，堆可以放入内存

基本原理及要点：最大堆求前 n 小，最小堆求前 n 大。方法，比如求前 n 小，我们比较当前元素与最大堆里的最大元素，如果它小于最大元素，则应该替换那个最大元素。这样最后得到的 n 个元素就是最小的 n 个。适合大数据量，求前 n 小，n 的大小比较小的情况，这样可以扫描一遍即可得到所有的前 n 元素，效率很高。

扩展：双堆，一个最大堆与一个最小堆结合，可以用来维护中位数。

问题实例：

1)100w 个数中找最大的前 100 个数。

用一个 100 个元素大小的最小堆即可。

五、双层桶划分----其实本质上就是【分而治之】的思想，重在“分”的技巧上！

适用范围：第 k 大，中位数，不重复或重复的数字

基本原理及要点：因为元素范围很大，不能利用直接寻址表，所以通过多次划分，逐步确定范围，然后最后在一个可以接受的范围内进行。可以通过多次缩小，双层只是一个例子。

扩展：

问题实例：

1).2.5 亿个整数中找出不重复的整数的个数，内存空间不足以容纳这 2.5 亿个整数。

有点像鸽巢原理，整数个数为 2^{32} ，也就是，我们可以将这 2^{32} 个数，划分为 2^8 个区域（比如用单个文件代表一个区域），然后将数据分离到不同的区域，然后不同的区域在利用 bitmap 就可以直接解决了。也就是说只要有足够的磁盘空间，就可以很方便的解决。

2).5 亿个 int 找它们的中位数。

这个例子比上面那个更明显。首先我们将 int 划分为 2^{16} 个区域，然后读取数据统计落到各个区域里的数的个数，之后我们根据统计结果就可以判断中位数落到那个区域，同时知道这个区域中的第几大数刚好是中位数。然后第二次扫描我们只统计落在这个区域中的那些数就可以了。

实际上，如果不是 int 是 int64，我们可以经过 3 次这样的划分即可降低到可以接受的程度。即可以先将 int64 分成 2^{24} 个区域，然后确定区域的第几大数，在将该区域分成 2^{20} 个子区域，然后确定是子区域的第几大数，然后子区域里的数的个数只有 2^{20} ，就可以直接利用 direct addr table 进行统计了。

六、数据库索引

适用范围：大数据量的增删改查

基本原理及要点：利用数据的设计实现方法，对海量数据的增删改查进行处理。

七、倒排索引(Inveted index)

适用范围：搜索引擎，关键字查询

基本原理及要点：为何叫倒排索引？一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。

以英文为例，下面是要被索引的文本：

T0 = "it is what it is"

T1 = "what is it"

T2 = "it is a banana"

我们就能得到下面的反向文件索引：

"a": {2}

"banana": {2}

"is": {0, 1, 2}

"it": {0, 1, 2}

"what": {0, 1}

检索的条件"what", "is"和"it"将对应集合的交集。

正向索引开发出来用来存储每个文档的单词的列表。正向索引的查询往往满足每个文档有序频繁的全文查询和每个单词在校验文档中的验证这样的查询。在正向索引中，文档占据了中心的位置，每个文档指向了一个它所包含的索引项的序列。也就是说文档指向了它包含的那些单词，而反向索引则是单词指向了包含它的文档，很容易看到这个反向的关系。

扩展：

问题实例：文档检索系统，查询那些文件包含了某单词，比如常见的学术论文的关键字搜索。

八、外排序

适用范围：大数据的排序，去重

基本原理及要点：外排序的归并方法，置换选择败者树原理，最优归并树

扩展：

问题实例：

1).有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 个字节，内存限制大小是 1M。返回频数最高的 100 个词。

这个数据具有很明显的特点，词的大小为 16 个字节，但是内存只有 1m 做 hash 有些不够，所以可以用来排序。内存可以当输入缓冲区使用。

九、trie 树

适用范围：数据量大，重复多，但是数据种类小可以放入内存

基本原理及要点：实现方式，节点孩子的表示方式

扩展：压缩实现。

问题实例：

1).有 10 个文件，每个文件 1G，每个文件的每一行都存放的是用户的 query，每个文件的 query 都可能重复。要你按照 query 的频度排序。

2).1000 万字符串，其中有些是相同的(重复)，需要把重复的全部去掉，保留没有重复的

字符串。请问怎么设计和实现？

3).寻找热门查询：查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个，每个不超过 255 字节。

十、分布式处理 mapreduce

适用范围：数据量大，但是数据种类小可以放入内存

基本原理及要点：将数据交给不同的机器去处理，数据划分，结果归约。

扩展：

问题实例：

1).The canonical example application of MapReduce is a process to count the appearances of each different word in a set of documents:

2).海量数据分布在 100 台电脑中，想个办法高效统计出这批数据的 TOP10。

3).一共有 N 个机器，每个机器上有 N 个数。每个机器最多存 $O(N)$ 个数并对它们操作。
如何找到 $N/2$ 个数的中数(median)？

经典问题分析

上千万 or 亿数据（有重复），统计其中出现次数最多的前 N 个数据，分两种情况：可一次读入内存，不可一次读入。

可用思路：trie 树+堆，数据库索引，划分子集分别统计，hash，分布式计算，近似统计，外排序

所谓的是否能一次读入内存，实际上应该指去除重复后的数据量。如果去重后数据可以放入内存，我们可以为数据建立字典，比如通过 map，hashmap，trie，然后直接进行统计即可。当然在更新每条数据的出现次数的时候，我们可以利用一个堆来维护出现次数最多的前 N 个数据，当然这样导致维护次数增加，不如完全统计后在求前 N 大效率高。

如果数据无法放入内存。一方面我们可以考虑上面的字典方法能否被改进以适应这种情形，可以做的改变就是将字典存放到硬盘上，而不是内存，这可以参考数据库的存储方法。

当然还有更好的方法，就是可以采用分布式计算，基本上就是 map-reduce 过程，首先可以根据数据值或者把数据 hash(md5)后的值，将数据按照范围划分到不同的机子，最好可以让数据划分后可以一次读入内存，这样不同的机子负责处理各种的数值范围，实际上就是 map。得到结果后，各个机子只需拿出各自的出现次数最多的前 N 个数据，然后汇总，选出所有的数据中出现次数最多的前 N 个数据，这实际上就是 reduce 过程。

实际上可能想直接将数据均分到不同的机子上进行处理，这样是无法得到正确的解的。因为一个数据可能被均分到不同的机子上，而另一个则可能完全聚集到一个机子上，同时还可能存在具有相同数目的数据。比如我们要找出现次数最多的前 100 个，我们将 1000 万的数据分布到 10 台机器上，找到每台出现次数最多的前 100 个，归并之后这样不能保证找到真正的第 100 个，因为比如出现次数最多的第 100 个可能有 1 万个，但是它被分到了 10 台机子，这样在每台上只有 1 千个，假设这些机子排名在 1000 个之前的那些都是单独分布在一台机子上的，比如有 1001 个，这样本来具有 1 万个的这个就会被淘汰，即使我们让每台机子选出出现次数最多的 1000 个再归并，仍然会出错，因为可能存在大量个数为 1001 个的发生聚集。因此不能将数据随便均分到不同机子上，而是要根据 `hash` 后的值将它们映射到不同的机子上处理，让不同的机器处理一个数值范围。

而外排序的方法会消耗大量的 `IO`，效率不会很高。而上面的分布式方法，也可以用于单机版本，也就是将总的数据根据值的范围，划分成多个不同的子文件，然后逐个处理。处理完毕之后再对这些单词的及其出现频率进行一个归并。实际上就可以利用一个外排序的归并过程。

另外还可以考虑近似计算，也就是我们可以通过结合自然语言属性，只将那些真正实际中出现最多的那些词作为一个字典，使得这个规模可以放入内存。

ok，更多请参见本文总结：[教你如何迅速秒杀掉：99%的海量数据处理面试题](#)。以上有任何问题，欢迎指正。谢谢大家。

版权所有。转载本 **BLOG** 内任何文章，请以超链接形式注明出处。

海量数据处理面试题与 Bit-map 详解

作者：小桥流水，redfox66，July。

前言

本博客内曾经整理过有关海量数据处理的 10 道面试题（[十道海量数据处理面试题与十个方法大总结](#)），此次除了重复了之前的 10 道面试题之后，重新多整理了 7 道。仅作各位参考，不作它用。

同时，[程序员编程艺术系列](#)将重新开始创作，第十一章以后的部分题目来源将取自下文中的 17 道海量数据处理的面试题。因为，我们觉得，下文的每一道面试题都值得重新思考，重新深究与学习。再者，编程艺术系列的前十章也是这么来的。若您有任何问题或建议，欢迎不吝指正。谢谢。

第一部分、十五道海量数据处理面试题

1. 给定 a、b 两个文件，各存放 50 亿个 url，每个 url 各占 64 字节，内存限制是 4G，让你找出 a、b 文件共同的 url？

方案 1：可以估计每个文件安的大小为 $50G \times 64 = 320G$ ，远远大于内存限制的 4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

1. 遍历文件 a，对每个 url 求取 $hash(url) \% 1000$ ，然后根据所取得的值将 url 分别存储到 1000 个小文件（记为 a_0, a_1, \dots, a_{999} ）中。这样每个小文件的大约为 300M。
2. 遍历文件 b，采取和 a 相同的方式将 url 分别存储到 1000 小文件中（记为 b_0, b_1, \dots, b_{999} ）。这样处理后，所有可能相同的 url 都在对应的小文件 ($a_0 vs b_0, a_1 vs b_1, \dots, a_{999} vs b_{999}$) 中，不对应的小文件不可能有相同的 url。然后我们只要求出 1000 对小文件中相同的 url 即可。
3. 求每对小文件中相同的 url 时，可以把其中一个小文件的 url 存储到 hash_set 中。然后遍历另一个小文件的每个 url，看其是否在刚才构建的 hash_set 中，如果是，那么就是共同的 url，存到文件里面就可以了。

方案 2：如果允许有一定的错误率，可以使用 Bloom filter，4G 内存大概可以表示 340 亿 bit。将其中一个文件中的 url 使用 Bloom filter 映射为这 340 亿 bit，然后挨个读取另外一个文件的 url，检查是否与 Bloom filter，如果是，那么该 url 应该是共同的 url（注意会有一定的错误率）。

读者反馈@crowgns:

1. hash 后要判断每个文件大小，如果 hash 分的不均衡有文件较大，还应继续 hash 分文件，换个 hash 算法第二次再分较大的文件，一直分到没有较大的文件为止。这样文件标号可以用 A1-2 表示（第一次 hash 编号为 1，文件较大所以参加第二次 hash，编号为 2）
2. 由于 1 存在，第一次 hash 如果有大文件，不能用直接 set 的方法。建议对每个文件都先用字符串自然顺序排序，然后具有相同 hash 编号的（如都是 1-3，而不能 a 编号是 1，b 编号是 1-1 和 1-2），可以直接从头到尾比较一遍。对于层级不一致的，如 a1, b 有 1-1, 1-2-1, 1-2-2，层级浅的要和层级深的每个文件都比较一次，才能确认每个相同的 uri。

2. 有 10 个文件，每个文件 1G，每个文件的每一行存放的都是用户的 query，每个文件的 query 都可能重复。要求你按照 query 的频度排序。

方案 1：

1. 顺序读取 10 个文件，按照 $\text{hash}(\text{query}) \% 10$ 的结果将 query 写入到另外 10 个文件（记为 a_0, a_1, \dots, a_9 ）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。
2. 找一台内存存在 2G 左右的机器，依次对 a_0, a_1, \dots, a_9 用 $\text{hash_map}(\text{query}, \text{query_count})$ 来统计每个 query 出现的次数。利用快速/堆/归并排序按照出现次数进行排序。将排序好的 query 和对应的 query_count 输出到文件中。这样得到了 10 个排好序的文件（记为 b_0, b_1, \dots, b_9 ）。
3. 对 b_0, b_1, \dots, b_9 这 10 个文件进行归并排序（内排序与外排序相结合）。

方案 2：

一般 query 的总量是有限的，只是重复的次数比较多而已，可能对于所有的 query，一次性就可以加入到内存了。这样，我们就可以采用 trie 树/hash_map 等直接来统计每个 query 出现的次数，然后按出现次数做快速/堆/归并排序就可以了

（**读者反馈@店小二：**原文第二个例子中：“找一台内存存在 2G 左右的机器，依次对用 $\text{hash_map}(\text{query}, \text{query_count})$ 来统计每个 query 出现的次数。”由于 query 会重复，作为 key 的话，应该使用 hash_multimap 。hash map 不允许 key 重复。**@hywangw**:店小二所述的肯定是错的， $\text{hash_map}(\text{query}, \text{query_count})$ 是用来统计每个 query 的出现次数 又不是存储他们的值 出现一次 把 count+1 就行了 用 multimap 干什么？多谢 hywangw)。

方案 3：

与方案 1 类似，但在做完 hash，分成多个文件后，可以交给多个文件来处理，采用分

布式的架构来处理（比如 MapReduce），最后再进行合并。

3. 有一个 **1G** 大小的一个文件，里面每一行是一个词，词的大小不超过 **16** 字节，内存限制大小是 **1M**。返回频数最高的 **100** 个词。

方案 1：顺序读文件中，对于每个词 x ，取 $hash(x) \% 5000$ ，然后按照该值存到 5000 个小文件（记为 $x_0, x_1, \dots, x_{4999}$ ）中。这样每个文件大概是 200k 左右。如果其中的有的文件超过了 1M 大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过 1M。对每个小文件，统计每个文件中出现的词以及相应的频率（可以采用 trie 树 /hash_map 等），并取出出现频率最大的 100 个词（可以用含 100 个结点的最小堆），并把 100 词及相应的频率存入文件，这样又得到了 5000 个文件。下一步就是把这 5000 个文件进行归并（类似与归并排序）的过程了。

4. 海量日志数据，提取出某日访问百度次数最多的那个 IP。

方案 1：首先是这一天，并且是访问百度的日志中的 IP 取出来，逐个写入到一个大文件中。注意到 IP 是 32 位的，最多有 2^{32} 个 IP。同样可以采用映射的方法，比如模 1000，把整个大文件映射为 1000 个小文件，再找出每个小文中出现频率最大的 IP（可以采用 hash_map 进行频率统计，然后再找出频率最大的几个）及相应的频率。然后再在这 1000 个最大的 IP 中，找出那个频率最大的 IP，即为所求。

5. 在 **2.5** 亿个整数中找出不重复的整数，内存不足以容纳这 **2.5** 亿个整数。

方案 1：采用 2-Bitmap（每个数分配 2bit，00 表示不存在，01 表示出现一次，10 表示多次，11 无意义）进行，共需内存 $2^{32} * 2\text{bit} = 1\text{GB}$ 内存，还可以接受。然后扫描这 2.5 亿个整数，查看 Bitmap 中相对应位，如果是 00 变 01，01 变 10，10 保持不变。扫描完事后，查看 bitmap，把对应位是 01 的整数输出即可。

方案 2：也可采用上题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

6. 海量数据分布在 **100** 台电脑中，想个办法高效统计出这批数据的 **TOP10**。

方案 1：

1. 在每台电脑上求出 TOP10，可以采用包含 10 个元素的堆完成（TOP10 小，用最大堆，TOP10 大，用最小堆）。比如求 TOP10 大，我们首先取前 10 个元素调整成最小堆，如果发现，然后扫描后面的数据，并与堆顶元素比较，如果比堆顶元素大，那么用该元素替换堆顶，然后再调整为最小堆。最后堆中的元素就是 TOP10 大。
2. 求出每台电脑上的 TOP10 后，然后把这 100 台电脑上的 TOP10 组合起来，共 1000 个数据，再利用上面类似的方法求出 TOP10 就可以了。

(更多可以参考：第三章、寻找最小的 k 个数，以及第三章续、Top K 算法问题的实现)

读者反馈@QinLeopard:

第 6 题的方法中，是不是不能保证每个电脑上的前十条，肯定包含最后频率最高的前十条呢？

比如说第一个文件中：A(4), B(5), C(6), D(3)

第二个文件中：A(4), B(5), C(3), D(6)

第三个文件中：A(6), B(5), C(4), D(3)

如果要选 Top(1)，选出来的结果是 A，但结果应该是 B。

@July：我想，这位读者可能没有明确提议。本题目中的 **TOP10** 是指最大的 10 个数，而不是指出现频率最多的 10 个数。但如果说，现在有另外一提，要你求频率最多的 10 个，相当于求访问次数最多的 10 个 IP 地址那道题，即是本文中上面的第 4 题。特此说明。

7. 怎么在海量数据中找出重复次数最多的一个？

方案 1：先做 hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）。

8. 上千万或上亿数据（有重复），统计其中出现次数最多的 N 个数据。

方案 1：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用 hash_map/搜索二叉树/红黑树等来进行统计次数。然后就是取出前 N 个出现次数最多的数据了，可以用第 6 题提到的堆机制完成。

9. 1000 万字符串，其中有些是重复的，需要把重复的全部去掉，保留没有重复的字符串。 请怎么设计和实现？

方案 1：这题用 trie 树比较合适，hash_map 也应该能行。

10. 一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词，请给出思想，给出时间复杂度分析。

方案 1：这题是考虑时间效率。用 trie 树统计每个词出现的次数，时间复杂度是 $O(n \cdot le)$ (le 表示单词的平均长度)。然后是找出出现最频繁的前 10 个词，可以用堆来实现，前面的题中已经讲到了，时间复杂度是 $O(n \cdot \lg 10)$ 。所以总的时间复杂度，是 $O(n \cdot le)$ 与 $O(n \cdot \lg 10)$ 中较大的哪一个。

11. 一个文本文件，找出前 10 个经常出现的词，但这次文件比较长，说是上亿行或十亿行，总之无法一次读入内存，问最优解。

方案 1：首先根据用 `hash` 并求模，将文件分解为多个小文件，对于单个文件利用上题的方法求出每个文件件中 10 个最常出现的词。然后再进行归并处理，找出最终的 10 个最常出现的词。

12. 100w 个数中找出最大的 100 个数。

- 方案 1：采用局部淘汰法。选取前 100 个元素，并排序，记为序列 L 。然后一次扫描剩余的元素 x ，与排好序的 100 个元素中最小的元素比，如果比这个最小的要大，那么把这个最小的元素删除，并把 x 利用插入排序的思想，插入到序列 L 中。依次循环，知道扫描了所有的元素。复杂度为 $O(100w \cdot 100)$ 。
- 方案 2：采用快速排序的思想，每次分割之后只考虑比轴大的一部分，知道比轴大的一部分在比 100 多的时候，采用传统排序算法排序，取前 100 个。复杂度为 $O(100w \cdot 100)$ 。
- 方案 3：在前面的题中，我们已经提到了，用一个含 100 个元素的最小堆完成。复杂度为 $O(100w \cdot \lg 100)$ 。

13. 寻找热门查询：

搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为 1-255 字节。假设目前有一千万个记录，这些查询串的重复度比较高，虽然总数是 1 千万，但是如果去除重复和，不超过 3 百万个。一个查询串的重复度越高，说明查询它的用户越多，也就越热门。请你统计最热门的 10 个查询串，要求使用的内存不能超过 1G。

- (1) 请描述你解决这个问题的思路；
- (2) 请给出主要的处理流程，算法，以及算法的复杂度。

方案 1：采用 `trie` 树，关键字域存该查询串出现的次数，没有出现为 0。最后用 10 个元素的最小堆来对出现频率进行排序。

关于此问题的详细解答，请参考此文的第 3.1 节：[第三章续、Top K 算法问题的实现](#)。

14. 一共有 N 个机器，每个机器上有 N 个数。每个机器最多存 $O(N)$ 个数并对它们操作。如何找到 N^2 个数中的中数？

方案 1：先大体估计一下这些数的范围，比如这里假设这些数都是 32 位无符号整数（共有 2^{32} 个）。我们把 0 到 $2^{32}-1$ 的整数划分为 N 个范围段，每个段包含 $(2^{32})/N$ 个整数。比如，第一个段位 0 到 $2^{32}/N-1$ ，第二段为 $(2^{32})/N$ 到 $(2^{32})/N-1$ ，...，第 N 个段为 $(2^{32})(N-1)/N$ 到 $2^{32}-1$ 。然后，扫描每个机器上的 N 个数，把属于第一个区段的数放到第一个机器上，属于第二个区段的数放到第二个机器上，...，属于第 N 个区段的数放到第 N 个机器上。注意这个过程每个机器上存储的数应该是 $O(N)$ 的。下面我们依次统计

每个机器上数的个数，一次累加，直到找到第 k 个机器，在该机器上累加的数大于或等于 $(N^2)/2$ ，而在第 $k-1$ 个机器上的累加数小于 $(N^2)/2$ ，并把这个数记为 x 。那么我们要找的中位数在第 k 个机器中，排在第 $(N^2)/2-x$ 位。然后我们对第 k 个机器的数排序，并找出第 $(N^2)/2-x$ 个数，即为所求的中位数的复杂度是 $O(N^2)$ 的。

方案 2：先对每台机器上的数进行排序。排好序后，我们采用归并排序的思想，将这 N 个机器上的数归并起来得到最终的排序。找到第 $(N^2)/2$ 个便是所求。复杂度是 $O(N^2 \lg N^2)$ 的。

15. 最大间隙问题

给定 n 个实数 $x_1, x_2, x_3, \dots, x_n$ ，求着 n 个实数在实轴上向量 2 个数之间的最大差值，要求线性的时间算法。

方案 1：最先想到的方法就是先对这 n 个数据进行排序，然后一遍扫描即可确定相邻的最大间隙。但该方法不能满足线性时间的要求。故采取如下方法：

1. 找到 n 个数据中最大和最小数据 \max 和 \min 。
2. 用 $n-2$ 个点等分区间 $[\min, \max]$ ，即将 $[\min, \max]$ 等分为 $n-1$ 个区间（前闭后开区间），将这些区间看作桶，编号为 $1, 2, \dots, n-2, n-1$ ，且桶 i 的上界和桶 $i+1$ 的下届相同，即每个桶的大小相同。每个桶的大小为： $dblAvrGap = \frac{(\max - \min)}{n-1}$ 。实际上，这些桶的边界构成了一个等差数列（首项为 \min ，公差为 $d=dblAvrGap$ ），且认为将 \min 放入第一个桶，将 \max 放入第 $n-1$ 个桶。
3. 将 n 个数放入 $n-1$ 个桶中：将每个元素 $x[i]$ 分配到某个桶（编号为 $index$ ），其中

$$index = \left\lfloor \frac{(x[i] - \min)}{dblAvrGap} \right\rfloor + 1$$
，并求出分到每个桶的最大最小数据。
4. 最大间隙：除最大最小数据 \max 和 \min 以外的 $n-2$ 个数据放入 $n-1$ 个桶中，由抽屉原理可知至少有一个桶是空的，又因为每个桶的大小相同，所以最大间隙不会在同一桶中出现，一定是某个桶的上界和气候某个桶的下界之间隙，且该量筒之间的桶（即便好在该连个便好之间的桶）一定是空桶。也就是说，最大间隙在桶 i 的上界和桶 j 的下界之间产生 $j >= i+1$ 。一遍扫描即可完成。

16. 将多个集合合并成没有交集的集合

给定一个字符串的集合，格式如： $\{aaa, bbb, ccc\}, \{bbb, ddd\}, \{eee, fff\}, \{ggg\}, \{ddd, hhh\}$ 。要求将其中交集不为空的集合合并，要求合并完成的集合之间无交集，例如上例应输出 $\{aaa, bbb, ccc, ddd, hhh\}, \{eee, fff\}, \{ggg\}$ 。

- (1) 请描述你解决这个问题的思路；

(2) 给出主要的处理流程，算法，以及算法的复杂度；

(3) 请描述可能的改进。

方案 1：采用并查集。首先所有的字符串都在单独的并查集中。然后依扫描每个集合，顺序合并将两个相邻元素合并。例如，对于 $\{aaa, bbb, ccc\}$ ，首先查看 **aaa** 和 **bbb** 是否在同一个并查集中，如果不在，那么把它们所在的并查集合并，然后再看 **bbb** 和 **ccc** 是否在同一个并查集中，如果不在，那么也把它们所在的并查集合并。接下来再扫描其他的集合，当所有的集合都扫描完了，并查集代表的集合便是所求。复杂度应该是 $O(N \lg N)$ 的。改进的话，首先可以记录每个节点的根结点，改进查询。合并的时候，可以把大的和小的进行合，这样也减少复杂度。

17. 最大子序列与最大子矩阵问题

数组的最大子序列问题：给定一个数组，其中元素有正，也有负，找出其中一个连续子序列，使和最大。

方案 1：这个问题可以动态规划的思想解决。设 $b[i]$ 表示以第 i 个元素 $a[i]$ 结尾的最大子序列，那么显然 $b[i+1] = b[i] > 0 ? b[i] + a[i+1] : a[i+1]$ 。基于这一点可以很快用代码实现。

最大子矩阵问题：给定一个矩阵（二维数组），其中数据有大有小，请找一个子矩阵，使得子矩阵的和最大，并输出这个和。

方案 2：可以采用与最大子序列类似的思想来解决。如果我们确定了选择第 i 列和第 j 列之间的元素，那么在这个范围内，其实就是一个最大子序列问题。如何确定第 i 列和第 j 列可以词用暴搜的方法进行。

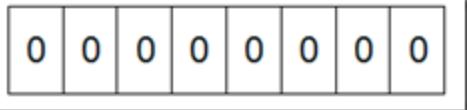
第二部分、海量数据处理之 Bit-map 详解

Bloom Filter 已在上一篇文章[海量数据处理之 Bloom Filter 详解](#)中予以详细阐述，本文接下来着重阐述 Bit-map。有任何问题，欢迎不吝指正。

什么是 Bit-map

所谓的 Bit-map 就是用一个 bit 位来标记某个元素对应的 Value，而 Key 即是该元素。由于采用了 Bit 为单位来存储数据，因此在存储空间方面，可以大大节省。

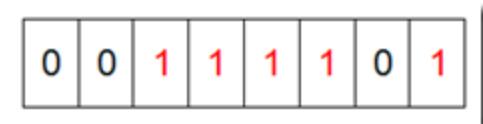
如果说了这么多还没明白什么是 Bit-map，那么我们来看一个具体的例子，假设我们要对 0-7 内的 5 个元素(4,7,2,5,3)排序（这里假设这些元素没有重复）。那么我们就可以采用 Bit-map 的方法来达到排序的目的。要表示 8 个数，我们就只需要 8 个 Bit (1Bytes)，首先我们开辟 1Byte 的空间，将这些空间的所有 Bit 位都置为 0(如下图：)



然后遍历这 5 个元素，首先第一个元素是 4，那么就把 4 对应的位置为 1（可以这样操作 $p+(i/8)|(0x01<<(i \% 8))$ ）当然了这里的操作涉及到 Big-ending 和 Little-ending 的情况，这里默认为 Big-ending），因为是从零开始的，所以要把第五位置为一（如下图）：



然后再处理第二个元素 7，将第八位置为 1，接着再处理第三个元素，一直到最后处理完所有的元素，将相应的位置为 1，这时候的内存的 Bit 位的状态如下：



然后我们现在遍历一遍 Bit 区域，将该位是一的位的编号输出 (2, 3, 4, 5, 7)，这样就达到了排序的目的。下面的代码给出了一个 BitMap 的用法：排序。

```
//定义每个 Byte 中有 8 个 Bit 位
#include <memory.h>
#define BYTESIZE 8
void SetBit(char *p, int posi)
{
    for(int i=0; i < (posi/BYTESIZE); i++)
    {
        p++;
    }

    *p = *p|(0x01<<(posi%BYTE SIZE)); //将该 Bit 位赋值 1
    return;
}

void BitMapSortDemo()
{
    //为了简单起见，我们不考虑负数
    int num[] = {3,5,2,10,6,12,8,14,9};

    //BufferLen 这个值是根据待排序的数据中最大值确定的
    //待排序中的最大值是 14，因此只需要 2 个 Bytes(16 个 Bit)
    //就可以了。
    const int BufferLen = 2;
```

```

char *pBuffer = new char[BufferLen];

//要将所有的 Bit 位置为 0, 否则结果不可预知。
memset(pBuffer,0,BufferLen);
for(int i=0;i<9;i++)
{
    //首先将相应 Bit 位上置为 1
    SetBit(pBuffer,num[i]);
}

//输出排序结果
for(int i=0;i<BufferLen;i++)//每次处理一个字节(Byte)
{
    for(int j=0;j<BYTESIZE;j++)//处理该字节中的每个 Bit 位
    {
        //判断该位上是否是 1, 进行输出, 这里的判断比较笨。
        //首先得到该第 j 位的掩码 (0x01<<j), 将内存区中的
        //位和此掩码作与操作。最后判断掩码是否和处理后的
        //结果相同
        if((*pBuffer&(0x01<<j)) == (0x01<<j))
        {
            printf("%d ",i*BYTESIZE + j);
        }
    }
    pBuffer++;
}
}

int _tmain(int argc, _TCHAR* argv[])
{
    BitMapSortDemo();
    return 0;
}

```

可进行数据的快速查找, 判重, 删除, 一般来说数据范围是 int 的 10 倍以下

基本原理及要点

使用 bit 数组来表示某些元素是否存在, 比如 8 位电话号码

扩展

Bloom filter 可以看做是对 bit-map 的扩展(关于 Bloom filter, 请参见: [海量数据处理之 Bloom filter 详解](#))。

问题实例

1)已知某个文件内包含一些电话号码，每个号码为**8**位数字，统计不同号码的个数。

8位最多 99 999 999，大概需要 99m 个 bit，大概 10 几 m 字节的内存即可。（可以理解为从 0-99 999 999 的数字，每个数字对应一个 Bit 位，所以只需要 99M 个 Bit==1.2MBytes，这样，就用了小小的 1.2M 左右的内存表示了所有的 8 位数的电话）

2)2.5 亿个整数中找出不重复的整数的个数，内存空间不足以容纳这**2.5**亿个整数。

将 bit-map 扩展一下，用 2bit 表示一个数即可，0 表示未出现，1 表示出现一次，2 表示出现 2 次及以上，在遍历这些数的时候，如果对应位置的值是 0，则将其置为 1；如果是 1，将其置为 2；如果是 2，则保持不变。或者我们不用 2bit 来进行表示，我们用两个 bit-map 即可模拟实现这个 2bit-map，都是一样的道理。

参考：

1. <http://www.cnblogs.com/youwang/archive/2010/07/20/1781431.html>。
2. <http://blog.redfox66.com/post/2010/09/26/mass-data-4-bitmap.aspx>。

完。

教你如何迅速秒杀掉：99%的海量数据处理面试题

作者：July

出处：结构之法算法之道 blog

前言

一般而言，标题含有“秒杀”，“99%”，“史上最全/最强”等词汇的往往都脱不了哗众取宠之嫌，但进一步来讲，如果读者读罢此文，却无任何收获，那么，我也甘愿背负这样的罪名，:-)，同时，此文可以看做是对这篇文章：[十道海量数据处理面试题与十个方法大总结的一般抽象性总结](#)。

毕竟受文章和理论之限，本文将摒弃绝大部分的细节，只谈方法/模式论，且注重用最通俗最直白的语言阐述相关问题。最后，有一点必须强调的是，全文行文是基于面试题的分析基础之上的，具体实践过程中，还是得具体情况具体分析，且场景也远比本文所述的任何一种情况复杂得多。

OK，若有任何问题，欢迎随时不吝赐教。谢谢。

何谓海量数据处理？

所谓海量数据处理，无非就是基于海量数据上的存储、处理、操作。何谓海量，就是数据量太大，所以导致要么是无法在较短时间内迅速解决，要么是数据太大，导致无法一次性装入内存。

那解决办法呢？针对时间，我们可以采用巧妙的算法搭配合适的数据结构，如 Bloom filter/Hash/bit-map/堆/数据库或倒排索引/trie 树，针对空间，无非就一个办法：大而化小：分而治之/hash 映射，你不是说规模太大嘛，那简单啊，就把规模大化为规模小的，各个击破不就完了嘛。

至于所谓的单机及集群问题，通俗点来讲，单机就是处理装载数据的机器有限(只要考虑 cpu，内存，硬盘的数据交互)，而集群，机器有多辆，适合分布式处理，并行计算(更多考虑节点和节点间的数据交互)。

再者，通过本 blog 内的有关海量数据处理的文章：[Big Data Processing](#)，我们已经大致知道，处理海量数据问题，无非就是：

1. 分而治之/hash 映射 + hash 统计 + 堆/快速/归并排序；

2. 双层桶划分
3. Bloom filter/Bitmap;
4. Trie 树/数据库/倒排索引;
5. 外排序;
6. 分布式处理之 Hadoop/Mapreduce。

下面，本文第一部分、从 `set/map` 谈到 `hashtable/hash_map/hash_set`，简要介绍下 `set/map/multiset/multimap`，及 `hash_set/hash_map/hash_multiset/hash_multimap` 之区别(万丈高楼平地起，基础最重要)，而本文第二部分，则针对上述那 6 种方法模式结合对应的海量数据处理面试题分别具体阐述。

第一部分、从 `set/map` 谈到 `hashtable/hash_map/hash_set`

稍后本文第二部分中将多次提到 `hash_map/hash_set`，下面稍稍介绍下这些容器，以作为基础准备。一般来说，STL 容器分两种，

- 序列式容器(`vector/list/deque/stack/queue/heap`)，
- 关联式容器。关联式容器又分为 `set`(集合)和 `map`(映射表)两大类，以及这两大类的衍生体 `multiset`(多键集合)和 `multimap`(多键映射表)，这些容器均以 RB-tree 完成。此外，还有第 3 类关联式容器，如 `hashtable`(散列表)，以及以 `hashtable` 为底层机制完成的 `hash_set`(散列集合)/`hash_map`(散列映射表)/`hash_multiset`(散列多键集合)/`hash_multimap`(散列多键映射表)。也就是说，`set/map/multiset/multimap` 都内含一个 RB-tree，而 `hash_set/hash_map/hash_multiset/hash_multimap` 都内含一个 `hashtable`。

所谓关联式容器，类似关联式数据库，每笔数据或每个元素都有一个键值(key)和一个实值(value)，即所谓的 Key-Value(键-值对)。当元素被插入到关联式容器中时，容器内部结构(RB-tree/`hashtable`)便依照其键值大小，以某种特定规则将这个元素放置于适当位置。

包括在非关联式数据库中，比如，在 MongoDB 内，文档(document)是最基本的数据组织形式，每个文档也是以 Key-Value (键-值对) 的方式组织起来。一个文档可以有多个 Key-Value 组合，每个 Value 可以是不同的类型，比如 String、Integer、List 等等。

```
{ "name": "July",
  "sex": "male",
  "age": 23 }
```

`set/map/multiset/multimap`

`set`，同 `map` 一样，所有元素都会根据元素的键值自动被排序，因为 `set/map` 两者的所有各种操作，都只是转而调用 RB-tree 的操作行为，不过，值得注意的是，两者都不允许两

个元素有相同的键值。

不同的是：`set` 的元素不像 `map` 那样可以同时拥有实值(`value`)和键值(`key`)，`set` 元素的键值就是实值，实值就是键值，而 `map` 的所有元素都是 `pair`，同时拥有实值(`value`)和键值(`key`)，`pair` 的第一个元素被视为键值，第二个元素被视为实值。

至于 `multiset/multimap`，他们的特性及用法和 `set/map` 完全相同，唯一的差别就在于它们允许键值重复，即所有的插入操作基于 RB-tree 的 `insert_equal()` 而非 `insert_unique()`。

hash_set/hash_map/hash_multiset/hash_multimap

`hash_set/hash_map`，两者的一切操作都是基于 `hashtable` 之上。不同的是，`hash_set` 同 `set` 一样，同时拥有实值和键值，且实质就是键值，键值就是实值，而 `hash_map` 同 `map` 一样，每一个元素同时拥有一个实值(`value`)和一个键值(`key`)，所以其使用方式，和上面的 `map` 基本相同。但由于 `hash_set/hash_map` 都是基于 `hashtable` 之上，所以不具备自动排序功能。为什么？因为 `hashtable` 没有自动排序功能。

至于 `hash_multiset/hash_multimap` 的特性与上面的 `multiset/multimap` 完全相同，唯一的差别就是它们 `hash_multiset/hash_multimap` 的底层实现机制是 `hashtable`(而 `multiset/multimap`，上面说了，底层实现机制是 RB-tree)，所以它们的元素都不会被自动排序，不过也都允许键值重复。

所以，综上，说白了，什么样的结构决定其什么样的性质，因为 `set/map/multiset/multimap` 都是基于 RB-tree 之上，所以有自动排序功能，而 `hash_set/hash_map/hash_multiset/hash_multimap` 都是基于 `hashtable` 之上，所以不含有自动排序功能，至于加个前缀 `multi_` 无非就是允许键值重复而已。

此外，

- 关于什么 `hash`，请看 blog 内此篇文章：http://blog.csdn.net/v_JULY_v/article/details/6256463；
- 关于红黑树，请参看 blog 内系列文章：http://blog.csdn.net/v_july_v/article/category/774945，
- 关于 `hash_map` 的具体应用：<http://blog.csdn.net/sdhongjun/article/details/4517325>，
- 关于 `hash_set`：<http://blog.csdn.net/morewindows/article/details/7330323>。

OK，接下来，请看本文第二部分、处理海量数据问题之六把密匙。

第二部分、处理海量数据问题之六把密匙

密匙一、分而治之 /Hash 映射 + Hash 统计 + 堆/快速/归并排序

1、海量日志数据，提取出某日访问百度次数最多的那个 IP。

既然是海量数据处理，那么可想而知，给我们的数据那就一定是海量的。针对这个数据的海量，我们如何着手呢？对的，无非就是分而治之/hash 映射 + hash 统计 + 堆/快速/归并排序，说白了，就是先映射，而后统计，最后排序：

1. 分而治之/hash 映射：针对数据太大，内存受限，只能是：把大文件化成(取模映射)小文件，即 16 字方针：大而化小，各个击破，缩小规模，逐个解决
2. hash 统计：当大文件转化了小文件，那么我们便可以采用常规的 `hash_map(ip, value)` 来进行频率统计。
3. 堆/快速排序：统计完了之后，便进行排序(可采取堆排序)，得到次数最多的 IP。

具体而论，则是：“首先是这一天，并且是访问百度的日志中的 IP 取出来，逐个写入到一个大文件中。注意到 IP 是 32 位的，最多有个 2^{32} 个 IP。同样可以采用映射的方法，比如模 1000，把整个大文件映射为 1000 个小文件，再找出每个小文中出现频率最大的 IP（可以采用 `hash_map` 进行频率统计，然后再找出频率最大的几个）及相应的频率。然后再在这 1000 个最大的 IP 中，找出那个频率最大的 IP，即为所求。”--十道海量数据处理面试题与十个方法大总结。

关于本题，还有几个问题，如下：

1、Hash 取模是一种等价映射，不会存在同一个元素分散到不同小文件中去的情况，即这里采用的是 mod1000 算法，那么相同的 IP 在 hash 后，只可能落在同一个文件中，不可能被分散的。

2、那到底什么是 hash 映射呢？简单来说，就是为了便于计算机在有限的内存中处理 big 数据，从而通过一种映射散列的方式让数据均匀分布在对应的内存位置(如大数据通过取余的方式映射成小树存放在内存中，或大文件映射成多个小文件)，而这个映射散列方式便是我们通常所说的 hash 函数，设计的好 hash 函数能让数据均匀分布而减少冲突。尽管数据映射到了另外一些不同的位置，但数据还是原来的数据，只是代替和表示这些原始数据的形式发生了变化而已。

此外，有一朋友 quicktest 用 python 语言实践测试了下本题，地址如下：<http://blog.csdn.net/quicktest/article/details/7453189>。谢谢。OK，有兴趣的，还可以再了解下一致性 hash 算法，见 blog 内此文第五部分：http://blog.csdn.net/v_july_v/article/details/6879101。

2、寻找热门查询：搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为 1-255 字节。

假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门），请你统计最热门的 10 个查询串，要求使用的内存不能超过 1G。

由上面第 1 题，我们知道，数据大则划为小的，但如果数据规模比较小，能一次性装入内存呢？比如这第 2 题，虽然有一千万个 Query，但是由于重复度比较高，因此事实上只有 300 万的 Query，每个 Query 255Byte，因此我们可以考虑把他们都放进内存中去，而现在只是需要一个合适的数据结构，在这里，Hash Table 绝对是我们优先的选择。所以我们放弃分而治之/hash 映射的步骤，直接上 hash 统计，然后排序。So，

1. hash 统计：先对这批海量数据预处理(维护一个 Key 为 Query 字串，Value 为该 Query 出现次数的 HashTable，即 `hash_map(Query, Value)`)，每次读取一个 Query，如果该字串不在 Table 中，那么加入该字串，并且将 Value 值设为 1；如果该字串在 Table 中，那么将该字串的计数加一即可。最终我们在 $O(N)$ 的时间复杂度内用 Hash 表完成了统计；
2. 堆排序：第二步、借助堆这个数据结构，找出 Top K，时间复杂度为 $N \cdot \log K$ 。即借助堆结构，我们可以在 \log 量级的时间内查找和调整/移动。因此，维护一个 K (该题目中是 10) 大小的小根堆，然后遍历 300 万的 Query，分别和根元素进行对比所以，我们最终的时间复杂度是： $O(N) + N \cdot O(\log K)$ ，(N 为 1000 万， N' 为 300 万)。

别忘了这篇文章中所述的堆排序思路：“维护 k 个元素的最小堆，即用容量为 k 的最小堆存储最先遍历到的 k 个数，并假设它们即是最大的 k 个数，建堆费时 $O(k)$ ，并调整堆（费时 $O(\log k)$ ）后，有 $k_1 > k_2 > \dots > k_{\min}$ (k_{\min} 设为小顶堆中最小元素)。继续遍历数列，每次遍历一个元素 x ，与堆顶元素比较，若 $x > k_{\min}$ ，则更新堆（用时 $\log k$ ），否则不更新堆。这样下来，总费时 $O(k \cdot \log k + (n-k) \cdot \log k) = O(n \cdot \log k)$ 。此方法得益于在堆中，查找等各项操作时间复杂度均为 $\log k$ 。”--[第三章续、Top K 算法问题的实现](#)。

当然，你也可以采用 trie 树，关键字域存该查询串出现的次数，没有出现为 0。最后用 10 个元素的最小堆来对出现频率进行排序。

3、有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 字节，内存限制大小是 1M。返回频数最高的 100 个词。

由上面那两个例题，分而治之 + hash 统计 + 堆/快速排序这个套路，我们已经开始有了屡试不爽的感觉。下面，再拿几道再多多验证下。请看此第 3 题：又是文件很大，又是内存受限，咋办？还能怎么办呢？无非还是：

1. 分而治之/hash 映射：顺序读文件中，对于每个词 x ，取 $\text{hash}(x) \% 5000$ ，然后按照该值存到 5000 个小文件（记为 $x_0, x_1, \dots, x_{4999}$ ）中。这样每个文件大概是 200k 左右。如果其中的有的文件超过了 1M 大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过 1M。
2. hash 统计：对每个小文件，采用 trie 树/hash_map 等统计每个文件中出现的词以及相应的频率。
3. 堆/归并排序：取出出现频率最大的 100 个词（可以用含 100 个结点的最小堆），

并把 100 个词及相应的频率存入文件，这样又得到了 5000 个文件。最后就是把这 5000 个文件进行归并（类似于归并排序）的过程了。

4、海量数据分布在 100 台电脑中，想个办法高效统计出这批数据的 TOP10。

此题与上面第 3 题类似，

1. 堆排序：在每台电脑上求出 TOP10，可以采用包含 10 个元素的堆完成（TOP10 小，用最大堆，TOP10 大，用最小堆）。比如求 TOP10 大，我们首先取前 10 个元素调整成最小堆，如果发现，然后扫描后面的数据，并与堆顶元素比较，如果比堆顶元素大，那么用该元素替换堆顶，然后再调整为最小堆。最后堆中的元素就是 TOP10 大。
2. 求出每台电脑上的 TOP10 后，然后把这 100 台电脑上的 TOP10 组合起来，共 1000 个数据，再利用上面类似的方法求出 TOP10 就可以了。

上述第 4 题的此解法，经读者反应有问题，如举个例子如求 2 个文件中的 top2，照上述算法，如果第一个文件里有：

- a 49 次
- b 50 次
- c 2 次
- d 1 次

第二个文件里有：

- a 9 次
- b 1 次
- c 11 次
- d 10 次

虽然第一个文件里出来 top2 是 b(50 次), a(49 次)，第二个文件里出来 top2 是 c(11 次), d(10 次)，然后 2 个 top2: b(50 次) a(49 次) 与 c(11 次) d(10 次) 归并，则算出所有的文件的 top2 是 b(50 次), a(49 次), 但实际上 a(58 次), b(51 次)。是否真是如此呢？若真如此，那作何解决呢？

正如老梦所述：

首先，先把所有的数据遍历一遍做一次 hash(保证相同的数据条目划分到同一台电脑上进行运算)，然后根据 hash 结果重新分布到 100 台电脑中，接下来的算法按照之前的即可。

最后由于 a 可能出现在不同的电脑，各有一定的次数，再对每个相同条目进行求和（由于上一步骤中 hash 之后，也方便每台电脑只需要对自己分到的条目内进行求和，不涉及到别的电脑，规模缩小）。

5、有 10 个文件，每个文件 1G，每个文件的每一行存放的都是用户的 query，每个文件的

query 都可能重复。要求你按照 **query** 的频度排序。

直接上：

1. **hash 映射**: 顺序读取 10 个文件, 按照 $\text{hash}(\text{query}) \% 10$ 的结果将 **query** 写入到另外 10 个文件(记为)中。这样新生成的文件每个的大小大约也 1G(假设 **hash** 函数是随机的)。
2. **hash 统计**: 找一台内存存在 2G 左右的机器, 依次对用 **hash_map(query, query_count)** 来统计每个 **query** 出现的次数。注: **hash_map(query, query_count)** 是用来统计每个 **query** 的出现次数, 不是存储他们的值, 出现一次, 则 **count+1**。
3. **堆/快速/归并排序**: 利用快速/堆/归并排序按照出现次数进行排序, 将排序好的 **query** 和对应的 **query_count** 输出到文件中, 这样得到了 10 个排好序的文件(记为)。最后, 对这 10 个文件进行归并排序(内排序与外排序相结合)。

除此之外, 此题还有以下两个方法:

方案 2: 一般 **query** 的总量是有限的, 只是重复的次数比较多而已, 可能对于所有的 **query**, 一次性就可以加入到内存了。这样, 我们就可以采用 **trie** 树/**hash_map** 等直接来统计每个 **query** 出现的次数, 然后按出现次数做快速/堆/归并排序就可以了。

方案 3: 与方案 1 类似, 但在做完 **hash**, 分成多个文件后, 可以交给多个文件来处理, 采用分布式的架构来处理(比如 **MapReduce**), 最后再进行合并。

6、给定 **a**、**b** 两个文件, 各存放 50 亿个 **url**, 每个 **url** 各占 64 字节, 内存限制是 4G, 让你找出 **a**、**b** 文件共同的 **url**?

可以估计每个文件的大小为 $5G \times 64 = 320G$, 远远大于内存限制的 4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

1. **分而治之/hash 映射**: 遍历文件 **a**, 对每个 **url** 求取, 然后根据所取得的值将 **url** 分别存储到 1000 个小文件(记为, 这里漏写了个 **a1**)中。这样每个小文件的大约有 300M。遍历文件 **b**, 采取和 **a** 相同的方式将 **url** 分别存储到 1000 小文件中(记为)。这样处理后, 所有可能相同的 **url** 都在对应的小文件()中, 不对应的小文件不可能有相同的 **url**。然后我们只要求出 1000 对小文件中相同的 **url** 即可。
2. **hash 统计**: 求每对小文件中相同的 **url** 时, 可以把其中一个小文件的 **url** 存储到 **hash_set** 中。然后遍历另一个小文件的每个 **url**, 看其是否在刚才构建的 **hash_set** 中, 如果是, 那么就是共同的 **url**, 存到文件里面就可以了。

OK, 此第一种方法: 分而治之/**hash** 映射 + **hash** 统计 + 堆/快速/归并排序, 再看最后 4 道题, 如下:

7、怎么在海量数据中找出重复次数最多的一个?

方案 1：先做 hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）。

8、上千万或上亿数据（有重复），统计其中出现次数最多的前 N 个数据。

方案 1：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用 hash_map/搜索二叉树/红黑树等来进行统计次数。然后就是取出前 N 个出现次数最多的数据了，可以用第 2 题提到的堆机制完成。

9、一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词，请给出思想，给出时间复杂度分析。

方案 1：这题是考虑时间效率。用 trie 树统计每个词出现的次数，时间复杂度是 $O(n \cdot le)$ (le 表示单词的平均长度)。然后是找出出现最频繁的前 10 个词，可以用堆来实现，前面的题中已经讲到了，时间复杂度是 $O(n \cdot lg 10)$ 。所以总的时间复杂度，是 $O(n \cdot le)$ 与 $O(n \cdot lg 10)$ 中较大的哪一个。

10. 1000 万字符串，其中有些是重复的，需要把重复的全部去掉，保留没有重复的字符串。请怎么设计和实现？

- 方案 1：这题用 trie 树比较合适，hash_map 也行。
- 方案 2：从 xbjzju：，1000w 的数据规模插入操作完全不现实，以前试过在 stl 下 100w 元素插入 set 中已经慢得不能忍受，觉得基于 hash 的实现不会比红黑树好太多，使用 vector+sort+unique 都要可行许多，建议还是先 hash 成小文件分开处理再综合。

上述方案 2 中读者 xbjzju 的方法让我想到了一些问题，即是 set/map，与 hash_set/hash_map 的性能比较？共计 3 个问题，如下：

- 1、hash_set 在千万级数据下，insert 操作优于 set？这位 blog：<http://t.cn/zOibP7t> 给的实践数据可靠不？
- 2、那 map 和 hash_map 的性能比较呢？谁做过相关实验？

```
set US hash_set US hash_table(强化版) 性能测试
数据容量 10000000个 查询次数 10000000次
容器中数据范围 [0, 40000000) 查询数据范围[0, 40000000)
--by MoreWindows( http://blog.csdn.net/MoreWindows ) --

-----插入数据-----
set中有数据8061105个
set 的 insert操作 用时 18782毫秒
hash_set中有数据8061105个
hash_set 的 insert操作 用时 7722毫秒
hash_table中有数据8061105个
Hash_table 的 insert操作 用时 4930毫秒
```

- 3、那查询操作呢，如下段文字所述？

可以发现在hash_table中最长的链表也只有5个元素，**长度为1和长度为2的链表中的数据占全部数据的89%以上。因此绝大部分查询将仅仅访问哈希表1次到2次。**这样的查询效率当然会比set（内部使用红黑树——类似于二叉平衡树）高的多。有了这个图示，无疑已经可以证明hash_set会比set快速高效了。但hash_set还可以动态的增加表的大小，因此我们再实现一个表大小可增加的hash_table。

或者小数据量时用 map，构造快，大数据量时用 hash_map？

rbtree PK hashtable

据朋友N邦卡猫N做的做的红黑树和 hash table 的性能测试中发现：当数据量基本上 int 型 key 时，hash table 是 rbtree 的 3-4 倍，但 hash table 一般会浪费大概一半内存。

因为 hash table 所做的运算就是个%，而 rbtree 要比较很多，比如 rbtree 要看 value 的数据，每个节点要多出 3 个指针（或者偏移量）如果需要其他功能，比如，统计某个范围内的 key 的数量，就需要加一个计数成员。

且 1s rbtree 能进行大概 50w+ 次插入，hash table 大概是差不多 200w 次。不过很多时候，其速度可以忍了，例如倒排索引差不多也是这个速度，而且单线程，且倒排表的拉链长度不会太大。正因为基于树的实现其实不比 hashtable 慢到哪里去，所以数据库的索引一般都是用的 B/B+ 树，而且 B+ 树还对磁盘友好(B 树能有效降低它的高度，所以减少磁盘交互次数)。比如现在非常流行的 NoSQL 数据库，像 MongoDB 也是采用的 B 树索引。关于 B 树系列，请参考本 blog 内此篇文章：从 B 树、B+ 树、B* 树谈到 R 树。

OK，更多请待后续实验论证。接下来，咱们来看第二种方法，双层桶划分。

密匙二、双层桶划分

双层桶划分----其实本质上还是分而治之的思想，重在“分”的技巧上！

适用范围：第 k 大，中位数，不重复或重复的数字

基本原理及要点：因为元素范围很大，不能利用直接寻址表，所以通过多次划分，逐步

确定范围，然后最后在一个可以接受的范围内进行。可以通过多次缩小，双层只是一个例子。

扩展：

问题实例：

11、2.5亿个整数中找出不重复的整数的个数，内存空间不足以容纳这2.5亿个整数。

有点像鸽巢原理，整数个数为 2^{32} ，也就是，我们可以将这 2^{32} 个数，划分为 2^8 个区域（比如用单个文件代表一个区域），然后将数据分离到不同的区域，然后不同的区域在利用bitmap就可以直接解决了。也就是说只要有足够的磁盘空间，就可以很方便的解决。

12、5亿个int找它们的中位数。

这个例子比上面那个更明显。首先我们将int划分为 2^{16} 个区域，然后读取数据统计落到各个区域里的数的个数，之后我们根据统计结果就可以判断中位数落到那个区域，同时知道这个区域中的第几大数刚好是中位数。然后第二次扫描我们只统计落在这个区域中的那些数就可以了。

实际上，如果不是int是int64，我们可以经过3次这样的划分即可降低到可以接受的程度。即可以先将int64分成 2^{24} 个区域，然后确定区域的第几大数，在将该区域分成 2^{20} 个子区域，然后确定是子区域的第几大数，然后子区域里的数的个数只有 2^{20} ，就可以直接利用direct addr table进行统计了。

密匙三：Bloom filter/Bitmap

Bloom filter

关于什么是Bloom filter，请参看blog内此文：

- 海量数据处理之 Bloom Filter 详解

适用范围：可以用来实现数据字典，进行数据的判重，或者集合求交集

基本原理及要点：

对于原理来说很简单，位数组+k个独立hash函数。将hash函数对应的值的位数组置1，查找时如果发现所有hash函数对应位都是1说明存在，很明显这个过程并不保证查找的结果是100%正确的。同时也不支持删除一个已经插入的关键字，因为该关键字对应的位会牵动到其他的关键字。所以一个简单的改进就是counting Bloom filter，用一个counter数组代替位数组，就可以支持删除了。

还有一个比较重要的问题，如何根据输入元素个数n，确定位数组m的大小及hash函数个数。当hash函数个数 $k=(\ln 2) * (\ln n)$ 时错误率最小。在错误率不大于E的情况下，m至少要等于 $n \cdot \ln(1/E)$ 才能表示任意n个元素的集合。但m还应该更大些，因为还要保证bit数组里至少一半为0，则m应该 $\geq n \cdot \ln(1/E) + n \cdot \ln 2$ （大概就是 $n \cdot \ln(1/E) + n \cdot \ln 2$ 倍（lg表示以2为底的对数）。

举个例子我们假设错误率为 0.01，则此时 m 应大概是 n 的 13 倍。这样 k 大概是 8 个。

注意这里 m 与 n 的单位不同， m 是 bit 为单位，而 n 则是以元素个数为单位(准确的说是不同元素的个数)。通常单个元素的长度都是有很多 bit 的。所以使用 bloom filter 内存上通常都是节省的。

扩展：

Bloom filter 将集合中的元素映射到位数组中，用 k (k 为哈希函数个数) 个映射位是否全 1 表示元素在不在这个集合中。Counting bloom filter (CBF) 将位数组中的每一位扩展为一个 counter，从而支持了元素的删除操作。Spectral Bloom Filter (SBF) 将其与集合元素的出现次数关联。SBF 采用 counter 中的最小值来近似表示元素的出现频率。

13、给你 A,B 两个文件，各存放 50 亿条 URL，每条 URL 占用 64 字节，内存限制是 4G，让你找出 A,B 文件共同的 URL。如果是三个乃至 n 个文件呢？

根据这个问题我们来计算下内存的占用， $4G=2^{32}$ 大概是 40 亿 * 8 大概是 340 亿， $n=50$ 亿，如果按出错率 0.01 算需要的大概是 650 亿个 bit。现在可用的是 340 亿，相差并不多，这样可能会使出错率上升些。另外如果这些 urlip 是一一对应的，就可以转换成 ip，则大大简单了。

同时，上文的第 5 题：给定 a 、 b 两个文件，各存放 50 亿个 url，每个 url 各占 64 字节，内存限制是 4G，让你找出 a 、 b 文件共同的 url？如果允许有一定的错误率，可以使用 Bloom filter，4G 内存大概可以表示 340 亿 bit。将其中一个文件中的 url 使用 Bloom filter 映射为这 340 亿 bit，然后挨个读取另外一个文件的 url，检查是否与 Bloom filter，如果是，那么该 url 应该是共同的 url（注意会有一定的错误率）。

Bitmap

- 关于什么是 Bitmap，请看 blog 内此文第二部分：http://blog.csdn.net/v_july_v/article/details/6685962。

下面关于 Bitmap 的应用，直接上题，如下第 9、10 道：

14、在 2.5 亿个整数中找出不重复的整数，注，内存不足以容纳这 2.5 亿个整数。

方案 1：采用 2-Bitmap（每个数分配 2bit，00 表示不存在，01 表示出现一次，10 表示多次，11 无意义）进行，共需内存 $2^{32} * 2 \text{ bit}=1 \text{ GB}$ 内存，还可以接受。然后扫描这 2.5 亿个整数，查看 Bitmap 中相对应位，如果是 00 变 01，01 变 10，10 保持不变。扫描完事后，查看 bitmap，把对应位是 01 的整数输出即可。

方案 2：也可采用与第 1 题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

15、腾讯面试题：给 40 亿个不重复的 `unsigned int` 的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那 40 亿个数当中？

方案 1：从内存申请 512M 的内存，一个 bit 位代表一个 `unsigned int` 值。读入 40 亿个数，设置相应的 bit 位，读入要查询的数，查看相应 bit 位是否为 1，为 1 表示存在，为 0 表示不存在。

密匙四、Trie 树/数据库/倒排索引

Trie 树

适用范围：数据量大，重复多，但是数据种类小可以放入内存

基本原理及要点：实现方式，节点孩子的表示方式

扩展：压缩实现。

问题实例：

- 上面的**第 2 题**：寻找热门查询：查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个，每个不超过 255 字节。
- 上面的**第 5 题**：有 10 个文件，每个文件 1G，每个文件的每一行都存放的是用户的 query，每个文件的 query 都可能重复。要你按照 query 的频度排序。
- 1000 万字符串，其中有些是相同的(重复)，需要把重复的全部去掉，保留没有重复的字符串。请问怎么设计和实现？
- 上面的**第 8 题**：一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词。其解决方法是：用 Trie 树统计每个词出现的次数，时间复杂度是 $O(n*le)$ (le 表示单词的平均长度)，然后是找出出现最频繁的前 10 个词。

更多有关 Trie 树的介绍，请参见此文：[从 Trie 树（字典树）谈到后缀树](#)。

数据库索引

适用范围：大数据量的增删改查

基本原理及要点：利用数据的设计实现方法，对海量数据的增删改查进行处理。

- 关于数据库索引及其优化，更多可参见此文：<http://www.cnblogs.com/pkuoliver/archive/2011/08/17/mass-data-topic-7-index-and-optimize.html>；
- 关于 MySQL 索引背后的数据结构及算法原理，这里还有一篇很好的文章：<http://www.codinglabs.org/html/theory-of-mysql-index.html>；
- 关于 B 树、B+ 树、B* 树及 R 树，本 blog 内有篇绝佳文章：http://blog.csdn.net/v_JULY_v/article/details/6530142。

倒排索引(Inverted index)

适用范围：搜索引擎，关键字查询

基本原理及要点：为何叫倒排索引？一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。

以英文为例，下面是要被索引的文本：

T0 = "it is what it is"

T1 = "what is it"

T2 = "it is a banana"

我们就能得到下面的反向文件索引：

"a": {2}

"banana": {2}

"is": {0, 1, 2}

"it": {0, 1, 2}

"what": {0, 1}

检索的条件"what", "is"和"it"将对应集合的交集。

正向索引开发出来用来存储每个文档的单词的列表。正向索引的查询往往满足每个文档有序频繁的全文查询和每个单词在校验文档中的验证这样的查询。在正向索引中，文档占据了中心的位置，每个文档指向了一个它所包含的索引项的序列。也就是说文档指向了它包含的那些单词，而反向索引则是单词指向了包含它的文档，很容易看到这个反向的关系。

扩展：

问题实例：文档检索系统，查询那些文件包含了某单词，比如常见的学术论文的关键字搜索。

关于倒排索引的应用，更多请参见：

- 第二十三、四章：杨氏矩阵查找，倒排索引关键词 Hash 不重复编码实践，
- 第二十六章：基于给定的文档生成倒排索引的编码与实践。

密匙五、外排序

适用范围：大数据的排序，去重

基本原理及要点：外排序的归并方法，置换选择败者树原理，最优归并树

扩展：

问题实例：

1).有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 个字节，内存限制大小是 1M。返回频数最高的 100 个词。

这个数据具有很明显的特点，词的大小为 16 个字节，但是内存只有 1M 做 hash 明显不够，所以可以用来排序。内存可以当输入缓冲区使用。

关于多路归并算法及外排序的具体应用场景，请参见 blog 内此文：

- 第十章、如何给 10^7 个数据量的磁盘文件排序

密匙六、分布式处理之 Mapreduce

MapReduce 是一种计算模型，简单的说就是将大批量的工作（数据）分解（MAP）执行，然后再将结果合并成最终结果（REDUCE）。这样做好处是可以在任务被分解后，可以通过大量机器进行并行计算，减少整个操作的时间。但如果你要我再通俗点介绍，那么，说白了，Mapreduce 的原理就是一个归并排序。

适用范围：数据量大，但是数据种类小可以放入内存

基本原理及要点：将数据交给不同的机器去处理，数据划分，结果归约。

扩展：

问题实例：

1. The canonical example application of MapReduce is a process to count the appearances of each different word in a set of documents:
2. 海量数据分布在 100 台电脑中，想个办法高效统计出这批数据的 TOP10。
3. 一共有 N 个机器，每个机器上有 N 个数。每个机器最多存 $O(N)$ 个数并对它们操作。
如何找到 $N^{1/2}$ 个数的中数(median)？

更多具体阐述请参见 blog 内：

- 从 Hadoop 框架与 MapReduce 模式中谈海量数据处理，
- 及 MapReduce 技术的初步了解与学习。

其它模式/方法论，结合操作系统知识

至此，六种处理海量数据问题的模式/方法已经阐述完毕。据观察，这方面的面试题无外乎以上一种或其变形，然题目为何取为是：秒杀 99% 的海量数据处理面试题，而不是 100% 呢。OK，给读者看最后一道题，如下：

非常大的文件，装不进内存。每行一个 int 类型数据，现在要你随机取 100 个数。

我们发现上述这道题，无论是以上任何一种模式/方法都不好做，那有什么好的别的方法呢？我们可以看看：操作系统内存分页系统设计(说白了，就是映射+建索引)。

Windows 2000 使用基于分页机制的虚拟内存。每个进程有 4GB 的虚拟地址空间。基于分页机制，这 4GB 地址空间的一些部分被映射了物理内存，一些部分映射硬盘上的交换文件，一些部分什么也没有映射。程序中使用的都是 4GB 地址空间中的虚拟地址。而访问物理内存，需要使用物理地址。关于什么是物理地址和虚拟地址，请看：

- 物理地址 (physical address): 放在寻址总线上的地址。放在寻址总线上，如果是读，电路根据这个地址每位的值就将相应地址的物理内存中的数据放到数据总线中传输。如果是写，电路根据这个 地址每位的值就将相应地址的物理内存中放入数据总线上的内容。物理内存是以字节(8位)为单位编址的。
- 虚拟地址 (virtual address): 4G 虚拟地址空间中的地址，程序中使用的都是虚拟地址。 使用了分页机制之后，4G 的地址空间被分成了固定大小的页，每一页或者被映射到物理内存，或者被映射到硬盘上的交换文件中，或者没有映射任何东西。对于一般程序来说，4G 的地址空间，只有一小部分映射了物理内存，大片大片的部分是没有映射任何东西。物理内存也被分页，来映射地址空间。对于 32bit 的 Win2k，页的大小是 4K 字节。CPU 用来把虚拟地址转换成物理地址的信息存放在叫做页目录和页表的结构里。

物理内存分页，一个物理页的大小为 4K 字节，第 0 个物理页从物理地址 0x00000000 处开始。由于页的大小为 4KB，就是 0x1000 字节，所以第 1 页从物理地址 0x00001000 处开始。第 2 页从物理地址 0x00002000 处开始。可以看到由于页的大小是 4KB，所以只需要 32bit 的地址中高 20bit 来寻址物理页。

返回上面我们的题目：非常大的文件，装不进内存。每行一个 int 类型数据，现在要你随机取 100 个数。针对此题，我们可以借鉴上述操作系统中内存分页的设计方法，做出如下解决方案：

操作系统中的方法，先生成 4G 的地址表，在把这个表划分为小的 4M 的小文件做个索引，二级索引。30 位前十位表示第几个 4M 文件，后 20 位表示在这个 4M 文件的第几个，等等，基于 key value 来设计存储，用 key 来建索引。

但如果现在只有 10000 个数，然后怎么去随机从这一万个数里面随机取 100 个数？请读者思考。

参考文献

1. 十道海量数据处理面试题与十个方法大总结;
2. 海量数据处理面试题集锦与 Bit-map 详解;
3. 十一、从头到尾彻底解析 Hash 表算法;
4. 海量数据处理之 Bloom Filter 详解;
5. 从 Trie 树（字典树）谈到后缀树;
6. 第三章续、Top K 算法问题的实现;
7. 第十章、如何给 10^7 个数据量的磁盘文件排序;
8. 从 B 树、B+树、B*树谈到 R 树;

9. 第二十三、四章：杨氏矩阵查找，倒排索引关键词 Hash 不重复编码实践；
10. 第二十六章：基于给定的文档生成倒排索引的编码与实践；
11. 从 Hadoop 框架与 MapReduce 模式中谈海量数据处理；
12. 第十六~第二十章：全排列，跳台阶，奇偶排序，第一个只出现一次等问题；
13. http://blog.csdn.net/v_JULY_v/article/category/774945；
14. STL 源码剖析第五章，侯捷著；
15. 2012 百度实习生招聘笔试题：<http://blog.csdn.net/hackbuteer1/article/detail/s/7542774>。

后记

经过上面这么多海量数据处理面试题的轰炸，我们依然可以看出这类问题是有一定的解决方案/模式的，所以，不必将其神化。然这类面试题所包含的问题还是比较简单的，若您在这方面有更多实践经验，欢迎随时来信与我不吝分享：zhoulei0907@yahoo.cn。当然，自会注明分享者及来源。

不过，相信你也早就意识到，若单纯论海量数据处理面试题，本 blog 内的有关海量数据处理面试题的文章已涵盖了你能在网上所找到的 70~80%。但有点，必须负责任的敬告大家：无论是这些海量数据处理面试题也好，还是算法也好，**面试时**，70~80% 的人不是倒在这两方面，而是倒在基础之上(诸如语言，数据库，操作系统，网络协议等等)，所以，**无论任何时候，基础最重要**，没了基础，便什么都不是。如果你问我什么叫做掌握了基础，很简单，我可以举个例子，如到<http://forum.csdn.net/BList/Cpp/>，如果你几乎能解决那里的全部问题，那么你的 c/c++ 基础便够了。

最后，推荐国外一面试题网站：<http://www.careercup.com/>，以及个人正在读的 Redis/MongoDB 绝佳站点：<http://blog.nosqlfan.com/>。

OK，本文若有任何问题，欢迎随时不吝留言，评论，赐教，谢谢。完。

九月腾讯，创新工场，淘宝等公司最新面试三十题（第 171-200 题）

引言

曾记否，去年的 10 月份也同此刻一样，是找工作的高峰期，本博客便是最初由整理微软等公司面试题而发展而来的。如今，又即将迈入求职高峰期--10 月份，所以，也不免关注了网上和个人建的算法群 Algorithms1-12 群内朋友发布和讨论的最新面试题。特此整理，以飨诸位。至于答案，望诸位共同讨论与思考。

最新面试十三题

好久没有好好享受思考了。ok，任何人有任何意见或问题，欢迎不吝指导：

1. 五只猴子分桃。半夜，第一只猴子先起来，它把桃分成了相等的五堆，多出一只。于是，它吃掉了一个，拿走了一堆； 第二只猴子起来一看，只有四堆桃。于是把四堆合在一起，分成相等的五堆，又多出一个。于是，它也吃掉了一个，拿走了一堆；其他几只猴子也都是 这样分的。问：这堆桃至少有多少个？（朋友说，这是小学奥数题）。

参考答案：先给这堆桃子加上 4 个,设此时共有 X 个桃子,最后剩下 a 个桃子.这样:

第一只猴子分完后还剩: $(1-1/5)X=(4/5)X$;

第二只猴子分完后还剩: $(1-1/5)(4/5)X$;

第三只猴子分完后还剩: $(1-1/5)(4/5)^2X$;

第四只猴子分完后还剩: $(1-1/5)(4/5)^3X$;

第五只猴子分完后还剩: $(1-1/5)(4/5)^4X=(1024/3125)X$;

得: $a=(1024/3125)X$;

要使 a 为整数,X 最小取 3125.

减去加上的 4 个,所以,这堆桃子最少有 3121 个。

2. 已知有个 rand7() 的函数，返回 1 到 7 随机自然数，让利用这个 rand7() 构造 rand10() 随机 1~10。

（参考答案：这题主要考的是对概率的理解。程序关键是要算出 rand10，1 到 10，十个数字出现的考虑都为 10%.根据排列组合，连续算两次 rand7 出现的组合数是 $7^2=49$ ，这 49 种组合每一种出现考虑是相同的。怎么从 49 平均概率的转换为 1 到 10 呢？方法是：

1.rand7 执行两次，出来的数为 a1.a2.

2.如果 $a1*7+a2<40$, $b=(a1*7+a2)/10+1$,如果 $a1*7+a2>=40$,重复第一步）。参考代码如下所示：

```
int rand7()
```

```

{
    return rand()%7+1;
}

int rand10()
{
    int a71,a72,a10;
    do
    {
        a71 = rand7()-1;
        a72 = rand7()-1;
        a10 = a71 *7 + a72;
    } while (a10 >= 40);
    return (a71*7+a72)/4+1;
}

```

3. 如果两个字符串的字符一样，但是顺序不一样，被认为是兄弟字符串，问如何在迅速匹配兄弟字符串（如，bad 和 adb 就是兄弟字符串）。思路：判断各自素数乘积是否相等。
4. 要求设计一个 DNS 的 Cache 结构，要求能够满足每秒 5000 以上的查询，满足 IP 数据的快速插入，查询的速度要快。
5. 一个未排序整数数组，有正负数，重新排列使负数排在正数前面，并且要求不改变原来的正负数之间相对顺序 比如： input: 1,7,-5,9,-12,15 ans: -5,-12,1,7,9,15 要求时间复杂度 $O(N)$,空间 $O(1)$ 。（此题一直没看到令我满意的答案，一般达不到题目所要求的：时间复杂度 $O(N)$ ，空间 $O(1)$ ，且保证原来正负数之间的相对位置不变）。

updated: 设置一个起始点 j, 一个翻转点 k,一个终止点 L

从右侧起

起始点在第一个出现的负数，翻转点在起始点后第一个出现的正数,终止点在翻转点后出现的第一个负数(或结束)

如果无翻转点，则不操作

如果有翻转点，则待终止点出现后，做翻转，即 $ab \Rightarrow ba$ 这样的操作

翻转后，负数串一定在左侧，然后从负数串的右侧开始记录起始点，继续往下找下一个翻转点

例子中的就是

1, 7, -5, 9, -12, 15

第一次翻转: 1, 7, -5, -12, 9, 15 \Rightarrow 1, -12, -5, 7, 9, 15

第二次翻转: -5, -12, 1, 7, 9, 15

N 维翻转空间占用为 $O(1)$ 复杂度是 $2N$; 在有一个负数的情况下，复杂度最大是 $2N$;

在有 i 个负数的情况下，复杂度最大是 $2N+2i$ ，但是不会超过 $2N+N$ 实际的复杂度在 $O(3N)$ 以内

但从最终时间复杂度分析，此方法是否真能达到 $O(N)$ 的时间复杂度，还待后续考证。
感谢 John_Lv, MikovChain。2012.02.25。

1, 7, -5, -6, 9, -12, 15 (后续：此种情况未能处理)

1 7 -5 -6 -12 9 15

1 -12 -5 -6 7 9 15

-6 -12 -5 1 7 9 15

更多请参考此文，程序员编程艺术第二十七章：重新排列数组（不改变相对顺序&时间 $O(N)$ &空间 $O(1)$ ，半年未被 KO）http://blog.csdn.net/v_july_v/article/details/7329314。

6. 淘宝面试题：有一个一亿节点的树，现在已知两个点，找这两个点的共同的祖先。
7. 海量数据分布在 100 台电脑中，想个办法高效统计出这批数据的 TOP10。（此题请参考本博客内其它文章）。
8. 某服务器流量统计器，每天有 1000 亿的访问记录数据，包括时间、url、ip。设计系统实现记录数据的保存、管理、查询。要求能实现一下功能：
 - (1) 计算在某一时间段（精确到分）时间内的，某 url 的所有访问量。
 - (2) 计算在某一时间段（精确到分）时间内的，某 ip 的所有访问量。
9.
假设某个网站每天有超过 10 亿次的页面访问量，出于安全考虑，网站会记录访问客户端访问的 ip 地址和对应的时间，如果现在已经记录了 1000 亿条数据，想统计一个指定时间段内的区域 ip 地址访问量，那么这些数据应该按照何种方式来组织，才能尽快满足上面的统计需求呢，
设计完方案后，并指出该方案的优缺点，比如在什么情况下，可能会非常慢？（参考答案：用 B+ 树来组织，非叶子节点存储（某个时间点，页面访问量），叶子节点是访问的 IP 地址。这个方案的优点是查询某个时间段内的 IP 访问量很快，但是要统计某个 IP 的访问次数或是上次访问时间就不得不遍历整个树的叶子节点。或者可以建立二级索引，分别是时间和地点来建立索引。）
10.
腾讯 1. 服务器内存 1G，有一个 2G 的文件，里面每行存着一个 QQ 号（5-10 位数），怎么最快找出出现过最多次的 QQ 号。（此题与稍后下文的第 14 题重复，思路参考请见下文第 14 题）。
腾讯 2. 如何求根号 2 的值，并且按照我的需要列出指定小数位，比如根号 2 是 1.141 我要列出 1 位小数就是 1.12 位就是 1.14, 1000 位就是 1.141..... 等。。

11.

给定一个字符串的集合，格式如：{aaa bbb ccc}， {bbb ddd}， {eee fff}， {ggg}， {ddd
hhh}要求将其中交集不为空的集合合并，要求合并完成后的集合之间无交集，例如上例应输出{aaa bbb ccc ddd hhh}， {eee fff}， {ggg}。

12.

创新工场面试题：abcde 五人打渔，打完睡觉，a 先醒来，扔掉 1 条鱼，把剩下的分成 5 分，拿一份走了；b 再醒来，也扔掉 1 条，把剩下的分成 5 份，拿一份走了；然后 cde 都按上面的方法取鱼。问他们一共打了多少条鱼，写程序和算法实现。提示：共打了多少条鱼的结果有很多。但求最少打的鱼的结果是 3121 条鱼（应该找这 5 个人问问，用什么工具打了这么多条鱼）。

（<http://blog.csdn.net/nokiaguy/article/details/6800209>）。

13. 我们有很多瓶无色的液体，其中有一瓶是毒药，其它都是蒸馏水，实验的小白鼠喝了以后会在 5 分钟后死亡，而喝到蒸馏水的小白鼠则一切正常。现在有 5 只小白鼠，请问一下，我们用这五只小白鼠，5 分钟的时间，能够检测多少瓶液体的成分？

淘宝 2012 笔试（研发类）：<http://topic.csdn.net/u/20110922/10/e4f3641a-1f31-4d35-80da-7268605d2d51.html>（一参考答案）。

ok，这 13 道题加上此前本博客陆陆续续整理的微软面试 187 题：[重启开源，分享无限--诚邀你加入微软面试 187 题的解题中](#)，至此，本博客内已经整理了整整 **200 道**面试题。

后续整理

以下是后续整理的最新面试题，不断更新中（2011.09.26）.....

14、腾讯最新面试题：服务器内存 1G，有一个 2G 的文件，里面每行存着一个 QQ 号（5-10 位数），怎么最快找出出现过最多次的 QQ 号。

以下是一个人所建第 **Algorithms_12** 群内朋友的聊天记录：

首先你要注意到，数据存在服务器，存储不了（内存存不了），要想办法统计每一个 qq 出现的次数。

比如，因为内存是 1g，首先 你用 hash 的方法，把 qq 分配到 10 个（这个数字可以变动，比较）文件（在硬盘中）。

相同的 qq 肯定在同一个文件中，然后对每一个文件，只要保证每一个文件少于 1g 的内存，统计每个 qq 的次数，可以使用 `hash_map(qq, qq_count)` 实现。然后，记录每个文件的最大访问次数的 qq，最后，从 10 个文件中找出一个最大，即为所有的最大。更多读者可以参见此文：[海量数据处理面试题集锦与 Bit-map 详解](#)。

那若面试官问有没有更高效率的解法之类的？这时，你可以优化一下，但是这个速度很

快, `hash` 函数, 速度很快, 他肯定会问, 你如何设计, 用 `bitmap` 也行。

15、百度今天的笔试题: 在一维坐标轴上有 n 个区间段, 求重合区间最长的两个区间段。

16、华为社招现场面试 1: 请使用代码计算

1234567891011121314151617181920*2019181716151413121110987654321。

华为面试 2: 1 分 2 分 5 分的硬币, 组成 1 角, 共有多少种组合。

17、百度笔试题:

一、系统有很多任务, 任务之间有依赖, 比如 B 依赖于 A, 则 A 执行完后 B 才能执行

(1) 不考虑系统并行性, 设计一个函数 (`Task *Ptask,int Task_num`) 不考虑并行度, 最快的方法完成所有任务。

(2) 考虑并行度, 怎么设计

```
typedef struct{
    int ID;
    int * child;
    int child_num;
}Task;
```

提供的函数:

`bool doTask(int taskID);`无阻塞的运行一个任务;

`int waitTask(int timeout);`返回运行完成的任务 id, 如果没有则返回-1;

`bool killTask(int taskID);`杀死进程

二、必答题 (各种 `const`)

1、解释下面 `ptr` 含义和不同

`double* ptr = &value;`

//`ptr` 是一个指向 `double` 类型的指针, `ptr` 的值可以改变, `ptr` 所指向的 `value` 的值也可以改变

`const double* ptr = &value`

//`ptr` 是一个指向 `const double` 类型的指针, `ptr` 的值可以改变, `ptr` 所指向的 `value` 的值不可以改变

`double* const ptr=&value`

//`ptr` 是一个指向 `double` 类型的指针, `ptr` 的值不可以改变, `ptr` 所指向的 `value` 的值可以改变

`const double* const ptr=&value`

//`ptr` 是一个指向 `const double` 类型的指针, `ptr` 的值不可以改变, `ptr` 所指向的 `value` 的值也不可以改变

2、去掉 `const` 属性，例： `const double value = 0.0f; double* ptr = NULL;` 怎么才能让 `ptr` 指向 `value`？

强制类型转换，去掉 `const` 属性，如 `ptr = <const_cast double *>(&value);`

http://topic.csdn.net/u/20110925/16/e6248e53-1145-4815-8d24-9c9019d24bd8.html?seed=1665205011&r=75709169#r_75709169

18、如果用一个循环数组 `q[0..m-1]` 表示队列时,该队列只有一个队列头指针 `front`,不设队列尾指针 `rear`，求这个队列中从队列头到队列尾的元素个数（包含队列头、队列尾）（华赛面试题、腾讯笔试题）。

19、昨晚淘宝笔试题：

1. 设计相应的数据结构和算法，尽量高效的统计一片英文文章（总单词数目）里出现的所有英文单词，按照在文章中首次出现的顺序打印输出该单词和它的出现次数。

2、有一棵树（树上结点为字符串或者整数），请写代码将树的结构和数据写到一个文件中，并能通过读取该文件恢复树结构。

20、13 个球一个天平，现知道只有一个和其它的重量不同，问怎样称才能用三次就找到那个球？(<http://zhidao.baidu.com/question/66024735.html>)。

21、搜狗笔试题：一个长度为 `n` 的数组 `a[0],a[1],...,a[n-1]`。现在更新数组的各个元素，即 `a[0]` 变为 `a[1]` 到 `a[n-1]` 的积，`a[1]` 变为 `a[0]` 和 `a[2]` 到 `a[n-1]` 的积，...，`a[n-1]` 为 `a[0]` 到 `a[n-2]` 的积（就是除掉当前元素，其他所有元素的积）。程序要求：具有线性复杂度，且不能使用除法运算符。

思路：`left[i]` 标示着 `a[i]` 之前的乘积，`right[i]` 标示着 `a[i]` 之后的乘积，但不申请空间，那么 `a[i]=left[i]*right[i]`。不过，`left` 的计算从左往右扫的时候得出，`right` 是从右往左扫得出。因为就是当中某个元素 `a[i]` 的左边所有元素与右边所有元素的乘积，就这么简单。所以计算 `a[i]=left[i]*right[i]` 时，直接出结果。

22、后 2012 年 4 月 67 日的腾讯暑期实习生招聘笔试中，出了一道与上述 21 题类似的题，原题大致如下：

两个数组 `a[N]`，`b[N]`，其中 `A[N]` 的各个元素值已知，现给 `b[i]` 赋值，`b[i] = a[0]*a[1]*a[2]*...*a[N-1]/a[i];`

要求：

1. 不准用除法运算
2. 除了循环计数值，`a[N],b[N]` 外，不准再用其他任何变量（包括局部变量，全局变量等）

3. 满足时间复杂度 $O(n)$, 空间复杂度 $O(1)$ 。

说白了，你要我求 $b=a[0]*a[1]*\dots*a[i-1]*a[i+1]*\dots*a[N-1]/a[i]$ ，就是求：
 $a[0]*a[1]*\dots*a[i-1]*a[i+1]*\dots*a[N-1]$ 。只是我把 $a[i]$ 左边部分标示为 $\text{left}[i]$, $a[i]$ 右边部分标示为 $\text{right}[i]$, 而实际上完全不申请 $\text{left}[i]$, 与 $\text{right}[i]$ 变量, 之所以那样标示, 无非就是为了说明: 除掉当前元素 $a[i]$, 其他所有元素($a[i]$ 左边部分, 和 $a[i]$ 右边部分)的积。读者你明白了么?

下面是此 TX 笔试题的两段参考代码, 如下:

```
void array_multiplication(int A[], int OUTPUT[], int n) {
    int left = 1;
    int right = 1;
    for (int i = 0; i < n; i++) {
        OUTPUT[i] = 1;
        for (int j = 0; j < n; j++) {
            if (j < i)
                OUTPUT[i] *= left;
            else if (j > i)
                OUTPUT[i] *= right;
            left *= A[j];
            right *= A[n - 1 - j];
        }
    }
}
```

```
//ncicc
b[0] = 1;
for (int i = 1; i < N; i++)
{
    b[0] *= a[i-1];
    b[i] = b[0];
}
b[0] = 1;
for (i = N-2; i > 0; i--)
{
    b[0] *= a[i+1];
    b[i] *= b[0];
}
b[0] *= a[1];
```

from wasd6081058 上面第二段代码最后一行的意义是: 我们看第二个循环, 从 $N-2$ 到 1 ; 再看 `for` 循环中 $b[0]$ 的赋值, 从 $N-1$ 到 2 , 而根据题目要求 $b[i] = a[0]*a[1]*a[2]*\dots*a[N-1]/a[i]$, $b[0]$ 应等于 $a[1]*a[2]*\dots*a[N-1]$, 所以这里手动添加 $a[1]$ 。

23、腾讯高水平复试题:

1. 有不同的手机终端, 如 `iphone`, 安卓, `Symbian`, 不同的终端处理不一样, 设计一种服务器和算法实现对不同的终端的处理。
2. 设计一种内存管理算法。
3. A 向 B 发邮件, B 收到后读取并发送收到, 但是中间可能丢失了该邮件, 怎么设计一种

最节省的方法，来处理丢失问题。

4. 设计一种算法求出算法复杂度。

24、人人笔试 1: 一个人上台阶可以一次上 1 个，2 个，或者 3 个，问这个人上 n 层的台阶，总共有几种走法？

人人笔试 2：在人人好友里，A 和 B 是好友，B 和 C 是好友，如果 A 和 C 不是好友，那么 C 是 A 的二度好友，在一个有 10 万人的数据库里，如何在时间 $O(n)$ 里，找到某个人的十度好友。

25、淘宝算法面试题：两个用户之间可能互相认识，也可能是单向的认识，用什么数据结构来表示？如果一个用户不认识别人，而且别人也不认识他，那么他就是无效节点，如何找出这些无效节点？自定义数据接口并实现之，要求尽可能节约内存和空间复杂度。

26、淘宝笔试题：对于一颗完全二叉树，要求给所有节点加上一个 pNext 指针，指向同一层的相邻节点；如果当前节点已经是该层的最后一个节点，则将 pNext 指针指向 NULL；给出程序实现，并分析时间复杂度和空间复杂度。

27、腾讯面试题：给你 5 个球，每个球被抽到的可能性为 30、50、20、40、10，设计一个随机算法，该算法的输出结果为本次执行的结果。输出 A, B, C, D, E 即可。

28、搜狐笔试题：给定一个实数数组，按序排列（从小到大），从数组中找出若干个数，使得这若干个数的和与 M 最为接近，描述一个算法，并给出算法的复杂度。

29、阿里巴巴研究院（2009）：

1. 有无序的实数列 $V[N]$ ，要求求里面大小相邻的实数的差的最大值，关键是要线性空间和线性时间

2. 25 匹赛马，5 个跑道，也就是说每次有 5 匹马可以同时比赛。问最少比赛多少次可以知道跑得最快的 5 匹马

3. 有一个函数 int getNum()，每运行一次可以从一个数组 $V[N]$ 里面取出一个数， N 未知，当数取完的时候，函数返回 NULL。现在要求写一个函数 int get()，这个函数运行一次可以从 $V[N]$ 里随机取出一个数，而这个数必须是符合 $1/N$ 平均分布的，也就是说 $V[N]$ 里面任意一个数都有 $1/N$ 的机会被取出，要求空间复杂度为 $O(1)$

30、微软面试题：Given a head pointer pointing to a linked list ,please write a function that sort the list in increasing order. You are not allowed to use temporary array or memory copy

```
struct
{
    int data;
    struct S_Node *next;
}Node;
```

```
Node * sort_link_list_increasing_order (Node *pheader):
```

更新至 2011.09.30....

如果各位对上面的题目有好的思路,或者还有好的面试题分享,欢迎添加到本文评论下,
或者发至我的邮箱: zhoulei0709@yahoo.cn。谢谢大家。July、2011.09.30。

十月百度，阿里巴巴，迅雷搜狗最新面试七十题（第 201-270 题）

引言

当月早已进入 10 月份，十一过后，招聘，笔试，面试，求职渐趋火热。而在这一系列过程背后浮出的各大 IT 公司的笔试/面试题则蕴含着诸多思想与设计，细细把玩，思考一番亦能有不少收获。

上个月，本博客着重整理九月腾讯，创新工场，淘宝等公司最新面试十三题，此次重点整理百度，阿里巴巴，迅雷和搜索等公司最新的面试题。同上篇一样，答案望诸君共同讨论之，个人亦在慢慢思考解答。多谢。

最新面试十一题

1. 十月百度：一个数组保存了 N 个结构，每个结构保存了一个坐标，结构间的坐标都不相同，请问如何找到指定坐标的结构（除了遍历整个数组，是否有更好的办法）？（要么预先排序，二分查找。要么哈希。hash 的话，坐标(x, y)你可以当做一个 2 位数，写一个哈希函数，把 (x, y) 直接转成 “(x, y)” 作为 key，默认用 string 比较。或如 Edward Lee 所说，将坐标(x, y) 作为 Hash 中的 key。例如(m, n)，通过 (m, n) 和 (n, m) 两次查找看是否在 HashMap 中。也可以在保存时就规定 (x, y) , x < y，在插入之前做个判断。）
2. 百度最新面试题：现在有 1 千万个随机数，随机数的范围在 1 到 1 亿之间。现在要求写出一种算法，将 1 到 1 亿之间没有在随机数中的数求出来。（编程珠玑上有此类似的一题，如果有足够的内存的话可以用位图法，即开一个 1 亿位的 bitset，内存为 $100\text{m}/8 = 12.5\text{m}$ ，然后如果一个数有出现，对应的 bitset 上标记为 1，最后统计 bitset 上为 0 的即可。）
3. Alibaba 笔试题：给定一段产品的英文描述，包含 M 个英文字母，每个英文单词以空格分隔，无其他标点符号；再给定 N 个英文单词关键字，请说明思路并编程实现方法

```
String extractSummary(String description, String[] key words)
```

目标是找出此产品描述中包含 N 个关键字（每个关键词至少出现一次）的长度最短的子串，作为产品简介输出。（不限编程语言）20 分。（扫描过程始终保持一个 [left, right] 的 range，初始化确保 [left, right] 的 range 里包含所有关键字则停止。然后每次迭代：
1，试图右移动 left，停止条件为再移动将导致无法包含所有关键字。
2，比较当前 range's length 和 best length，更新最优值。
3，右移 right，停止条件为使任意一个关键字的计数+1。
4，重复迭代。

编程之美有最短摘要生成的问题，与此问题类似，读者可作参考。)

4. 搜狗：有 N 个正实数(注意是实数，大小升序排列) $x_1, x_2 \dots x_N$, 另有一个实数 M 。需要选出若干个 x , 使这几个 x 的和与 M 最接近。请描述实现算法，并指出算法复杂度（参考：[第五章、寻找满足条件的两个或多个数](#)）。
5. 迅雷：给你 10 台机器，每个机器 2 个 cpu, 2g 内存，现在已知在 10 亿条记录的数据库里执行一次查询需要 5 秒，问用什么方法能让 90% 的查询能在 100 毫秒以内返回结果。

（@geochway：将 10 亿条记录排序，然后分到 10 个机器中，分的时候是一个记录一个记录的轮流分，
确保每个机器记录大小分布差不多，每一次查询时，同时提交给 10 台机器，同时查询，
因为记录已排序，可以采用二分法查询。
如果无法排序，只能顺序查询，那就要看记录本身的概率分布，否则不可能实现。
一个机器 2 个 CPU 未必能起到作用，要看这两个 CPU 能否并行存取内存，取决于系统架构。）
6. 给定一个函数 `rand()` 能产生 0 到 $n-1$ 之间的等概率随机数，问如何产生 0 到 $m-1$ 之间等概率的随机数？
7. 腾讯：五笔的编码范围是 $a \sim y$ 的 25 个字母，从 1 位到 4 位的编码，如果我们把五笔的编码按字典序排序，形成一个数组如下：

a, aa, aaa, aaaa, aaab, aaac,, b, ba, baa, baaa, baab, baac, yyw,
yyyx, yyyy
其中 a 的 `Index` 为 0, aa 的 `Index` 为 1, aaa 的 `Index` 为 2, 以此类推。
1) 编写一个函数，输入是任意一个编码，比如 `baca`，输出这个编码对应的 `Index`；
2) 编写一个函数，输入是任意一个 `Index`，比如 12345，输出这个 `Index` 对应的编码。
8. **2011.10.09 百度笔试题**（下述第 8-12 题）：linux/unix 远程登陆都用到了 ssh 服务，当网络出现错误时服务会中断，linux/unix 端的程序会停止。为什么会这样？说下 ssh 的原理，解释中断的原理。
9. 一个最小堆，也是完全二叉树，用按层遍历数组表示。
 1. 求节点 $a[n]$ 的子节点的访问方式
 2. 插入一节点的程序 `void add_element(int *a,int size,int val);`
 3. 删除最小节点的程序。
- 10.a) 求一个全排列函数：如 `p([1,2,3])`，输出： [123],[132],[213],[231],[321],[323]。
b) 求一个组合函数：如 `p([1,2,3])`，输出： [1],[2],[3],[1,2],[2,3],[1,3],[1,2,3]。
这两问可以用伪代码（全排列请参考这里的第 67 题：[微软、Google 等公司非常好的面试题及解答【第 61-70 题】](#)）。

3. 通过某种 hash 算法，可以让用户稳定的均匀分布到一个区间内，这个区间的大小为 100%，分布的最小粒度为：0.1%。我们把这种区间叫做一层。现在有两个区间 A, B，如何让层 A 中的任意子区间段都均匀分布到层 B 的 100% 中？例如：层 A 中取 10%，这 10%会均匀分布到层 B 中，即：层 B 的每一个 10% 区间都会有 1% 的区间 A 中的 10%，也可以说层 B 的。如果现在有超过 10 层，每一层之间都需要有这种关系，又如何解决？

11.

12. 有这样一种编码：如， $N=134$, $M=f(N)=143$, $N=020$, $M=fun(N)=101$, 其中 N 和 M 的位数一样，N,M 可以均可以以 0 开头，N,M 的各位数之和要相等，即 $1+3+4=1+4+3$ ，且 M 是大于 N 中最小的一个，现在求这样的序列 S,N 为一个定值，其中 $S(0)=N$, $S(1)=fun(N)$, $S(2)=fun(S(1))$ 。

13. 有 1000 万条 URL，每条 URL 50 字节，只包含主机前缀，要求实现 URL 提示系统：

- (1) 要求实时更新匹配用户输入的地址，每输出一个字符，输出最新匹配 URL
- (2) 每次只匹配主机前缀，例如对 www.abaidu.com 和 www.baidu.com，用户输入 www.b 时只提示 www.baidu.com
- (3) 每次提供 10 条匹配的 URL
- (4) 以用户需求为主。

14. 海量记录，记录形式如下：TERMINAL URLNOCOUNT urlno1 urlno2 ... urlnoN
怎么考虑资源和时间这两个因素，实现快速查询任意两个记录的交集，并集等，设计相关的数据结构和算法。

15. 百度最新笔试题（感谢 xiongyangwan 提供的题目）：利用互斥量和条件变量设计一个消息队列，具有以下功能：

- 1 创建消息队列（消息中所含的元素）
- 2 消息队列中插入消息
- 3 取出一个消息（阻塞方式）
- 4 取出第一消息（非阻塞方式）

16. 百度移动终端研发笔试：系统设计题（40 分）

对已排好序的数组 A，一般来说可用二分查找可以很快找到。现有一特殊数组 A[], 它是循环递增的，如 $A[] = \{17\ 19\ 20\ 25\ 1\ 4\ 7\ 9\}$ ，试在这样的数组中找一元素 x，看看是否存在。请写出你的算法，必要时可写伪代码，并分析其空间、时间复杂度。

17.

```
#include<stdio.h>
#include<string.h>
void main()
{
    int a[2000];
    char *p = (char *)a;
    int i ;
    for( i = 0; i < 2000; i++)
```

```

    a[i] = -i -1;
    printf("%d\n", strlen(p));
}

```

写出输出结果

(onlyice: i = FFFFFF00H 的时候，才有'\0'出现，就是最后一个字节，C 风格字符串读到'\0'就终止了。FFFFFFFFFF00H 是 -256，就是 i 的值为 255 时 a[i] = FFFFFF00H)

18. 腾讯 10.09 测试笔试题：有 $N+2$ 个数， N 个数出现了偶数次，2 个数出现了奇数次（这两个数不相等），问用 $O(1)$ 的空间复杂度，找出这两个数，不需要知道具体位置，只需要知道这两个值。（@Rojay: xor 一次，得到 2 个奇数次的数之和 x 。第二步，以 x （展开成二进制）中有 1 的某位（假设第 i 位为 1）作为划分，第二次只 xor 第 i 位为 1 的那些数，得到 y 。然后 $x \text{ xor } y$ 以及 y 便是那两个数。）

19. @well: 一个整数数组，有 n 个整数，如何找其中 m 个数的和等于另外 $n-m$ 个数的和？

（与上面第 4 题类似，参考：[第五章、寻找满足条件的两个或多个数](#)）。

20. 阿里云笔试题：一个 HTTP 服务器处理一次请求需要 500 毫秒，请问这个服务器如何每秒处理 100 个请求。

21. 今天 10.10 阿里云笔试@土豆：1、三次握手；

TCP 连接是通过三次握手进行初始化的。三次握手的目的是同步连接双方的序列号和确认号并交换 TCP 窗口大小信息。以下步骤概述了通常情况下客户端计算机联系服务器计算机的过程：

1. 客户端向服务器发送一个SYN置位的TCP报文，其中包含连接的初始序列号x和一个窗口大小（表示客户端上用来存储从服务器发送来的传入段的缓冲区的大小）。
2. 服务器收到客户端发送过来的SYN报文后，向客户端发送一个SYN和ACK都置位的TCP报文，其中包含它选择的初始序列号y、对客户端的序列号的确认x+1和一个窗口大小（表示服务器上用来存储从客户端发送来的传入段的缓冲区的大小）。
3. 客户端接收到服务器端返回的SYN+ACK报文后，向服务器端返回一个确认号y+1和序号x+1的ACK报文，一个标准的TCP连接完成。

TCP 使用类似的握手过程来结束连接。这可确保两个主机均能完成传输并确保所有的数据均得以接收

TCP Client	Flags	TCP Server
1 Send SYN (seq=x)	----SYN--->	SYN Received
2 SYN/ACK Received	<---SYN/ACK----	Send SYN (seq=y), ACK (x+1)
3 Send ACK (y+1)	----ACK--->	ACK Received, Connection Established
x: ISN (Initial Sequence Number) of the Client		
y: ISN of the Server		

第一次是客户端发起连接；第二次表示服务器收到了客户端的请求；第三次表示客户端收到了服务器的反馈。这之后双方均确认了连接的有效性，如果第三次服务器未收到，假设一个C向S发送了SYN后无故消失了，那么S在发出SYN+ACK应答报文后是无法收到C的ACK报文的（第三次握手无法完成），这种情况下S一般会重试（再次发送SYN+ACK给客户端）并等待一段时间后丢弃这个未完成的连接，这段时间的长度我们称为SYN Timeout，一般来说这个时间是分钟的数量级（大约为30秒-2分钟）；

2、死锁的条件。（**互斥条件**（Mutual exclusion）：1、资源不能被共享，只能由一个进

程使用。2、请求与保持条件 (Hold and wait): 已经得到资源的进程可以再次申请新的资源。3、非剥夺条件 (No pre-emption): 已经分配的资源不能从相应的进程中被强制地剥夺。4、循环等待条件 (Circular wait): 系统中若干进程组成环路，该环路中每个进程都在等待相邻进程正占用的资源。**处理死锁的策略**: 1. 忽略该问题。例如鸵鸟算法，该算法可以应用在极少发生死锁的情况下。为什么叫鸵鸟算法呢，因为传说中鸵鸟看到危险就把头埋在地底下，可能鸵鸟觉得看不到危险也就没危险了吧。跟掩耳盗铃有点像。2. 检测死锁并且恢复。3. 仔细地对资源进行动态分配，以避免死锁。4. 通过破除死锁四个必要条件之一，来防止死锁产生。)

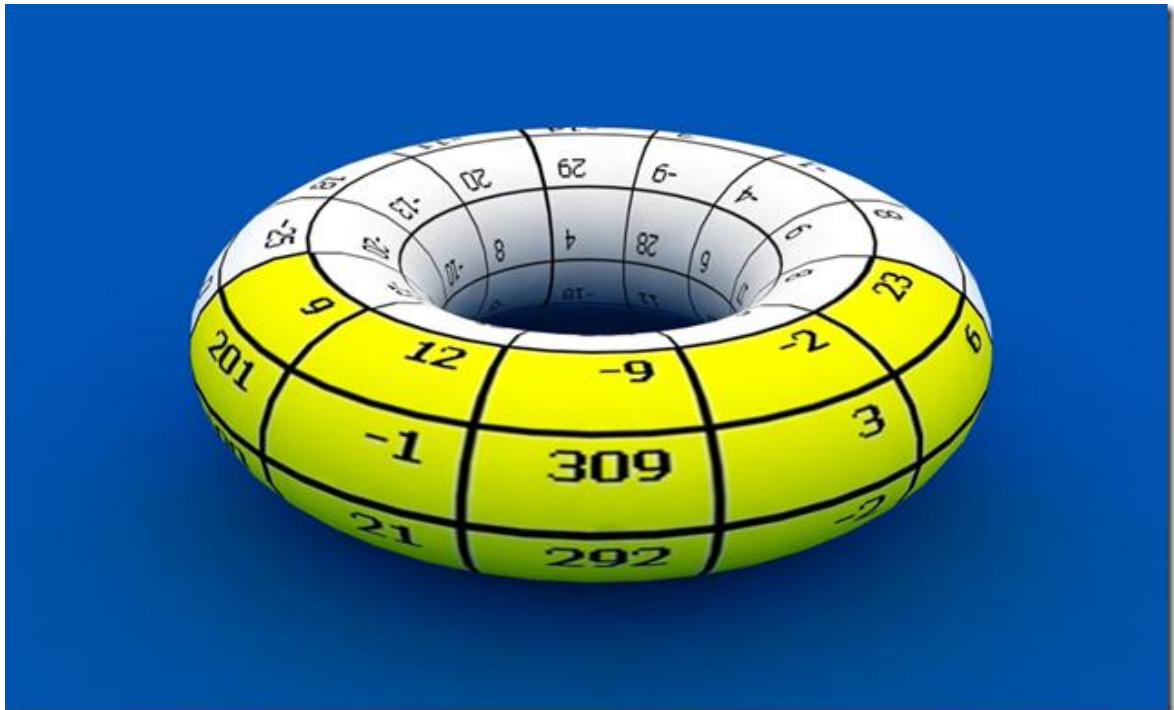
22.微软 2011 最新面试题 (以下三题，第 22、23、24 题皆摘自微软亚洲研究院的邹欣老师博客): 浏览过本人的[程序员编程艺术系列](#)的文章，一定对其中的这个问题颇有印象：[第七章、求连续子数组的最大和](#)。求数组最大子数组的和最初来源于编程之美，

-32, -10, 33, -23, 32, -12, 41, -12, 12

。我在编程艺术系列中提供了多种解答方式，然而这个问题若扩展到二维数组呢？

8	-10	-3	26	-11	-1	-6	12	17	6	28	4
20	-13	-20	-13	-15	-254	5	8	9	-4	-9	29
-11	18	-25	9	12	-9	-2	23	8	-1	3	-14
-16	-7	0	201	-1	309	3	6	-18	11	24	-8
-1	-7	11	100	21	292	-2	2	-18	-8	-10	9
26	-11	-19	-18	20	-981	2	-14	12	-14	1	27
9	-20	5	28	-15	26	-20	-8	-16	30	3	20
-6	-7	-5	-9	-16	-15	5	-16	22	-17	11	-18

再者，若数组首尾相连，像一个轮胎一样，又怎么办呢？聪明的同学还是给出了漂亮的答案，并且用 SilverLight/WPF 给画了出来，如下图所示：



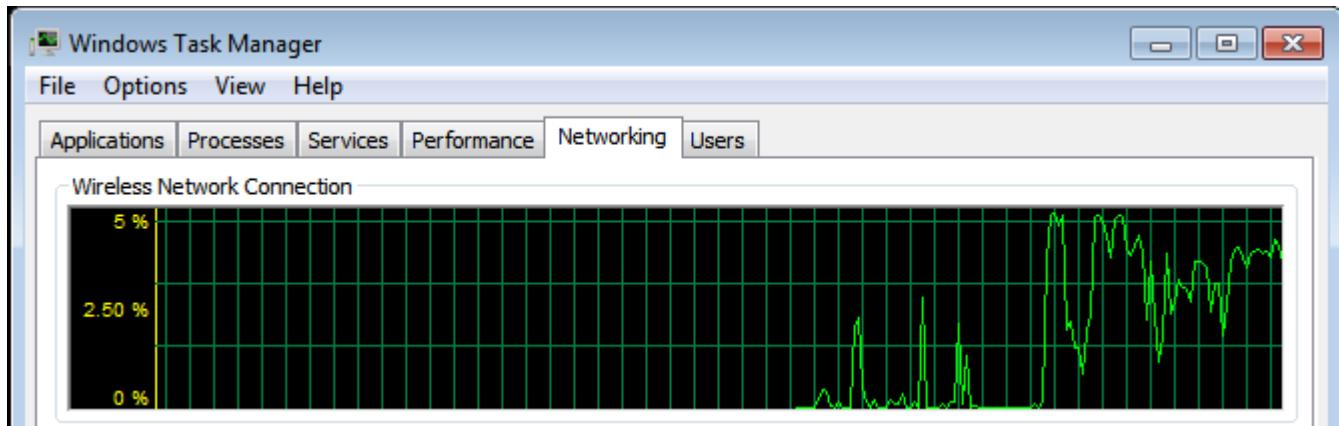
好，设想现在我们有一张纸带，两面都写满了像上第一幅图那样的数字，我们把纸带的一端扭转，和另一端接起来，构成一个莫比乌斯环（Möbius Strip, 如将一个长方形纸条 $ABCD$ 的一端 AB 固定，另一端 DC 扭转半周后，把 AB 和 CD 粘合在一起，得到的曲面就是麦比乌斯圈，也称莫比乌斯带。），如下图所示：



如上，尽管这个纸带扭了一下，但是上面还是有数组，还是有最大子数组的和，对么？在求最大子数组的和之前，我们用什么样的数据结构来表示这些数字呢？你可以用 Java, C, C#, 或其他语言的数据结构来描述这个莫比乌斯环上的数组。数据结构搞好了，算法自然就有了。（@风大哥：莫比乌斯带，用环形数组或者链表可以表示。环型数组的话， $1-N$ ，到 N 特殊处理一下，连到 1 就是环型数组了，一个纸带上正反两面各有 N 个数， $A_1 \dots A_n, B_1 \dots B_n$ ，那么就可以构造一个新的数组： $A_1-A_n-B_1-B_n$. 访问到 B_n 下一位就是 A_1 ，就是环形的数组了。从某个位置 k 开始，用 i, j 向一个方向遍历，直到 i 到达 k 位置，或者 $i=j$ ，被追上，用数组需要一点技巧，就是 J 再次过 k 需要打个标志，以便计算终止条件和输出。当然，如果用链表就更简单了。把链表首尾相接即可，即 A_n 执行 B_1, B_n 指向 A_1 即可。）

23. 《编程之美》的第一题是让 Windows 任务管理器的 CPU 使用率曲线画出一个正弦波。

我一直在想，能不能把 CPU 使用率边上的网络使用率也如法炮制一下呢？比如，也来一个正弦曲线？



24. 如果你没看过，也至少听说<人月神话> (The Mythical Man-month) 这本在软件工程领域很有影响的书。当你在微软学术搜索中输入“manmonth”这个词的时候，你会意外地碰到下面这个错误：

A screenshot of the Microsoft Academic Search interface. The search bar at the top contains the query "manmonth". Below the search bar, the page title is "Academic > Results for "manmonth" in All Domains". The main content area displays the message "We did not find any result related to "manmonth"". Underneath this message, there is a section titled "Search Tips:" with the following bullet points:

- Make sure words are spelled correctly.
- Try rephrasing keywords or using synonyms. E.g. use "face detection", instead of "face identific
- Try less specific keywords. E.g. use "decision tree", instead of "arc-4x adaboost decision tree".
- Use concise queries. E.g. use "neural network", instead of "recent papers about neural network"

经过几次试验之后，你发现必须要输入“man-month”才能得到希望的结果。这不就是只差一个‘-’符号么？为什么这个搜索引擎不能做得聪明一些，给一些提示 (Query Suggestion)？或者自动把用户想搜的结果展现出来 (Query Alteration)？我们在输入比较长的英文单词的时候，也难免会敲错一两个字母，网站应该帮助用户，而不是冷冰冰地拒绝用户啊。

微软的学术搜索 (Microsoft Academic Search) 索引了超过 3 千万的文献，2 千万的人名，怎么能以比较小的代价，对经常出现的输入错误提供提示？或直接显示相关结果，

避免用户反复尝试输入的烦恼？

你可能会说，这很难吧，但是另一家搜索引擎似乎轻易地解决了这个问题（谷歌，读者可以一试）。所以，还是有办法的。

这个题目要求你：

- 1) 试验不同的输入，反推出目前微软的学术搜索是如何实现搜索建议（Query Suggestion）的。
- 2) 提出自己的改进建议，并论证这个解决方案在千万级数据规模上能达到“足够好”的时间（speed）和空间（memory usage）效率。
- 3) 估计这事需要几个 人·月（man-month）才能做完？（备注：顺便给邹欣老师传个话，如果应届毕业生可以能做好上述全部三个题目，便可直接找他。<http://www.cnblogs.com/xinz/archive/2011/10/10/2205232.html>）。

25. 今天 10.10 阿里云部分笔试题目：

- 1、一个树被序列化为数组，如何反序列化。
- 2、如何将 100 百万有序数据最快插入到 STL 的 map 里。
- 3、有两个线程 a、b 分别往一条队列 push 和 pop 数据，在没有锁和信号量的情况下如何避免冲突访问。
- 4、写一个函数，功能是从字符串 s 中查找出子串 t，并将 t 从 s 中删除。

26. 将长度为 m 和 n 的两个升序数组复制到长度为 m+n 的数组里，升序排列。

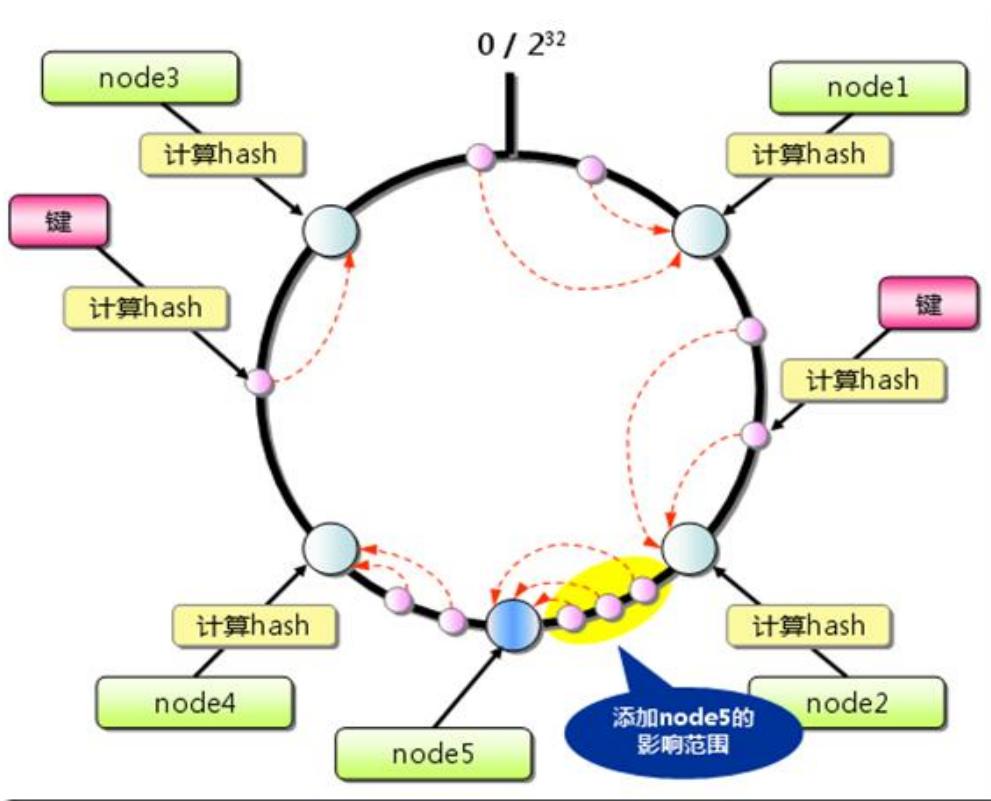
27. tencent2012 笔试题附加题

问题描述：例如手机朋友网有 n 个服务器，为了方便用户的访问会在服务器上缓存数据，因此用户每次访问的时候最好能保持同一台服务器。

已有的做法是根据 ServerIPIndex[QQNUM%n] 得到请求的服务器，这种方法很方便将用户分到不同的服务器上去。但是如果一台服务器死掉了，那么 n 就变为了 n-1，那么 ServerIPIndex[QQNUM%n] 与 ServerIPIndex[QQNUM%(n-1)] 基本上都不一样了，所以大多数用户的请求都会转到其他服务器，这样会发生大量访问错误。

问：如何改进或者换一种方法，使得：

- (1) 一台服务器死掉后，不会造成大面积的访问错误，
- (2) 原有的访问基本还是停留在同一台服务器上；
- (3) 尽量考虑负载均衡。（思路：往分布式一致哈希算法方面考虑。关于此算法，可参见此文：<http://blog.csdn.net/21aspnet/article/details/5780831>）



28. 腾讯面试题：A.txt 和 B.txt 两个文件，A.txt 有 1 亿个 QQ 号，B.txt 100W 个 QQ 号，用代码实现交、并、差。

29. 说出下面的运行结果

```
#include <iostream>
using namespace std;
class A
{
public:
    virtual void Fun(int number = 10)
    {
        std::cout << "A::Fun with number " << number<<endl;
    }
};
class B: public A
{
public:
    virtual void Fun(int number = 20)
    {
        std::cout << "B::Fun with number " << number<<endl;
    }
};
int main()
{
```

```
B b;  
A &a = b;  
a.Fun();  
return 0;  
} //虚函数动态绑定=>B, 非 A, 缺省实参是编译时候确定的=>10, 非 20。
```

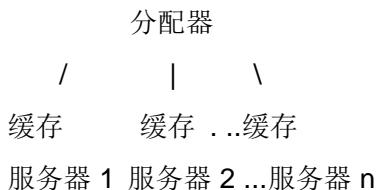
30.今晚阿里云笔试：有 101 根电线 每根的一头在楼底 另一端在楼顶 有一个灯泡 一个电池 无数根很短的电线 怎么样在楼上一次在楼下去一次将电线的对应关系弄清楚。

31.金山笔试题：

- 1、C++为什么经常将析构函数声明为虚函数？
- 2、inline 和#define 的如何定义 MAX，区别是什么。
- 3、const 的用法，如何解除 const 限制。
- 4、智能指针的作用和设计原理。
- 5、STL 中 vector 如何自己设计，关键设计点，函数声明，自定义删除重复元素的函数。
- 6、如何用一条 SQL 语句，删除表中某字段重复的记录。

32.淘宝：

在现代 web 服务系统的设计中，为了减轻源站的压力，通常采用分布式缓存技术，其原理如下图所示，前端的分配器将针对不同内容的用户请求分配给不同的缓存服务器向用户提供服务。



- 1) 请问如何设置分配策略，可以保证充分利用每个缓存服务器的存储空间（每个内容只在一个缓存服务器有副本）
- 2) 当部分缓存服务器故障，或是因为系统扩容，导致缓存服务器的数量动态减少或增加时，你的分配策略是否可以保证较小的缓存文件重分配的开销，如果不能，如何改进？
- 3) 当各个缓存服务器的存储空间存在差异时（如有 4 个缓存服务器，存储空间比为 4:9:15:7），如何改进你的策略，按照如上的比例将内容调度到缓存服务器？（思路：往 memcached 或者一致性 hash 算法方面考虑，但具体情况，具体分析。）

33.腾讯：50 个台阶，一次可一阶或两阶，共有几种走法（老掉牙的题了，详见微软面试 100 题 2010 版）。

```
long Fibonacci_Solution1(unsigned int n)  
{  
    int result[2] = {0, 1};  
    if(n < 2)  
        return result[n];
```

```
    return Fibonacci_Solution1(n - 1) + Fibonacci_Solution1(n - 2);
}
```

34. 有两个 float 型的数，一个为 fmax, 另一个为 fmin, 还有一个整数 n, 如果 $(fmax - fmin)/n$, 不能整除，怎么改变 fmax, fmin, 使改变后可以整除 n。

35. 2011.10.11 最新百度面试：

1、动态链接库与静态链接库的区别（静态链接库是.lib 格式的文件，一般在工程的设置界面加入工程中，程序编译时会把.lib 文件的代码加入你的程序中因此会增加代码大小，你的程序一运行.lib 代码强制被装入你程序的运行空间，不能手动移除.lib 代码。

动态链接库是程序运行时动态装入内存的模块，格式*.dll，在程序运行时可以随意加载和移除，节省内存空间。

在大型的软件项目中一般要实现很多功能，如果把所有单独的功能写成一个个.lib 文件的话，程序运行的时候要占用很大的内存空间，导致运行缓慢；但是如果将功能写成.dll 文件，就可以在用到该功能的时候调用功能对应的.dll 文件，不用这个功能时将.dll 文件移除内存，这样可以节省内存空间。）

2、指针与引用的区别（相同点：1. 都是地址的概念；

指针指向一块内存，它的内容是所指内存的地址；引用是某块内存的别名。

区别：

1. 指针是一个实体，而引用仅是个别名；
2. 引用使用时无需解引用(*), 指针需要解引用；
3. 引用只能在定义时被初始化一次，之后不可变；指针可变；
4. 引用没有 const, 指针有 const;
5. 引用不能为空，指针可以为空；
6. “sizeof 引用”得到的是所指向的变量(对象)的大小，而“sizeof 指针”得到的是指针本身(所指向的变量或对象的地址)的大小；
7. 指针和引用的自增(++)运算意义不一样；
8. 从内存分配上看：程序为指针变量分配内存区域，而引用不需要分配内存区域。）

3、进程与线程的区别（①从概念上：

进程：一个程序对一个数据集的动态执行过程，是分配资源的基本单位。

线程：一个进程内的基本调度单位。

线程的划分尺度小于进程，一个进程包含一个或者更多的线程。

②从执行过程中来看：

进程：拥有独立的内存单元，而多个线程共享内存，从而提高了应用程序的运行效率。

线程：每一个独立的线程，都有一个程序运行的入口、顺序执行序列、和程序的出口。

但是线程不能够独立的执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

③从逻辑角度来看：(重要区别)

多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但是，操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理及资源分配。)

4、函数调用入栈出栈的过程

5、c++对象模型与虚表

7、海量数据处理，以及如何解决 Hash 冲突等问题

8、系统设计，概率算法

36.今天腾讯面试：

一个大小为 N 的数组，里面是 N 个整数，怎样去除重复，

要求时间复杂度为 $O(n)$ ，空间复杂度为 $O(1)$ （此题答案请见@作者 hawksoft：

<http://blog.csdn.net/hawksoft/article/details/6867493>）。

37.一个长度为 10000 的字符串，写一个算法，找出最长的重复子串，如 abczzacbc，结果

是 bc（思路：后缀树/数组的典型应用，@well：就是求后缀数组的 height[] 的最大值）。

38.今晚 10.11 大华笔试题：建立一个 data structure 表示没有括号的表达式，而且找出所有等价（equivalent）的表达式

比如：

$3 \times 5 == 5 \times 3$

$2+3 == 3+2$

39.今晚 10.11 百度二面：判断一个数的所有因数的个数是偶数还是奇数（只需要你判断因

数的个数是偶数个还是奇数个，那么可以这么做@滨湖&&土豆：那只在计算质因数的过程中统计一下当前质因数出现的次数，如果出现奇数次则结果为偶，然后可以立即返回；如果每个质因数的次数都是偶数，那么结果为奇。如果该数是平方数 结果就为奇 否则就为偶了）。

40.比如 A 认识 B，B 认识 C，但是 A 不认识 C，那么称 C 是 A 的二度好友。找出某个人的所有十度好友。数据量为 10 万（BFS，同时记录已遍历过的顶点，遍历时遇到的已遍历过的顶点不插入队列。此是今晚 10.11 人人笔试题目，但它在上个月便早已出现在本人博客中，即此文第 23 题第 2 小题：[九月腾讯，创新工场，淘宝等公司最新面试十三题](#)）。

41.map 在什么情况下会发生死锁；stl 中的 map 是怎么实现的？（有要参加淘宝面试的朋友注意，淘宝喜欢问 STL 方面的问题）

42.昨日笔试：有四个人，他们每次一起出去玩的时候，用同时剪刀包袱锤的方式决定谁请客。设计一种方法，使得他们只需出一次，就可以决定请客的人，并且每个人请客的几率相同，均为 25%。

43. Given two sets of n numbers $a_1, a_2 \dots, a_n$ and $b_1, b_2 \dots, b_n$, find, in polynomial time, a permutation Π such that $\sum_i |a_i - b_{\Pi(i)}|$ is minimized? Prove your algorithm works.

有两个数组，在多项式时间里找到使 两数组元素 的差 的绝对值 的和 最小 的一种置换。并证明算法的有效性。注意，关键是证明。(此题个人去年整理过类似的一题，详见微软面试 100 题 2010 版第 32 题：http://blog.csdn.net/v_JULY_v/archive/2011/01/10/6126444.aspx)

44. 对已排好序的数组 A，一般来说可用二分查找 可以很快找到。

现有一特殊数组 A[], 它是循环递增的，如 $A[] = \{17 19 20 25 1 4 7 9\}$,

试在这样的数组中找一元素 x，看看是否存在。

请写出你的算法，必要时可写伪代码，并分析其空间 时间复杂度。

45. 网易：题意很简单，写一个程序，打印出以下的序列。

(a),(b),(c),(d),(e).....(z)

(a,b),(a,c),(a,d),(a,e).....(a,z),(b,c),(b,d).....(b,z),(c,d).....(y,z)

(a,b,c),(a,b,d)....(a,b,z),(a,c,d)....(x,y,z)

....

(a,b,c,d,.....x,y,z) (思路：全排列问题)

46.

```
int global = 0;

// thread 1
for(int i = 0; i < 10; ++i)
    global -= 1;

// thread 2
for(int i = 0; i < 10; ++i)
    global += 1;
```

之后 global 的可能的值是多少（多种可能）？

47. 今天 10.13 新浪笔试：

1、用隐喻说明 class 和 object 的区别，要求有新意。

2、DDL, DML, DCL 的含义，和距离

3、TCP 建立连接的三次握手

4、设计人民币面值，要求种类最好，表示 1——1000 的所有数，平均纸币张数最少

5、UML

48. 一个数组。里面的数据两两相同，只有两个数据不同，要求找出这两个数据。要求时间复杂度 O (N) 空间复杂度 O (1)。

49. 两个数相乘，小数点后位数没有限制，请写一个高精度算法。

50. 面试基础题：

- 1、静态方法里面为什么不能声明静态变量？
- 2、如果让你设计一个类，什么时候把变量声明为静态类型？
- 3、抽象类和接口的具体区别是什么？

51. 谷歌昨晚 10.13 算法笔试三题：

1. 一个环形公路，上面有 N 个站点，A₁, ..., A_N，其中 A_i 和 A_{i+1} 之间的距离为 D_i，A₁ 和 A₁ 之间的距离为 D₀。高效的求第 i 和第 j 个站点之间的距离，空间复杂度不超过 O(N)。它给出了部分代码如下：

```
#define N 25
double D[N]
...
void Preprocess()
{
    //Write your code1;
}
double Distance(int i, int j)
{
    //Write your code2;
}
```

2. 一个字符串，压缩其中的连续空格为 1 个后，对其中的每个字串逆序打印出来。比如 "abc efg hij" 打印为 "cba gfe jih"。

3. 将一个较大的钱，不超过 1000000(10⁶) 的人民币，兑换成数量不限的 100、50、10、5、2、1 的组合，请问共有多少种组合呢？（其它选择题考的是有关：操作系统、树、概率题、最大生成树有关的题，另外听老梦说，谷歌不给人霸笔的机会。）。

52. 谷歌在线笔试题：

输入两个整数 A 和 B，输出所有 A 和 B 之间满足指定条件的数的个数。指定条件：假设 C=8675 在 A 跟 B 之间，若 $(8+6+7+5)/4 > 7$ ，则计一个，否则不计。

要求时间复杂度：log(A)+log(B)。

已知二叉树的前序遍历为： - + a * b - c d / e f
后序遍历为： a b c d - * + e f / -
求其中序遍历？

53.

54. 十五道百度、腾讯面试基础测试题 @fengchaokobe：

1、写一个 C 的函数，输入整数 N，输出整数 M，M 满足：M 是 2 的 n 次方，且是不大于 N 中最大的 2 的 n 次方。例如，输入 4,5,6,7，都是输出 4。

- 2、C++中虚拟函数的实现机制。
- 3、写出选择排序的代码及快速排序的算法。
- 4、你认为什么排序算法最好？
- 5、tcp/ip 的那几层协议，IP 是否是可靠的？为什么？
- 6、进程和线程的区别和联系，什么情况下用多线程，什么时候用多进程？
- 7、指针数组和数组指针的区别。
- 8、查找单链表的中间结点。
- 9、最近在实验室课题研究或工作中遇到的技术难点，怎么解决的？
- 10、`sizeof` 和 `strlen` 的区别。
- 11、`malloc-free` 和 `new-delete` 的区别
- 12、大数据量中找中位数。
- 13、堆和栈的区别。
- 14、描述函数调用的整个过程。
- 15、在一个二维平面上有三个不在一条直线上的点。请问能够作出几条与这些点距离相同的线？

55. 搜狐的一道笔试题：

```
char *s="mysohu";
s[0]=0; //..
printf("%s",s);
```

输出是什么啊？

搜狐的一道大题：

数组非常长，如何找到第一个只出现一次的数字，说明算法复杂度。（与个人之前整理的微软面试 100 题中，第 17 题：在一个字符串中找到第一个只出现一次的字符。类似，读者可参考。）

56. 百度笔试 3. 假设有一台迷你计算机，1KB 的内存，1MHZ 的 cpu，已知该计算机执行的程序可出现确定性终止(非死循环)，问如何求得这台计算机上程序运行的最长时间，可以做出任何大胆的假设。

57. 微软 10.15 笔试：对于一个数组{1,2,3}它的子数组有{1,2}, {1,3}{2,3}, {1,2,3}，元素之间可以不是连续的，对于数组{5,9,1,7,2,6,3,8,10,4}，升序子序列有多少个？或者换一种表达为：数组 int a[]={5,9,1,7,2,6,3,8,10,4}。求其所有递增子数组(元素相对位置不变)的个数，例如：{5, 9}, {5, 7, 8, 10}, {1, 2, 6, 8}。

58. 今日腾讯南京笔试题：

M*M 的方格矩阵，其中有一部分为障碍，八个方向均可以走，现假设矩阵上有 Q+1 节点，从(X0, Y0)出发到其他 Q 个节点的最短路径。

其中， $1 \leq M \leq 1000$, $1 \leq Q \leq 100$ 。

59. 另外一个笔试题：

一个字符串 **S1**: 全是由不同的字母组成的字符串如: abcdefghijklmn

另一个字符串 **S2**: 类似于 **S1**, 但长度要比 **S1** 短。

问题是, 设计一种算法, 求 **S2** 中的字母是否均在 **S1** 中。(字符串包含问题, 详见程序员编程艺术系列第二章:http://blog.csdn.net/v_JULY_v/article/details/6347454)。

60. 检索一英语全文, 顺序输出检测的单词和单词出现次数。

61. 今天 10.15 下午网易游戏笔试题：给一个有序数组 **array[n]**, 和一个数字 **m**, 判断 **m** 是否是这些数组里面的数的和。(类似于微软面试 100 题 2010 年版第 4 题, 即相当于给定一棵树, 然后给定一个数, 要求把那些 相加的和等于这个数的 所有节点打印出来)。

62. 一个淘宝的面试题

文件 A:

uid username

文件 B:

username password

文件 A 是按照 uid 有序排列的, 要求有序输出合并后的 A,B 文件, 格式为 uid username password (AB 两个文件都很大, 内存装不下。)

63. 百度可能会问问 memcached (可下载此份文档看看: <http://tech.idv2.com/2008/08/17/memcached-pdf/>。源码下载地址: <http://www.oschina.net/p/memcached>), apache 之类的。

64. 今上午 10.16 百度笔试: 1.C++ STL 里面的 vector 的实现机制,

(1) 当调用 **push_back** 成员函数时, 怎么实现? (粗略的说@owen, 内存足则直接 **placement new** 构造对象, 否则扩充内存, 转移对象, 新对象 **placement new** 上去。具体的参见此文: http://blog.csdn.net/v_july_v/article/details/6681522)

(2) 当调用 **clear** 成员函数时, 做什么操作, 如果要释放内存该怎么做。(调用析构函数, 内存不释放。 **clear** 没有释放内存, 只是将数组中的元素置为空了, 释放内存需要 **delete**。)

2. 函数 **foo** 找错, 该函数的作用是将一个字符串中的 a-z 的字母的频数找出来

```
void foo(char a[100], int cnt[256])
{
    memset(cnt, 0, sizeof(cnt));
    while (*a != '\0')
    {
        ++cnt[*a];
```

```

        ++a;
    }
    for ( char c='a';c<='z';++c)
    {
        printf("%c:%d\n",c,cnt[c]);
    }
}
int main()
{
    char a[100]="百度 abc";
    int cnt[256];
    foo(a,cnt);
    return 0;
}

```

```

linux-6v95:/home/owenliang/csdn/cAndCpp # cat main.cpp
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void foo(char a[100], int cnt[256])
{
    memset(cnt,0,sizeof(int)*256);

    unsigned char *p=(unsigned char*)a;

    while( (*p)!='\0' )
    {
        ++cnt[*p++];      //保证*p在0-255内
    }

    for(char c='a';c<='z';++c)
    {
        printf("(%c,%d)\n",c,cnt[c]);
    }
}

int main()
{
    char a[]="百度abc";
    int cnt[256];

    foo(a,cnt);

    return 0;
}

```

65.腾讯长沙笔试：旅行商问题。

66.今天完美 10.16 笔试题：2D 平面上有一个三角形 ABC，如何从这个三角形内部随机取一个点，且使得在三角形内部任何点被选取的概率相同。

67.不用任何中间变量，实现 strlen 函数

68.笔试：联合赋值问题：

```
#include <stdio.h>
union A{
    int i;
    char x[2];
}a;
int main()
{
    a.x[0]=10;
    a.x[1]=1;
    printf("%d\n",a.i);
    return 0;
}
sizeof(a) = sizeof(int) = 4 byte
4 * 8 = 32 bit
a = > 00000000 00000000 00000000 00000000
a.x[0]=10 => 00000000 00000000 00000000 00001010
a.x[1]=1 => 00000000 00000000 00000001 00001010
a.i = 1*256 + 1*8 + 1*2 = 256+10 = 266
```

69.昨天做了中兴的面试题：

```
struct A{
    int a;
    char b;
    char c;
};
```

问 sizeof(A) 是多大？

70.你好：

今天 5 月 6 日百度笔试，遇到一个题目，没想到比较好的思路 在网上看了不太明朗，希望你帮我解答下

题目如下：

百度研发笔试题。设子数组 $A[0:k]$ 和 $A[k+1:N-1]$ 已排好序 ($0 \leq k \leq N-1$)。试设计一个合并这 2 个子数组为排好序的数组 $A[0:N-1]$ 的算法。要求算法在最坏情况下所用的计算时间为 $O(N)$ ，只用到 $O(1)$ 的辅助空间。

若论这道题的来源，则是在高德纳的计算机程序设计艺术第三卷第五章排序中，如下(第一张图是原题，第二张图是书上附的答案)：

18.[40] (M.A.Kronrod 给定仅含两个路段的 N 个记录的一个文件

$$K_1 \leq \dots \leq K_M \quad \text{和} \quad K_{M+1} \leq \dots \leq K_N$$

有可能在一个随机存取存储器中用 $O(N)$ 个操作对这个文件排序吗? 而且不论 M 和 N 的大小如何, 只许使用少量固定的附加存储空间(本节描述的所有合并算法都使用与 N 成比例的额外存储空间)。

开始于文件。)

18. 是的, 但它似乎是一项复杂的工件。要找出的头一个解作用下列巧妙的构造。[Doklady Akad Nauk SSSR 186(1969), 1256~1258]: 设 $n \approx \sqrt{N}$ 。把这个文件分成为 $m+2$ 个“段” $Z_1 \dots Z_m Z_{m+1} Z_{m+2}$, 其中 Z_{m+2} 包含 $(N \bmod n)$ 个记录, 而每一个其它的段恰包含 n 个记录。把 Z_{m+1} 的诸记录同包含 R_M 的段进行交换; 现在这个文件有 $Z_1 \dots Z_m A$ 的形式, 其中 $Z_1 \dots Z_m$ 中的每一个恰巧包含 n 个排好序的记录, 而且这里 A 是包含 s 个记录的一个辅助区域, 其中的 s 在范围 $n \leq s < 2n$ 中。

找出具有最小前导元素的段, 而且把整个该段同 Z_1 作交换; 如果有一个以上的段有最小前导元素, 则选择有最小尾元素的一个段。(这花费 $O(m+n)$ 个操作。) 然后找出具有次小前导元素和尾元素的段, 而且把它同 Z_2 进行交换, 等等。最后, 通过 $O(m(m+n)) = O(N)$ 个操作, 我们重新安排了 m 个段, 使得它们的前导元素是有序的。而且, 由于对于该文件原来的假定, 在 $Z_1 \dots Z_m$ 中的每个键码现在都有少于 n 个反序。

我们利用下列技巧, 可以合并 Z_1 和 Z_2 : 把 Z_1 同 A 的头 n 个元素 A' 进行交换; 然后以通常方式合并 Z_2 和 A' , 但当它们被输出时与 $Z_1 Z_2$ 的元素相交换。例如, 如果 $n=3$, 且 $x_1 < y_1 < x_2 < y_2 < x_3 < y_3$, 则我们有

	段 1	段 2	辅助区域
初始值的内容:	$x_1 x_2 x_3$	$y_1 y_2 y_3$	$a_1 a_2 a_3$
交换 Z_1 :	$a_1 a_2 a_3$	$y_1 y_2 y_3$	$x_1 x_2 x_3$
交换 x_1 :	$x_1 a_2 a_3$	$y_1 y_2 y_3$	$a_1 x_2 x_3$

71. 百度实习生笔试题:

一个单词如果交换其所含字母顺序, 得到的单词称为兄弟单词, 例如 `mary` 和 `army` 是兄弟单词, 即所含字母是一样的, 只是字母顺序不同, 用户输入一个单词, 要求在一个字典中找出该单词的所有兄弟单词, 并输出。给出相应的数据结构及算法。要求时间和空间复杂度尽可能低

目前思想:

```
struct {
    char data;
    int n;
};
```

根据数学定理: 任何一个大于 1 的自然数 N , 都可以唯一分解成有限个质数的乘积

$N=(P_1^{a_1} \cdot P_2^{a_2} \cdots \cdot P_n^{a_n})$, 这里 $P_1 < P_2 < \dots < P_n$ 是质数, 且唯一。

例如

`a=2 b=3 c=5 d=7 e=11...`

`f(abcd)=2*3*5*7=210`

然后字典里找乘积 210 的位数相同的一定是这 5 个字母组合的单词就是兄弟单词

72. 更新至 2012.05.06 下午.....

更多面试题，参见[横空出世，席卷 Csdn—评微软等数据结构+算法面试 100 题](#)（在此文中，集结了本博客已经整理的 236 道面试题）。

后记

些面试题看多了，自然会发现题目类型可能会千变万化，但解决问题的思路却只有那么几种。再者，写代码的时候，很多的细节需要务必注意，如返回值，函数参数的检查，特殊情况的处理等等，这是一个代码规范性的问题。有个消息：

1. 微软面试全部 100 题的答案如今已由一朋友阿财做出，微软面试 100 题 2010 年版全部答案集锦：http://blog.csdn.net/v_july_v/article/details/6870251，供诸君参考。

ok，日后一有最新的面试题，再整理，有任何问题，欢迎在本文评论下指出或来信指导（zhoulei0907@yahoo.cn），谢谢。July、2012.05.08。

十月下旬腾讯，网易游戏，百度最新校园招聘笔试题集锦(第 271-330 题)

引言

此文十月百度，阿里巴巴，迅雷搜狗最新面试十一题已经整理了最新的面试题 70 道，本文依次整理腾讯，网易游戏，百度等各大公司最新校园招聘的笔试题，后续将继续整理十月下旬的笔/面试题。

腾讯 2011.10.15 校园招聘会笔试题

1、下面的排序算法中，初始数据集的排列顺序对算法的性能无影响的是 (B)

- A、插入排序 **B、堆排序** C、冒泡排序 D、快速排序

2、以下关于 Cache 的叙述中，正确的是 (B)

- A、CPU 中的 Cache 容量应大于 CPU 之外的 Cache 容量
B、Cache 的设计思想是在合理成本下提高命中率
C、Cache 的设计目标是容量尽可能与主存容量相等
D、在容量确定的情况下，替换算法的时间复杂度是影响 Cache 命中率的关键因素

3、数据存储在磁盘上的排列方式会影响 I/O 服务的性能，一个圆环的磁道上有 10 个物理块，10 个数据记录 R1-----R10 存放在这个磁道上，记录的安排顺序如下表所示：

物理块	1	2	3	4	5	6	7	8	9	10
逻辑记录	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10

假设磁盘的旋转速度为 20ms/周，磁盘当前处在 R1 的开头处，若系统顺序扫描后将数据放入单缓冲区内，处理数据的时间为 4ms（然后再读取下个记录），则处理这 10 个记录的最长时间为 (C)

- A、180ms B、200ms **C、204ms** D、220ms

4、随着 IP 网络的发展，为了节省可分配的注册 IP 地址，有一些地址被拿出来用于私有 IP 地址，以下不属于私有 IP 地址范围的是 (C)（私网 IP 地址：10.0.0.0-10.255.255.255；172.16.0.0 - 172.31.255.255；192.168.0.0-192.168.255.255。故选 C）

- A、10.6.207.84 B、172.23.30.28 **C、172.32.50.80** D、192.168.1.100

5、下列关于一个类的静态成员的描述中，不正确的是 (D)

- A、该类的对象共享其静态成员变量
B、静态成员变量可被该类的所有方法访问

C、该类的静态方法只能访问该类的静态成员变量

D、该类的静态数据成员变量的值不可修改

6、已知一个线性表（38，25，74，63，52，48），假定采用散列函数 $h(key) = key \% 7$ 计算散列地址，并散列存储在散列表 A【0....6】中，若采用线性探测方法解决冲突，则在该散列表上进行等概率成功查找的平均查找长度为（C）

A、1.5

B、1.7

C、2.0

D、2.3

依次进行取模运算求出哈希地址：

A	0	1	2	3	4	5	6
记录	63	48		38	25	74	52
查找次数	1	3		1	1	2	4

74 应该放在下标为 4 的位置，由于 25 已经放在这个地方，所以 74 往后移动，放在了下标为 5 的位置上了。由于是等概率查找，所以结果为： $1/6 * (1+3+1+1+2+4) = 2.0$

7、表达式“X=A+B*（C--D）/E”的后缀表示形式可以为（C）

A、XAB+CDE/-*=

B、XA+BC-DE/*=

C、XABCD-*E/+=

D、XABCDE+*/=

8、（B）设计模式将抽象部分与它的实现部分相分离。

A、Singleton（单例）

B、Bridge（桥接）

C、Composite（组合）

D、Facade（外观）

9、下面程序的输出结果为多少？

```
void Func(char str_arg[100])
{
    printf("%d\n", sizeof(str_arg));
}

int main(void)
{
    char str[]="Hello";
    printf("%d\n", sizeof(str));
    printf("%d\n", strlen(str));
    char *p = str;
    printf("%d\n", sizeof(p));
    Func(str);
}
```

输出结果为： 6 5 4 4

对字符串进行 `sizeof` 操作的时候，会把字符串的结束符“\0”计算进去的，进行 `strlen` 操作求字符串的长度的时候，不计算\0的。

数组作为函数参数传递的时候，已经退化为指针了，Func 函数的参数 str_arg 只是表示一个指针，那个 100 不起任何作用的。

10、下面程序的输出结果是多少？

```
void Func(char str_arg[2])
{
    int m = sizeof(str_arg);           //指针的大小为 4
    int n = strlen(str_arg);          //对数组求长度，str_arg 后面的那个 2 没有任何意义，数组已经退化为指针了
    printf("%d\n",m);
    printf("%d\n",n);
}
int main(void)
{
    char str[]="Hello";
    Func(str);
}
```

输出结果为： 4 5

strlen 只是对传递给 Func 函数的那个字符串求长度，跟 str_arg 中的那个 2 是没有任何关系的，即使把 2 改为 200 也是不影响输出结果的。。

11、到商店里买 200 的商品返还 100 优惠券（可以在本商店代替现金）。请问实际上折扣是多少？

算法编程题：

1、给定一个字符串，求出其最长的重复子串。

思路：使用后缀数组，对一个字符串生成相应的后缀数组后，然后再排序，排完序依次检测相邻的两个字符串的开头公共部分。

这样的时间复杂度为：

生成后缀数组 $O(N)$

排序 $O(N \log N * N)$ 最后面的 N 是因为字符串比较也是 $O(N)$

依次检测相邻的两个字符串 $O(N * N)$

总的时间复杂度是 $O(N^2 \log N)$,

网易游戏 2011.10.15 校园招聘会笔试题

1、对于一个内存地址是 32 位、内存页是 8KB 的系统。0X0005F123 这个地址的页号与页内偏移分别是多少。

2、如果 X 大于 0 并小于 65536，用移位法计算 X 乘以 255 的值为： $-X + X \ll 8$

X<<8-X 是不对的，**X<<8**，已经把 X 的值改变了（订正：**X<<8** 是个临时变量，不会改变 X 的值，就像 **a+1** 不会改变 **a** 一样）。

3、一个包含 n 个节点的四叉树，每个节点都有四个指向孩子节点的指针，这 $4n$ 个指针中有 **3n+1** 个空指针。

4、以下两个语句的区别是：

```
int *p1 = new int[10];
int *p2 = new int[10]();
```

5、计算机在内存中存储数据时使用了大、小端模式，请分别写出 **A=0X123456** 在不同情况下的首字节是，大端模式：**0X12** 小端模式：**0X56** X86 结构的计算机使用 **小端** 模式。一般来说，大部分用户的操作系统（如 windows, FreeBSD, Linux）是小端模式的。少部分，如 **MAC OS**，是大端模式的。

6、在游戏设计中，经常会根据不同的游戏状态调用不同的函数，我们可以通过函数指针来实现这一功能，请声明一个参数为 **int ***，返回值为 **int** 的函数指针：

```
int (*fun)(int *)
```

7、在一冒险游戏里，你见到一个宝箱，身上有 N 把钥匙，其中一把可以打开宝箱，假如没有任何提示，随机尝试，问：

(1) 恰好第 K 次 ($1 \leq K \leq N$) 打开宝箱的概率是多少。

(2) 平均需要尝试多少次。

百度 2011.10.16 校园招聘会笔试题

一、算法设计

1、设 **rand (s, t)** 返回 [s,t] 之间的随机小数，利用该函数在一个半径为 R 的圆内找随机 n 个点，并给出时间复杂度分析。

2、为分析用户行为，系统常需存储用户的一些 **query**，但因 **query** 非常多，故系统不能全存，设系统每天只存 m 个 **query**，现设计一个算法，对用户请求的 **query** 进行随机选择 m 个，请给一个方案，使得每个 **query** 被抽中的概率相等，并分析之，注意：不到最后一刻，并不知用户的总请求量。

3、C++ STL 中 **vector** 的相关问题：

(1)、调用 **push_back** 时，其内部的内存分配是如何进行的？

(2)、调用 **clear** 时，内部是如何具体实现的？若想将其内存释放，该如何操作？

二、系统设计

正常用户端每分钟最多发一个请求至服务端，服务端需做一个异常客户端行为的过滤系统，

设服务器在某一刻收到客户端 A 的一个请求，则 1 分钟内的客户端任何其它请求都需要被过滤，现知每一客户端都有一个 IPv6 地址可作为其 ID，客户端个数太多，以至于无法全部放到单台服务器的内存 hash 表中，现需简单设计一个系统，使用支持高效的过滤，可使用多台机器，但要求使用的机器越少越好，请将关键的设计和思想用图表和代码表现出来。

三、求一个全排列函数：

如 p([1,2,3])输出：

[123]、[132]、[213]、[231]、[321]、[323]

求一个组合函数

如 p([1,2,3])输出：

[1]、[2]、[3]、[1,2]、[2,3]、[1,3]、[1,2,3]

这两问可以用伪代码。

迅雷 2011.10.21 笔试题

1、下面的程序可以从 1....n 中随机输出 m 个不重复的数。请填空

```
knuth(int n, int m)
{
    srand((unsigned int)time(0));
    for (int i=0; i<n; i++)
    {
        if (_____)
        {
            cout<<i<<endl;
            _____;
        }
    }
}
```

分别为：rand()% (n-i)<m 和 m--;

2、以下 prim 函数的功能是分解质因数。请填空

```
void prim(int m, int n)
{
    if (m>n)
    {
        while (_____) n++;
        _____;
        prim(m,n);
        cout<<n<<endl;
    }
}
```

分别为: $m \% n$ 和 m / n

3、下面程序的功能是输出数组的全排列。请填空

```
void perm(int list[], int k, int m)
{
    if (_____)
    {
        copy(list, list+m, ostream_iterator<int>(cout, " "));
        cout << endl;
        return;
    }
    for (int i=k; i<=m; i++)
    {
        swap(&list[k], &list[i]);
        _____;
        swap(&list[k], &list[i]);
    }
}
```

分别为: $k == m$ 和 $perm(list, k+1, m)$

二、主观题:

1、(40 分) 用户启动迅雷时，服务器会以 `uid,login_time/logout_time` 的形式记录用户的在线时间；用户在使用迅雷下载时，服务器会以 `taskid,start_time/finish_time` 的形式记录任务的开始时间和结束时间。有效下载时间是指用户在开始时间和结束时间之间的在线时间，由于用户可能在下载的时候退出迅雷，因此有效下载时间并非 `finish_time` 和 `start_time` 之差。假设登录记录保存在 `login.txt` 中，每一行代表用户的上下线记录；下载记录保存在 `task.txt` 中，每一行代表一个任务记录，记录的字段之间以空格分开。计算每个用户的有效下载时间和总在线时间的比例。注意：请尽量使用 **STL** 的数据结构和算法

2、(60 分) 在 8×8 的棋盘上分布着 n 个骑士，他们想约在某一个格中聚会。骑士每天可以像国际象棋中的马那样移动一次，可以从中间像 8 个方向移动（当然不能走出棋盘），请计算 n 个骑士的最早聚会地点和要走多少天。要求尽早聚会，且 n 个人走的总步数最少，先到聚会地点的骑士可以不再移动等待其他的骑士。

从键盘输入 n ($0 < n \leq 64$)，然后一次输入 n 个骑士的初始位置 x_i, y_i ($0 \leq x_i, y_i \leq 7$)。屏幕输出以空格分隔的三个数，分别为聚会点 (x, y) 以及走的天数。

盛大游戏 2011.10.22 校园招聘会笔试试题

1、下列代码的输出为：

```
#include "iostream"
#include "vector"
```

```

using namespace std;

int main(void)
{
    vector<int>array;
    array.push_back(100);
    array.push_back(300);
    array.push_back(300);
    array.push_back(500);
    vector<int>::iterator itor;
    for(itor=array.begin();itor!=array.end();itor++)
    {
        if(*itor==300)
        {
            itor = array.erase(itor);
        }
    }
    for(itor=array.begin();itor!=array.end();itor++)
    {
        cout<<*itor<<" ";
    }
    return 0;
}

```

- A、100 300 300 500 B、100 300 500 C、100 500 D、程序错误

vector 在 erase 之后，指向下一个元素的位置，其实进行 erase 操作时将后面所有元素都向前移动，迭代器位置没有移动。itor=array.erase(itor) erase 返回下一个元素的地址，相当于给 itor 一个新值。

2、下列代码的输出为：

```

class CParent
{
public:
    virtual void Intro()
    {
        printf("I'm a Parent, ");
        Hobby();
    }
    virtual void Hobby()
    {
        printf("I like football!");
    }
};

```

```

class CChild:public CParent
{
public:
    virtual void Intro()
    {
        printf("I'm a Child, ");
        Hobby();
    }
    virtual void Hobby()
    {
        printf("I like basketball!\n");
    }
};

int main(void)
{
    CChild *pChild = new CChild();
    CParent *pParent = (CParent*)pChild;
    pParent->Intro();
    return 0;
}

```

A、I'm a Child,I like football!

B、I'm a Child,I like basketball!

C、I'm a Parent,I like football!

D、I'm a Parent,I like basketball!

3、在 win32 平台下，以下哪种方式无法实现进程同步？

A、CriticalSection B、Event C、Mutex D、Semaphore

4、以下哪句的说法是正确的

A、在页式存储管理中，用户应将自己的程序划分为若干个相等的页

B、所有的进程都挂起时，系统将陷入死锁

C、执行系统调用可以被中断

D、进程优先数是进程调度的重要依据，必须根据进程运行情况动态改变

5、以下描述正确的是

A、虚函数是可以内联的，可以减少函数调用的开销提高效率

B、类里面可以同时存在函数名和参数都一样的虚函数和静态函数

C、父类的析构函数是非虚的，但是子类的析构函数是虚的，`delete` 子类对象指针会调用父类的析构函数

D、以上都不对

简答题：快速排序的思想是递归的，但是它的平均效率却是众多排序算法中最快的，为什么？

请结合本例说明你对递归程序的理解。

算法题：用你熟悉的编程语言，设计如下功能的函数：输入一个字符串，输出该字符串中所

有字母的全排列。程序请适当添加注释。

C++函数原型: void Print (const char *str)

输入样例: abc

输出结果: abc、acb、bca、bac、cab、cba

(以上部分整理自此君博客: <http://blog.csdn.net/Hackbuteer1>。十分感谢。有何不妥之处,还望海涵海涵。)

后续整理

1. 12个工厂分布在一条东西向高速公路的两侧,工厂距离公路最西端的距离分别是0、4、5、10、12、18、27、30、31、38、39、47.在这12个工厂中选取3个原料供应厂,使得剩余工厂到最近的原料供应厂距离之和最短,问应该选哪三个厂?

7、下面程序运行后的结果为: to test something

```
01. char str[] = "glad to test something";
02.     char *p = str;
03.     p++;
04.     int *p1 = static_cast<int *>(p);
05.     p1++;
06.     p = static_cast<char *>(p1);
07.     printf("result is %s\n",p);
```

2.

3. hash冲突时候的解决方法?

- 1)、开放地址法
- 2)、再哈希法
- 3)、链地址法
- 4)、建立一个公共溢出区

4.

```
int main()
{
    if()
    {
        printf("Hello ");
    }
    else
    {
        printf("World !!!");
    }
}
```

```

    }
    return 0;
}

```

在 if 里面请写入语句 使得打印出 hello world。

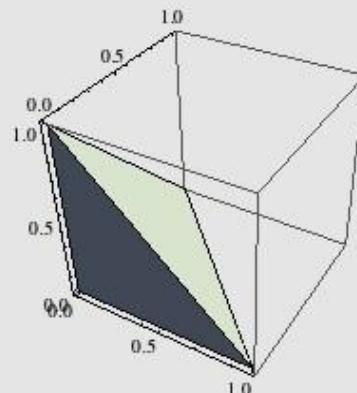
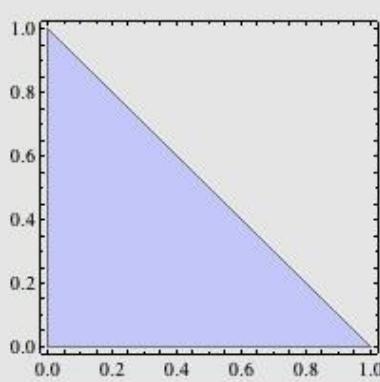
5. 今天 10.19 西山居笔试题:

分别写一个宏和函数来获取元素个数 如 count(a) 会得到 a 数组元素个数。

6. 平均要取多少个(0,1)中的随机数才能让和超过 1。(答案: e 次, 其中 e 是自然对数的底数)

为了证明这一点, 让我们先来看一个更简单的问题: 任取两个 0 到 1 之间的实数, 它们的和小于 1 的概率有多大? 容易想到, 满足 $x+y<1$ 的点 (x, y) 占据了正方形 $(0, 1) \times (0, 1)$ 的一半面积, 因此这两个实数之和小于 1 的概率就是 $1/2$ 。类似地, 三个数之和小于 1 的概率则是 $1/6$, 它是平面 $x+y+z=1$ 在单位立方体中截得的一个三棱锥。这个 $1/6$ 可以利用截面与底面的相似比关系, 通过简单的积分求得:

$$\int_{(0..1)} (x^2)^{1/2} dx = 1/6$$



可以想到, 四个 0 到 1 之间的随机数之和小于 1 的概率就等于四维立方体一角的“体积”, 它的“底面”是一个体积为 $1/6$ 的三维体, 在第四维上对其进行积分便可得到其“体积”

$$\int_{(0..1)} (x^3)^{1/6} dx = 1/24$$

依此类推, n 个随机数之和不超过 1 的概率就是 $1/n!$, 反过来 n 个数之和大于 1 的概率就是 $1 - 1/n!$, 因此加到第 n 个数才刚好超过 1 的概率就是

$$(1 - 1/n!) - (1 - 1/(n-1)!) = (n-1)/n!$$

因此, 要想让和超过 1, 需要累加的期望次数为

$$\sum_{n=2..\infty} n * (n-1)/n! = \sum_{n=1..\infty} n/n! = e$$

7. 今天支付宝 10.20 笔试题: 汉诺塔一共为 2^N , 2 个一样大小, 有编号顺序 每次只能移动一个 大的不能叠在小得上面 移动完之后, 相同大小的编号必须和原来一样 问最小要移动多少次? 如 A1 A2 B1 B2 C1 C2 这样叠, A<B<C.... B 不能放 A 上面, C 不能放 B A 上面, 移动到另外一个柱子后, 还必须是 A1 A2 B1 B2 C1 C2

8. socket 编程的问题

TCP 连接建立后，调用 `send` 5 次，每次发 100 字节，问 `recv` 最少要几次，最多要几次？

9. 迅雷笔试题：

下面的程序可以从 1....n 中随机输出 m 个不重复的数。请填空

```
knuth(int n, int m)
{
    srand((unsigned int)time(0));
    for (int i=0; i<n; i++)
        if (           )
    {
        cout<<i<<endl;
        (           );
    }
}
```

10. 四个线程 t1,t2,t3,t4,向 4 个文件中写入数据，t1 只能写入 1，t2 只能写入 2，t3 只能写入 3，t4 只能写入 4，对 4 个文件 A, B, C, D 写入如下内容

A:123412341234.....

B:234123412341....

C:341234123412....

D:412341234123....

怎么实现同步可以让线程并行工作？

11. 比如一个数组[1,2,3,4,6,8,9,4,8,11,18,19,100]

前半部分是是一个递增数组，后面一个还是递增数组，但整个数组不是递增数组，那么怎么最快的找出其中一个数？

12. 今日 10.21 迅雷笔试题：

1、一棵二叉树节点的定义（和平时我们定义的一样的） 它给出了一棵二叉树的根节点 说现在怀疑这棵二叉树有问题 其中可能存在某些节点不只有一个父亲节点 现要你编写一个函数判断给定的二叉树是否存在这样的节点 存在则打印出其父亲节点返回 `true` 否则返回 `false`

打印节点形式：

[当前节点][父亲节点 1][父亲节点的父亲节点][...]]

[当前节点][父亲节点 2][父亲节点的父亲节点][...]]

2、有一亿个整数，请找出最大的 1000 个，要求时间越短越好，空间占用越少越好

13. 在频繁使用小内存时，通常会先申请一块大的内存，每次使用小内存时都从大内存里取，最后大内存使用完后一次性释放，用算法实现。

14. 今天亚马逊 A 卷校招笔试题：

输入一个字符串，如何求最大重复出现的字符串呢？比如输入 `ttafcftrgabcd`,输出结果为 `abc`, `canffcancd`,输出结果为 `can`。

15. 今天 10.22 盛大：删除模式串中出现的字符，如“`welcome to asted`”,模式串为“`aeiou`”那么得到的字符串为“`wlcm t std`”，要求性能最优。

16. 数组中的数分为两组，让给出一个算法，使得两个组的和的差的绝对值最小

数组中的数的取值范围是 $0 < x < 100$ ，元素个数也是大于 0，小于 100

比如 `a[] = {2,4,5,6,7}`,得出的两组数 `{2, 4, 6}` 和 `{5, 7}`, `abs(sum(a1)-sum(a2))=0`;

比如 `{2, 5, 6, 10}`, `abs (sum(2,10)-sum(5,6))=1`,所以得出的两组数分别为 `{2, 10}` 和 `{5, 6}`。

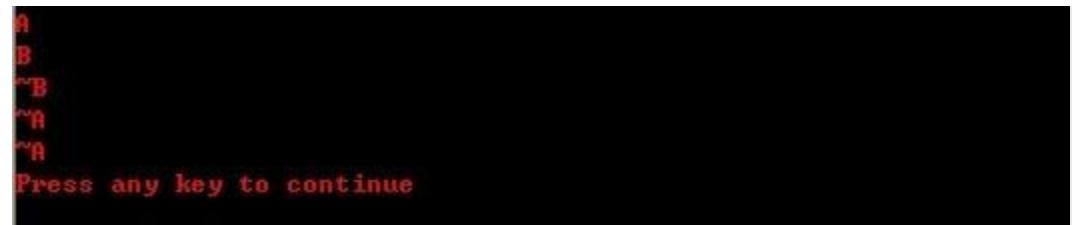
17. 百度北京研发一道系统设计题，如何快速访问 `ipv6` 地址呢？`ipv6` 地址如何存放？

18. 百度 2012 校招北京站笔试题系统设计：正常用户端每分钟最多发一个请求至服务端，服务端需做一个异常客户端行为的过滤系统，设服务器在某一刻收到客户端 A 的一个请求，则 1 分钟内的客户端任何其它请求都需要被过滤，现知每一客户端都有一个 IPv6 地址可作为其 ID，客户端个数太多，以至于无法全部放到单台服务器的内存 `hash` 表中，现需简单设计一个系统，使用支持高效的过滤，可使用多台机器，但要求使用的机器越少越好，请将关键的设计和思想用图表和代码表现出来。

19.

```
#include <iostream>
using namespace std;
class A
{
public:
    A(){cout<<"A"<<endl;}
    ~A(){cout<<"~A"<<endl;}
};
class B
{
public:
    B(A &a):_a(a)
    {
        cout<<"B"<<endl;
    }
    ~B(){cout<<"~B"<<endl;}
private:
    A _a;
};
```

```
int main()
{
    A a;
    B b(a);
    return 0;
    // 构造次序和析构次序是对称的，这种题解答都是有技巧的。
    //      拷贝构造就不说了，构造过程是：
    //      A A B ，那么析构必然是对称的：B A A。
}
```



A terminal window displaying memory dump output. The output shows the following sequence of characters: A, B, ~B, ~A, ~A. Below the dump, the text "Press any key to continue" is visible.

....

ok, 以上所有任何参考答案若有问题，欢迎不吝指正。谢谢。日后，继续整理十月下旬各大 IT 公司的笔/面试题，持续更新，直到十月月底。祝所有诸君找到自己合适而满意的 offer，工作。July、2011.10.17。

最新九月百度人搜，阿里巴巴，腾讯华为京东笔试面试二十题

引言

自发表上一篇文章至今（事实上，上篇文章更新了近 3 个月之久），blog 已经停了 3 个多月，而在那之前，自开博以来的 21 个月每月都不曾断过。正如上一篇文章[支持向量机通俗导论（理解 SVM 的三层境界）](#)末尾所述：“额，blog 许久未有更新了，因为最近实在忙，无暇顾及 blog。”与此同时，工作之余，也一直在闲心研究数据挖掘：“神经网络将可能作为 [Top 10 Algorithms in Data Mining](#) 之番外篇第 1 篇，同时，k-最近邻法(k-nearest neighbor, kNN)算法谈到 kd 树将可能作为本系列第三篇。这是此系列接下来要写的两个算法，刚好项目中也要用到 KD 树”。

但很显然，若要等到下一篇数据挖掘系列的文章时，说不定要到年底去了，而最近的这段时间，9 月，正是各种校招/笔试/面试火热进行的时节，自己则希望能帮助到这些找工作的朋友，故此，怎能无动于衷，于是，3 个多月后，blog 今天更新了。

再者，虽然如我的这条微博：<http://weibo.com/1580904460/yzs72mmFZ> 所述，blog 自 10 年 10 月开通至 11 年 10 月，一年的时间内整理了 300 多道面试题(这 300 道题全部集锦在此文中第一部分：http://blog.csdn.net/v_july_v/article/details/6543438)。但毕竟那些题已经是前年或去年的了，笔试面试题虽然每年类型变化不大，但毕竟它年年推陈出新，存着就有其合理性。

OK，以下是整理自 8 月下旬至 9 月中旬各大公司的笔试面试二十题，相信一定能给正在参加各种校招的诸多朋友多少帮助，学习参考或借鉴（如果你手头上有好的笔试/面试题，欢迎通过微博私信：<http://weibo.com/julyweibo>，或邮箱：zhoulei0907@yahoo.cn 发给我，或者干脆直接评论在本文下；同时，若你对以下任何一题有任何看法.想法.思路或建议，欢迎留言评论，大家一起讨论，共同享受思考的乐趣，谢谢）。

九月百度人搜，阿里巴巴，腾讯华为京东小米笔/面试二十题

1. 9 月 11 日，京东：

谈谈你对面向对象编程的认识

2. 8 月 20 日，金山面试，题目如下：

数据库 1 中存放着 a 类数据，数据库 2 中存放着以天为单位划分的表 30 张（比如 table_20110909,table_20110910,table_20110911），总共是一个月的数据。表 1 中的 a 类数据中有一个字段 userid 来唯一判别用户身份，表 2 中的 30 张表（每张表结构相同）

也有一个字段 `userid` 来唯一识别用户身份。如何判定 `a` 类数据库的多少用户在数据库 `2` 中出现过？

来源：<http://topic.csdn.net/u/20120820/23/C6B16CCF-EE15-47C0-9B15-77497291F2B9.html>。

3. 百度实习笔试题（2012.5.6）

简答题 1

一个单词单词字母交换，可得另一个单词，如 `army->mary`，成为兄弟单词。提供一个单词，在字典中找到它的兄弟。描述数据结构和查询过程。评点：同去年 9 月份的一道题，见此文第 3 题：http://blog.csdn.net/v_july_v/article/details/6803368。

简答题 2

线程和进程区别和联系。什么是“线程安全”

简答题 3

C 和 C++ 怎样分配和释放内存，区别是什么

算法题 1

一个 `url` 指向的页面里面有另一个 `url`，最终有一个 `url` 指向之前出现过的 `url` 或空，这两种情形都定义为 `null`。这样构成一个单链表。给两条这样单链表，判断里面是否存在同样的 `url`。`url` 以亿级计，资源不足以 `hash`。

算法题 2

数组 `al[0,mid-1]` 和 `al[mid,num-1]`，都分别有序。将其 `merge` 成有序数组 `al[0,num-1]`，要求空间复杂度 $O(1)$

系统设计题

百度搜索框的 `suggestion`，比如输入北京，搜索框下面会以北京为前缀，展示“北京爱情故事”、“北京公交”、“北京医院”等等搜索词。

如何设计使得空间和时间复杂度尽量低。评点：老题，直接上 `Trie` 树+Hash，`Trie` 树的介绍见：[从 Trie 树（字典树）谈到后缀树](#)。

4. 人搜笔试

1. 快排每次以第一个作为主元，问时间复杂度是多少？($O(N \log N)$)

2. $T(N) = N + T(N/2) + T(2N)$ ，问 $T(N)$ 的时间复杂度是多少？($O(N)$)

3. 从 $(0,1)$ 中平均随机出几次才能使得和超过 1？(e)

4. 编程题：

一棵树的节点定义格式如下：

```
struct Node{
    Node* parent;
    Node* firstChild; // 孩子节点
    Node* sibling; // 兄弟节点
}
```

要求非递归遍历该树。

思路：采用队列存储，来遍历节点。

5. 算法题：

有 N 个节点，每两个节点相邻，每个节点只与 2 个节点相邻，因此， N 个顶点有 $N-1$ 条边。每一条边上都有权值 w_i ，定义节点 i 到节点 $i+1$ 的边为 w_i 。

求：不相邻的权值和最大的边的集合。

5. 人搜面试，所投职位：搜索研发工程师：面试题目回忆

1、删除字符串开始及末尾的空白符，并且把数组中间的多个空格（如果有）符转化为 1 个。

2、求数组（元素可为正数、负数、0）的最大子序列和。

3、链表相邻元素翻转，如 $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g$ ，翻转后变为： $b \rightarrow a \rightarrow d \rightarrow c \rightarrow f \rightarrow e \rightarrow g$

4、链表克隆。链表的结构为：

```
typedef struct list {
    int data; //数据字段
    list *middle; //指向链表中某任意位置元素(可指向自己)的指针
    list *next; //指向链表下一元素
} list;
```

5、100 万条数据的数据库查询速度优化问题，解决关键点是：根据主表元素特点，把主表拆分并新建副表，并且利用存储过程保证主副表的数据一致性。（不用写代码）

6、求正整数 n 所有可能的和式的组合（如： $4=1+1+1+1$ 、 $1+1+2$ 、 $1+3$ 、 $2+1+1$ 、 $2+2$ ）

7、求旋转数组的最小元素（把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。输入一个排好序的数组的一个旋转，输出旋转数组的最小元素。例如数组{3, 4, 5, 1, 2}为{1, 2, 3, 4, 5}的一个旋转，该数组的最小值为 1。）

8、找出两个单链表里交叉的第一个元素

9、字符串移动（字符串为*号和 26 个字母的任意组合，把*号都移动到最左侧，把字母移到最右侧并保持相对顺序不变），要求时间和空间复杂度最小

10、时间复杂度为 $O(1)$ ，怎么找出一个栈里的最大元素

11、线程、进程区别

12、static 在 C 和 C++ 里各代表什么含义

13、const 在 C/C++ 里什么意思

14、常用 linux 命令

15、解释 Select/Poll 模型

6. 百度，网易，阿里巴巴等面试题：<http://blog.csdn.net/hopeztm/article/category/1201028>；

7. 8月30日，网易有道面试题

```
var tt = 'aa';
function test()
{
    alert(tt);
    var tt = 'dd';
    alert(tt);
}
test();
```

8. 8月31日，百度面试题：不使用随机数的洗牌算法，详情：<http://topic.csdn.net/u/20120831/10/C837A419-DFD4-4326-897C-669909BD2086.html>;

9. 9月6日，阿里笔试题：平面上有很多点，点与点之间有可能有连线，求这个图里环的数目。

10. 9月7日，一道华为上机题：

题目描述：选秀节目打分，分为专家评委和大众评委，`score[]` 数组里面存储每个评委打的分数，`judge_type[]` 里存储与 `score[]` 数组对应的评委类别，`judge_type == 1`，表示专家评委，`judge_type == 2`，表示大众评委，`n` 表示评委总数。打分规则如下：专家评委和大众评委的分数先分别取一个平均分（平均分取整），然后，`总分 = 专家评委平均分 * 0.6 + 大众评委 * 0.4`，总分取整。如果没有大众评委，则 `总分 = 专家评委平均分`，总分取整。函数最终返回选手得分。

函数接口 `int cal_score(int score[], int judge_type[], int n)`

上机题目需要将函数验证，但是题目中默认专家评委的个数不能为零，但是如何将这种专家数目为 0 的情形排除出去。

来源：<http://topic.csdn.net/u/20120907/15/c30eead8-9e49-41c2-bd11-c277030ad17a.html>；

11. 9月8日，腾讯面试题：

假设两个字符串中所含有的字符和个数都相同我们就叫这两个字符串匹配，

比如：`abcda` 和 `adabc`，由于出现的字符个数都是相同，只是顺序不同，

所以这两个字符串是匹配的。要求高效！

又是跟上述第 3 题中简单题一的兄弟节点类似的一道题，我想，你们能想到的，这篇 blog 里：http://blog.csdn.net/v_JULY_v/article/details/6347454 都已经有了。

12. 阿里云，搜索引擎中 5 亿个 url 怎么高效存储；

13. 创新工场微博，前几天才发布的难道不少人的牛题：http://t.qq.com/iwrecruiting?pgv_ref=im.WBlog.guest&ptlang=2052；

14. 4**9 的笔试题，比较简单：

- 1.求链表的倒数第二个节点
- 2.有一个整数数组，求数组中第二大的数

15. 阿里巴巴二道题（之前第 16 题）

第一道：

对于给定的整数集合 S ，求出最大的 d ，使得 $a+b+c=d$ 。 a,b,c,d 互不相同，且都属于 S 。集合的元素个数小于等于 2000 个，元素的取值范围在 $[0, 1000]$ ，假定可用内存空间为 100MB，硬盘使用空间无限大，试分析时间和空间复杂度，找出最快的解决方法。

阿里巴巴第二道(研发类)

笔试题 1，原题大致描述有一大批数据，百万级别的。数据项内容是：用户 ID、科目 ABC 各自的成绩。其中用户 ID 为 0~1000 万之间，且是连续的，可以唯一标识一条记录。科目 ABC 成绩均在 0~100 之间。有两块磁盘，空间大小均为 512M，内存空间 64M。

- 1) 为实现快速查询某用户 ID 对应的各科成绩，问磁盘文件及内存该如何组织；
- 2) 改变题目条件，ID 为 0~10 亿之间，且不连续。问磁盘文件及内存该如何组织；
- 3) 在问题 2 的基础上，增加一个需求。在查询各科成绩的同时，获取该用户的排名，问磁盘文件及内存该如何组织。

笔试题 2：代码实现计算字符串的相似度。

16. 9月14日，小米笔试，给一个浮点数序列，取最大乘积子序列的值，例如 -2.5, 4, 0, 3, 0.5, 8, -1，则取出的最大乘积子序列为 3, 0.5, 8。

17. 9月15日，中兴面试：

小端系统

```
union{
    int i;
    unsigned char ch[2];
}Student;

int main()
{
    Student student;
    student.i=0x1420;
    printf("%d %d",student.ch[0],student.ch[1]);
    return 0;
}
```

输出结果为？（答案：32 20）

18. 一道有趣的 Facebook 面试题：

给一个二叉树，每个节点都是正或负整数，如何找到一个子树，它所有节点的和最大？

点评：

②某猛将兄：后序遍历，每一个节点保存左右子树的和加上自己的值。额外一个空间存放最大值。

③陈利人：同学们，如果你面试的是软件工程师的职位，一般面试官会要求你在短时间内写出一个比较整洁的，最好是高效的，没有什么 bug 的程序。所以，光有算法不够，还得多实践。

写完后序遍历，面试官可能接着与你讨论，**a)**. 如果要求找出只含正数的最大子树，程序该如何修改来实现？**b)**. 假设我们将子树定义为它和它的部分后代，那该如何解决？**c)**. 对于**b**，加上正数的限制，方案又该如何？总之，一道看似简单的面试题，可能能变换出各种花样。

比如，面试官可能还会再提两个要求：第一，不能用全局变量；第二，有个参数控制是否要只含正数的子树。其它的，随意，当然，编程风格也很重要。

19. 谷歌面试题：

有几百亿的整数，分布的存储到几百台通过网络连接的计算机上，你能否开发出一个算法和系统，找出这几百亿数据的中值？就是在一组排序好的数据中居于中间的数。显然，一台机器是装不下所有的数据。也尽量少用网络带宽。

20. 9月19日，IGT面试：

你走到一个分叉路口，有两条路，每个路口有一个人，一个说假话，一个说真话，你只能问其中一个人仅一个问题，如何问才能得到正确答案？点评：
答案是，问其中一个人：另一个人会说你的路口是通往正确的道路么？

21. 9月19日，创新工厂笔试题：

给定一整型数组，若数组中某个下标值大的元素值小于某个下标值比它小的元素值，称这是一个反序。

即：数组 $a[]$ ；对于 $i < j$ 且 $a[i] > a[j]$ ，则称这是一个反序。

给定一个数组，要求写一个函数，计算出这个数组里所有反序的个数。

点评：

归并排序，至于有的人说是否有 $O(N)$ 的时间复杂度，我认为答案是否定的，正如老梦所说，下限就是 $n \lg n$ ， n 个元素的数组的排列共有的排列是 $n \lg n$, $n!$ 。

22. 持续更新，待续...2012.09.19；

23.

（上述所有题目收集整理自或是我一些算法群内的面试题讨论，或朋友提供，或网络帖子，由于整理匆忙，有部分题目未注明详细来源，若以上任何一题目出自你的空间或者发的帖子而未有注明，请于本文评论中告知我，一定即刻补上，感谢诸位，谢谢）

后记

经过上面这么多笔试面试题目的了解，你自会看到，除了少数几个特别难的算法题，大部分都是考察的基础，故校招笔试面试的关键是你的 80% 的基础知识和编程实践能力 + 20% 的算法能力（特别强调算法能力的则此项比例加大）。

再强调一下开头所述的一两点：

1. 如果你有好的笔试面试题，欢迎通过私信或邮件提供给我统一整理出来（对于好的题目提供者，你尽可以在私信：<http://weibo.com/julyweibo>，或邮件：zhoulei0907@yahoo.cn，里提出你的要求，如贴出你的微博昵称，或个人主页，或免费回赠编程艺术+算法研究的两个 PDF 文档：<http://weibo.com/1580904460/yzpYDAuYz>），以供他人借阅；
2. 如果你对以上任何一题有好的思路或解法，更欢迎不吝分享，**show me your answer or code !**

当然，若你对以上任何一题有疑问，而原文中没有给出详细答案，也欢迎在评论中告知，我会补上，大家也可以一起讨论。**thank you.**

OK，以上的一切都是我喜欢且我乐意做的，我愿同各位朋友享受这一切。(如果你身边有正在参加校招/笔试/面试的朋友，欢迎把此文转给他/她，举手之劳，助人无限)，谢谢。完。
July，二零一二年九月。

结语

感谢本 PDF 文档的制作者小龟，他的微博主页是：<http://weibo.com/guicomeon>，他是最早在微博上响应我帮忙制作本微软面试 100 题系列 PDF 文档的：<http://weibo.com/1580904460/yAvqj8tDN>；

感谢所有在微软面试 100 题的维护帖子上贡献了他们的想法，思路，建议，及代码的朋友：<http://topic.csdn.net/u/20101126/10/b4f12a00-6280-492f-b785-cb6835a63dc9.html>，还记得那些日子，我们一起跟帖做题，那般充实而富有激情；

感谢所有在本 blog：http://blog.csdn.net/v_JULY_v，上针对这 100 题系列任何一题发表他们的见解，留言及评论的朋友；

因为有你们，我才能从 2010 年 10 月份开始整理微软面试 100 题的前 20 题而坚持到现在，才有这份稍具规模的系列文档，才能有机会帮助许许多多千千万万找工作的朋友们；

感谢各位的无私，奉献，blog 上见！

最后，还是开头这句话，有任何问题，欢迎随时不吝批评指正，联系方式如开头所述：

- 邮箱: zhoulei0907@yahoo.cn
- 微博: <http://weibo.com/julyweibo>

谢谢，感谢诸位。July、二零一二年 9 月 20 日，于上海。



数据库
技术丛书

MySQL DBA 修炼之道

陈晓勇 著

数据库技术专家撰写，多年数据库领域的经验结晶。实战性强，从架构、调优、运维、开发、测试等多个角度对MySQL管理与维护进行了全方位的归纳和总结，包含大量来自实际生产环境的经典案例，并深入阐述MySQL DBA进阶实战的技巧和方法！

 机械工业出版社
China Machine Press

资源由 www.eimhe.com 美河学习在线收集提供

数据库技术丛书

MySQL DBA修炼之道

陈晓勇 著

ISBN: 978-7-111-55841-5

本书纸版由机械工业出版社于2017年出版，电子版由华章分社（北京华章图文信息有限公司，北京奥维博世图书发行有限公司）全球范围内制作与发行。

版权所有，侵权必究

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @华章数媒

微信公众号 华章电子书（微信号：hzebook）

目录

推荐序

前言

第一部分 入门篇

第1章 理解MySQL

- 1.1 MySQL介绍
- 1.2 MySQL的基础架构和版本
- 1.3 查询执行过程概述
- 1.4 MySQL权限
- 1.5 长连接、短连接、连接池
- 1.6 存储引擎简介
- 1.7 MySQL复制架构
- 1.8 一些基础概念

第2章 MySQL安装部署和入门

- 2.1 如何选择MySQL版本
- 2.2 官方版本的安装
- 2.3 其他MySQL分支的安装
- 2.4 安装InnoDB Plugin
- 2.5 常用命令
- 2.6 MySQL的主要参数设置

第二部分 开发篇

第3章 开发基础

- 3.1 相关基础概念
- 3.2 数据模型
- 3.3 SQL基础
- 3.4 PHP开发
- 3.5 索引
- 3.6 ID主键
- 3.7 字符集和国际化支持

第4章 开发进阶

- 4.1 范式和反范式
- 4.2 权限机制和安全
- 4.3 慢查询日志
- 4.4 应用程序性能管理
- 4.5 数据库设计
- 4.6 导入导出数据
- 4.7 事务和锁
- 4.8 死锁
- 4.9 其他特性

第5章 开发技巧

- 5.1 存储树形数据
- 5.2 转换字符集
- 5.3 处理重复值
- 5.4 分页算法
- 5.5 处理NULL值
- 5.6 存储URL地址
- 5.7 归档历史数据
- 5.8 使用数据库存储图片
- 5.9 多表UPDATE
- 5.10 生成全局唯一ID
- 5.11 使用SQL生成升级SQL

第6章 查询优化

- 6.1 基础知识
- 6.2 各种语句优化
- 6.3 OLAP业务优化

第7章 研发规范

- 7.1 命名约定
- 7.2 索引
- 7.3 表设计
- 7.4 SQL语句
- 7.5 SQL脚本
- 7.6 数据架构的建议
- 7.7 开发环境、测试环境的配置参数建议
- 7.8 数据规划表
- 7.9 其他规范

第三部分 测试篇

第8章 测试基础

- 8.1 基础概念
- 8.2 性能测试的目的
- 8.3 基准测试
- 8.4 性能/基准测试的步骤
- 8.5 测试的注意事项

第9章 测试实践

- 9.1 硬件测试
- 9.2 MySQL测试
- 9.3 应用数据库性能测试

第四部分 运维篇

第10章 基础知识

- 10.1 文件和I/O管理
- 10.2 MySQL如何进行灾难恢复

- 10.3 变量设置、配置文件和主要参数
- 10.4 MySQL Query Cache和优化器
- 10.5 SHOW INNODB STATUS解析

第11章 MySQL的监控

- 11.1 非数据库的监控
- 11.2 数据库的监控
- 11.3 数据库监控的实现
- 11.4 数据库监控的可视化

第12章 MySQL复制

- 12.1 基础知识
- 12.2 配置主从复制
- 12.3 配置主主复制
- 12.4 配置级联复制、环形复制
- 12.5 跨IDC复制
- 12.6 多主复制
- 12.7 延时复制
- 12.8 半同步复制
- 12.9 在线搭建从库
- 12.10 配置日志服务器
- 12.11 常见的复制问题及处理方法

第13章 迁移、升级、备份、恢复数据库

- 13.1 升级
- 13.2 新业务部署上线
- 13.3 迁移
- 13.4 生产环境常用的备份策略
- 13.5 常用备份方式和恢复方法

第14章 运维技巧和常见问题处理

- 14.1 MySQL运维技巧
- 14.2 常见问题
- 14.3 故障和性能问题处理

第15章 运维管理

- 15.1 规模化运维
- 15.2 服务器采购
- 15.3 运维规则

第五部分 性能调优与架构篇

第16章 基础理论和工具

- 16.1 性能调优理论
- 16.2 诊断工具
- 16.3 调优方法论

第17章 应用程序调优

- 17.1 程序访问调优

17.2 应用服务器调优

第18章 MySQL Server调优

18.1 概述

18.2 MySQL的主要参数

18.3 MySQL内存优化

18.4 MySQL CPU优化

18.5 MySQL I/O优化

第19章 操作系统、硬件、网络的优化

19.1 基本概念

19.2 文件系统的优化

19.3 内存

19.4 CPU

19.5 I/O

19.6 网络

第20章 可扩展的架构

20.1 做好容量规划

20.2 扩展和拆分

20.3 读写分离

20.4 切勿过度设计

20.5 可扩展的方法

20.6 使用云数据库

第21章 高可用性

21.1 概述

21.2 单点故障

21.3 MySQL数据库切换

21.4 跨IDC同步

第22章 其他产品的选择

22.1 列式数据库产品

22.2 NoSQL产品的选择

参考文献

将本书献给三岁的女儿陈观之。

推荐序

我之前也看过很多数据库相关的图书，但是没有一本能像这本书一样，让我读起来感觉那么轻松愉快，读完后觉得必须要收藏一本作为案头必备。

本书的作者是互联网一线的数据库开发、运维专家，书中的内容是其对10多年工作中所遇问题的思考和总结，围绕着MySQL徐徐展开，犹如庖丁解牛，对MySQL的核心逻辑解释得相当清晰和透彻。本书以一个数据库专家的视角，解析其观察到的方方面面，内容涉及“业务系统设计”“测试体系”“运维管理”等。本书的很多内容已经不仅仅是从一个DBA的角度出发，更多的是从一个系统架构师和运维管理者的角度来思考问题。读完全书，你将会对整个研发、运维体系的相关领域都有一个概要的认识。这种提纲挈领的架构，对于某个知识领域的学习是非常有价值的。

之所以说本书读起来令人轻松愉悦，是因为书中提及的很多问题都是我所关心的，而笔者均以很简练的语言给予了回答和梳理，让人理解起来非常清晰、不费劲。我边看边忍不住想，这风格分明就是UC内部的培训资料嘛，有很强烈的亲切感。

本书对实战中的很多问题，都给出了详细的解题思路，方案成熟、观点中肯，体现了对技术应有的严谨和敬畏，我相信对从事DBA工作的很多技术人员来说，本书具有非常重要的参考价值。

毫不夸张地说，MySQL开源项目推动了整个互联网产品的发展。我们从中获益不少，同时也深刻体会到自由分享精神对社会进步的贡献。从晓勇写的这本书中，我也能感受到这一分享理念。我非常赞赏这种分享精神，也希望更多的技术人员都能有此回报社会的情怀。

20年前，互联网刚刚起步，工程师是靠掌握一批指令和娴熟的操作来执行运维工作的。现如今，开发和运维体系已经渐趋成熟，不少企业更是将基础运维工作交给云服务厂商，研发和运维人员得以从烦琐的技术细节中解放出来，从而更专注于业务分析和产品设计，这个进步是巨大的。

往后看，我们正从IT时代过渡到DT时代。在DT企业中，工程师使用贝叶斯变换和机器学习来操作数据，就好比当初使用“if()...else()”来编写程序一样，巨大的技术变革正在来临。

在这风起云涌之际，技术让我们再次感受到年轻和无知，希望我们能从MySQL出发，保持旺盛的好奇心和探索精神，迈向下一个崭新的时代。

梁捷（Jack）

UC联合创始人，神马搜索总裁

2016年12月

前言

为什么要写本书

本书主要讲述MySQL DBA的必备技能，包括MySQL的安装部署、开发、测试、监控和运维，此外，读者还可从中学习到系统架构的一些知识。

我从业10多年，先是在传统行业做开发工程师、系统管理员、Oracle DBA，2008年因为机缘巧合投身互联网，开始从事MySQL运维工作。相对于成熟的商业数据库，MySQL缺乏高质量的技术文档和图书，我在接触MySQL的过程中，也感觉市面上的相关图书还存在一些不足，难以系统化地学习MySQL。

从一名Oracle DBA转型为一名MySQL DBA，从传统领域转投到互联网公司，即便我之前有丰富的经验，在学习MySQL的过程中也仍然走了一些弯路。成为一名MySQL DBA并不难，但成为一名高水平的MySQL DBA则需要时间、知识、技能、经验和意识的积累。

我在学习MySQL的过程中，有时会去看技术论坛，或者通过MSN群等聊天工具咨询他人一些问题，也得到过一些朋友的帮助。国内存在一批高素质的MySQL DBA，但由于各种现实因素，有心写一本关于MySQL DBA实战的书的人很少，所以市面上缺乏高质量的相关图书不足为奇。2013年年初，华章公司的策划编辑杨绣国找到我，说希望我能写一本关于MySQL的书，我当时很犹豫，虽然我有时会在网上回答一些问题，也定期撰写个人博客，但是，写一本书，对于我来说，是一个艰巨的任务。经过一些交流，我慢慢明确了自己的想法，其实我一直是想写一本书的，既然我对市面上的相关图书不太满意，那么就自己写一本吧，当时我唯一欠缺的是写作经验以及时间。

我写这本书的目的是想做一个尝试——引领感兴趣的读者进入MySQL数据库运维领域。国内互联网行业正在高速发展，迫切需要大量的MySQL人才，希望这本书可以帮助一些读者顺利进入数据库领域。而且，我也想将自己的一些心得分享给读者，希望热爱数据库技术的同行们在工作中少走弯路。

在技术领域工作多年后，文字写作对我来说其实已经很陌生了，弗朗西斯·培根说过，“阅读使人充实，谈论使人机敏，写作使人精确”。在本书的写作的过程中，其实我自己也获得了很多，不仅学到了更多的知识，对于自己的精神也是一种洗礼。写作真的是一种积极而富有价值的创作，我们只有正确地掌握所讲述的内容，才能为言行思想带来正能量。

希望在这个世界上，有越来越多的人愿意分享，且能享受分享的乐趣。

读者对象

本书的主要读者是MySQL DBA，在现实中，许多公司并没有配备专职的数据库维护人员，数据库的维护工作往往也是由开发工程师和系统管理员负责的，因此这本书也适用于他们。

这是一本偏向实战的技术书籍，不会过多地涉及技术的细节和原理，我会尽量直接地给出解决方案；本书除了讲MySQL技能，还花了大量篇幅讲述架构；本书不仅讲述技术，也讲述技术之外的一些运维管理规则。对数据库的使用、维护和管理感兴趣的运维工程师、架构师、运维经理、开发工程师、测试工程师都可以将本书作为参考图书，而了解其他领域会有助于你的职业发展。

本书也适合希望转行到数据库运维领域的人士。许多人想从事IT工作，但当下时间宝贵，要想进入一个行业或改变职业方向，往往会花费巨大的时间成本，所以这本书将尽量做到简单、易懂，以节省大家的学习成本。

如何阅读本书

本书将分为5个部分，分别从入门、开发、测试、运维、性能与架构这几个方面来介绍MySQL的使用。对于初次接触MySQL的读者，建议按照章节顺序逐步学习。对于已经有一定经验的读者，则可以选择自己感兴趣的篇章，跳过自己已经熟悉的内容。

第一部分讲述了MySQL的基础架构、权限机制、常用的存储引擎、复制架构、安装及常用命令等知识。如果读者是初次接触MySQL，那么可能还需要在这一部分上花一些时间。在掌握Linux和MySQL的基本使用方法之后，就可以开始第二部分的学习了。

第二部分将介绍MySQL数据库开发相关的基础知识和技巧。基础知识包括关系数据模型、字符集、常用的SQL语法、范式、索引和事务等。由于开发的领域很广，所以本部分仅仅选取了一些常用的技巧分享给大家。最后会结合实际生产，提供一份开发规范供大家参考。

第三部分介绍了数据库基准测试所需要的理论知识和常用的测试工具。本部分将介绍一个MySQL的基准测试模型。

第四部分介绍了MySQL运维工作的各项职责：监控、复制、迁移、升级、备份和恢复，然后通过一些案例向读者传授一些维护技巧及处理问题的方法。读者还将学习到规模化运维MySQL的一些知识和规则。

第五部分介绍了性能调优的一些理论知识，以及从应用程序到数据库，再到存储等各个环节的优化。由于架构和性能优化密切相关，本部分也介绍了一些MySQL DBA需要熟悉的架构优化知识。初次接触MySQL的读者对于架构优化的内容可能会感到难以理解，但随着经验的增长，再理解这些内容将不会再有问题。

本书假设读者已经对软硬件有了一定的认识，掌握了一门脚本语言，并且对Unix或Linux有一定的使用经验，对于数据库有了基本的认识。阅读本书时，读者不需要预先准备好上述的所有知识，但需要有意识地在阅读本书之外不断地补充自己的基础知识。我会对以上内容做深入的讲解，但如果读者有基础会更好，好的基础有利于快速吸收知识和深入思考问题。如果读者还不会使用Linux和编写Shell脚本，那么，建议尽快搭建一个学习环境。

由于DBA需要和研发、测试、产品、运营、监控等团队进行合作，所以对于相关领域所涉及的数据库知识，本书也会做一些介绍。但是，由于经验侧重的关系，本书将主要从DBA的角度来讲述这些知识和技能。

本书主要基于MySQL官方5.1版本写作，这也是目前最流行的版本，我会补充MySQL最新版本的少许内容，但跟踪MySQL新版本更合适的策略是关注官方发布的新特性说明、新版本的文档手册，跟踪业内专家的技术博客和社交媒体等。

通过阅读本书，读者可以学到MySQL的许多知识，但是仅通过阅读是难以获得技能和经验的。读者需要有一个适合自己的MySQL测试环境，并能够不断地思考和实践自己的想法，这样才能够掌握技能，并得到属于自己的经验。

勘误和支持

由于作者的水平有限，写作时间也很仓促，书中难免存在一些错误或不准确的地方，如有不妥之处，恳请读者批评指正。为此，我特意创建了在线支持页面<http://www.db110.com>。你可以将书中的错误发布在勘误表页面，若遇到任何问题，也可以访问Q&A页面，我将尽量在线上为你提供最满意的解答。书中的全部源文件都将发布在这个网站上。如果你有更多的宝贵意见，也欢迎你发送邮件至我的邮箱ucgary@gmail.com，很期待听到你们的真挚反馈。

致谢

感谢机械工业出版社华章公司的策划编辑杨绣国的努力工作，没有她的投入和耐心，就不可能有本书的面世。本书写作的时间较长，我有时会充满愧疚，是杨绣国编辑的包容和鼓励，最终引导我顺利完成全部书稿。

感谢UC的旧同事，和你们的共事，是我职业生涯最宝贵的财富，我将一直铭记在心。

最后，我要感谢我的家人和朋友，是你们的支持，让我能够坚持下来。

陈晓勇（Gary Chen）

中国，长沙，2016年12月

第一部分 入门篇

本篇首先介绍MySQL的应用领域、基础架构和版本，然后介绍MySQL的基础知识，如查询的执行过程、权限机制、连接、存储引擎，最后阐述一些基础概念。

第1章 理解MySQL

本章将介绍MySQL的一些常识，以及目前MySQL的发展现状。然后简要说明MySQL的基础架构、存储引擎、运行机制，以及工作中应该如何使用MySQL，为后面章节的学习做个铺垫。

1.1 MySQL介绍

1.1.1 应用领域和适用场景

MySQL是目前世界上最流行的开源关系数据库。在国内，MySQL大量应用于互联网行业，比如，大家所熟知的百度、腾讯、阿里、京东、网易、新浪等都在使用MySQL。搜索、社交、电商、游戏后端的核心存储往往都是MySQL，有的具有上千台甚至几千台MySQL数据库主机。可以说，支撑互联网公司日常运转的主要数据库就是MySQL。近年来，随着业务的发展，互联网公司产生了许多成熟的架构和技术，这也促使MySQL不断变得更加成熟和稳健。但MySQL的应用并未局限于互联网应用，许多软件开发商也把MySQL集成到了自己的产品中，这样一来，传统行业的大公司也都可以在企业内部大量使用MySQL存储企业数据了，包括政府信息系统，同样也在大量使用MySQL数据库。

MySQL的定位是通用的数据库，各种类型的应用一般都能利用到MySQL存取数据的优势。业内生产实践也证明，MySQL更适合中小型数据库、OLTP业务，以目前的软硬件产品水平来看，如果单机数据超过几个TB将难以高效利用MySQL。

MySQL可以作为传统的关系型数据库产品使用，也可以当作一个key-value产品来使用，由于它具有优秀的灾难恢复功能，因此相对于目前市场上的一些key-value产品会更有优势。

我们所说的MySQL更适合OLTP业务、中小型数据库，并不是说MySQL仅限于此，数据的存储往往是一个架构问题，如果配合架构，MySQL也是可以存储海量数据的。海量数据没有一个明确的标准，对于MySQL来说，我们可以简单地认为海量数据是指单个实例难以处理的几十亿以上的数据。不过，MySQL对于海量数据的分析就不擅长了，你可能还需要其他产品来协助解决这方面的问题。一般而言，中小型公司最佳的选择仍然是MySQL，毕竟在这类公司里，海量数据并不常见。下面让我们来看看部分知名互联网公司的MySQL主机规模，一些公开资料显示如下。

- Facebook 2008年有10000台服务器，其中包括1800台MySQL服务器，到2013年已经突破了20万台服务器，按40：1计算，MySQL服务器至少也有五千台了。

- Twitter早在2011年就有2000~4000台服务器，绝大部分数据后端的持久化存储都是MySQL服务器。

- 对于国内的几大互联网公司，如阿里、百度、腾讯，依据公开的信息，它们均有千台以上MySQL服务器的规模。

这些大型互联网公司都注重使用MySQL，而且往往也在内部维护了一个MySQL的分支，同时它们也积极参与到MySQL社区，促使MySQL不断改进。

1.1.2 为什么那么多公司和机构选择使用MySQL

它们选择使用MySQL的主要原因有以下两点。

- 低成本、高效能。

- 处于起步阶段的团队、小公司需要一个开放的系统来适应发展的需要。

互联网公司，特别是处于起步阶段的公司，需要一个低成本的系统来构建服务，从而可以把更多的资金用于业务的扩张。LAMP（其中的“M”指的就是MySQL）的组合已被广泛应用——目前世界上的大部分网站使用的都是LAMP（或者LNMP）组合。由于它是免费的，LAMP自然就成了第一选择，一般而言，选择成熟可靠、使用人数广泛的产品，公司的技术风险也会大大降低。同时MySQL是一个开放的系统，源代码开放，社区成熟活跃，在公司发展壮大的过程中，可以不断从外部获取成熟的

思想和解决方案。可以说MySQL已经构建了成熟的生态圈，使用它的人往往能得到许多益处，而且相对于目前市场上的其他产品，MySQL也具备许多优势。

一些公司出于节省成本和扩展性的考虑，尝试把某些业务从商业数据库迁移到MySQL上，比如阿里，由于数据库集群的规模巨大，传统的基于小型机和高端的存储架构难以扩展，且支出成本庞大，所以把大部分业务逐步迁移到PC服务器的MySQL集群上，成功地降低了成本。

1.1.3 MySQL的优势是什么，它解决了什么问题

MySQL是一个轻量级的通用关系型数据库，具有稳定、易安装、易使用、高性能等特点，可配合架构进行扩展。它的安装包不大，百MB级别，安装简单方便，入门也很简单，而一些商业化的关系型数据库产品往往安装包庞大，且配置使用复杂，需要开发人员或DBA花费几倍的时间去掌握产品的使用。

MySQL起初也有很多Bug，而且不太稳定，但经过十多年的发展，目前的MySQL（5.0/5.1）已经很稳定了。新的5.5/5.6/5.7也发布了GA版本，正在持续完善中，截至2015年年底，我们可以看到MySQL 5.1/5.5已经大量应用于生产环境了。

此外，MySQL也是一个高性能的产品，它不仅适用于中小型公司，还能稳定高效地处理大数据。业内存在一种误解，认为MySQL的扩展性不好，若超过一定的数据量时，性能就会下降。其实这更多的是一个架构问题，配合成熟的架构，比如在应用层切分数据，MySQL的扩展性就不再是什么问题了，而且很多数据是能够分片到各个MySQL节点的。Facebook、Twitter、Google等都在大量使用MySQL存储海量数据。

一些人倾向于用NoSQL产品来存储数据，其实，NoSQL产品，特别是一些key-value单机产品，相对于MySQL来说并没有什么优势。MySQL同样可以把数据存储为key-value的形式，并且，NoSQL的产品还不是很稳定，一旦数据丢失就可能会导致很严重的损失，又往往因为数据模型简单，所以应用范围狭小。MySQL成熟稳定且拥有丰富的数据类型，它的关系模型可以满足项目不断增加的商业需求。

1.2 MySQL的基础架构和版本

1.2.1 软件架构中数据库的定位

数据库一般位于整个软件架构的后端，而不直接服务于用户，数据的展示、应用逻辑的处理都是由其他层次的程序来实现的。比较流行的一种软件架构的分类是“双层”、“三层”、“多层”架构。客户端直接和数据库服务器通信，比如通过ODBC、JDBC连接数据库，一般称为“双层架构”或“client-server”架构。若客户端和数据库之间有一个中间服务器（如Web服务器，中间件），则由中间服务器负责转发请求给数据库服务器，这种模式称为“三层架构”。在很多较大规模的Web应用中，在Web服务器和数据库服务器之间还可能存在一个应用服务器，这种结构称为“四层架构”。

本书探讨的MySQL是基于目前互联网最常见的架构，如，网站应用、移动互联网应用。它们一般是三层架构，这三层架构分别如下。

- 1) 用户接口层，即各种终端，比如，运行在最终用户计算机上的浏览器。
- 2) 业务逻辑和数据处理层，即应用程序服务器，比如，PHP、Java EE、ASP.NET、Ruby on Rails等应用服务。网站处理网络访问请求的过程可能是这样的：由Nginx接受用户请求，处理静态页面，并且将动态请求转发给后端的PHP服务，PHP服务处理完动态请求后，将结果返还给Nginx，Nginx再返还给用户。有时也称该层为中间件（middle ware）。
- 3) DBMS，即后端数据存储，如MySQL、PostgreSQL、Redis、Memcached等产品。

相应地，在软件系统架构设计中也存在一种分层设计的方法学。我们熟知的三层架构（3-tier application）是一种应用广泛的分层设计，它把应用分解为表现层、业务逻辑层、数据访问层3个层次。三层（多层）架构主要的好处是提供了一个灵活的、可重用的模型，开发者可以通过简单地修改某一层的功能或增加某一层的功能来实现某种需求，而不需要修改整个应用程序。

- 表现层（UI），即直接和用户交互的界面。
- 业务逻辑层（BLL），即对业务逻辑进行处理，处理用户的请求，它将许多最终用户的业务逻辑集中到了应用服务器上。
- 数据访问层（DAL），直接操作数据库，即针对数据的增加、删除、修改、查找等操作。

传统行业的商业数据库往往还承载了许多业务逻辑的功能，这其中就会经常用到存储过程、触发器。互联网世界的开源数据库虽然也有存储过程、触发器之类的特性，但绝大部分场合下并不会用到这些非核心的基本特性，开发者把数据库更多地看作一个存储数据的容器，并已将核心业务逻辑从数据库功能中分离了出来。

本书主要是讲述MySQL的使用，由于MySQL的优化与软件整体架构的其他组件的关系密切，所以对于Web服务器、缓存产品、队列等产品，也会做一些简单介绍。作为一个合格的DBA，有必要了解各种应用服务的运行机制，以及是否需要对它们进行优化。

1.2.2 MySQL的基础架构

MySQL是一种关系数据库产品。关系数据库，顾名思义，是建立在关系模型基础上的数据库。现实世界中，实体与实体之间的各种联系一般都可以用关系模型来表示。经过数十年的发展，关系数据库在理论和工业实践中都已经很成熟了。

数据库产品的架构一般可以分为应用层、逻辑层、物理层，对于MySQL，同样可以理解为如下的3个层次。

·应用层。负责和客户端、用户进行交互，需要和不同的客户端和中间服务器进行交互，建立连接，记住连接的状态，响应它们的请求，返回数据和控制信息（错误信息、状态码等）。

·逻辑层。负责具体的查询处理、事务管理、存储管理、恢复管理，以及其他附加功能。查询处理器负责查询的解析、执行。当接收到客户端的查询时，数据库就会分配一个线程来处理它。先由查询处理器（优化器）生成执行计划，然后交由计划执行器来执行，执行器有时需要访问更底层的事务管理器、存储管理器来操作数据，事务管理器、存储管理器主要负责事务管理、并发控制、存储管理。这其中，将由事务管理器来确保“ACID”特性，通过锁管理器来控制并发，由日志管理器来确保数据持久化，存储管理器一般还包括一个缓冲管理器，由它来确定磁盘和内存缓冲之间的数据传输。

·物理层。实际物理磁盘（存储）上的数据库文件，比如，数据文件、日志文件等。

图1-1是MySQL官方文档的一个基础架构图，其中Connectors可以理解为各种客户端、应用服务；Connection Pool可以理解为应用层，负责连接、验证等功能；Management Services&Utilities、SQL Interface、Parser、Optimizer、Caches&Buffers、Pluggable Storage Engines可以理解为数据库的大脑——逻辑层；最下方的Files&Logs可以理解为物理层。

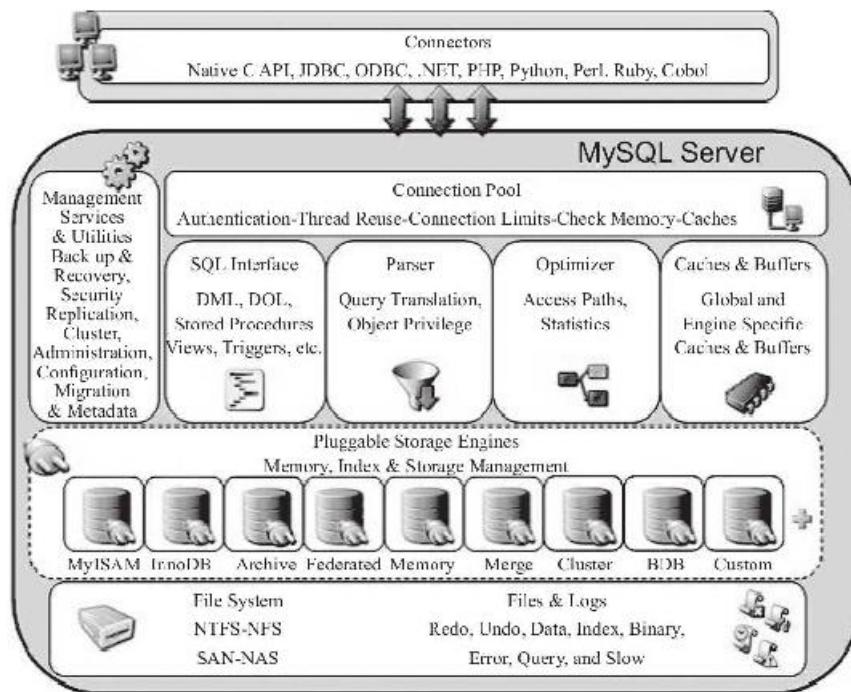


图1-1 MySQL的基础架构

1.2.3 MySQL的版本及特性

1. MySQL支持的平台

MySQL支持目前市面上的大部分平台，包括32位和64位平台，一般情况下程序运行在64位平台上比32位更快。MySQL支持的平台如下所示。

·Solaris

·Linux

·Windows

·AIX

·Mac OS

·HPUX

2.MySQL许可协议

Oracle以双重授权（Dual Licensed）的方式发布MySQL，它们是GPL和商业许可协议（Commercial License）。如果你在一个遵循GPL的自由（开源）项目中使用MySQL，那么你可以遵循GPL协议使用MySQL，无论是否将其用作商用。

如果某些商业软件中结合了MySQL或修改了MySQL源码，但又不愿意按GPL协议公开软件源码，那么就必须和Oracle公司达成商业许可协议。简而言之，如果你违反了GPL，则需要购买商业许可。

GPL授予用户以下权利。

- 以任何目的运行此程序的自由。
- 再发行复印件的自由。
- 改进程序，并公开发布改进内容的自由。

需要注意的是，GPL只限制了对外分发的软件，也就是说，如果该软件只在内部使用，无论开不开源都没有关系。如何使用开源软件并不受GPL的约束，只有在你基于开源软件，修改开源软件的源码时才受GPL约束，如果你的应用程序只是用到了MySQL，无论是否商用，都不需要考虑开源。

3.MySQL版本

MySQL目前可分为4个版本：MySQL社区版、MySQL标准版、MySQL企业版、MySQL集群版。

（1）MySQL社区版

可免费下载使用的开源版本，遵循GPL协议，包括如下的这些特性。

- 可插拔的存储引擎架构
- 多存储引擎支持InnoDB、MyISAM、NDB（MySQL Cluster即采用NDB存储引擎）、Memory、Merge、Archive、CSV等
- 复制
- 分区
- 存储过程、触发器、视图
- 信息数据库（Information-Schema）
- MySQL连接器
- MySQL工作台（MySQL Workbench）

目前已经发布了MySQL 5.0、MySQL 5.1、MySQL 5.5、MySQL 5.6、MySQL 5.7一共5个GA版本。一般来说，后面的版本

比前面的版本功能更强、扩展性更好。

以下3个版本是给商业用户使用的，商业客户可灵活选择多个版本，以满足特殊的商业和技术需求。

(2) MySQL标准版

和社区版差别不大，提供社区版所支持的各种特性。

(3) MySQL企业版

MySQL企业版提供7×24小时的技术支持服务，用户可直接联系MySQL专业支持工程师，获取关于MySQL应用程序开发、部署和管理的全方位支持。

MySQL企业版提供了更全面的高级功能、管理工具和技术支持，例如：MySQL企业级备份可为数据库提供在线“热”备份，从而降低数据丢失的风险。它支持完全、增量和部分备份，以及时间点恢复和备份压缩。

MySQL线程池提供了一个高效的线程处理模型，旨在降低客户端连接和语句执行线程的管理开销。

MySQL企业级安全性提供了一些立即可用的外部身份验证模块，可将MySQL轻松集成到现有的安全基础架构中。

其他特性还有MySQL企业级审计、MySQL企业级监视器（MySQL Enterprise Monitor）和MySQL查询分析器（MySQL Query Analyzer）等。

MySQL的一些新特性出现在了企业版中，但并没有出现在社区版，这导致很多人对于MySQL产生了疑虑，但MySQL的生态已经建立成熟，官方版本和其他分支也都在稳定地发展改进中，一般的中小公司选择社区版本即可。一些行业、领域要求更好的服务，更高的稳定性，或者有其他复杂的业务需求，对于它们企业版是一个很好的选择。

(4) MySQL集群（MySQL Cluster）版

Oracle收购MySQL之后，对MySQL Cluster做了大量改进，这也是Oracle力推的产品。集群版是一种分布式、无共享（share-nothing）的架构，也就是说把数据分布在各个节点的内存里。据官方宣称，集群版可比单机数据库提供更高的可用性，高达99.999%。它还有一些好处，比如自动分片、动态添加节点、支持跨IDC复制、减少维护成本等。但这个产品比较复杂，国内也缺少精通MySQL Cluster的专家，如果一定要使用，建议做好充分的测试验证工作。

据说现在的MySQL Cluster版本已经允许存储部分数据到硬盘上，但由于主要数据需要存放在内存中，因此其部署成本会比较高。另外，随着MySQL Cluster节点的增多，节点之间通信、同步的代价也越来越大，所以其扩展性也是有限的。对于海量数据，MySQL Cluster可能不是很好的方案，从理论上来讲，仅仅把热点数据加载到内存是更经济的做法。

1.2.4 MySQL的开发周期

Oracle公司是一家成熟的商业公司，拥有一流的工程能力和执行力，自收购MySQL以来，就增加了相应的开发人员，并且提供了更成熟的开发模式，目前MySQL的开发进度比收购之前高了很多，许多第三方的优化补丁也都在官方版本中得到了实现。而之前MySQL的400多名开发人员分布在25个国家，70%的开发人员在家工作，导致了交流沟通不畅，产品开发进展缓慢。

目前MySQL的发展路线更清晰，开发周期大致分为4个阶段。

1) 新特性开发。

2) 发布实验室版本。

实验室版本可以提前预览到一些正在开发的特性，供用户试用，但是不保证这些特性会被整合到里程碑版本和GA版本。

3) 发布里程碑版本 (Development Milestone Releases)。

这个时候的版本称为RC (Release Candidate候选) 版本，有充分的文档支持，在所有支持的平台上发布，可以让用户试用，以收集反馈。一般平均3~6个月发布一个DMR (里程碑版本)。

4) 发布GA版本 (Generally Availability Releases)。

GA版本是建议用于生产系统的版本。一般18~24个月为一个周期。

1.3 查询执行过程概述

图1-2抽象化地描述了客户端和数据库交互的过程。

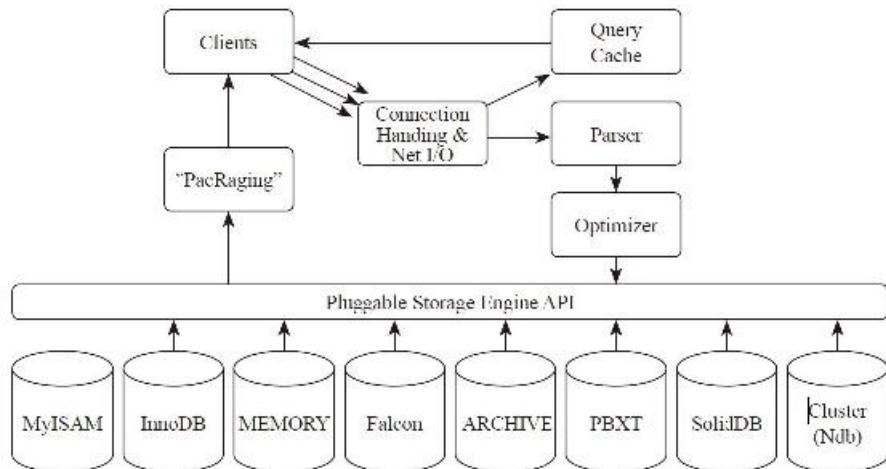


图1-2 客户端与数据库交互抽象架构图

如图1-2所示，客户端（Clients）发布查询的流程如下，首先连接MySQL（Connection Handling），然后发布查询，如果缓存（Query Cache）中有结果集，则直接返回结果集。如果没有被缓存，那么，MySQL解析查询（Parser）将通过优化器（Optimizer）生成执行计划，然后运行执行计划通过API（Pluggable Storage Engine API）从存储引擎获取数据，并返回给客户端。

什么是执行计划（查询计划）呢？执行计划就是一系列的操作步骤。SQL是声明性语言，它只告诉数据库要查询什么，但并不告诉数据库如何去查。数据库所要做的就是基于算法和统计信息计算出一条最佳的访问路径。这个工作是由优化器来完成的。优化器会比较不同的执行计划，然后选择其中最优的一套。

1.4 MySQL权限

1.4.1 MySQL权限机制

MySQL权限控制包含如下两个阶段。

阶段1：服务器检查是否允许你连接。

阶段2：假定你能连接，服务器将检查你发出的每一个请求，查看你是否有足够的权限实施它。例如，如果你从数据库表中选择（SELECT）行或从数据库中删除表，那么服务器要确定你是否对表有SELECT权限或对数据库有DROP权限。

MySQL是通过用户名、密码、IP（主机名）3个要素来验证用户的。当你想要访问MySQL服务器时，MySQL客户端程序一般会要求你指定如下参数。

- MySQL服务器的IP（主机名），端口

- 用户名

- 密码

以下是连接MySQL服务器的一个示例，你需要以实际的IP、端口、用户名、密码代替相应的内容。

```
shell> mysql -h host_ip_address -u user_name -p your_password -P server_port
```

1.4.2 赋予权限和回收权限

一般在生产环境下，程序账号有增加、删除、查询、修改这4项功能即可。

如下命令用于赋予查询、插入、修改、删除权限，并进行密码设置。

```
mysql> grant select,insert,update,delete on db_name.* to user_name@ '10.%' identified by 'password';
```

如下命令用于收回上面所赋予的权限。

```
mysql> revoke select,insert,update,delete on db_name.* from user_name@ '10.%';
```

1.5 长连接、短连接、连接池

当数据库服务器和客户端位于不同的主机时，就需要建立网络连接来进行通信。客户端必须使用数据库连接来发送命令和接收应答、数据。通过提供给客户端数据库的驱动指定连接字符串后，客户端就可以和数据库建立连接了。可以查阅程序语言手册来获知通过何种方式使用短连接、长连接。

1.5.1 短连接

短连接是指程序和数据库通信时需要建立连接，执行操作后，连接关闭。短连接简单来说就是每一次操作数据库，都要打开和关闭数据库连接，基本步骤是：连接→数据传输→关闭连接。

在慢速网络下使用短连接，连接的开销会很大；在生产繁忙的系统中，连接也可能会受到系统端口数的限制，如果要每秒建立几千个连接，那么连接断开后，端口不会被马上回收利用，必须经历一个‘FIN’阶段的等待，直到可被回收利用为止，这样就可能会导致端口资源不够用。在Linux上，可以通过调整`/proc/sys/net/ipv4/ip_local_port_range`来扩大端口的使用范围；调整`/proc/sys/net/ipv4/tcp_fin_timeout`来减少回收延期（如果想在应用服务器上调整这个参数，一定要慎重！）。

另外一个办法是主机使用多个IP地址。端口数的限制其实是基于同一个IP:PORT的，如果主机增加了IP，MySQL就可以监听多个IP地址，客户端也可以选择连接某个IP:PORT，这样就增加了端口资源。

1.5.2 长连接

长连接是指程序之间的连接在建立之后，就一直打开，被后续程序重用。使用长连接的初衷是减少连接的开销，尽管MySQL的连接比其他数据库要快得多。

以PHP程序为例，当收到一个永久连接的请求时，PHP将检查是否已经存在一个（前面已经开启了的）相同的永久连接。如果存在，则将直接使用这个连接；如果不存在，则建立一个新的连接。所谓“相同”的连接是指用相同的用户名和密码到相同主机的连接。

从客户端的角度来说，使用长连接有一个好处，可以不用每次创建新连接，若客户端对MySQL服务器的连接请求很频繁，永久连接将更加高效。对于高并发业务，如果可能会碰到连接的冲击，推荐使用长连接或连接池。

从服务器的角度来看，情况则略有不同，它可以节省创建连接的开销，但维持连接也是需要内存的。如果滥用长连接的话，可能会使用过多的MySQL服务器连接。现代的操作系统可以拥有几千个MySQL连接，但很有可能绝大部分都是睡眠（sleep）状态的，这样的工作方式不够高效，而且连接占据内存，也会导致内存的浪费。

对于扩展性好的站点来说，其实大部分的访问并不需要连接数据库。如果用户需要频繁访问数据库，那么可能会在流量增大的时候产生性能问题，此时长短连接都是无法解决问题的，所以应该进行合理的设计和优化来避免性能问题。

如果客户端和MySQL数据库之间有连接池或Proxy代理，一般在客户端推荐使用短连接。

对于长连接的使用一定要慎重，不可滥用。如果没有每秒几百、上千的新连接请求，就不一定需要长连接，也无法从长连接中得到太多好处。在Java语言中，由于有连接池，如果控制得当，则不会对数据库有较大的冲击，但PHP的长连接可能导致数据库的连接数超过限制，或者占用过多的内存。

对此，研发工程师、系统运维工程师、DBA需要保持沟通，确定合理的连接策略，千万不要不假思索就采用长连接。

1.5.3 连接池

由于一些数据库创建和销毁连接的开销很大，或者相对于所执行的具体数据操作，连接所耗的资源过多，此时就可能需要添加连接池来改进性能。

数据库连接池是一些网络代理服务或应用服务器实现的特性，如J2EE服务器，它实现了一个持久连接的“池”，允许其他程序、客户端来连接，这个连接池将被所有连接的客户端共享使用，连接池可以加速连接，也可以减少数据库连接，降低数据库服务器的负载。

1.5.4 持久连接和连接池的区别

长连接是一些驱动、驱动框架、ORM工具的特性，由驱动来保持连接句柄的打开，以便后续的数据库操作可以重用连接，从而减少数据库的连接开销。而连接池是应用服务器的组件，它可以通过参数来配置连接数、连接检测、连接的生命周期等。

如果连接池或长连接使用的连接数很多，有可能会超过数据库实例的限制，那么就需要留意连接相关的设置了，比如连接池的最小、最大连接数设置，以及php-fpm的进程个数等，否则程序将不能申请新的连接。

1.6 存储引擎简介

运行如下命令可查看表的引擎。

```
mysql> show table status like 'sys_accont' \G
***** 1. row *****
Name: sys_accont
Engine: InnoDB
```

其中，`Engine`栏位表示使用的是何种引擎。

MySQL不同于其他数据库，它的存储引擎是“可插拔”的，意思就是MySQL Server的核心基础代码和存储引擎是分离的，你可以使用最适合应用的引擎，也就是说MySQL支持不同的表使用不同的引擎。MySQL拥有20多个引擎，下面介绍几个常用的引擎。

1.6.1 InnoDB引擎

在MySQL 5.5及以后的版本中，InnoDB是MySQL的默认引擎，这些年来，InnoDB一直在持续改进，处理能力不断提高，其优秀的性能和可维护性使它成为生产中普遍推荐使用的引擎。它的优点有：

- 灾难恢复性好。
- 支持全部4种级别的事务。默认的事务隔离级别是可重复读（Repeatable Read），它的事务支持是通过多版本并发控制（MVCC）来提供的。
- 使用行级锁。
- 对于InnoDB引擎中的表，其数据的物理组织形式是簇表（Cluster Table），数据按主键来组织，也就是说主键索引和数据是在一起的，数据按主键的顺序物理分布。数据表的另一种常见形式是非簇表，其索引是有序的，而数据是无序的。
- 实现了缓冲管理，不仅能缓冲索引也能缓冲数据，并且会自动创建散列索引以加快数据的获取。相比之下，MyISAM只是缓存了索引。
- 支持外键。
- 支持热备份。



注意 若无特殊说明，本书都是基于InnoDB引擎论述的。

1.6.2 MyISAM引擎

MyISAM是MySQL5.0/5.1的默认引擎，但MySQL官方的重心早已不在MyISAM引擎上了，近些年来，MyISAM一直没有大的改进，由于它有许多缺陷，如不支持事务、灾难恢复性差，所以不建议在生产环境中使用。

以下是MyISAM的一些特性。

- 可以配合锁，实现操作系统下的复制备份、迁移。

- 使用表级锁，并发性差。
- 支持全文检索（MySQL InnoDB在5.6以后也支持全文检索）。
- 主机宕机后，MyISAM表易损坏，灾难恢复性不佳。
- 无事务支持。
- 只缓存索引，数据的缓存是利用操作系统缓冲区来实现的。可能引发过多的系统调用且效率不佳。
- 数据紧凑存储，因此可获得更小的索引和更快的全表扫描性能。

1.6.3 MEMORY存储引擎

MEMORY存储引擎提供“内存”表，也不支持事务、外键。

使用内存表（内存引擎）可以显著提高访问数据的速度，可用于缓存会频繁访问的、可以重构的数据、计算结果、统计值、中间结果，但也有如下这些不足之处。

- 使用的是表级锁，虽然内存访问快，但如果频繁地读写，表级锁可能会成为瓶颈所在。
- 只支持固定大小的行。VARCHAR类型的字段会存储为固定长度的CHAR类型，浪费空间。
- 不支持TEXT、BLOB字段。当有些查询需要使用到临时表（使用的也是MEMORY存储引擎）时如果表中有TEXT、BLOB字段，那么会转化为基于磁盘的MyISAM表，严重降低性能。
- 由于内存资源成本昂贵，一般不建议设置过大的内存表，如果内存表满了，就会在MySQL错误日志里发现类似“The table ‘table_name’ is full”这样的错误，可通过清除数据或调整内存表参数来避免报错。
- 服务器重启后数据会丢失，复制维护时需要小心，具体请参考第12章。

1.6.4 ARCHIVE存储引擎

ARCHIVE存储引擎是被设计用来存储企业中的大量流水数据的存储引擎。ARCHIVE引擎使用zlib无损数据压缩，让数据都保存在压缩的存档表中。当数据被插入时，它们被压缩。

它只支持INSERT和SELECT，支持自增键及其上的索引，不支持其他索引。它适合做日志记录、用户行为分析，不需要UPDATE、DELETE和索引的数据。

1.6.5 选择合适的引擎

表1-1列举了MySQL部分引擎的特性：是否支持事务、锁级别、是否支持热备份。其中，5.0版本、5.1版本默认的引擎是MyISAM，5.5版本、5.6版本默认的引擎是InnoDB。

表1-1 MySQL引擎特性对比

存储引擎	事务支持	锁级别	热备份	MySQL Server 版本
InnoDB	Yes	Row	Yes	5.1, 5.5, 5.6
MyISAM/Merge	No	Table	No	5.1, 5.5, 5.6
Memory	No	Table	No	5.1, 5.5, 5.6
Marta	Yes	Row	No	5.1, 5.5, 5.6
Falcon	Yes	Row	Yes	5.6
PBXT	Yes	Row	Yes	5.1, 5.5, 5.6
FEDERATED	No	None	None	5.1, 5.5, 5.6
NDB	Yes	Row	Yes	MySQL Cluster
Archive	No	Row	No	5.1, 5.5, 5.6
CSV	No	Table	No	5.1, 5.5, 5.6

那么如何选择合适的引擎呢？以下是选择引擎时需要考虑的一些因素。

- 是否需要事务支持。
- 是否为高并发， InnoDB实现了行锁， 这方面的表现大大优于MyISAM。
- 索引， 不同存储引擎的索引实现不尽相同。
- 是否需要外键。
- 高效缓冲数据， InnoDB缓冲数据而MyISAM只缓冲了索引。
- 备份， 是否需要支持热备份。

我们可以灵活地选择引擎，但是从维护的角度来说，维护统一的存储引擎会更方便，所以或者全部是MyISAM，或者全部是InnoDB引擎在现实生产中更常见，也易于管理。

1.6.6 选择何种平台

业内普遍的做法是把MySQL部署在Linux系统下，所以如果不加特别说明，本书指的都是Linux下的MySQL部署、使用。为什么互联网公司的生产环境一般使用Linux操作系统，而不考虑在Windows上部署安装MySQL呢？部分原因如下所示。

一般来说，部署在Unix/Linux环境下的软件程序往往有更高的运行效率。因为这样一个事实：不同的操作系统在它们所采用的进程和线程模型方面有着相当大的差异。Unix/Linux编程模型对Apache和MySQL等软件进行优化的工作不仅开始得最早，进行得也最全面彻底，而Windows在这方面就远远落后了。

Oracle公司在收购MySQL后，对Windows版本做了一些增强，这样做更多的是出于商业的考虑，Windows PC和Windows Server的市场占有率高，无论是作为开发环境或独立软件供应商的后台数据库，Windows下的MySQL都有其巨大的商业价值，而且可以对MS SQL Server构成一定的威胁，但如果想要获得更好的性能、更高的吞吐量，仍然只有在Linux平台上才能实现。

1.7 MySQL复制架构

下面简要叙述下MySQL的各种复制模式，为了方便理解，假设有A、B、C三个MySQL实例，它们的复制模式有如下几种。

- 主从模式 A→B
- 主主模式 A←→B
- 链式复制模式 A→B→C
- 环形复制模式 A→B→C→A



说明 箭头的意思是复制到。

以上4种模式为复制的主要模式，生产中一般建议部署为主从模式，这也是最稳健的一种方式。

为了方便切换，在一定程度上提高可用性，也可以选择主主模式。需要注意的是，主主模式必须确保任何时刻都只有一个数据库是主动（Active）状态，也就是说同一个时刻只能写入一个主（Master）节点，否则可能导致数据异常。

链式或环形复制在生产中很少用到，它们的主要缺点在于，随着节点的增加，整个复制系统的稳健性会下降。

后续运维章节（第12章）对复制会有更多的叙述。各种复制模式的基础都是主从模式，可以说，掌握了主从模式也就掌握了其他各种模式。

1.8 一些基础概念

为了方便后续阅读，让大家对部分概念的理解保持一致，从而更好地理解书中的内容，这里有必要先对下面的这些概念进行阐述。

1. MySQL Server、MySQL实例、MySQL数据库

MySQL数据库指的是实际存在的物理操作系统文件的集合，也可以指逻辑数据的集合。为了访问、处理数据，我们需要一个数据库管理系统，也就是MySQL Server（也称为MySQL服务器）。

MySQL实例指的是MySQL进程及其所持有的内存结构，我们对数据的操作实际上是通过MySQL实例来访问物理数据库文件的。在实际生产中，可以用一个IP:PORT组合来表示一个实例。如“192.168.7.101:3307”这个MySQL实例表示在主机上起起了一个MySQL服务，它的服务端口是3307。如果没有特别说明，本书中的实例一词就是指MySQL实例。

现实语境中，我们一般使用实例来描述对于数据库的操作，对于MySQL数据库、MySQL Server、MySQL实例并没有进行严格的区分，没有特别说明的话，大家可以将它们看作是同等的。

2. 可扩展性

可扩展性也称为伸缩性，指的是系统不断增长其承载能力的能力。它是能满足不断增长的负荷而自身的性能仍然尚可的这样一种能力。

3. 可用性

可用性可以定义为系统保持正常运行时间的百分比，比如一个系统一共运行了100分钟，有99分钟是正常运行的，那么可用性就是99%。

4. 单点故障

单点故障是指系统中的某个部分，一旦失败，将会导致整个系统无法工作。为了消除单点故障，一般需要增加冗余组件或冗余系统。比如服务器的电源冗余、网卡冗余、磁盘RAID阵列，冗余的服务器，备用的数据中心等。如果要设计高可用的服务，单点故障是需要尽量避免的。

5. 读写分离

由于数据库只能接受有限的读请求。对于读请求较多的应用，数据库可能会成为瓶颈，为了增加读的能力，提高扩展性，因此引入了读写分离的技术。比如，利用复制技术配置多个从库，以承担更多的读请求，或者应用程序直接访问读库，或者通过一个负载均衡软件分发读请求。写入操作和一些读操作仍然访问主库。由于MySQL的复制是异步的，所以需要留意复制延时对于读写分离的影响。



小结 本章主要讲述了MySQL的基础架构、查询的执行过程，以及MySQL常见的部署方式。MySQL支持许多存储引擎，大家有必要熟悉和了解最常使用的两个引擎：MyISAM、InnoDB。

第2章 MySQL安装部署和入门

第1章介绍了MySQL的一些基础知识，本章将为读者介绍MySQL的部署、安装及一些常用命令和参数的设置。

2.1 如何选择MySQL版本

在选择MySQL的版本时，要根据生产情况来决定，是对现有生产环境中的数据库进行版本升级呢？还是部署新的数据库呢？如果已经在生产环境中部署了MySQL，那么我们不需要急着将其升级到最新版本，旧的版本已经在生产环境中长期稳定地运行，而新版本刚出来时，往往并不是那么稳定，通常都会有一些Bug需要修复。不稳定版本将导致生产系统的不稳定，所以，如果不是急需新版本的某种特性，或者旧版本有严重的安全隐患，建议继续使用旧的MySQL版本即可。如果新版本已经稳定成熟且生产环境中的版本过于陈旧，那么可以考虑升级旧的MySQL版本。MySQL的发展已经有10多年了，截至2016年6月，Oracle已经发布了MySQL 5.5、MySQL 5.6、MySQL 5.7，其中MySQL 5.5已经比较成熟，读者可以考虑把生产环境中的MySQL 5.0和MySQL 5.1升级到MySQL 5.5，如果需要MySQL 5.6的一些新特性，那么可以考虑将非核心的一些系统升级到MySQL 5.6。

升级到新版本，往往可以获得一定程度上的性能提升，所以，有计划地把生产环境中的MySQL 5.0、MySQL 5.1系统升级为最新的稳定成熟版本是值得的。如果升级的代价比较大，那么保持现状也是可以的。如果生产数据库的部署是标准的，那么可以考虑编写一个自动升级的脚本。先统一升级从库，再升级主库。由于升级主库可能对服务的可用性造成影响，因此需要和相关方协调好时间计划。如果前端有带数据库自动切换功能的中间件，或者应用层能够比较好地处理主从切换，那么把数据库流量临时切换到从库，可以大大减少对生产服务的影响。

对MySQL的分支选择也要慎重，2008年SUN公司收购了MySQL AB，但次年Oracle又收购了SUN，MySQL也是交易的一部分，这之后，Oracle的一系列举动让许多用户和开发者开始质疑MySQL在Oracle旗下的命运，进而开始选择其他替代品。对于MySQL分支的选择，本书不做过大的叙述，现实中，已经有一些重量级公司放弃了MySQL，转向MySQL的其他分支，如MariaDB、Percona Server，但对于绝大部分中小公司来说，使用官方的MySQL或其他分支（如MariaDB），都是比较好的选择，能够满足绝大部分的需求。笔者的建议是如果公司尚在起步阶段，选择Oracle官方的版本即可。我们选择一个产品往往会基于一个重要的理由，它必须是由一个可靠的、成熟的公司或组织来维护的，这能够确保这个产品会得到长久、稳定的支撑。技术发展的目的是解放生产力，如果官方版本仍然能够为企业带来好处，那么坚持使用原来的产品往往是一种比较好的选择，开源和闭源的分裂将是长期的，也是可以共存的，只要是对企业有利的，就不应该拒绝继续使用，除非你有明确的理由放弃它。

2.2 官方版本的安装

下面将以Linux下MySQL 5.1和MySQL 5.5的安装为例进行讲解。为了避免冲突，可以考虑先卸载Linux下自带的MySQL安装包，可使用“`rpm -qa | grep MySQL`”检测是否安装了MySQL相关包。

推荐大家使用二进制版本的安装，主要原因是简单方便，而且官方的二进制包也是经过了充分的测试验证和参数优化的。使用源代码编译的方式安装可能会有一定性能的提升，但在实际应用中，可能会由于编译源码而出现各种问题，如果不清楚编译的参数，建议还是使用二进制版本。此外，无论是使用二进制版本还是源码编译，大规模的部署都必须尽量做到自动化安装，否则安装部署的成本会比较高。

2.2.1 二进制包的安装

首先登录官网，下载二进制版本，步骤如下。

- 1) 进入www.mysql.com。
- 2) 选择downloads (GA)。
- 3) 单击Download from MySQL Developer Zone。
- 4) 单击MySQL Community Server。
- 5) 选择相应的平台、版本，比如，选择64位Linux平台下的MySQL二进制包“Linux-Generic (glibc 2.5) (x86, 64-bit)，Compressed”。

下面开始二进制版本的安装。

1.在root下安装MySQL

这种安装方式为默认方式，这里以“`mysql-5.1.45-linux-x86_64-icc-glibc23.tar.gz`”为例进行讲解。

以root身份登录，运行如下命令安装MySQL。

```
useradd mysql
cd /usr/local
tar zxvf /tmp/mysql-5.1.45-linux-x86_64-icc-glibc23.tar.gz
ln -s mysql-5.1.45-linux-x86_64-icc-glibc23 mysql
cd mysql
cp support-files/my-large.cnf /etc/my.cnf
chown -R mysql .
chgrp -R mysql .
scripts/mysql_install_db --user=mysql
chown -R root .
chown -R mysql data
mv data /home/mysql/
ln -s /home/mysql/data .
```

上面的命令中移动data目录到其他分区（/home/mysql），是因为/usr/local下的磁盘空间可能不够。一般数据目录会存放到和操作系统不一样的分区或磁盘中。

下面是安装后的目录及文件说明。

安装后在安装目录mysql/bin中有如下内容。

- mysqld**: MySQL服务主程序。
- mysqld_safe**: MySQL服务启动脚本。
- mysql**: MySQL命令行工具。
- mysqladmin**: MySQL客户端（管理数据库）。
- perror**: 显示错误码（状态码）含义。
- mysqlbinlog**: 是处理二进制日志文件的实用工具。

将MySQL配置为自启动服务，并启动。

```
cp support-files/mysql.server /etc/init.d/mysqld
chkconfig mysqld on
/etc/init.d/mysqld start
```

运行如下命令设置MySQL root密码。

```
/usr/local/mysql/bin/mysqladmin -u root password 'your_password'
```

之后，使用MySQL自带的脚本或手动执行命令强化安全，删除匿名用户。自动化的方式是在root用户下执行如下命令。

```
./bin/mysql_secure_installation
```

然后按照提示操作，删除匿名账户和空密码的账户。

手动删除匿名账户的操作方法如下。

```
shell> mysql -u root
mysql> DELETE FROM mysql.user WHERE User = '';
mysql> FLUSH PRIVILEGES
```



说明 如果要手动修改授权表（使用INSERT、UPDATE或DELETE等），应该在mysql命令提示符下执行FLUSH PRIVILEGES或mysqladmin flush-privileges告诉服务器再装载授权表，否则更改将不会生效。

建议使用/usr/bin/mysql_secure_installation脚本进行安全配置，它会帮你删除匿名账号。安装完成后，注意把要执行命令的路径添加到系统的PATH变量里，命令如下。

```
vi ~
mysql/.bash_profile
export PATH=/usr/local/mysql/bin:$PATH
```

2. 安装在特定的用户下面

首先，编辑一份自己的配置文件，指定PORT、SOCKET等参数变量。安装和启动的时候需要指定这个配置文件，其他操作和默认安装类似。比如，要安装到“\$HOME/app”下，命令如下。

```
cd $HOME/app
tar zxvf /path/mysql-5.1.45-linux-x86_64-icc-glibc23.tar.gz
ln -s mysql-5.1.45-linux-x86_64-icc-glibc23 mysql
cd mysql
```

```
scripts/mysql_install_db --defaults-file=/home/garychen/app/mysql/my.cnf --user=garychen
```

如果配置文件没有指定数据目录的话，则默认是在/home/garychen/app/mysql/data下。

启动方式如下。

```
./bin/mysqld_safe --defaults-file=/home/garychen/app/mysql/my.cnf --user=garychen &
```



注意 defaults-file参数必须作为第一个参数。

此外，如果是生产环境下的大批量部署，一般建议定制自己的自动化安装脚本，或者通过自动化平台安装。

2.2.2 源码编译安装

本书不建议一般使用者使用源码编译的方式进行安装，如果决定编译安装，最好想想是否真的值得这样做，它可能对于性能提升并无多大作用，但却可能会带来潜在的不稳定因素，你必须确保自己对某些编译选项很熟悉，因为许多生产问题都来自于错误的编译方式。

可采用如下的命令查看已经安装的MySQL编译选项。

```
cat /usr/local/mysql/bin/mysqlbug | grep CONFIGURE_LINE
```

下面以MySQL 5.5为例讲解源码编译安装的基本步骤。

- 1) 下载“MySQL-5.5.33.tar.gz”。
- 2) 确认系统已经安装了cmake。
- 3) 编译安装MySQL，命令如下。

```
# 创建运行  
MySQL的用户  
  
shell> groupadd mysql  
shell> useradd -r -g mysql mysql  
# 开始编译安装  
  
shell> tar zxvf mysql-VERSION.tar.gz  
shell> cd mysql-VERSION  
shell> cmake . -LH # overview with help text  
shell> cmake .  
shell> make-j 8  
shell> make install  
# 安装后配置、初始化数据库  
  
shell> cd /usr/local/mysql  
shell> chown -R mysql .  
shell> chgrp -R mysql .  
shell> scripts/mysql_install_db --user=mysql  
shell> chown -R root .  
shell> chown -R mysql data  
#启动  
MySQL Server  
shell> cp support-files/my-medium.cnf /etc/my.cnf  
shell> bin/mysqld_safe --user=mysql &  
#添加到自启动服务  
  
shell> cp support-files/mysql.server /etc/init.d/mysql.server  
shell> chkconfig mysql.server on  
#设置  
root密码  
  
/usr/local/mysql/bin/mysqladmin -u root password 'your_password'  
#类似二进制安装，还需要进行安全强化，运行  
  
. /bin/mysql_secure_installation
```


2.3 其他MySQL分支的安装

一些其他MySQL的分支，提供了更高的性能和更多的特性，如Percona Server、MariaDB等，它们的二进制版本安装类似于官方版本，读者可参考对应分支的安装文档进行部署安装。注意，安装前一定要仔细阅读它们的安装文档。

2.4 安装InnoDB Plugin

对于MySQL 5.0、MySQL 5.1版本，有时我们可能会想要安装InnoDB Plugin，因为它较之Built-in版本新增了一些特性。而且一些性能测试也表明，InnoDB Plugin的性能、伸缩性明显优于MySQL 5.1里内置的InnoDB。不过，在这么做之前要先留意一下不同的InnoDB Plugin版本和MySQL版本的兼容性。对于源代码编译的MySQL，一般可以用编译的InnoDB代替内建的InnoDB，但是二进制版本的InnoDB插件通常只适用于特定的MySQL版本。

使用二进制版本安装启用InnoDB Plugin的具体步骤如下。

- 1) 确认MySQL没有在运行。如果正在运行，那么应该先设置变量`innodb_fast_shutdown`。

```
SET GLOBAL innodb_fast_shutdown=0;
```

然后再关闭数据库（对于大数据库而言，可能耗时会较多）。

- 2) 在参数文件[mysqld]节中增加以下参数。

```
shell>vi my.cnf
ignore_builtin_innodb
plugin-load=innodb=ha_innodb_plugin.so
plugin_dir=/usr/local/mysql/lib/plugin
```

- 3) 启动数据库，启动数据库后执行下面的语句。

```
INSERT INTO mysql.plugin VALUES('INNODB', 'ha_innodb_plugin.so') ;
INSTALL PLUGIN INNODB SONAME 'ha_innodb_plugin.so';
INSTALL PLUGIN INNODB_TRX SONAME 'ha_innodb_plugin.so';
INSTALL PLUGIN INNODB_LOCKS SONAME 'ha_innodb_plugin.so';
INSTALL PLUGIN INNODB_LOCK_WAIT SONAME 'ha_innodb_plugin.so';
INSTALL PLUGIN INNODB_CMP SONAME 'ha_innodb_plugin.so';
INSTALL PLUGIN INNODB_CMP_RESET SONAME 'ha_innodb_plugin.so';
INSTALL PLUGIN INNODB_CMPMEM SONAME 'ha_innodb_plugin.so';
INSTALL PLUGIN INNODB_CMPMEM_RESET SONAME 'ha_innodb_plugin.so';
```

- 4) 关闭数据库，然后再去掉参数文件my.cnf中的`plugin-load`和`plugin_dir`行，之后重新启动数据库，运行“`SELECT@@@innodb_version;`”以确认版本。

2.5 常用命令

本节先介绍几个常用命令，如mysql、mysqladmin、mysqldump的简单用法。后续章节还会再详述这些命令的使用。

2.5.1 使用mysql命令

首先，需要留意区分MySQL的大小写。标准的说法是，MySQL指MySQL服务器，mysql指客户端。

从Unix/Linux系统下发展出来的MySQL有着优良的设计，客户工具的所有选项都可以保存到一个“~/.my.cnf”的用户级配置文件里的[client]部分中，而且它把适用于MySQL的选项集中在了[MySQL]部分。可以先把默认的用户名、密码、端口等在“.my.cnf”文件中配置好，以便简化登录。

另外，要说明一下，本章阐述的一些命令，为了显示方便，可能会省略用户名、密码、socket文件的功能连接参数。

首先给出连接并登录数据库时会涉及的命令，分别如下。

通过IP、端口远程连接的命令。

```
mysql -h ip_address -P your_port -u username -p
```

通过TCP/IP协议进行本地连接的命令。

```
mysql -u username -h 127.0.0.1 -P your_port
```

通过socket文件进行本地连接的命令。

```
mysql -u username -S /path/to/mysql.sock
```

阅读在线帮助的命令。

```
mysql> help contents
```

退出的命令。

```
mysql > exit
```

简单查询的命令。

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> SELECT SIN(PI()/4), (4+1)*5;
```

MySQL客户端还提供了一些简写命令，这些简写命令只能出现在命令行的中间或末尾，具体如下。

```
mysql> help
      List of all MySQL commands:
      Note that all text commands must be first on line and end with ';'
ego      (\G) Send command to MySQL server, display result vertically.
system   (\!) Execute a system shell command.
tee      (\T) Set outfile [to_outfile]. Append everything into given outfile.
pager    (\P) Set PAGER [to_pager]. Print the query results via PAGER.
edit    (\e) Edit command with $EDITOR.
```

下面来看一个示例。

```
mysql> pager cat > /tmp/log.txt
mysql> pager less -n -i -S -F
```

命令less的“-S”选项可以让你用方向键进行浏览，这对于长行的显示很有用。其中的参数说明分别如下。

·**-i**: 搜索时忽略大小写，但如果搜索的字符串中包含大写字母，那么这个选项不起作用。

·**-n**: 禁用行号功能，加速浏览大文件。

·**-F**: 如果屏幕可以显示的话，就直接退出。

使用以下命令，不仅可以将结果输出到屏幕上，还可以通过tee命令记录到文件中。

```
mysql> pager cat | tee /dr1/tmp/res.txt \
| tee /dr2/tmp/res2.txt | less -n -i -S
```

使用如下命令，会列出所有可见的数据库。

```
mysql> SHOW DATABASES;
```

切换到**test**数据库时的命令如下。

```
mysql> USE test #如果有许多表，使用
use db_name可能会比较慢，可以使用
mysql -A进行加速
```

显示当前数据库的命令如下。

```
mysql> SELECT DATABASE();
```

创建数据库**menagerie**的命令如下。

```
MySQL > CREATE DATABASE menagerie;
```

删除数据库的命令如下。

```
mysql> DROP DATABASE IF EXISTS menagerie;
```

创建用户，并赋予其对**menagerie**库的权限的命令如下。

```
mysql> GRANT select,insert,update,delete ON menagerie.* TO 'your_name' @ 'your_client_host';
```

列出当前数据库下所有表的命令如下。

```
mysql> SHOW TABLES;
```

查看表结构的命令如下。

```
mysql>DESC pet;
mysql>SHOW FULL TABLES;      #多了第二列，用于显示
```

Table_type

输入表名、列名等信息时，可以按TAB键补全，“~A”可关闭这个功能。

创建表的命令如下。

```
CREATE TABLE shop (
    article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
    dealer  CHAR(20)          DEFAULT ''      NOT NULL,
    price   DOUBLE(16,2)       DEFAULT '0.00' NOT NULL,
    PRIMARY KEY(article, dealer));
```

插入初始化数据的命令如下。

```
INSERT INTO shop VALUES
(1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
(3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

查询数据的命令如下。

```
SELECT * FROM shop;
```

执行SQL文件的3种方式如下。

```
mysql -e "source batch-file"
mysql -h host -u user -p < batch-file
mysql> source /path/filename;
```

如果有长的屏幕输出，可以转储到文本或使用more进行查看。

```
mysql < batch-file | more
mysql < batch-file > mysql.out
```

表2-1针对mysql客户端的提示给出了解释。

表2-1 mysql客户端的提示说明

prompt	解释	prompt	解释
mysql>	等待下一个命令	">"	等待下一行，等待下一个结束字符串反引号"
>>	等待多行命令的下一行	?>	等待下一行，等待下一个结束标记字符反引号?
>>>	等待下一行，等待下一个结束字符串单引号'	/>>	等待下一行，等待下一个注释结束标记/*

如果输入错了，需要清除当前的输入字符，可输入\c来实现。在如下示例中，少输入了单引号，我们使用\c清除所有的输入字符，回到提示符下。

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
      '> '\c
      \c前还需要输入单引号
mysql>
```

修改用户密码的命令如下。

```
mysql> SET PASSWORD FOR user_name@ip_address = password('1234');
```

显示当前连接、客户端、数据库字符集等信息的命令如下。

```
mysql> STATUS
```

显示MySQL支持的排序方式的命令如下。

```
mysql> SHOW COLLATION;
```

下面的命令将展示前一条命令的警告信息。

```
mysql> SHOW WARNINGS;
```

展示可用引擎的命令如下。

```
mysql> SHOW ENGINES;
```

还可以使用下面的语句代替SHOW ENGINES，并检查你感兴趣的存储引擎的变量值。

```
mysql> SHOW VARIABLES LIKE 'have%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_archive  | YES   |
| have_bdb      | NO    |
```

SHOW命令的精确输出将随所使用的MySQL版本（和启用的特性）的不同而有变化。第2列的值表示各特性支持的级别，如表2-2所示。

表2-2 第2列的值及其含义

值	含义
YES	支持该特性并且已经激活
NO	不支持该特性
DISABLED	支持该特性但被禁用

如下命令可得到表的引擎（engine）。

```
mysql> USE information_schema;
mysql> SELECT table_name,engine FROM information_schema.tables WHERE table_schema = 'Your Database Name';
```

如下命令可查看当前连接和服务器的事务隔离模式。

```
SELECT @@tx_isolation,@@global.tx_isolation;
```

如下命令可查询是否自动提交事务，

```
SELECT @@autocommit;
```

如下命令可用于查询sql_mode。

```
SELECT ROUTINE_SCHEMA, ROUTINE_NAME, SQL_MODE FROM INFORMATION_SCHEMA.ROUTINES;
SELECT EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE, TRIGGER_NAME, SQL_MODE FROM INFORMATION_SCHEMA.TRIGGERS;
```

也可以通过设置OS环境变量的方式来改变连接的socket文件和TCP端口，命令如下。

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock  
shell> MYSQL_TCP_PORT=3307  
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
```

2.5.2 使用mysqladmin命令

在使用mysqladmin命令时，如下命令可显示参数设置。

```
mysqladmin -p variables |grep log_queries_not_using_indexes
```

设置root密码的命令如下。

```
myaqladmin -u root -p password "new password"
```

如下命令可显示状态变量，一般使用-r参数显示两次命令执行期间的增量值。

```
mysqladmin extended-status -uroot -r -i 10
```

其中，“extended-status”显示的是服务器状态变量和值。

-r: 显示当前状态变量和上一次运行命令状态变量的差值。

-i: 重复执行命令的间隔时间。

如下命令可显示当前连接的线程。

```
mysqladmin -uroot -pnemo1234admin processlist
```

如下命令可用于关闭数据库。

```
mysqladmin shutdown
```

2.5.3 使用mysqldump命令

在使用mysqldump命令时，如下命令可用于备份数据库。

```
mysqldump -uroot --hex-blob db_name > db_name.sql
```

增加压缩功能的命令如下。

```
mysqldump -uroot --hex-blob db_name | gzip > db_name.sql.gz
```

也可以使用如下mysql命令恢复数据。

```
mysql < db_name.sql
```

2.6 MySQL的主要参数设置

研发、测试人员往往熟悉SQL语句的撰写、表结构的设计，而不熟悉MySQL的配置，这可能会导致一些困惑，比较常见的是，在线上运行良好的查询，到了线下就变慢了，下面介绍几个常见的参数配置。一般情况下，配置好这几个参数可以满足大部分开发环境和测试环境的要求。

(1) innodb_buffer_pool_size

为了提升写性能，可以把要写的数据先在缓冲区（buffer）里合并，然后再发送给下一级存储。这样做可提高I/O操作的效率。InnoDB Buffer Pool就是InnoDB用来缓存它的数据和索引的内存缓冲区，可由`innodb_buffer_pool_size`设置其大小。理论上，将这个值设置得越高，访问数据需要的磁盘I/O就越少。常见的做法是让这个值大于热点数据，这样可以获得比较好的性能。如果不清楚环境的数据量和访问模式，建议将其设置为机器物理内存大小的70%~80%。

(2) innodb_log_file_size

日志组里每个日志文件的大小。在32位计算机上日志文件的合并大小必须小于4GB，默认大小是5MB，在生产环境下，这个值太小了。官方文档推荐的值为从1MB到1/N的缓冲池大小，其中N是日志组里日志文件的数目（由`innodb_log_files_in_group`变量来确定，一般默认为2）。值越大，在缓冲池中需要检查点刷新的行为就越少，因此也越节约磁盘I/O，但更大的日志文件也意味着在崩溃时恢复得更慢。建议将日志文件的大小设置为256MB或更大，这样可以满足一般情况下的需要。

(3) innodb_flush_log_at_trx_commit，建议设置为2

这个选项的默认值是1。当设置为2时，在每个事务提交时，日志缓冲被写到文件中，但不对日志文件做刷新到磁盘的操作。对日志文件的刷新每秒才发生一次。所以，理论上，操作系统崩溃或掉电只会丢失最后一秒的事务。

(4) sync_binlog，建议设置为0

如果是`autocommit`模式，那么每执行一个语句就会向二进制日志写入一次，否则每个事务写入一次。如果`sync_binlog`的值为正，那么每当`sync_binlog`参数设定的语句或事务数被写入二进制日志后，MySQL服务器就会将它的二进制日志同步到硬盘上。默认值是0，不与硬盘同步。值为1是最安全的选择，因为崩溃时，你最多丢掉二进制日志中的一个语句或事务。但是，这也是最慢的选择，成本昂贵。

另外，在MySQL中，数据库对应数据目录中的目录。数据库中的每个表至少对应数据库目录中的一个文件（也可能是多个，取决于存储引擎）。因此，所使用操作系统的大小写敏感性决定了数据库名和表名的大小写敏感性。在大多数Unix中数据库名和表名对大小写敏感，而在Windows中对大小写则不敏感。我们应设置变量`lower_case_table_names=0`，这也是Unix/Linux系统的默认值。开发环境、测试环境的MySQL也建议部署在Unix/Linux平台，尽可能和生产环境一致。



小结 本章介绍了如何在生产环境中部署MySQL，一般情况下，掌握二进制版本的安装即可，本章也介绍了几个常用的命令`mysql`、`mysqladmin`、`mysqldump`，这些命令的选项较多，全部掌握不太现实，但对它们的常用用法应该熟悉。

第二部分 开发篇

本篇首先讲述数据库开发的一些基础知识，如关系数据模型、常用的SQL语法、范式、索引、事务等，然后介绍编程开发将会涉及的数据库的一些技巧，最后结合生产实际，提供一份开发规范供大家参考。

第3章 开发基础

本章将为读者介绍MySQL数据库相关的开发基础，首先，介绍一些基础概念，然后讲解关系数据模型和SQL基础。由于在互联网开发者中，PHP开发者占据了相当大的比重，因此这里也将简要介绍下PHP开发者应该掌握的一些基础知识和开发注意事项。最后，要接触的是MySQL数据库更深层次的内容——索引、主键、字符集等。

3.1 相关基础概念

(1) 框架

在软件开发过程中，研发人员经常借助框架（framework）来辅助自己进行软件开发。成熟的框架可以帮助处理很多细节性的问题，并完成一些基础性的工作，如生成访问数据库的代码、简化网络编程，这样开发者就会有更多的时间和精力专注于业务逻辑的设计。但目前仍存在的一个问题，一些框架对于数据库的使用不符合我们的预期，或者说不友好，故而有必要先了解一下开发框架是如何存取数据的。大家有兴趣的话，可深入学习和使用如下这些业内使用比较广泛的一些框架，如 Django（Python）、Ruby on Rails（Ruby）、Zend Framework（PHP）、Spring（JAVA）等。

(2) 数据模型

数据模型（data model）是数据的定义和格式，即数据是如何组织的。关系数据模型是以二维表的结构来表示实体与实体之间的联系，每个二维表又可称为关系。关系可以看作是一系列记录的集合。如，员工关系表（见表3-1）和项目关系表（见表3-2）。

表3-1 员工关系表

员工编号	姓名	性别	职位	部门
123	张三	男	软件开发经理	应用研发部

表3-2 项目关系表

项目编号	名称	项目经理	研发	测试	运维	DBA	产品
1110	彩票	王五	张三	李四	曾六	王二	陈九

从以上两个关系表中可以看出，项目表和员工表是存在某种关系的。众多的关系表，以及关系表之间的关系，构成了关系数据模型，而支持关系模型的数据库管理系统则称之为关系数据库管理系统。

其他的模型还有XML和图数据模型（graph data model）等。

XML是一种层次结构的数据结构，使用标签、标签值来标识信息，如下面的这个xml文件。

```
<note>
  <to>George</to>
  <from>John</from>
  <heading>Reminder</heading>
  <body>Don't forget the meeting!</body>
</note>
```

而图数据模型存储的数据则是以点、线的方式进行存储的。

(3) schema

schema可译作“模式”，不同的数据库管理系统，schema的意义会有些不同。依据维基百科的定义：schema指的是用数据库管理语言描述的数据结构，它定义了数据是如何组织构建的。

典型的关系数据模型，是以数据库表的形式来组织数据的，数据存储于一系列设计好的表中。也就是说，关系数据库的schema就是数据库中各种关系的结构化描述。一般来说，数据建模就是设计数据表的过程，一般在项目初期就设计好表结构，在开发过程中可能会不断地调整表结构，但一旦应用上线，表结构往往就不会频繁变更了。若项目积累了大量数据，这时再修改表结构可能会很耗时，从而严重影响在线服务，所以前期进行一个优良的数据库表设计是很有必要的，这也考验着开发人员的数据建模能力。数据库表的设计一般由经验丰富的开发人员来负责，如果DBA时间精力允许，也会参与到重要的项目数据库表设计中。

MySQL中的schema可以看作是数据库（database）的同义词。我们创建一个schema，其实就是创建一个数据库（create database）。而在其他数据库中， schema的概念则略有不同。

(4) 结构化数据

结构化数据通常是指被记录信息的类型，格式等属性是固定的，一般可存储于关系数据库或电子表格中，可以用数据记录的形式进行表达和存储，如产品及其零部件的名称、代号、设计日期、类型等信息。结构化数据往往需要预先定义好业务数据类型的模型，确定这些数据类型是如何存储、处理和访问的。例如确定业务数据的哪些字段信息需要存储，以及这些信息的数据类型（数字、货币、字符串、日期等）和数据输入的校验（如字符个数、日期范围等）。很长时间以来，关系数据库或电子表格软件是处理结构化数据的最佳工具，所以业内也有人简单地把存储在关系数据库中能用二维表格表示的数据称为结构化数据，如来自于企业内部已经被转换成固定规则、格式的数据，而把不方便用关系数据库存取的数据称为非结构化数据，如市场比较和分析报告、股票行情等就是以非结构化的、不可预测的格式呈现的数据。

(5) 非结构化数据

有些信息无法用数字或统一的结构来表示，或者说没有一个预定义的数据模型，如文本、照片和图形图像、声音、视频、网页、PDF文件、PowerPoint演示文稿、电子邮件、博客、Wiki和文字处理文档等，我们将其称之为非结构化数据。

(6) 半结构化数据

半结构化数据介于结构化数据和非结构化数据之间，它可看作是一种结构化数据，但是缺乏严格的数据模型，半结构化数据可通过标签或其他类型的标记识别数据中的某些元素，但半结构化数据不具有刚性结构。XML和其他标记语言经常被用来管理半结构化数据。

例如，文字处理软件现在可以定义元数据，用于显示作者的姓名和创建日期，但数据的主体——文本文件仍然是非结构化数据。电子邮件有发件人、收件人、日期、时间和其他标识信息，但电子邮件消息的内容和附件仍然是非结构化数据。照片或图形图像能使用一些关键字进行标识，如创作者、日期、地点和关键字，从而能够组织和定位照片和图形图像，但图像本身是非结构化数据。

相对于非结构化数据，结构化数据往往存储于关系数据库中，可以利用关系数据库进行高效地存储和检索，但现实中的数据并不是总能被固定的结构来描述的，生活也并不总是合适整齐的小盒子。非结构化数据和半结构化数据是现实世界的主要数据，而且正在以惊人的速度激增，它们的增长比结构化数据的增长更快，在大数据时代，非结构化（半结构化）数据的提取、存储和管理是一个难点，非结构化数据能否被有效地管理和应用，这对于企业未来的发展道路影响深远。

(7) DDL

数据定义语言（Data Definition Language, DDL）是负责数据结构定义与数据库对象定义的语言。为了设计schema，如创建数据库，创建表，这时就需要用到数据定义语言。我们常用的有CREATE、ALTER、DROP语句。例如，创建数据库的语句如下。

```
CREATE DATABASE database_name;
```

创建表的语句如下。

```
CREATE TABLE table_name (id INT, name VARCHAR(10));
```

添加字段的语句如下。

```
ALTER TABLE table_name ADD COLUMN column_name INT ;
```

删除表的语句如下。

```
DROP TABLE table_name;
```

(8) DML

数据操作语言（Data Manipulation Language, DML）是用来查询和修改数据的语句，包括SELECT、INSERT、UPDATE、DELETE 4种语句，分别代表查询、插入、更新与删除，有很多开发人员将它们称之为“CRUD”（Create、Read、Update和Delete），对应的操作见表3-3。

表3-3 CRUD对应的SQL语句

Operation	SQL
Create	INSERT
Read (Retrieve)	SELECT
Update (Modify)	UPDATE
Delete (Destroy)	DELETE

3.2 数据模型

3.2.1 关系数据模型介绍

目前数据库领域使用最广泛的就是关系数据模型，业内主流的数据库产品都是建立在关系数据模型之上的，如Oracle、MS SQLServer、MySQL、PostgreSQL、DB2。关系型数据库系统的技术发展了几十年，已经相当成熟，在数据库中也得到了高效的实现。关系型数据库管理系统的标准语言——结构化查询语言（SQL），是一种高级的非过程化编程语言，它已经成为事实上工业标准而被广泛使用，而且也变成了一项必须被程序员掌握的标准技能。

下面仍然以3.1节的两个表为例（见表3-4和表3-5），说明一些概念。

表3-4 员工关系表（employee）

员工编号	姓名	性别	岗位	部门	绩效得分
123	张三	男	软件开发经理	应用研发部	80
124	王刚	男	系统工程师	运维部	NULL
125	陈华	男	网络工程师	运维部	90
126	王家林	男	系统工程师	游戏研发部	100
127	张卫	男	开发工程师	游戏研发部	NULL

表3-5 项目关系表（project）

项目编号	名称	项目经理	研发	测试	运维	DBA	产品
1110	彩票	456	123	458	125	234	235

从表3-4和表3-5可以看出，关系数据模型是由一系列的“关系”组成的。“关系”也就是我们所说的表（table）。每个表也存在一个或多个属性（字段），如“员工编号”、“姓名”、“性别”。每个字段均有对应的数据类型（type），如“整型”、“字符型”、“枚举类型”。关系模型建立后，就可以在这些关系（表）中插入、修改、删除、查询数据了。

1.关于NULL

如果某个字段的值是未知的或未定义的，数据库会提供一个特殊的值NULL来表示。NULL值很特殊，在关系数据库中应该小心处理。例如对表employee，运行查询语句“select*from employee where绩效得分<=85 or>绩效得分>85；”可能很多人认为这样能获取所有记录，但实际上，由于王刚和张卫的绩效得分是未知的（NULL），因此他们不会被包含在查询结果中。

2.关于key和索引

key常指表中能唯一标识一笔记录的字段（属性）或多个字段的组合。现实中，key和索引可以简单地看作同义词，key不一定唯一标识一笔记录，本书以后的论述中会使用“索引”、“主键索引”、“唯一索引”这些术语。我们可以通过某个记录的索引/key去查找记录。数据库管理系统为了高效地检索记录，往往会创建各种索引结构加速检索记录，或者按照索引/key的顺序存储记录，所以基于记录的索引/key会很容易查找到记录。关系数据库中的表之间的关联往往也是通过索引来进行关联的，比如上面的project表，项目组成员存储的是员工编号，可以通过员工编号和另外一张员工关系表——employee表（员工编号字段上有主键索引）进行关联。

3.2.2 实体-关系建模

由于设计人员、研发人员和最终用户看待和使用数据的方式不同，因此可能会导致数据库的设计不能反映真实的需求，以

及后期出现的扩展性问题，为了能够更准确地理解数据的本质，理解使用这些数据的方法，我们需要有一个通用的模型，这个模型和技术实现无关。实体关系图（ER模型）就是这样一个通用模型的例子。以下介绍ER建模的一些关键概念。

(1) ER建模

1976年Peter Chen首次提出了Entity Relationship Modeling（实体关系建模）概念，并发明了陈氏表示法（Peter Chen's notation）。随着问题复杂度的增加，适应范围的增广，截至今天出现了许多ER模型的表示法，如Barker ER Information Engineering（IE）和IDEF1X或Crow's foot表示法。各种表示法都有它们的优缺点和适用领域，但它们都基于同样的建模概念。

ER建模是一种自上而下的数据库设计方法。我们通过标识模型中必须要表示的重要数据（称为实体）及数据之间的关系开始ER建模，然后增加细节信息，如实体和关系所要具有的信息（称为属性）。该方法的输出是实体类型、关系类型和约束条件的清单。

(2) UML

UML（Unified Modeling Language，统一建模语言）是一种分析人员和开发人员广泛使用的标准建模语言，它可以以图形化的方式表示实体、关系。UML最初用于软件设计，目前已经扩展到业务和数据库设计。UML包括分析、实施、部署过程中指定任何事项所必需的元素和图表。通过使用几种图表和数十种元素，UML能表达不同程度的系统抽象。对于ER建模，我们只需要了解常用于ER建模的一些视图和表示即可。

(3) 实体

实体代表现实世界的一组对象集合，可以粗略地认为它是名词，如学生、雇员、订单、演员、电影。实体一般用矩形来表示。

(4) 关系

关系指特定实体之间的关系。可以粗略地认为是动词，如公司拥有员工、演员演电影。关系用线来表示。一般为二元关系。

关系的基数指参与关系的实体数目。二元关系的基数就是我们所说的一对一、一对多、多对多。在数据库设计中，需要选择合适的基数表示法，如IDEF1X表示法、关系表示法或Crow's foot表示法。本书中的例子一般使用Crow's foot表示法，下面简要介绍下Crow's foot表示法。

对于Crow's foot表示法，实体表示为矩形框，关系表示为矩形框之间的线，线两端的形状表示关系的基数。空心圆表示零或多，单阴影线标记表示一或多，单阴影线标记和空心圆表示零或一，双阴影线标记表示恰好为一。

许多建模工具都可以使用Crows'foot表示法，如ARIS、System Architect、Visio、PowerDesigner、MySQL Workbench等。

属性指实体或关系的特征，如实体雇员的姓名、地址、生日、身份证ID等。如果要一起显示实体和属性，那么就把代表实体的矩形分为两部分，上半部分显示实体名，下半部分列出属性名。

在图3-1中，Artist（艺术家）实体和Song（歌曲）实体的关系是艺术家演唱歌曲。



图3-1 Artist实体和Song实体的关系

这两个实体使用的是Crow's foot表示法，靠近Song实体一端的符号表示“0、1或更多”，靠近Artist一端的符号表示“1且只有1个”，所以图3-1表示一个艺术家可以演唱0首、1首或者多首歌曲。

关于ER建模更详细的信息，请阅读其他相关书籍。

3.2.3 其他数据模型

1.XML数据模型

对于结构化数据，除了关系模型，还可以使用XML数据模型存取数据。XML（eXtensible Markup Language）是可扩展标记语言，最开始设计XML的目的是为了在Internet上交换数据。标记是指计算机所能理解的信息符号，通过此种标记，计算机之间可以处理包含各种信息的文章等。如何定义这些标记？既可以选择国际通用的标记语言，比如HTML，也可以使用像XML这样由相关人士自由决定的标记语言，这就是语言的可扩展性。

XML被广泛用作跨平台之间数据交互的形式，主要针对数据的内容，通过不同的格式化描述手段（XSL、CSS等）来完成最终的形式展现（生成对应的HTML、PDF或其他的文件格式）。

常用的查询语言是XPath，即XML路径语言（XML path language），它是一种用来确定XML文档中某部分的位置的语言。XPath基于XML的树状结构，提供在数据结构树中找寻节点的能力。

在图3-2中，一个形式良好的XML文档或XML字符串，经过CSS或XSL解析器的解析，最终生成客户端可接受的展现形式。

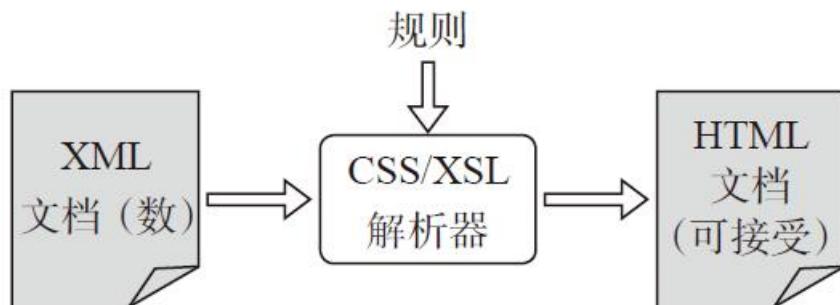


图3-2 XML文档（数据）解析输出的过程

以下是一个XML的例子。

```
<?xml version="1.0" ?>
<person sex="female">
    <firstname>Anna</firstname>
    <lastname>Smith</lastname>
</person>
```

可以看到XML的格式和HTML文件比较类似，但两者也有不同之处。XML被设计为传输和存储数据，其标签描述的是数据的内容。HTML被设计用来显示数据，其标签是用来格式化数据的。

由于XML文件的标签描述的是数据的内容，因此XML文件可以看作“自描述”的文件。

一个形式良好的XML主要包括如下三个基本部分。

·元素，如上面的person。元素允许嵌套，如person包含子元素firstname、lastname。元素有开始标签和关闭标签，如上面的

<person></person>。

·属性，元素还可以拥有属性，如上面例子中的sex=“female”。

·文本，如上面例子中的Anna、Smith。

XML作为一项数据交换的标准被广泛使用，因此某种意义上，XML也是关系数据模型的竞争者。表3-6对关系数据模型和XML数据模型做了简要对比。

表3-6 关系数据模型和XML数据模型的对比

比较项	关系	XML
数据结构	二维表	层次结构(树形结构)
模式	预先定义好模式，有固定的模式后才适合存入数据	很灵活，“自描述”，数据和 schema 是混合在一起的，相当于模式是可以灵活变化的
查询	简单友好的查询语言(SQL)	XPATH，不那么易用，需要一些技巧
排序	无(InnoDB按照主键顺序存储数据是一个特例)，查询一般需要使用 ORDER BY 子句进行排序	可随意排序。XML 文档中的子元素可以按照自己的规则进行排序
实现	关系数据库系统已经很成熟了，原有的实现	往往是关系数据库的附加功能。虽然可以以 XML 的形式存取数据，但内部实现仍然是关系数据模型，需要转化为关系数据模型进行存取。这样关系数据库可以同时处理关系数据和 XML 数据，扩展了它的功能

在下面的XML文档中，第一本书没有输入price（价格）信息，而后面的两本书添加了price信息，这种数据结构的一致性在XML中是允许存在的，这就意味着，可以在以后给<book>元素添加或删除子元素，因此大大增加了灵活性。

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year> </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J. K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

2.JSON数据模型

JSON (JavaScript Object Notation) 与XML类似，也适用于存储半结构化数据。JSON比XML出现得更晚，不像XML那样有比较完善的工具支持，但由于JSON更简洁，更符合程序语言的数据表达方式，因此，在互联网开发中，一般选择JSON而不是XML，JavaScript的很多工具包如jQuery、ExtJS等都大量使用了JSON。事实上，JSON已经成为了一种前端与服务器端的数据交换格式，前端程序通过Ajax发送JSON对象到后端，服务器端脚本对JSON进行解析，将其还原成服务器端对象，然后进行一些处理，反馈给前端的仍然是JSON对象。

尽管JSON是JavaScript的一个子集，但JSON是独立于语言的文本格式，并且采用了类似于C语言家族的一些习惯。许多程序语言都有解析器，可用来处理JSON数据。

JSON用于描述数据结构，有如下几种存在形式。

·对象：是一个无序的“名称/值”对集合。一个对象以“{”（左大括号）开始，“}”（右大括号）结束。每个“名称”后跟一个“：“（冒号）；“名称/值”对之间使用“,”（逗号）分隔。

·数组：是值（value）的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间使用“,”（逗号）分隔。

·值：可以是双引号括起来的字符串（string）、数值（number）、true、false、NULL、对象（object）或数组（array）。这些结构可以嵌套。

下面来举一个例子。

```
"employees": [
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]
```

在上面的例子中，对象employees是包含三个对象的数组。每个对象代表一条关于某人（有姓和名）的记录。

相对于传统的关系型数据库，一些基于文档存储的NoSQL非关系型数据库则选择JSON作为其数据存储格式，比较知名的产品有：MongoDB、CouchDB、RavenDB等。下面两个表（表3-7和表3-8）列举了其与关系数据模型和XML数据模型的不同。

表3-7 JSON与关系数据模型的比较

比较项	关系	JSON
数据结构	二维表	嵌套的集合（数组），具有层级结构
模式	预先定义好模式，有固定的模式后才能存入数据	很灵活，数据和 schema 是混合在一起的，相当于模式是可以灵活变化的
查询	简单友好的查询语言（SQL）	般自己写程序处理
排序	无（InnoDB按照主键顺序存储数据是一个特例），查询一般需要使用 order by 子句进行排序	array（数组）是有序的
实现	关系数据库系统已经很成熟了，原生的实现	一般程序语言操作，一些 NoSQL 系统选择 JSON 作为其数据存储格式

表3-8 JSON与XML数据模型的比较

比较项	XML	JSON
表达数据的方式	更多冗余信息，需要占用更多字符	更简洁
复杂度	更复杂	更简单
验证数据、约束、模型定义	DTDs、XSDs 等手段广泛应用	有 JSON schema，但应用很少
操作接口	比较重，不适合用编程语言进行操作	编程简单，是程序语言易于使用的数据结构
查询	有 XPath、XQuery、XSLT 等成熟手段	工具不成熟，往往需要自己编写代码来处理

虽然XML会比JSON的存储占据更多字节，但如果只是用到XML的一个子集，用基本的结构，同样也是很简洁的。那为什么有人会觉得XML复杂呢？更多的原因是XML拥有大量的特性，设定很多，深入了解需要花费很多功夫，而JSON的简单模型，很快就可以掌握了。

JSON与XML最大的不同之处在于XML是一个完整的标记语言，而JSON不是，JSON仅仅是一种表达传输数据的方式，正如名字所言，JavaScript对象表示法（JavaScript Object Notation, JSON）是通过字符来表示一个对象的。XML的设计理念与JSON不同。XML利用标记语言的特性提供了绝佳的扩展性能，如果数据模型复杂多变，想要单独定义自己传输数据的模型，那么XML将是一个很好的选择，但是我们所使用的数据结构往往不需要变动，这时JSON更简洁，它的数据结构很像程序语言定义的数据结构，在你预先知道JSON结构的情况下，可以写出实用美观、可读性强的代码，如果要存储或传输的数据格式出现了变化，此时就需要重新编码来解析存储，这方面的成本往往是可以接受的。

3.3 SQL基础

SQL是一种高级查询语言，它是声明式的，也就是说，只需要描述希望怎么获取数据，而不用考虑具体的算法实现。

3.3.1 变量

MySQL里的变量可分为用户变量和系统变量。

1. 用户变量

用户变量与连接有关。也就是说，一个客户端定义的变量不能被其他客户端看到或使用。当客户端退出时，该客户端连接的所有变量将自动释放。这点不同于在函数或存储过程中通过DECLARE语句声明的局部变量，局部变量的生存周期在它被声明的“**BEGIN...END**”块内。对于用户变量的值，可以先保存在用户变量中，然后以后再引用它；这样就可以将值从一个语句传递到另外一个语句。

用户变量的形式为`@var_name`。

设置用户变量的一个途径是执行SET语句，语法如下。

```
SET @var_name= expr[, @var_name= expr] ...
```

对于SET，可以使用“`=`”或“`:=`”作为分配符。分配给每个变量的expr可以为整数、实数、字符串或NULL值。如：

```
mysql> SET @t1=0, @t2=0, @t3=0;
```

或：

```
SET @minMid=(select min(id) FROM table_name) ;
```

2. 系统变量

MySQL服务器维护着两种系统变量：全局变量影响MySQL服务的整体运行方式；会话变量影响具体客户端连接的操作。

当服务器启动时，它将所有全局变量初始化为默认值。这些默认值可以在选项文件中或在命令行中对指定的选项进行更改。服务器启动后，通过连接服务器并执行SET GLOBAL var_name语句，可以动态更改这些全局变量。要想更改全局变量，必须具有SUPER权限。

服务器还为每个连接的客户端维护一系列的会话变量。在连接时使用相应全局变量的当前值对客户端的会话变量进行初始化。对于动态会话变量，客户端可以通过SET SESSION var_name语句更改它们。设置会话变量不需要特殊权限，但客户端只能更改自己的会话变量，而不能更改其他客户端的会话变量。

访问全局变量的任何客户端都可以看见对全局变量所做的更改。然而，它只影响更改后连接的客户的相应会话变量，而不会影响目前已经连接的客户端的会话变量（即使客户端执行SET GLOBAL语句也不影响）。也就是说，如果你的连接是短连接，那么修改全局变量后，客户端有重连的操作，就会立刻影响到客户端。而对于长连接、连接池来说，连接可能一直在MySQL里没有被销毁，也就不会有重连的操作，所以这种情况下对全局变量的修改一般不会影响到客户端。

可以使用如下几种语法形式来设置或检索全局变量或会话变量（下面的例子使用sort_buffer_size作为示例变量名）。

要想设置一个GLOBAL变量的值，可使用下面的语法。

```
mysql> SET GLOBAL sort_buffer_size=value;
mysql> SET @@global.sort_buffer_size=value;
```

要想设置一个SESSION变量的值，可使用下面的语法。

```
mysql> SET SESSION sort_buffer_size=value;
mysql> SET @@session.sort_buffer_size=value;
mysql> SET sort_buffer_size=value;
```

如果设置变量时不指定GLOBAL、SESSION或LOCAL，则默认使用SESSION。

要想检索一个GLOBAL变量的值，可使用下面的语法。

```
mysql> SELECT @@global.sort_buffer_size;
mysql> SHOW GLOBAL VARIABLES LIKE 'sort_buffer_size';
```

要想检索一个SESSION变量的值，可使用下面的语法。

```
mysql> SELECT @@sort_buffer_size;
mysql> SELECT @@session.sort_buffer_size;
mysql> SHOW VARIABLES LIKE 'sort_buffer_size';
```

当用SELECT@@var_name搜索一个变量时（也就是说，不指定GLOBAL、SESSION），MySQL会返回SESSION值（如果存在SESSION变量的话），否则返回GLOBAL值。

对于SHOW VARIABLES，如果不指定GLOBAL、SESSION的话，MySQL会返回SESSION值。

3.3.2 保留字

MySQL显式保留了表3-9（摘自官方文档）中的关键字。其中大多数关键字被标准SQL用作列名和/或表名（例如GROUP）。少数被保留了，因为MySQL需要它们。在生产环境下，常犯的一个错误是，使用了MySQL保留的关键字作表名、列名，这会导致部署、升级失败或留下隐患。

表3-9 MySQL保留的关键字

保留字	保留字	保留字
ADD	ALTER	ALTER
ANALYZE	AND	AS
ASC	ASENSITIVE	BEFORE
BETWEEN	BIGINT	BINARY
BLOB	BOTH	BY
CALL	CASCADE	CASE
CHANGEL	CHAR	CHARACTER
CHECK	COLLATE	COLUMN

(续)

保留字	保留字	保留字
CONDITION	CONNECTION	CONSTRAINT
CONTINUE	CONVERT	CREATE
CROSS	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR
DATABASE	DATABASES	DAY_HOUR
DAY_MICROSECOND	DAY_MINUTE	DAY_SECOND
DEC	DECIMAL	DECLARE
DEFAULT	DELAYED	DELETE
DESC	DESCRIBE	DETERMINISTIC
DISTINCT	DISTINCTROW	DIV
DOUBLE	DROP	DUAL
EACH	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	FALSE
FETCH	FLOAT	FLOAT4
FLOAT8	FOR	FORCE
FOREIGN	FROM	FULLTEXT
GOTO	GRANT	GROUP
HAVING	HIGH_PRIORITY	HOUR_MICROSECOND
HOUR_MINUTE	HOUR_SECOND	IF
IGNORE	IN	INDEX
INFILE	INNER	INOLT
INSENSITIVE	INSERT	INT
INT1	INT2	INT3
INT4	INT5	INTEGER
INTERVAL	INTO	IS
ITERATE	JOIN	KEY
KEYS	KILL	LABEL
LEADING	LEAVE	LEFT
LIKE	LIMIT	LINEAR
LINES	LOAD	LOCALTIME
LOCALTIMESTAMP	LOCK	LONG
LONGLOB	LONGTEXT	LOOP
LOW_PRIORITY	MATCH	MEDIUMLOB
MEDIUMINT	MEDIUMTEXT	MIDDLEINT
MINUTE_MICROSECOND	MINUTE_SECOND	MOD
MODIFIERS	NATURAL	NOT
NO_WRITE_TO_BINLOG	NULL	NUMERIC

保留字	保留字	保留字
ON	OPTIMIZE	OPTION
OPTIONALLY	OR	ORDER
OUT	OUTER	OUTFILE
PRECISION	PRIMARY	PROCEDURE
PURGE	RAID0	RANGE
READ	READS	REAL
REFERENCES	REGEXP	RELEASE
RENAME	REPEAT	REPLACE
REQUIRE	RESTRICT	RETURN
REVOKE	RIGHT	RLIKE
SCHEMA	SCHEMAS	SECOND_MICROSECOND
SELECT	SENSITIVE	SEPARATOR
SET	SHOW	SMALLINT
SPATIAL	SPECIFIC	SQL
SQLEXCEPTION	SQLSTATE	SQLWARNING
SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS	SQL_SMALL_RESULT
SSL	STARTING	STRAIGHT_JOIN
TABLE	TERMINATED	THEN
TINYBLOB	TINYINT	TINYTEXT
TO	TRAILING	TRIGGER
TRUE	UNDO	UNION
UNIQUE	UNLOCK	UNSIGNED
UPDATE	USAGE	USE
USING	UTC_DATE	UTC_TIME
UTC_TIMESTAMP	VALUES	VARBINARY
VARCHAR	VARCHARACTER	VARYING
WHEN	WHERE	WHILE
WITH	WRITE	XSOH
XOR	YEAR_MONTH	ZEROFILL

3.3.3 MySQL注释

MySQL服务器支持如下3种注释风格。

- 从“#”字符至行尾。

- 从“--”序列到行尾。请注意，“--”（双破折号）注释风格要求第2个破折号的后面至少要跟一个空格符（例如空格、tab、换行符等）。之所以要求使用空格，是为了防止出现非预期结果。比如，对于语句“UPDATE account SET credit=credit--1”，则是表示credit的值减去-1，这样的语法是合格的，而不会误认为“--1”是注释。



说明 建议不要使用“--”这样的方式，生产环境可能由于忘记在“--”后面加空格从而导致误操作。

- /*序列到后面的*/序列。结束序列不一定在同一行中，因此该语法允许注释跨过多行。

下面的例子显示了3种风格的注释。

```
mysql> SELECT 1+1;      # This comment continues to the end of line
mysql> SELECT 1+1;      -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

MySQL对标准SQL进行了扩展，如果使用了它们，将无法把代码移植到其他数据库的服务器上。可以用“/*...*/”注释掉这些扩展。如下例子中，MySQL服务器能够解析并执行注释中的代码，就像对待其他SQL语句一样，但其他数据库服务器将忽略

这些扩展。

```
SELECT /*! STRAIGHT_JOIN */ col_name FROM table1,table2 WHERE ...
```

如果在字符“!”后添加了版本号，那么仅当MySQL的版本等于或高于指定的版本号时才会执行注释中的语法，比如下面这条语句。

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

这就意味着，如果你的版本号为3.23.02或更高，那么MySQL服务器将使用TEMPORARY关键字。

3.3.4 数据类型

MySQL支持常用的数据类型：数值类型、日期/时间类型和字符串（字符）类型。

1. 数值类型

数值类型可分为两类：整型和实数。对于实数，MySQL支持确切精度的值（定点数）和近似精度的值（浮点数）。确切精度的数值类型有DECIMAL类型，近似精度的数值类型有单精度（FLOAT）或双精度（DOUBLE）两种类型。

(1) 整型

整型包括TINYINT、SMALLINT、MEDIUMINT、INT、BIGINT，表3-10展示了各种整型的空间占用及表示的数值范围。

表3-10 各种整型占用的字节数及数值范围

类型	字节	最小值 (带符号的/无符号的)	最大值 (带符号的/无符号的)
(续)			
TINYINT	1	-128 0	127 255
SMALLINT	2	-32768 0	32767 65535
MEDIUMINT	3	-8388608 0	8388607 16777215
INT	4	-2147483648 0	2147483647 4294967295
BIGINT	8	-9223372056854775808 0	9223372036854775807 18446744073709551615

在表3-10中，无符号（unsigned）属性可扩展一倍的最大值上限。



注意 MySQL整型可设置一个“width”属性，这点很容易让人混淆。实际上，这不是一个精度，只是告诉客户端工具显示多少个字符而已，如INT (11)：11表示的不是数值范围，只是显示宽度（告诉交互式工具显示宽度，如MySQL客户端）。

由于MySQL的内部类型只支持到秒级别的精度，因此可以用BIGINT来存储精度到毫秒的时间戳。

开发数据库应用的时候，需要注意的是，应保留足够的范围来满足未来的数据增长需要，对于超过数值范围的插入/修改数

据，MySQL将报错失败，例如对于SMALLINT类型，值的范围为-32768~32767，那么在自增ID列（后文会详述）的值已经到了32767后，还继续插入记录，就会报错“Duplicate entry '32767' for key 'PRIMARY'”，而且更新的值超过最高阈值时也会报错，如“Out of range value for column 'id' at row 1”。可以设置unsigned属性来扩展数据范围。

(2) DECIMAL和NUMERIC类型（定点数）

定点数也就是DECIMAL型，指的是数据的小数点的位置是固定不变的。也就是说，小数点后面的位数是固定的。

DECIMAL和NUMERIC在MySQL中被视为相同的类型。它们用于保存必须为确切精度的值，例如货币数据。当声明该类型的列时，可以（并且通常要）指定精度和标度；比如，在DECIMAL(M,D)中，M是精度，表示数据的总长度，也就是十进制数字的位数，不包括小数点；D是标度，表示小数点后面的数字位数。在MySQL 5.1中，M的范围是1~65，D的范围是0~30且不能大于M。例如下面这条语句，5是精度，2是标度。

```
salary DECIMAL(5,2)
```

对于数值123456789.12345，可以这样定义，M=14，D=5。

在MySQL 5.1中以二进制格式保存DECIMAL和NUMERIC的值。如果值太大超出了BIGINT的范围，也可以用DECIMAL存储整型。

定点数表达法的缺点在于其形式过于僵硬，固定的小数点位置决定了固定位数的整数部分和小数部分，不利于同时表达特别大的数或特别小的数。

(3) FLOAT和DOUBLE类型（浮点数）

浮点数（floating-point number）是属于有理数中某个特定子集的数的表示法，在计算机中用于近似地表示任意某个实数。具体来说，这个实数是由一个整数或定点数（即尾数）乘以某个基数（计算机中通常是2）的整数次幂（指数）得到的，这种表示方法类似于基数为10的科学记数法。比如123.45可以用十进制科学计数法表达为“ 1.2345×10^2 ”，其中1.2345为尾数，10为基数，2为指数。浮点数利用指数达到了浮动小数点的效果，从而可以灵活地表达更大范围的实数。

在MySQL中，对于浮点列类型，单精度值（FLOAT）使用4个字节，双精度值（DOUBLE）使用8个字节。浮点数可以比整型、定点数表示更大的数值范围。

为了保证最大可能的可移植性，对于使用近似数值存储的代码，应使用FLOAT或DOUBLE来表示，不规定精度或位数。由于浮点数存在误差问题，如果用到浮点数，要特别注意误差的问题，并尽量避免做浮点数比较。

MySQL允许使用非标准语法：FLOAT(M,D)或DOUBLE(M,D)。这里，“(M,D)”表示该值一共显示了M位整数，其中D位整数位于小数点后面。例如，定义为FLOAT(7,4)的一个列可以显示为-999.9999。MySQL保存值时会进行四舍五入，因此如果在FLOAT(7,4)列内插入999.00009，近似结果是999.0001。

浮点型（FLOAT/DOUBLE）对比定点类型（DECIMAL）使用的空间更少，所以为了减少存储空间，应尽量不要使用DECIMAL，除非是在保存确切精度的值时，比如货币数据。

2.日期/时间类型

表示时间值的日期和时间类型有DATETIME、DATE、TIMESTAMP、TIME和YEAR等。每个时间类型都有一个有效值范围，TIMESTAMP类型有其特有的自动更新特性。

如果试图插入一个不合法的日期，MySQL将给出警告或错误。可以使用`ALLOW_INVALID_DATES` SQL模式让MySQL接受某些日期，例如'1999-11-31'。在这种模式下，MySQL只验证月的范围是否为从0到12，日的范围是否为从0到31。有时应用程序希望保存一个特定的不合法日期，以便将来进行处理，这时可以利用这个模式。但更常见的处理方式是设置一个不可能的特定的合法日期值，如'9999-01-01 00:00:00'。

如果没有使用`NO_ZERO_DATE`的SQL模式，默认情况下，MySQL只允许在DATE或DATETIME列保存月和日是零的日期。这在应用程序中需要保存一个你不知道确切日期的生日时非常有用，在这种情况下，只需要将日期保存为'1999-00-00'或'1999-01-00'即可。

如果不使用`NO_ZERO_DATE` SQL模式，MySQL还允许将'0000-00-00'保存为“伪日期”。这在某些情况下比使用NULL值更方便，并且数据和索引占用的空间更小。

MySQL以标准输出格式检索给定日期或时间类型的值，但它会尽力解释你指定的各种输入值格式。尽管MySQL在尝试解释几种格式的值时，日期总是以“年–月–日”的顺序（例如，'98-09-04'）来处理的，而不是以“月–日–年”或“日–月–年”的顺序（例如，'09-04-98'、'04-09-98'）。

包含两位年值的日期会令人产生困惑，因为不知道世纪。MySQL使用以下规则解释两位年值的日期。

·70~99范围的年值均转换为1970~1999。

·00~69范围的年值均转换为2000~2069。

(1) DATETIME、DATE和TIMESTAMP类型

当需要同时包含日期和时间信息的值时，建议使用DATETIME（日期时间组合）类型。MySQL以'YYYY-MM-DD HH:MM:SS'的格式检索和显示DATETIME值，但允许使用字符串或数字为DATETIME列分配值。支持的范围为'1000-01-01 00:00:00'到'9999-12-31 23:59:59'。DATETIME类型占8个字节。

当只需要日期值而不需要时间部分时，建议使用DATE（日期）类型。MySQL用'YYYY-MM-DD'格式检索和显示DATE值，但允许使用字符串或数字为DATE列分配值。支持的范围是'1000-01-01'到'9999-12-31'。DATE类型占3个字节。

TIMESTAMP（时间戳）列用于在进行INSERT或UPDATE操作时记录日期和时间。TIMESTAMP列的显示格式与DATETIME列相同。换句话说，显示宽度固定在19个字符，并且格式为'YYYY-MM-DD HH:MM:SS'。TIMESTAMP的范围是从'1970-01-01 00:00:01'UTC到'2038-01-09 03:14:07'UTC。TIMESTAMP类型占4个字节。

TIMESTAMP的值以UTC格式进行保存，存储时会对当前的时区进行转换，检索时再转换回当前的时区。当前时区对应的是`time_zone`系统变量。

控制TIMESTAMP列的初始化和更新的规则如下。

若将TIMESTAMP类型字段定义为`default current_timestamp`，那么插入一条记录时，该TIMESTAMP字段自动被赋值为当前时间。

若将TIMESTAMP类型字段定义为`on update current_timestamp`，那么修改一条记录时，该TIMESTAMP字段自动被修改为当前时间。

可以将这些类型联合使用，如`default current_timestamp on update current_timestamp`。

可以给TIMESTAMP字段指定一个默认值，也可以在SQL语句中指定TIMESTAMP字段的值。

可以使用任何常见格式指定DATETIME、DATE和TIMESTAMP的值。

对于'YYYY-MM-DD HH:MM:SS'或'YY-MM-DD HH:MM:SS'格式的字符串，允许“不严格”语法：任何标点符号都可以用作日期部分或时间部分之间的间隔符。例如，'98-12-3111:30:45'、'98.12.3111+30+45'、'98/12/3111*30*45'和'98@12@3111^30^45'是等价的。

对于'YYYY-MM-DD'或'YY-MM-DD'格式的字符串，也允许使用“不严格的”语法。例如，'98-12-31'、'98.12.31'、'98/12/31'和'98@12@31'是等价的。

(2) TIME (时间) 类型

该时间类型的范围是'-838:59:59'到'838:59:59'。MySQL以'HH:MM:SS'格式检索和显示TIME值（或者对于大的小时值采用'HHH:MM:SS'格式），但允许使用字符串或数字为TIME列分配值。TIME类型占3个字节。

(3) YEAR (两位或四位格式的年) 类型

YEAR类型表示两位或四位格式的年。MySQL以YYYY格式显示YEAR值，但允许使用字符串或数字为YEAR列分配值。

默认是四位格式，在四位格式中，允许的值是1901~2155和0000。

在两位格式中，如果是两位字符串，那么范围为'00'~'99'。'00'~'69'和'70'~'99'范围的值被分别转换为2000~2069和1970~1999范围的YEAR值。

如果是两位整数，范围为1~99。1~69和70~99范围的值被分别转换为2001~2069和1970~1999范围的YEAR值。请注意，两位整数范围与两位字符串范围稍有不同，因为你不能直接将零指定为数字并将它解释为2000。你必须将它指定为一个字符串'0'或'00'，或者它被解释为0000。

YEAR类型占1个字节。

3.字符串类型

字符串类型指CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM和SET。

(1) CHAR和VARCHAR类型

CHAR与VARCHAR类型类似，但它们保存和检索数据的方式不同。

CHAR和VARCHAR类型声明的长度表示你想要保存的最大字符数。例如，CHAR(30)可以占用30个字符。注意，在CHAR(M)、VARCHAR(M)声明里，M是字符个数而不是字节。

如果分配给CHAR或VARCHAR列的值超过了列的最大长度，则对值进行裁剪以使其长度适合。如果被裁剪掉的字符不是空格，则会产生一条警告。

CHAR是固定长度的字符串，它的长度固定为创建表时声明的长度。长度范围为0到255个字符。当保存CHAR值时，在它们的右边填充空格以达到指定的长度。当检索到CHAR值时，尾部的空格会被删除掉，这是MySQL服务器级别控制的，和存储引擎无关。CHAR类型适合存储大部分值的长度都差不多的数据，例如MD5值。

VARCHAR列中的值为可变长度的字符串。长度可以指定为0到65535之间的值（VARCHAR的最大有效长度由最大记录长

度和使用的字符集确定。整体最大长度是65532字节）。相对于固定长度的字符串，它需要更少的存储空间。在保存VARCHAR的值时，只保存需要的字符数，然后用1~2个字节来存储值的长度，所以如果是很短的值（如仅一个字符），那么耗费的存储空间比CHAR还会多些，所以，如果想存储很短的类型，使用CHAR会更合适。VARCHAR可选的一种场景是最长记录的长度值比平均长度的值大得多。

保存VARCHAR的值时不会进行填充。当值保存和检索时尾部的空格仍会保留，这一点符合标准SQL。

实际上，各存储引擎存取VARCHAR和CHAR的方法不尽相同。比如，内存引擎使用固定长度的行，会在内存中分配最大可能空间给VARCHAR类型，所以CHAR(5)和VARCHAR(200)在存储“hello”字符串时占据的空间大小是一样的，但VARCHAR(200)会耗费更大的内存空间。

（2）BINARY和VARBINARY类型

BINARY和VARBINARY类似于CHAR和VARCHAR，不同的是，它们包含的是二进制字符串而不是非二进制字符串。也就是说，它们包含的是字节字符串而不是字符字符串，它们的长度是字节长度而不是字符长度。这说明它们没有字符集，并且排序和比较也是基于字节的二进制值进行的。

相对来说，二进制字符串的比较比字符字符串的比较更为简单有效。

对于“随机”字符串，如MD5()、SHA1()或UUID()生成的值会导致数据非常分散，没有明显的热点数据，还可能导致数据库缓存不能很好的工作。因此建议把MD5()、UUID()之类的值再散列下，生成整型值。

（3）BLOB和TEXT类型

BLOB是一个二进制大对象，可以容纳可变数量的数据。BLOB类型共有4种：TINYBLOB、BLOB、MEDIUMBLOB和LONGBLOB。BLOB是SMALLBLOB的同义词。

TEXT类型也有4种：TINYTEXT、TEXT、MEDIUMTEXT和LONGTEXT。它们分别对应上面的4种BLOB类型，有相同的最大长度和存储需求。TEXT是SMALLTEXT的同义词。

BLOB用于存储二进制字符串（字节字符串），而TEXT列则被视为非二进制字符串（字符字符串）的存储方式，它是有字符集和排序规则的，这两种类型都用于存储大量数据，具体的存储方式按存储引擎各有不同。

在大多数情况下，可以将BLOB列视为能够存储足够大数据的VARBINARY列。同样，也可以将TEXT列视为VARCHAR列。但是，BLOB和TEXT在以下几个方面不同于VARBINARY和VARCHAR。

- 保存或检索BLOB和TEXT列的值时不用删除尾部的空格。
- 对于BLOB和TEXT列的索引，必须指定索引前缀的长度。
- BLOB和TEXT列不能有默认值。
- 排序时只使用该列的前max_sort_length个字节。max_sort_length的默认值是1024。

BLOB或TEXT对象的最大长度由其类型来确定，但在客户端和服务器之间实际可以传递的最大数据量是由可用内存数量和通信缓存区的大小来确定的。可以通过更改max_allowed_packet变量的值更改消息缓存区的大小，但必须同时修改服务器和客户端的程序。

使用BLOB、TEXT等大字段可能会导致严重的性能问题，比如导致产生磁盘临时表。

MySQL的临时表分为“内存临时表”和“磁盘临时表”，其中内存临时表使用MySQL的MEMORY存储引擎，磁盘临时表使用MySQL的MyISAM存储引擎。由于MEMORY存储引擎不支持BLOB和TEXT类型，所以如果有查询使用了BLOB或TEXT列且需要隐式使用临时表（MEMORY存储引擎）来进行排序，那么将不得不使用磁盘临时表，磁盘比内存慢得多，这会导致很严重的性能问题。

(4) ENUM类型

ENUM（枚举）类型是一个字符串对象，其值通常选自一个允许值列表，该列表是在创建表时被定义的。

对于ENUM类型，需要慎重使用，如果候选值的集合可能发生改变，那么使用它就不见得是一个好主意。对于一些属性有固定数量的候选值的场景，可以使用其他更通用的方案，这样也能更方便地迁移到其他数据库，如用TINYINT类型代替ENUM类型，可以靠应用程序去维护字符串值和TINYINT的映射关系，或者增加一个表来存储映射关系，或者就直接存储更“自然”的字符串值。在现实世界中，空间大小一般已经不再是一个问题，自然、直观往往是更值得考虑的因素。

这里省略了对SET类型的介绍，感兴趣的读者可自行查阅相关图书。

4.数据类型存储需求

常用数值类型的存储需求见表3-11。

表3-11 常用数值类型的存储需求

数据类型	存储需求
TINYINT	1个字节
SMALLINT	2个字节
MEDIUMINT	3个字节
INT、INTEGER	4个字节
BIGINT	8个字节
FLOAT(p)	如果 $0 \leq p \leq 24$ ，则为4个字节；如果 $25 \leq p \leq 31$ ，则为8个字节
FLOAT	4字节

日期和时间类型的存储需求见表3-12。

表3-12 日期和时间类型的存储需求

列类型	存储需求	列类型	存储需求	列类型	存储需求
DATE	3字节	DATETIME	8字节	YEAR	1字节
TIME	3字节	TIMESTAMP	4字节		

字符串类型的存储需求见表3-13，其中的“L”代表字符串的字节长度。

表3-13 字符串类型的存储需求

列类型	存储需求
CHAR(M)	$M \times w$ 字节， $0 \leq M \leq 255$ ， w 是字符串字符的最大长度， M 是字符的个数
BINARY(M)	M 字节， $0 \leq M \leq 255$ 。这里 M 指的是字节
VARCHAR(M)、VARBINARY(M)	如果列值长度为 $0 \sim 255$ 个字节，那么需要 $L + 1$ 字节。如果列值长度超过 255 个字节，那么需要 $L + 2$ 字节

(续)

列类型	存储需求
TINYBLOB、TINYTEXT	$L + 1$ 字节， $L < 2^8$
BLOB、TEXT	$L + 2$ 字节， $L < 2^{16}$
MEDIUMBLOB、MEDIUMTEXT	$L + 3$ 字节， $L < 2^{24}$
LONGBLOB、LONGTEXT	$L + 4$ 字节， $L < 2^{32}$
ENUM('value1','value2',...)	1或2个字节，取决于枚举值的个数（最多 65 535 个值）
SET('value1','value2',...)	1, 2, 3, 4或8个字节，取决于 set 成员的数目（最多 64 个成员）

要想计算用于保存具体CHAR、VARCHAR或TEXT列值的字节数，需要考虑该列使用的字符集。例如utf8字符集，存储汉字是3个字节，存储英文字符是1个字节。

以上VARCHAR、VARBINARY、BLOB和TEXT类型都是可变长度的类型，它们的存储需求取决于如下3个因素。

- 列值的实际长度。
- 列的最大可能长度，如行长度有65536个字节的限制。
- 字符集。

例如，一个VARCHAR(255)列可以容纳一个最大长度为255个字符的字符串。如果该列使用latin1字符集（每个字符占一个字节），那么所需的实际存储为字符串字节的长度(L)，再加上一个字节以记录字符串的长度。对于字符串'ABCD'，L等于4，存储需求是5个字节。如果该列使用UCS2双字节字符集，那么存储要求为10个字节：'ABCD'的长度为8个字节，再需要2个字节来存储长度，因为它的最大长度大于255个字节（此时VARCHAR(255)最多为510个字节）。

可以存储在VARCHAR或VARBINARY列的字节还受到最大行长（65535字节）的限制。很显然，对于VARCHAR列，如果存储多字节字符，实际能够存储的字符会更少。例如，utf8字符集每个字符最多三个字节，所以使用utf8字符集的VARCHAR列可以被声明为最多21844个字符。

5.选择合适的数据类型

MySQL支持许多数据类型，选择合适的数据类型可以获得更好的性能，从而更节省空间。

以下是一些指导原则。

(1) 各表使用一致的数据类型

字段在每个表中都应该使用一样的数据类型、长度，因为以后可能需要进行JOIN（连接）操作，这样做是为了避免无谓的转换或可能出现不期望的结果。我们不仅要考虑数据类型是如何存储的，也要清楚数据类型是如何计算和比较的。

(2) 小往往更好

选择更短的数据类型。更短的类型意味着更少的磁盘空间、更少的内存空间、更少的CPU处理时间。例如，如果列值的范围为从1~99999，若使用整数，则MEDIUMINT UNSIGNED是比较好的数据类型。在所有可以表示该列值的类型中，该类型使用的存储最少。

(3) 简单类型更好

简单的数据类型能够进行更快的处理。例如，整型值比字符类型运算得更快，因为字符的字符集和排序规则使字符的比较运算变得更为复杂。生产环境中经常会看到用字符或整型来存储时间，为了使数据更友好、自然，建议还是使用MySQL内建的类型来存储日志时间会更好。使用无符号整型来存储IP地址（IP本质上是一个无符号的整型，点分的形式只是为了方便我们阅读）也是常用的好办法，可用INET_ATON()和INET_NTOA执行转换。

(4) 尽可能避免NULL值

应尽量显式定义“not NULL”，如果查询涉及的是NULL值的字段，MySQL会很难去优化查询。可使用0、空字符串或特殊的值来代替NULL存储。当然，也不要去刻意追求“not NULL”，因为更改NULL字段为“not NULL”，对性能的提升可能没什么太大的作用，让设计更自然、更具可理解性应该更值得看重。熟悉Oracle数据库的读者需要留意，MySQL会索引NULL值，而Oracle

则不会。

3.3.5 函数

以下介绍常用的函数和操作符。

1. 数值函数

(1) 算数操作符

可使用常见的算数操作符。例如‘+’、‘-’、‘*’、‘/’、DIV（整除）。

(2) 数学函数

·**ABS(X)**: X的绝对值。

·**CEIL(X)**: 返回不小于X的最小整数值。

·**FLOOR(X)**: 返回不大于X的最大整数值。

·**CRC32(X)**: 计算循环冗余码校验值并返回一个32比特无符号值。

·**RAND()、RAND(N)**: 返回一个随机浮点值v，范围在0到1之间（即其范围为 $0 \leq v \leq 1.0$ ）。若已指定一个整数参数N，则它被用作种子值，用来产生重复序列。

注意不要使用此函数做随机排序，如下的语句形式效率会很差，仅适合很小的表。

```
SELECT * FROM table_name ORDER BY RAND() LIMIT 1;
```

·**SIGN(X)**: 返回X的符号。

·**TRUNCATE(X,D)**: 返回被舍去至小数点后D位的数字X。若D的值为0，则结果不带有小数点或不带有小数部分。

·**ROUND(X)、ROUND(X,D)**: 返回参数X，其值接近于最近似的整数。在有两个参数的情况下，返回X，其值保留到小数点后D位，而第D位的保留方式为四舍五入。若要保留X值到小数点左边的D位，可将D设为负值，例如，**ROUND (123.45, -1)**的结果是120，**ROUND (167.8, -2)**的结果是200。

2. 字符类型处理函数

·**CHAR_LENGTH(str)**: 返回值为字符串str的长度，长度的单位为字符。一个多字节字符算作一个单字符。对于一个包含5个二字节的字符集，**LENGTH()**的返回值为10，而**CHAR_LENGTH()**的返回值为5。

·**LENGTH(str)**: 返回值为字符串str的长度，单位为字节。

·**CONCAT(str1,str2,...)**: 返回结果为连接参数产生的字符串。如下查询将拼接'My'、'S'、'QL'3个字符串。

```
mysql> SELECT CONCAT('My', 'S', 'QL');
-> 'MySQL'
```

·**LEFT(str,len)**: 从字符串str开始，返回最左len个字符。

·**RIGHT(str,len)**: 从字符串str开始，返回最右len个字符。

·**SUBSTRING(str,pos)、SUBSTRING(str,pos,len)**: 不带有len参数的格式是从字符串str返回一个子字符串，起始于位置pos。带有len参数的格式是从字符串str返回一个长度同len字符相同的子字符串，起始于位置pos。

如下查询将返回字符串Quadratically第5个字符之后的所有字符。

```
mysql> SELECT SUBSTRING('Quadratically',5);
-> 'ratically'
```

如下查询将返回字符串Quadratically第5个字符之后的6个字符。

```
mysql> SELECT SUBSTRING('Quadratically',5,6);
-> 'ratica'
```

·**LOWER(str)**: 返回字符串str转化为小写字母的字符。

·**UPPER(str)**: 返回字符串str转化为大写字母的字符。

3.日期和时间函数

·**NOW()**: 返回当前日期和时间的值，其格式为'YYYY-MM-DD HH:MM:SS'或YYYYYMMDDHHMMSS。

·**CURTIME()**: 将当前时间以'HH:MM:SS'或HHMMSS的格式返回。

·**CURDATE()**: 将当前日期按照'YYYY-MM-DD'或YYYYYMMDD格式的值返回。

·**DATEDIFF(expr1,expr2)**: 是返回开始日期expr1与结束日期expr2之间相差的天数，计算中只用到这些值的日期部分。返回值为正数或负数。

·**DATE_ADD(date,INTERVAL expr type)、DATE_SUB(date,INTERVAL expr type)**: 这些函数执行日期运算。date是一个DATETIME或DATE值，用来指定起始时间。expr是一个表达式，用来指定从起始日期添加或减去的时间间隔值。type为关键词，它指示了表达式被解释的方式。type常用的值有SECOND、MINUTE、HOUR、DAY、WEEK、MONTH、YEAR。示例代码如下所示。

```
mysql> SELECT DATE_ADD('1997-12-31 23:59:59', INTERVAL 1 SECOND);
-> '1998-01-01 00:00:00'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59', INTERVAL 1 DAY);
-> '1998-01-01 23:59:59'
```

·**DATE_FORMAT(date,format)**: 下面的代码会根据format字符串安排date值的格式。常用的日期格式'YYYY-MM-DD HH:MM:SS'，对应的format为'%Y-%m-%d %H:%i:%S'，示例代码如下所示。

```
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
-> '22:23:00'
```

·**STR_TO_DATE(str,format)**: 是DATE_FORMAT()函数的倒转。它将获取一个字符串str和一个格式字符串format。

若格式字符串包含日期和时间部分，则STR_TO_DATE()返回一个DATETIME值，若该字符串只包含日期部分或只包含时间部分，则返回一个DATE或TIME值。示例代码如下所示。

```
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
2004-04-31
```

3.3.6 操作符及优先级

运算符的优先级决定了不同的运算符在表达式中计算的先后顺序。一般情况下，级别高的运算符先进行计算，如果级别相同，MySQL则会按照表达式的顺序从左到右依次计算。以下是按照从低到高的优先级列出的各种运算操作符。

·:=

·||、OR、XOR

·&&、AND

·NOT

·BETWEEN、CASE、WHEN、THEN、ELSE

·=、<=>、>=、>、<=、<、<>、!=、IS、LIKE、REGEXP、IN

·|

·&

·<<、>>

·-、+

·*、/ (DIV) 、% (MOD)

·^ (按位异或)

·- (负号) 、~ (按位取反)

·!

如果不能确定优先级，可以使用圆括号()来改变优先级，并且这样会使计算过程更加清晰。比如，如下的查询，我们会先计算最里层括号里面的表达式(2+3)，然后计算外层的表达式。这点类似于我们学过的算术运算。

```
select 1*(3-2)*(3+3+3*(2+3));
```

3.3.7 MySQL示例employees数据库

MySQL提供了一个练习用的示范数据库employees。可以从Employees DB on Launchpad (<https://launchpad.net/test-db/> 中下载)，在页面右侧选择“Latest version is 1.0.6”，建议下载“employees_db-full-1.0.6”。

employees示例数据库一共有6张表，约400万条记录，包含160MB的数据。

首先是安装数据库，安装命令如下。

```
cd tmp  
wget  
https://launchpad.net/test-db/employees-db-1/1.0.6/+download/employees_db-full-1.0.6.tar.bz2  
tar jxf employees_db-full-1.0.6.tar.bz2  
cd employees_db
```

默认导入数据是InnoDB引擎，如果需要指定其他引擎，可以修改employees.sql文件，取消注释相应的引擎，命令如下。

```
set storage_engine = InnoDB;  
-- set storage_engine = MyISAM;  
-- set storage_engine = Falcon;  
-- set storage_engine = PBXT;  
-- set storage_engine = Maria;
```

使用MySQL命令将数据导入到实例中。

```
mysql -t < employees.sql
```

通过以下命令验证范例数据导入是否正确。

```
time mysql -t < test_employees_sha.sql
```

实体关系图3-3描述了employees示例数据库各表的结构和它们之间的关系。

图3-3中的各表通过一些字段相互关联，如dept_emp表中存储了部门职员的信息，通过dept_emp.emp_no可以到employees表中去查询职员的记录。各表之间的关系是通过连线和特殊符号标明的，钥匙标记表示这是主键，关系中属于“多”的这一边用一个类似鸟爪的图形来表示，如dept_emp表，主键是联合主键（emp_no,dept_no），employees和dept_emp表就是一对多的关系，由于职员可能在不同时期属于不同的部门，那么employees表中一条职员的记录可能存在dept_emp表中存在多条对应记录。读者可自行下载此数据库，验证各表的数据和彼此之间的联系。

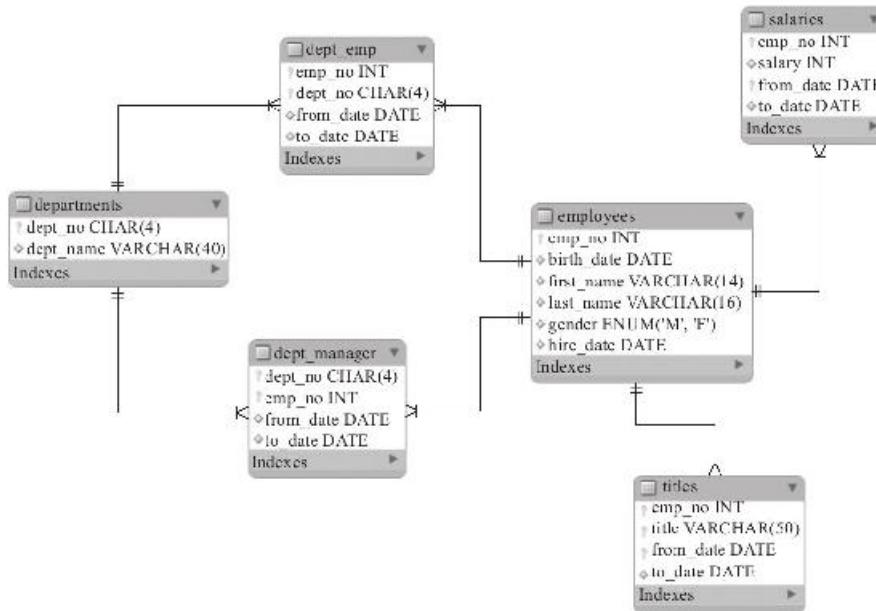


图3-3 employees示例数据库实体关系图

3.3.8 SQL语法

结构化查询语言（Structured Query Language, SQL）是一种高级编程语言，是数据库中的标准数据查询语言，这种语言是描述性的，很容易上手，你不需要了解数据是如何存储的也能编写出语句查询和修改数据，这项技能并非IT人士的专有领域，

其他非计算机行业的人，虽然不会编程，但也可以根据自己的业务需求，用SQL在公司的数据平台上查询数据。所以，我们设计的库表，如果有其他业务部门要使用，而且是通过SQL的方式进行查询，那么表名、列名就需要考虑下自然性和易用性。

美国国家标准学会（ANSI）对SQL进行规范后，将其作为关系式数据库管理系统的标准语言，而后在国际标准组织的支持下成为了国际标准。不过各种通行的数据库系统在其实践过程中都对SQL规范作了某些改编和扩充。所以，实际上不同数据库系统之间的SQL并不能完全相互通用。一般情况下，扩展语法后功能虽有所增强，但可能会导致移植性变差，而且对于整个系统的吞吐率、性能可能也不会有明显的改善。因此本书不会对MySQL的扩展语法进行详细介绍。下面使用自带的MySQL命令行工具来演示示例。

1.SQL常见操作

使用如下命令可查看MySQL支持的选项。

```
shell> mysql -  
help
```

使用如下命令可连接MySQL Server。

```
shell> mysql -h host -P port -u user -  
p  
Enter password: *****
```

连接登录成功后，可以按Ctrl+D退出，或者输入QUIT退出，命令如下。

```
mysql> QUIT  
Bye
```

下面来运行一个简单的查询，通过以下语句可查询当前MySQL Server的版本和当前日期。

```
mysql> SELECT VERSION(), CURRENT_DATE;  
+-----+-----+  
| VERSION() | CURRENT_DATE |  
+-----+-----+  
| 5.6.15 | 2013-12-29 |  
+-----+-----+  
1 row in set (0.00 sec)
```

使用如下命令可创建数据库employees。

```
mysql> CREATE DATABASE employees;
```

显示数据库，切换到test数据库时可使用如下命令。

```
mysql> SHOW DATABASES;  
mysql> USE test
```

使用如下命令可显示当前的数据库。

```
SELECT DATABASE();
```

使用如下命令可显示数据库下的表。

```
mysql> SHOW TABLES
```

2.数据定义语句（DDL）

以下介绍常用的DDL语句。

(1) 创建和删除表

可使用CREATE TABLE语句创建表。

```
CREATE TABLE employees_2 (
    emp_no int(11) NOT NULL,
    birth_date date NOT NULL,
    first_name varchar(14) NOT NULL,
    last_name varchar(16) NOT NULL,
    gender enum('M','F') NOT NULL,
    hire_date date NOT NULL,
    primary key (emp_no)
) engine=innodb default charset=latin1
```

可使用DESC语句验证创建的表结构。

```
mysql> DESC employees_2
```

使用DROP TABLE语句删除表。

```
DROP TABLE employees_2;
```

(2) 使用ALTER TABLE语句修改表结构

首先创建表t1。

```
mysql> CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

把表t1重新命名为t2。

```
mysql> ALTER TABLE t1 RENAME t2;
```

把列a从INTEGER类型更改为TINYINT NOT NULL（名称保持不变），并把列b从CHAR(10)更改为CHAR(20)，同时把列b重新命名为列c。

```
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

添加一个新的TIMESTAMP列，名称为d。

```
mysql> ALTER TABLE t2 ADD d TIMESTAMP;
```

在列d和列a中添加索引。

```
mysql> ALTER TABLE t2 ADD INDEX (d), ADD INDEX (a);
```

删除列c。

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

添加一个新的AUTO_INCREMENT整数列，名称为c。

```
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,  
ADD PRIMARY KEY (c);
```

(3) 使用CREATE INDEX语句创建索引

在表lookup的列id上创建索引。

```
CREATE INDEX id_index ON lookup (id);
```

在customer表的name列上创建一个索引，索引使用name列的前10个字符。

```
CREATE INDEX part_of_name ON customer (name(10));
```

在tbl_name表的a、b、c列上创建一个复合索引。

```
CREATE INDEX idx_a_b_c ON tbl_name(a,b,c);
```

(4) 使用DROP INDEX语句删除索引

删除表tbl_name上的index_name索引时使用如下命令。

```
DROP INDEX index_name ON tbl_name;
```

(5) 修改字符集和排序规则

可使用如下命令更改排序字符集。

```
ALTER TABLE test.tt1 CHANGE v2 v2 VARCHAR(10) CHARACTER SET utf8 COLLATE utf8_general_ci;
```

可使用如下命令更改排序规则。

```
ALTER TABLE table_name CHANGE col_a col_a VARCHAR(50) CHARACTER SET latin1 COLLATE latin1_bin
```

3.数据操作语句（DML）

以下是一些查询语句常用的语法和示例。需要留意的是，我们日常所说的查询语句，不仅包括SELECT查询语句，也包括INSERT、UPDATE、DELETE等修改数据的语句。在创建表之后，就可以导入数据了，导入数据的方式有二：或采用LOAD DATA语句，或采用INSERT语句。

以下主要介绍INSERT、SELECT、UPDATE、DELETE语句，LOAD DATA语句在以后的章节中会有介绍。

(1) INSERT语句

语法如下所示。

```
INSERT INTO table_name (column1, column2....)  
VALUES (value1, value2...);
```

具体示例如下。

```
INSERT INTO employees.employees (emp_no, birth_date, first_name, last_name, gender, hire_date) VALUES ('111111', '1976-11-11', 'Gary', 'Chen', 'M', '1998-08-20');
```

MySQL支持用一条INSERT语句插入多条记录，这样可以加快数据的导入，比如下面的示例。

```
INSERT INTO employees.employees  
(emp_no, birth_date, first_name, last_name, gender, hire_date)  
VALUES  
('111112', '1977-11-11', 'hua', 'chen', 'M', '2000-02-02'),  
('111113', '1988-12-02', 'feng', 'yu', 'M', '2013-12-20'),  
('111114', '1993-02-01', 'yong', 'chen', 'M', '2010-10-01');
```

(2) 修改数据 (UPDATE)

语法如下所示。

```
UPDATE table_name SET column_name1 = value1, column_name2 = value2, column_name3 = value3 ...  
[WHERE conditions];
```

以下命令可将员工编号为10001的员工姓名修改为gary wang。

```
UPDATE employees set first_name='gary',last_name='wang' where emp_no=10001;
```

(3) 删除 (DELETE)

语法如下所示。

```
DELETE FROM table_name [WHERE conditions];
```

以下命令可删除员工编号为1000000的员工记录。

```
DELETE from employees WHERE emp_no = 1000000;
```

(4) SELECT语句

语法如下所示。

```
SELECT column_names FROM table_name [WHERE ...conditions];
```

可使用如下命令查询表employees的所有数据。

```
SELECT * FROM employees;
```

可使用如下命令查询表employees的emp_no、birth_date、first_name、last_name这几个特定列的数据。

```
SELECT emp_no,birth_date,first_name,last_name FROM employees;
```

查询employees表中出生日期晚于'1960-01-01'的员工，使用WHERE子句，加上比较操作符“>”即可，命令如下。

```
SELECT emp_no,birth_date,first_name,last_name FROM employees WHERE birth_date > '1960-01-01';
```

可使用如下命令查询employees表中出生日期早于'1960-01-01'的员工。

```
SELECT emp_no,birth_date,first_name,last_name FROM employees WHERE birth_date < '1960-01-01';
```

可使用如下命令查询employees表中first_name等于Divier的员工。

```
SELECT * FROM employees WHERE first_name='Divier';
```

SELECT查询是需要我们重点掌握的，下文将详细讨论SELECT查询。

(1) SQL模式匹配

SQL有两个通配符，“_”匹配任意单个字符，“%”匹配任意多个字符（包括0个字符）。

模式匹配默认是区分大小写的，它一般使用LIKE或NOT LIKE这些比较操作符，比如，要查询employees表中first_name列以字母D开始的员工记录，可使用如下命令。

```
SELECT * FROM employees WHERE first_name LIKE 'D%';
```

查询employees表中first_name列以Ang开头，一共5个字符，last_name以Con开头，一共5个字符的记录时可使用如下命令。

```
mysql> SELECT emp_no,first_name,last_name,birth_date FROM employees WHERE first_name LIKE 'Ang__' and last_name LIKE 'Con__';
+-----+-----+-----+-----+
| emp_no | first_name | last_name | birth_date |
+-----+-----+-----+-----+
| 485400 | Angel     | Conry    | 1963-12-22 |
| 492878 | Angus     | Conia   | 1956-02-14 |
+-----+-----+-----+-----+
2 rows in set (0.14 sec)
```

(2) 逻辑操作符与或非 (AND、OR、NOT)

可以用逻辑操作符组合成多个筛选条件，示例如下。

选择employees表first_name列等于Parto，而且last_name列等于Alpay的记录。

```
SELECT emp_no,birth_date,first_name,last_name,gender,hire_date FROM employees WHERE first_name='Parto' AND last_name='Alpay';
```

选择employees表中'1995-01-31'或'1996-11-21'入职的员工。

```
SELECT emp_no,birth_date,first_name,last_name,gender,hire_date FROM employees WHERE hire_date='1995-01-31' OR hire_date='1996-11-21';
```

选择employees表中last_name列不是以字母A开头的所有记录。

```
SELECT * FROM employees WHERE last_name NOT LIKE 'A%';
```

(3) 范围操作符IN和BETWEEN

选择employees表中分别在'1964-06-01'、'1964-06-02'和'1964-06-04'这3天出生的员工时可使用如下命令。

```
SELECT * FROM employees WHERE birth_date IN ('1964-06-01','1964-06-02','1964-06-04');
```

选择employees表中在'1964-06-01'至'1964-06-04'期间出生的员工时可使用如下命令。

```
SELECT * FROM employees WHERE birth_date BETWEEN '1964-06-01' AND '1964-06-04';
```

(4) 限制获取记录数（使用LIMIT子句）

只获取employees表中的5条记录（没顺序）时可使用如下命令。

```
SELECT * FROM employees LIMIT 5;
```

(5) 排序（ORDER BY）

查询按出生日期排序的最老的10名员工时可使用如下命令。

```
SELECT * FROM employees ORDER BY birth_date ASC LIMIT 10;
```

查询按出生日期排序第100至109名的员工时可使用如下命令。

```
SELECT * FROM employees ORDER BY birth_date ASC LIMIT 100,10;
```

(6) 数据计算

MySQL提供了一些计算函数，下面给出了计算日期的示例，后续章节会专门叙述此类常用的函数。

以下命令将计算employees表中员工的年龄，并且按first_name、last_name排序，返回记录数限制在10条。

```
SELECT
    emp_no,
    first_name,
    last_name,
    birth_date,
    curdate(),
    timestampdiff(year,
        birth_date,
        curdate()) as age
FROM
    employees
ORDER BY first_name , last_name
LIMIT 10;
```

(7) 使用DISTINCT获取不重复的唯一值

以下命令将查询employees雇员表里唯一的first_name值。

```
SELECT DISTINCT first_name FROM employees ;
```

如果以上语句没有关键字DISTINCT，那么返回的记录里first_name会有许多重复值。

(8) 聚集函数COUNT、MIN、MAX、AVG、SUM

查询表employees的总记录数时可使用如下命令。

```
SELECT COUNT(*) FROM employees;
```

查询表employees的最小员工号时可使用如下命令。

```
SELECT MIN(emp_no) FROM employees;
```

查询表employees的最大员工号时可使用如下命令。

```
SELECT MAX(emp_no) FROM employees;
```

查询雇员的平均薪水时可使用如下命令。

```
SELECT AVG(salary) FROM salaries WHERE to_date='9999-01-01' ;
```

salaries表存储了所有员工在不同时期的薪水，where to_date='9999-01-01'可用于筛选出员工目前的薪水。

查询雇员薪水总额时可使用如下命令。

```
SELECT SUM(salary) FROM salaries WHERE to_date='9999-01-01' ;
```

以下查询将统计'1986-06-26'和'1985-11-21'分别有多少人入职。

```
SELECT SUM(hire_date='1986-06-26') AS sum_1986_06_26,SUM(hire_date='1985-11-21') AS sum_1985_11_21 FROM employees;
```

查询语句里的hire_date='1986-06-26'是一个计算表达式，满足条件时为1，不满足条件时为0。sum对表达式返回的值进行求和，这样就实现了统计。当然这种写法不常见，也不推荐使用。

(9) 分组统计GROUP BY子句

一般将GROUP BY语句和聚集函数一起使用，从而实现分组统计。

查询employees表，按照first_name分组，并根据first_name出现的次数按降序排序。

```
SELECT first_name,COUNT(*) cnt FROM employees GROUP BY first_name ORDER BY cnt DESC;
```

也支持如下形式的对多个列同时进行聚集计算。

```
SELECT MAX(a),MAX(b),MAX(c) FROM table_name WHERE ... GROUP BY d ;
```

我们可以在GROUP BY语句后添加HAVING子句，并对聚集结果进行筛选。

以下命令将查询employees表，按照first_name分组，列出重复次数大于270的first_name，并按照first_name重复的次数按降序排序。

```
SELECT first_name,COUNT(*) cnt FROM employees GROUP BY first_name HAVING cnt > 270 ORDER BY cnt DESC ;
```



注意 SELECT之后的选择列表的每个列都要来自于GROUP BY的列，有些数据库有严格的约定，必须满足此条件，MySQL对此没有强制要求，但需要清楚一个事实，数据库并不能保证其他列共有一个值，其他列的值可能是不固定的，目前其他列的返回结果是分组结果中的第一条记录，是按物理存储顺序进行排序的，但这个排序并不可靠，MySQL也没有确保后续版本会这么定义。SQL_MODE添加ONLY_FULL_GROUP_BY可以避免这种错误。

(10) 并集操作(UNION和UNION ALL)

有时我们需要对两个结果集进行合并操作。UNION和UNION ALL都是将两个结果集合并为一个，但UNION比UNION ALL更快。UNION实际上是UNION DISTINCT，在进行表连接后会筛选掉重复的记录，所以在表连接后会对所产生的结果集进行排序运算，删除重复的记录再返回结果。而UNION ALL则是不管有没有重复记录，都直接返回合并后的记录。实际应用中，两个需要合并的结果集一般不会产生重复记录，所以建议在能够使用UNION ALL的情况下尽量使用UNION ALL，否则对于很大的结果集，可能会导致查询耗时很长。

UNION ALL的使用示例如下。

```
SELECT * FROM a
UNION ALL
SELECT * FROM b;
```

UNION的使用示例如下。

```
SELECT * FROM a
UNION
SELECT * FROM b;
```

(11) NULL值

NULL值的判断一般使用IS NULL或IS NOT NULL，不能使用以上的比较操作符=、<、>，因为NULL是一个特殊的值，表示这个值是未知的或没有定义的。

以下命令将查询employees表中first_name列以字母D开头的员工且last_name值不是NULL的记录。

```
SELECT * FROM employees WHERE first_name LIKE 'D%' AND last_name IS NOT NULL;
```

上述命令添加的条件“AND last_name IS NOT NULL”仅是为了演示，实际上，由于last_name列上有约束——必须是NOT NULL，所以“AND last_name IS NOT NULL”这个条件总是满足的。

对于GROUP BY子句，两个NULL值可以认为是相等的。

对于“ORDER BY...ASC”，NULL显示在前。对于“ORDER BY...DESC”，NULL值显示在后。

0或空字符串实际上都是有值的，所以在一个NOT NULL的列上插入0或空字符串是允许的。

4.JOIN（连接）

MySQL使用JOIN来连接多个表查询数据，主要使用的JOIN算法只有一种，那就是nested-loop join。

nested-loop join算法实现的机制很简单，就是从驱动表中选取数据作为循环基础数据，然后以这些数据作为查询条件到下一个表中进行查询，如此往复。这个实现机制类似于foreach函数的遍历。因此带来的问题就是连接的表越多，函数嵌套的层数就越多，算法复杂度呈指数级增长。

因此，设计查询要尽量减少连接的表的个数。

驱动表是指：在使用多表嵌套连接时，首先，全表扫描该驱动表，然后用驱动表返回的结果集逐行去匹配被驱动的表（可以利用索引），数据库基于成本可能会选择小表作为驱动表，而被驱动表使用索引进行连接。

JOIN语句的含义是把两张（或者多张）表的属性通过它们的值组合在一起，一般会遇到如下3种连接。

·等值连接 ([INNER]JOIN)

·左外连接 (LEFT JOIN)

·右外连接 (RIGHT JOIN)

示例用表见图3-4~图3-6所示。

emp_no	birth_date	first_name	last_name	gender	hire_date
10001	1953-09-02	gary	wang	M	1986-06-26
10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
10003	1959-12-03	Parto	Bamford	M	1986-08-28
10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12
10006	1953-04-20	Anneke	Preusig	F	1989-06-02
10007	1957-05-23	Tzvetan	Zielinski	F	1989-02-10

图3-4 职员表

在部门职员表中，如果是在职员工，那么to_date的值为'9999-01-01'。

dept_no	dept_name
d005	Development
d003	Human Resources
d004	Production
d008	Research
d007	Sales

图3-5 部门表

emp_no	dept_no	from_date	to_date
10001	d005	1986-06-26	9999-01-01
10002	d007	1996-08-03	9999-01-01
10003	d004	1995-12-03	9999-01-01
10004	d004	1986-12-01	9999-01-01
10005	d003	1989-09-12	9999-01-01
10006	d005	1990-08-05	9999-01-01
10007	d008	1989-02-10	9999-01-01

图3-6 部门职员表

(1) 内连接

内连接 (INNER JOIN) 是应用程序中普遍应用的“连接”操作，它一般都是默认的连接类型。内连接基于连接谓词将两张表(如A和B)的列组合在一起，从而产生新的结果表。

内连接可以被进一步分为等值连接、自然连接和交叉连接。较常用的是等值连接。

以下是等值连接的示例。

查询目前在职的所有员工的姓名及其所在的部门时可使用如下语句。

```
SELECT
    de.emp_no, first_name, last_name, dept_name
FROM
    dept_emp de
    INNER JOIN
```

```
employees e ON de.emp_no = e.emp_no
    INNER JOIN
departments d ON d.dept_no = de.dept_no
WHERE
    to_date = '9999-01-01';
```

输出结果如图3-7所示。

自然连接（**natural join**）是等值连接的进一步特例化。两表做自然连接时，两表中名称相同的所有列都将被比较，这是隐式的。自然连接得到的结果表中，两表中名称相同的列只出现一次。一般应该避免使用自然连接，因为我们无法指定连接列，且这种写法隐藏了我们的JOIN关系，如果以后数据模型发生了变化，可能会导致出现非预期的结果。

交叉连接（笛卡儿积） 把表视为行记录的集合，交叉连接即返回这两个集合的笛卡儿积。这其实等价于内连接的连接条件为“永真”，或者连接条件不存在的情况。

示例如下。

emp_no	first_name	last_name	dept_name
10038	Huan	Lortz	Customer Service
10049	Basil	Tramer	Customer Service
10060	Breannda	Billingsley	Customer Service
10088	Jungsoon	Syrzycki	Customer Service
10112	Yuichiro	Swick	Customer Service
10126	Kayoko	Valtorta	Customer Service
10128	Babette	Lamba	Customer Service

图3-7 在职员工的姓名和部门的等值连接

```
SELECT * FROM a JOIN b;
SELECT * FROM a,b;
```

SQL定义了两种不同的语法方式来表示“连接”。一种是“显式连接符号”，显式地使用关键字**JOIN**，另一种是“隐式连接符号”，它使用所谓的“隐式连接符号”。隐式连接符号把需要连接的表放到**SELECT**语句的**FROM**部分，并用逗号隔开。这样就构成了一个“交叉连接”，**WHERE**语句可能会放置一些过滤谓词（过滤条件）。那些过滤谓词在功能上等价于显式连接符号。

如上所述的例子中，查询目前在职的所有员工的姓名及所在部门时，可以写成如下的形式。

```
SELECT
    de.emp_no, first_name, last_name, dept_name
FROM
    dept_emp de,
    employees e,
    departments d
WHERE
    de.dept_no = d.dept_no
        and de.emp_no = e.emp_no
        and de.to_date = '9999-01-01';
```

它等价于：

```
SELECT
    de.emp_no, first_name, last_name, dept_name
FROM
    dept_emp de
        INNER JOIN
    employees e ON de.emp_no = e.emp_no
        INNER JOIN
    departments d ON d.dept_no = de.dept_no
WHERE
    to_date = '9999-01-01';
```

ON表达式是任何可以用于**WHERE**子句的条件表达式，一般来说，你应该只在**ON**表达式里指定如何**JOIN**表，而把筛选结果集的条件放到**WHERE**子句中。

(2) 外连接 (OUT JOIN)

并未要求连接的两表的每一条记录在对方的表中都必须有一条匹配的记录。连接表保留所有的记录，甚至这条记录没有匹配的记录也要保留。外连接可依据连接表保留左表、右表或全部表的行而进一步分为左外连接、右外连接和全外连接。全外连接一般没有什么意义，MySQL并不直接支持全外连接，但可以通过左右外连接的并集（UNION）来模拟实现。

1) 左外连接 (LEFT JOIN、LEFT OUTER JOIN)

左外连接也简称为左连接 (LEFT JOIN)，若A和B两表进行左外连接，那么结果表中将包含“左表”(即表A)的所有记录，即使那些记录在“右表”B中没有符合连接条件的匹配。这就意味着即使ON语句在表B中的匹配项是0条，连接操作也还是会返回一条记录，只不过这条记录中的来自于表B的每一列的值都为NULL。

之前的示例曾演示过插入一些记录（员工号111111、111112、111113、111114）到employees表中，但还没有指定新插入员工记录的部门。现在用如下语句联合查询下雇员表和部门-雇员表。

```
SELECT
    de.dept_no, first_name, last_name
FROM
    employees e
        LEFT JOIN
    dept_emp de ON e.emp_no = de.emp_no
WHERE
    e.emp_no IN (10001,10002,10003,111111 , 111112, 111113, 111114);
```

结果如图3-8所示，可以看到有些员工没有部门，dept_no（部门代码）显示为NULL。

2) 右外连接 (RIGHT JOIN、RIGHT OUT JOIN)

右外连接也简称右连接 (RIGHT JOIN)，它与左外连接完全类似，只不过是连接表的顺序相反而已。如果A表右连接B表，那么“右表”B中的每一行在连接表中至少会出现一次。如果B表的记录在“左表”A中未找到匹配行，则连接表中来源于A表中的列的值将设为NULL，示例如下。

```
SELECT *
FROM A RIGHT JOIN B
    ON A.id = B.id
```

实际上，显式的右外连接很少使用，因为它的可读性不佳，所以总是被改写成左连接。

dept_no	first_name	last_name
d005	gary	wang
d007	Bezalel	Simmel
d004	Parto	Bamford
NULL	Gary	Chen
NULL	hua	chen
NULL	feng	yu
NULL	yong	chen

图3-8 查询一些员工的部门

如果JOIN的层次比较多，则需要留意一下可读性，如果不能确定优先级，那么建议使用括号来明确优先级，以避免犯错误。例如，在以下的例子中，由于逗号的优先级比JOIN表达式低，因此可能会导致我们犯错误。

```
SELECT t1.id,t2.id,t3.id
```

```
FROM t1,t2  
LEFT JOIN t3 ON (t3.id=t1.id)  
WHERE t1.id=t2.id;
```

上述代码实际上是：

```
SELECT t1.id,t2.id,t3.id  
FROM t1,( t2 LEFT JOIN t3 ON (t3.id=t1.id) )  
WHERE t1.id=t2.id;
```

但这其实并不是我们的本意，应该写成如下的形式（用括号来提升优先级别）。

```
SELECT t1.id,t2.id,t3.id  
FROM (t1,t2)  
LEFT JOIN t3 ON (t3.id=t1.id)  
WHERE t1.id=t2.id;
```

使用JOIN命令操作表的时候，需要留意如果表按条件筛选的记录是不确定的，可能就会导致非预期的结果。下面以图3-9和图3-10所示的数据为例来进行说明。

id	code
1	a
2	b
3	c
4	d
5	e

图3-9 t1表

id	code	name
1	a	name1
2	b	name2
3	c	name3
4	d	name4
5	a	name5
6	a	name6

图3-10 t2表

对于如下的查询：

```
SELECT  
    t1.id as t1_id,  
    t1.code as t1_code,  
    t2.id as t2_id,  
    t2.code as t2_code,  
    t2.name  
FROM
```

```
t1
      JOIN
t2 ON t1.code = t2.code AND t2.code = 'b';
```

由于code='b'在两个表中都可以唯一确定一条记录，因此查询会返回合理的结果（如图3-11所示）。

t1_id	t1_code	t2_id	t2_code	name
2	b	2	b	name2

图3-11 返回正确的结果

而对于如下查询：

```
SELECT
    t1.id as t1_id,
    t1.code as t1_code,
    t2.id as t2_id,
    t2.code as t2_code,
    t2.name
FROM
    t1
        JOIN
    t2 ON t1.code = t2.code AND t2.code = 'a';
```

由于t2表code='a'会返回多个值。最终的查询结果返回了3条记录（如图3-12所示），因此可能不是我们所需要的结果。

t1_id	t1_code	t2_id	t2_code	name
1	a	1	a	name1
1	a	5	a	name5
1	a	6	a	name6

图3-12 返回错误的结果

5.子查询

子查询是指查询语句里的SELECT语句。比如下面这条语句。

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

在这个示例中，SELECT*FROM t1是外部查询（外部语句），SELECT column1 FROM t2是子查询。

我们可以说，这个子查询是嵌套在外部查询中的，子查询嵌套的层次不宜过多，否则性能可能会很差。

许多人认为子查询的可读性更好，子查询在现实中的应用也很广泛，但MySQL对于子查询的优化不佳，由于子查询一般可以改写成JOIN语句，因此一般建议使用JOIN的方式查询数据。下面来看看示例。

查询薪水大于150000的员工的姓名。

```
SELECT
    emp_no, first_name, last_name
FROM
    employees
WHERE
    emp_no IN (SELECT
                    emp_no
                FROM
                    salaries)
```

```
WHERE
    to_date = '9999-01-01'
        AND salary > 150000);
```

可将上述语句改写成JOIN的方式，查询语句如下。

```
SELECT
    employees.emp_no, first_name, last_name
FROM
    employees
        JOIN
    salaries ON salaries.emp_no = employees.emp_no
WHERE
    salaries.to_date = '9999-01-01'
        AND salaries.salary > 150000;
```



说明 以上的例子使用了通配符“*”，生产环境DBA一般会建议不要使用“select*”这样的方式查询数据，这里使用通配符主要是为了书写方便，避免分散注意力，从而关注更重要的部分。

3.4 PHP开发

3.4.1 概述

一般的流行语言，如PHP、C、Perl、Java都对MySQL提供了完善支持，这其中PHP是最常用的使用MySQL数据库的语言，互联网普遍使用的是LAMP/LNMP架构，这里的P可以理解为就是PHP，可以说PHP的应用范围相当广泛，尤其是在Web程序的开发上，比如，我们熟知的Facebook，就是PHP、MySQL的重度使用者。作为互联网开发者，我们有必要熟悉MySQL在各种语言环境下的使用，尤其是PHP。

以下简要介绍PHP与MySQL开发，PHP（全称为Hypertext Preprocessor，即超文本预处理器）是一种开源的通用计算机脚本语言，它是服务器端的解释语言，可以嵌入到HTML页面中，这些代码在每次页面访问时都将被执行。PHP代码将在Web服务器中被解释并且生成HTML或访问者看到的其他输出界面。

如果想要学习PHP+MySQL开发，那么首先需要搭建一个适合练习的开发环境。对于操作系统，建议使用Linux；Web服务器建议使用Apache或Nginx；数据库当然是MySQL了。网上有很多搭建LAMP（Linux、Apache、MySQL、PHP）环境的文档，大家可以搜索查看。

也有一些集成的自动安装包，如XAMPP，可以一键安装帮你部署好所有环境，但还是建议手动部署下环境，从而对Web服务器的配置文件、PHP的配置文件有一定的了解。

3.4.2 客户端访问过程

下面简单介绍下传统网站的访问数据流。在客户端（用户）与数据库服务器之间往往还会涉及Web服务器和负载均衡设备。作为一个开发者，需要清楚数据是如何在客户端和服务器之间进行传递的。下面简单说明下客户端访问数据库服务器中间经过的环节，这里以Windows PC浏览器为例进行说明，并解释一些软硬件基础概念。

1. 访问DNS服务

用户在浏览器的地址栏输入网址域名，浏览器会查询这个域名与IP的对应关系是否已经存在于本机的host文件中，如果没有，则会把请求发送给本机指定的域名系统（Domain Name System，DNS）服务器。

什么是域名系统服务器呢？

计算机世界是以IP地址来定位服务器或PC的，DNS这项服务的目的就是将域名翻译成IP，使用户可以更方便地访问互联网。DNS服务器有一定的层级，如果某个DNS服务器不知道如何翻译，就会问另外一个，再不知道，再问下一个，这样就会有一个递归的过程。幸运的是，DNS可以缓存查询结果，这样我们就不需要经历重复冗长的过程去查找一个域名映射的IP地址。

DNS系统中，常见的资源记录类型有如下两种。

- 主机记录（A记录）：用于名称解析的重要记录，它将特定的主机名映射到对应主机的IP地址上。
- 别名记录（CNAME记录）：用于将某个别名指向到某个A记录上，这样的好处是修改IP的时候改A记录就可以了，对于有大量子域名的网站可以简化操作、统一维护域名指向。

DNS查询有两种方式：递归和迭代。DNS客户端设置使用的DNS服务器一般都是递归服务器，它负责全权处理客户端的

DNS查询请求，直到返回最终结果。而DNS服务器之间一般采用迭代查询的方式。

下面以查询zh.wikipedia.org为例。

客户端发送查询报文“query zh.wikipedia.org”至DNS服务器，DNS服务器首先检查自身缓存，如果存在记录则直接返回结果。

如果记录老化或不存在，则DNS服务器向根域名服务器发送查询报文“query zh.wikipedia.org”，根域名服务器返回“.org”域的权威域名服务器地址。

DNS服务器向“.org”域的权威域名服务器发送查询报文“query zh.wikipedia.org”，得到“.wikipedia.org”域的权威域名服务器地址。

DNS服务器向“.wikipedia.org”域的权威域名服务器发送查询报文“query zh.wikipedia.org”，得到主机zh的A记录，存入自身缓存并返回给客户端。

DNS系统还有一个很重要的概念：TTL (Time To Live的缩写)，简单地说，它表示的是一条域名解析记录在DNS服务器上的缓存时间。当一个递归域名服务器查询权威域名服务器获取某个域名的映射时，它会将该记录缓存上一定的时间，这个时间就是TTL指定的时间（以秒为单位）。如果在一台Linux机器上反复运行命令“dig www.mysql.com”，就会发现这个缓存时间在减少，为什么呢？因为在你的DNS缓存中，这笔记录能够保存的时间开始倒计数，如果TTL没有归零，缓存服务器会简单地用已缓存的记录答复查询请求。若这个数字归零后，下次再有人重新搜寻这笔记录时，你的DNS就需要从权威域名服务器重新获取记录。也就是说，如果更改了域名的指向，那么最长需要TTL时间才会完全生效。

了解DNS系统不仅仅是运维团队的事情，研发人员也有必要清楚其大概的机制。我们在部署程序或设计迁移方案的时候，需要清楚是否要申请域名、创建新的CNAME记录，是否需要修改TTL生效时间，是否需要修改A记录。另外需要注意的是，虽然有TTL的机制，但由于国内移动网络的特殊性，DNS的修改可能长期不能生效，由于其特殊的分布式数据库的设计，如果遭到域名污染，往往也会造成巨大的影响，这种问题很难解决。

2. 经过负载均衡软硬件设备

经过负载均衡软硬件设备如F5、Haproxy、LVS后，再把请求转发给后端的网络服务。

负载均衡（Load Balance），即将负载（工作任务）进行平衡、分摊到多个操作单元上进行执行，例如Web服务器、FTP服务器、企业关键应用服务器和其他关键任务服务器等，从而共同完成工作任务。当后端的一台服务器宕机或过载，负载均衡软硬件设备将不再转发流量到这台服务器，转而发送到备用的服务器上，从而实现自动故障冗余切换。

最早的负载均衡技术是通过DNS来实现的，在DNS中为多个地址配置同一个名字，因而查询这个名字的客户机将得到其中的一个地址，从而使得不同的客户访问不同的服务器，达到负载均衡的目的。DNS负载均衡是一种简单而且有效的方法，但是它不能区分服务器的差异，也不能反映服务器的当前运行状态，对于高并发、大流量的请求，很容易导致负载并不均衡。现实中，它可能作为更上层的负载均衡存在，完成粗粒度的流量调度任务，比如在机房之间使用DNS负载均衡，在机房内部使用其他负载均衡方式。

F5负载均衡器是应用交付网络的全球领导者F5 Networks公司提供的一个负载均衡器专用设备，一般需要配置双机故障冗余切换。F5主要应用于传统行业内，如电信、移动、银行等，也有许多互联网公司使用F5设备，虽然F5设备比较昂贵，但在一定规模下，它可以降低企业的成本，代替系统管理员、工程师管理各种资源。互联网公司用得比较多的是F5 BIG-IP LTM（本地流量管理器），由于国内网络的复杂性，也有使用BIG-IP广域网流量管理器（BIG-IP GTM）的。F5设备除了负载均衡外，还有

一些其他的功能，如利用压缩技术降低带宽支出、减少连接数等。

F5等硬件设备毕竟是商业化的产品，比较昂贵，在一定规模下使用可以获得比较好的投资回报率，但在公司初创时，或者公司已经流量很大的时候，F5设备的成本优势则并不明显，目前很多公司的F5设备已经逐步被LVS等其他软件替代，所以，对于互联网公司，一般建议使用开源软件实现负载均衡，常用的有LVS、Haproxy等，它们和其他技术配合使用可以实现很好的扩展性，无论是在流量很小还是流量很大的情况下，都能够满足需要。网上有很多关于LVS、Haproxy的资料，这里不再赘述。

3. 经过反向代理服务

反向代理是代理服务器的一种，比如Squid、Vanish等。它根据客户端的请求，从后端的服务器上获取资源，然后再将这些资源返回给客户端。常用的代理服务为Squid，它可以作为缓存服务器，可以过滤流量保证网络安全，也可以作为代理服务器链中的一环，向上级代理转发数据或直接连接互联网，一些网站往往在前端增加Squid反向代理加速响应、提高吞吐量。Squid可以缓存内容，特别是一些静态的数据，比如图片和文件，如果反向代理靠近用户的网络，那么用户就会得到延时很低的高质量访问，这正是CDN技术的核心。

4. 到达Web服务器

Web服务器包括Nginx、Apache、Lighttpd、Tomcat、Resin等。

Apache HTTP Server（简称Apache）是Apache软件基金会的一个开放源代码的网页服务器，是使用最广泛、最流行Web服务器端软件之一。Apache功能最完备，但占用的资源比较多，支持的连接数也比Nginx少，所以目前在互联网界，已经被Nginx抢了风头。

Nginx（发音同engine x）是一款由俄罗斯程序员Igor Sysoev所开发的轻量级的网页服务器，它也可以用作反向代理、负载均衡器，但更常见的功能是Web服务器。它是一款面向性能设计的HTTP服务器，以事件驱动的方式编写，很注重效率，所以在许多评测中，相比于Apache都有更高的性能，能支撑更多的并发请求。

在常见的网络架构中，Nginx往往配合php-fpm使用，Nginx负责处理静态请求，把PHP等动态请求抛给后端的php-fpm处理。或者Nginx处理前端的静态请求，把Apache放在后端处理一些动态请求。所以各种Web服务器之间可能也有一定的层级关系或功能分工，这点需要了解清楚。

5. 调用应用服务器

应用程序服务器是通过很多协议来为应用程序提供商业逻辑的服务器。

根据我们的定义，作为应用程序服务器，它将通过各种协议（包括HTTP），把商业逻辑暴露给客户端应用程序。Web服务器主要是处理向浏览器发送HTML以供浏览，而应用程序服务器则提供访问商业逻辑的途径以供客户端应用程序使用。

这里所说的应用服务器更多地属于内部调用的范畴。一般Web服务器可以高效地处理简单的响应请求，但如果有关复杂的商业逻辑的话，把这些业务逻辑放到独立的应用服务器上，然后通过调用的方式来获取信息会更安全、性能更高、开发也更方便。

6. 访问数据库

客户端不直接和数据库打交道，如果处理逻辑需要访问数据，则由应用服务器或Web服务器访问数据库，获取数据。

以上架构是比较普通的三层/四层架构，架构中也可能有一个缓存（Cache）服务，以减轻数据库的压力。Web服务器、应用服务器到数据库服务器中间可能存在数据中间件，但更常规的做法是通过在Web服务器、应用服务程序配置文件里指定IP或内网域名来配置数据库路由。

生产环境里，如果出现了性能问题，研发人员往往第一时间就会怀疑是数据库出现了性能问题，但事实往往并非如此，从上面的叙述可知，用户的访问请求经过了许多环节，有DNS、负载均衡设备、Web服务器，而且长距离网络中的数据传输物理上还需要经过许多设备，如路由器、交换机等，由于国内网络的特殊性和复杂性，有时会碰到网络丢包，丢包很可能导致性能问题。所以，如果出现了性能问题或访问异常，研发、运维人员就需要仔细甄别到底是哪个环节出现了问题，配合各种监控和日志记录，是有可能快速定位到问题症结所在的。



注意 数据库一般位于内网，若没有外网IP，则不需要暴露给外网。

3.4.3 开发工具

这里主要介绍一款常用的基于Web的管理工具phpMyAdmin。

phpMyAdmin是一个以PHP为基础的MySQL的数据库管理工具。让管理者可以通过Web接口操作数据库，也就是于远端管理MySQL数据库。

研发人员并不像DBA那样经常通过命令行来操作数据库，所以往往会不熟悉命令行，借由这套图形工具，可以方便地建立、修改、删除数据库及表、浏览数据、插入数据、删除数据。

phpMyAdmin也是很好的学习工具，可以生成SQL语句，验证语法，也可以生成常用的PHP语法。对于研发人员，若要求熟练使用命令行操作数据库，那么这个要求确实高了些，他们应该把更主要的精力放在程序代码的编写上，所以，建议熟悉此类图形工具。

需要注意的是，未经配置的phpMyAdmin不安全，容易受到攻击，建议只将其部署在开发和测试环境中。在生产环境中，如果需要部署phpMyAdmin，那么一定要先经过安全评估，确认没有安全漏洞，而这往往是需要通过改造phpMyAdmin来实现的。

还有一些客户端工具，如Workbench、toad for MySQL、SQLyog、Navicat for MySQL。这其中，SQLyog功能最强大，但是，它是收费软件，Workbench出自官方，对比以前的版本功能已经有了长足的进步，而且也一直在努力改进中，建议大家优先使用这个工具。

3.4.4 操作数据

如果想要做Web开发，那么读者需要熟悉HTML、XML、JavaScript、Ajax等知识。关于这些知识和PHP的基本语法和特性这里不做介绍，本章主要讲述PHP相对于数据的一些常用操作。如果需要详细了解PHP开发知识，可以参考官方文档或《PHP与MySQL Web开发》^[1]一书。

PHP提供了3种API可用于访问MySQL，分别是MySQL扩展、MySQLi扩展、PDO扩展。官方的建议是使用MySQLi或PDO，因为MySQL扩展在未来可能被废弃掉。MySQLi只支持MySQL数据库，如果有跨数据库平台的需要，那么就要使用PDO了。由于目前MySQL扩展仍然有广泛的应用，以下示例仍以MySQL扩展为例，同时也将列出MySQLi的代码供读者参考。

1.PHP连接数据库

在访问并处理数据库中的数据之前，必须先创建到达数据库的连接。

PHP提供了两个连接MySQL的函数mysql_pconnect()和mysql_connect()。

mysql_connect()函数的语法如下。

```
mysql_connect(servername,username,password);
```

涉及的参数详见表3-14。

表3-14 mysql_connect参数描述

参数	描述
servername	可选。规定要连接的服务器。默认是“localhost:3306”
username	可选。规定登录所使用的用户名。默认值是拥有服务器进程的用户的名称
password	可选。规定登录所用的密码。默认是“”

脚本一结束，就会关闭连接。如需提前关闭连接，请使用mysql_close()函数。

2.选择数据库

在执行语句前，需要先选择数据库。通过mysql_select_db()函数可选取数据库，它的语法如下。

```
bool mysql_select_db(db_name);
```

示例test_conn_use_db.php的语句如下。

```
<html>
<head>
<title>Selecting MySQL Database</title>
</head>
<body>
<?php
$dbhost = '127.0.0.1:3306';
$dbuser = 'garychen';
$dbpass = 'garychen';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
$db_selected = mysql_select_db('employees');
if (! $db_selected) {
    die ('Can\'t use employees : ' . mysql_error());
}
echo "<br />";
echo 'use employees successfully';
mysql_close($conn);
?>
</body>
</html>
```

使用MySQLi连接数据库，语句如下。

```
<html>
<head>
<title> connect database </title>
</head>
<body>
<?php
$host="127.0.0.1";
$port=3306;
$socket="";
$user="garychen";
$password="garychen";
$dbname="employees";
$con = new mysqli($host, $user, $password, $dbname, $port, $socket)
      or die ('Could not connect to the database server' . mysqli_connect_error());
```

```
echo 'connect to employees database successfully';
$ccon->close();
?>
</body>
</html>
```



注意 无论是指定“localhost”还是“localhost:port”作为servername，MySQL客户端库都将覆盖之并尝试连接到本地套接字。如果希望使用TCP/IP连接，则用“127.0.0.1”替代“localhost”。

3. 查询数据

先使用mysql_query()函数向MySQL发送查询或命令。然后使用mysql_fetch_array函数返回数据。

示例test_select.php语句如下。

```
<html>
<head>
<title>Selecting MySQL Database</title>
</head>
<body>
<?php
$ccon = mysql_connect("127.0.0.1", "garychen", "garychen");
if (!$ccon)
{
    die('Could not connect: ' . mysql_error());
}
mysql_select_db("employees", $ccon);
$result = mysql_query("select * from departments");
while($row = mysql_fetch_array($result))
{
    echo $row['dept_no'] . " " . $row['dept_name'];
    echo "<br />";
}
mysql_close($ccon);
?>
</body>
</html>
```

上面这个例子在\$result变量中存放了由mysql_query()函数返回的数据。接下来，使用mysql_fetch_array()函数以数组的形式从记录集返回第一行。随后对于mysql_fetch_array()函数的每个调用都会返回记录集中的下一行。while loop语句会循环记录集中的所有记录。为了输出每行的值，这里使用了PHP的\$row变量（\$row['dept_no']和\$row['dept_name']）。

4. 使用MySQLi查询数据

首先新建连接（new mysqli），然后预处理查询语句（使用prepare方法），执行这个语句（使用execute方法），最后，把结果集的列绑定到两个变量（使用bind_result方法），并获取实际数据（使用fetch方法），打印输出。

示例test_select2.php语句如下。

```
<html>
<head>
<title>select table</title>
</head>
<body>
<?php
$host="127.0.0.1";
$port=3306;
$socket="";
$user="garychen";
$password="garychen";
$dbname="employees";
$ccon = new mysqli($host, $user, $password, $dbname, $port, $socket)
        or die ('Could not connect to the database server' . mysqli_connect_error());
echo 'connect to employees database successfully';
echo "<br />";
echo "select departments table";
echo "<br />";
$query = "select * from departments";
if ($stmt = $ccon->prepare($query)) {
    $stmt->execute();
```

```
$stmt->bind_result($field1, $field2);
while ($stmt->fetch()) {
    printf("%s, %s\n", $field1, $field2);
    echo "<br />";
}
$stmt->close();
}
$con->close();
?>
</body>
</html>
```

5.插入记录

类似SELECT查询，先连接数据库（使用`mysql_connect`函数），然后选择数据库（使用`mysql_select_db`函数），最后发送`INSERT`语句给MySQL（使用`mysql_query`函数）。

示例`test_insert.php`语句如下。

```
<html>
<head>
<title>Insert records</title>
</head>
<body>
<?php
$con = mysql_connect("127.0.0.1", "garychen", "garychen");
if (!$con)
{
    die('Could not connect: ' . mysql_error());
}
mysql_select_db("employees", $con);
$sql="INSERT INTO employees (emp_no, birth_date, first_name, last_name, gender, hire_date)
VALUES (500000, '1990-08-01', 'Peter', 'wang', 'M', '2011-11-11')";
if (!mysql_query($sql,$con))
{
    die('Error: ' . mysql_error());
}
echo "1 record added";
mysql_close($con);
?>
</body>
</html>
```

6.使用MySQLi插入记录

新建连接（`new mysqli`），然后使用`mysqli_query`方法操作数据库，执行查询。

示例`test_insert2.php`语句如下。

```
<html>
<head>
<title>insert records</title>
</head>
<body>
<?php
$host="127.0.0.1";
$port=3306;
$socket="";
$user="garychen";
$password="garychen";
$dbname="employees";
$con = new mysqli($host, $user, $password, $dbname, $port, $socket)
        or die ('Could not connect to the database server' . mysqli_connect_error());
echo 'connect to employees database successfully';
echo "<br />";
$sql="INSERT INTO employees (emp_no, birth_date, first_name, last_name, gender, hire_date)
VALUES (500003, '1990-09-01', 'Peter', 'zhang', 'M', '2011-12-12')";
if (!mysqli_query($con,$sql))
{
    die('Error: ' . mysqli_error($con));
}
echo "1 record added";
$con->close();
?>
</body>
</html>
```

更新数据和删除语句的操作这里不再赘述，大家可参考如下链接。

关于MySQL: http://www.tutorialspoint.com/php/mysql_update_php.htm

关于MySQLi: http://www.w3schools.com/php/php_mysql_update.asp

3.4.5 PHP数据库开发建议

对于开发人员来说，了解数据流和生产环境的物理部署是很重要的，开发人员不应该局限于自己的领域。

能够熟练查阅MySQL官方文档，虽然研发人员不必熟悉MySQL的每一个细节，但要善于查找资料，找到答案，一般我们经历的问题都是可以从文档或网络中找到答案的。

注意安全问题，防止SQL注入、跨站脚本攻击等恶意攻击。详细内容可参考4.2节“权限机制和安全”。

阅读源码，了解它们是如何访问数据的。一些网站，如sourceforge提供了许多优秀的源码。

注重用户体验，数据库的访问优化，往往和用户体验相关。如果要访问数据库或数据接口，优化的方式不外乎两个，或者减少对数据的访问，或者让查询执行得更快。比如一些大页面只改动了几行字，那么查询时可能就只需要检索小部分数据，而不是重载整个页面查询很多数据。比如地图，只需要加载部分块区域，而不是整个图重新绘制。比如翻页，传统的一些算法效率往往很差，影响用户体验。

要有性能分析器，如果想要开发高质量的程序，那么需要对自己的程序进行分析，特别是对于数据库访问的分析，比如记录数据库性能访问日志。一些工具也有助于分析网站的响应，如firebug。

[1] 该书第4版由机械工业出版社2009年出版，书号为978-7-111-26281-7。——编辑注

3.5 索引

3.5.1 索引介绍

数据库索引，是数据库管理系统中一个排序的数据结构，用于协助快速查询、更新数据库表中的数据。它类似于书本上的索引，通过索引可以更便捷地找到书里面的内容而不需要查阅整本书。对于海量数据的检索，索引往往是最有效的。

目前MySQL主要支持的几种索引有：B树索引（B-tree）、散列索引（hash）、空间索引（R-tree）和全文索引（full-text）。如果没有特别指明，本书指的就是B-Tree索引。由于索引是在存储引擎层实现的，所以不同的存储引擎的索引实现会有一些差异。以下所述的是一些较通用的索引知识。

逻辑上又可以分为：单列索引、复合索引（多列索引）、唯一（Unique）索引和非唯一（Non Unique）索引。

如果索引键值的逻辑顺序与索引所服务的表中相应行的物理顺序相同，那么该索引被称为簇索引（cluster index），也称为聚集索引、聚簇索引，也就是说数据和索引（B+树）在一起，记录被真实地保存在索引的叶子中，簇索引也称为索引组织表，反之为非聚集索引。我们常用的InnoDB表其实使用的就是聚集索引。

簇索引是一个很重要的概念，InnoDB作为最常使用的引擎，只有在熟悉了它的数据存储方式之后，才可能有针对性地对它进行调优。

簇索引的一些优点如下。

- 将相关的的数据保持在一起，叶子节点内可保存相邻近的记录。
- 因为索引和数据存储在一起，所以查找数据通常比非簇索引更快。由于主键是有序的，很显然，对于InnoDB表，最高效的存取方式是按主键存取唯一记录或进行小范围的主键扫描。

如果充分利用簇索引，它可以极大地提升性能，但簇索引也有许多不足之处。

·簇索引对I/O密集型的负荷性能提升最佳，但如果数据是在内存中（访问次序不怎么重要），那么簇索引并没有明显益处。

- 插入操作很依赖于插入的顺序，按primary key的顺序插入是最快的。
- 更新簇索引列的成本比较高，因为InnoDB不得不将更新的行移动到新的位置。
- 全表扫描的性能不佳，尤其是数据存储得不那么紧密时，或者因为页分裂（page split）而导致物理存储不连续。
- 二级索引的叶节点中存储了主键索引的值，如果主键采用的是较长的字符，那么索引可能会很大，且通过二级索引查找数据也需要进行两次索引查找。

3.5.2 使用索引的场景及注意事项

1. 何种查询可以应用索引

- (1) MySQL目前仅支持前导列

筛选记录的条件应能组成复合索引最左边的部分，即按最左前缀的原则进行筛选。随着日后的技术发展，MySQL或许能够更有效地利用复合索引多字段中的非前导列信息。

下面来看个例子，对于如下的复合索引idx_a_b_c:

```
CREATE INDEX idx_a_b_c ON tb1(a,b,c);
```

只有使用如下条件才可能应用到这个复合索引。

```
WHERE a=?  
WHERE a=? AND b=?  
WHERE a=? AND b=? AND c=?  
WHERE a=? AND c=? #注意这个查询仅仅利用了  
MySQL索引的  
a列信息
```

(2) 索引列上的范围查找

在对某个条件进行范围查找时，如果这个列上有索引，且使用的是WHERE...BETWEEN...AND...、>、<等范围操作符时，那么可能就会用到索引范围查找。一般应该避免大范围的索引范围查找，如果索引范围查找的成本太高，那么数据库可能会选择全表扫描的方式。



注意 IN(...)并不属于范围查找的范畴。

(3) JOIN列

在联合查询两个表时，比如查询语句为“SELECT a.col1,b.col2 FROM a JOIN b ON a.id=b.id”，其中id为主键，若a表是驱动表，那么数据库可能全表扫描a表，并用a表的每个id去探测b表的索引查找匹配的记录。

(4) WHERE子句

WHERE子句的条件列是复合索引前面的索引列再加上紧跟的另一个列的范围查找。

比如，对于如下的复合索引idx_a_b_c_d:

```
CREATE INDEX idx_a_b_c_d ON tb1(a,b,c,d);
```

只有使用如下条件才可能应用到这个复合索引。

```
WHERE a=? AND b=? AND c > 10000;  
WHERE a=? AND b=? AND c=? AND d<10000;
```



注意 MySQL索引仅支持最近一个范围的查询。也就是说，MySQL使用最左边的前缀，一直到碰到第一个范围的查找条件为止。

对于复合索引idx_a_b_c_d，我们来看如下的两个查询。

```
WHERE a=? AND b=? AND c > 10000 AND d<100000;
```

上面的例子中，d<100000这个筛选操作并不会走索引。

```
WHERE a>? AND b=? AND c= 10000 AND d = 100;
```

上面的例子中，`a`列上有范围查找，那么`b`、`c`、`d`等列上的索引信息将都不能被利用。

即对于复合索引，如果某部分索引已经用到了范围查找，那么这个列之后的索引信息将不能被利用。

所以如果想要创建索引，应该考虑把复合索引的范围查找列放到最后。

(5) MySQL优化器

MySQL优化器会做一些特殊优化，比如对于索引查找MAX（索引列），那么可以进行直接定位，在EXPLAIN输出的Extra信息里可以看到语句“Select tables optimized away”，意思是这个查询所包含的MIN、MAX操作可以直接利用索引信息来解决，而不需要去检索物理记录。优化器确定只需要返回一行结果即可。

2.注意事项和建议

1) WHERE条件中的索引列不能是表达式的一部分，MySQL也不支持函数索引。

2) InnoDB的非主键索引存储的不是实际记录的指针，而是主键的值，所以主键最好是整型值，如自增ID，基于主键存取数据是最高效的，使用二级索引存取数据则需要进行两次索引查找。

3) 最好是按主键的顺序导入数据，如果导入大量随机id的数据，那么可能需要运行OPTIMIZE TABLE命令来优化表。

4) 索引应尽量是高选择性的，而且需要留意“基数（cardinality）”值，基数指的是一个列中不同值的个数，显然，最大基数意味着该列中的每个值都是唯一的，最小基数意味着该列中的所有值都是相同的。索引列的基数相对于表的行数较高时（也就是说重复值更少），索引的工作效果更好。

一些基数很小的列，如性别可能就不适合建立索引。也存在这样一种特殊的情况，有些列虽然基数很小，但由于数据分布很不均匀因此也会导致某些值的记录数很少，那么这种情况也适合创建索引加速查找这部分数据。

5) 使用更短的索引。可以考虑前缀索引，前缀索引仅索引前面一部分字符（值），但应确保所选择的前缀的长度可以保证大部分值是唯一的。

示例如下：

```
ALTER TABLE test.test1 ADD KEY (col(6))
```

如下的SQL衡量了不同前缀索引的唯一值比例。

```
SELECT COUNT(DISTINCT LEFT(col_name, 3))/COUNT(*) AS sel3,
COUNT(DISTINCT LEFT(col_name, 4))/COUNT(*) AS sel4,
COUNT(DISTINCT LEFT(col_name, 5))/COUNT(*) AS sel5,
COUNT(DISTINCT LEFT(col_name, 6))/COUNT(*) AS sel6,
COUNT(DISTINCT LEFT(col_name, 7))/COUNT(*) AS sel7
FROM table_name;
```

6) 索引太多时可能会浪费空间，且降低修改数据的速度。所以，不要创建过多的索引，也不要创建重复的索引。MySQL允许在同样的列上创建多个索引而不会提示报错，一些其他分支的版本有统计信息可以甄别出没有被使用的索引。而对于官方版本，你可能需要借助工具清理掉过多的重复索引。

7) 如果是唯一值的列，创建唯一索引会更佳，也可以确保不会出现重复数据。

8) 使用覆盖索引（covering index）也可以大大提高性能。

所谓“覆盖索引”是指所有数据都可以从索引中得到，而不需要去读取物理记录。例如某个复合索引idx_a_b_c建立在表tbl的a、b、c列上，那么对于如下的SQL语句

```
select a,b from tbl where a=? and b=? and c=?;
```

MySQL可以直接从索引idx_a_b_c中获取所有数据。使用覆盖索引也可以避免第2点所说的二次索引查找。

在EXPLAIN命令输出的查询计划里，如果Extra列是“using index”，那就表示使用的是覆盖索引。EXPLAIN的使用在下节详述。

9) 利用索引来排序。MySQL有两种方式可以产生有序的结果。一种是使用文件排序（filesort）来对记录集进行排序，另一种是扫描有序的索引。我们应尽量利用索引来排序。

在文件排序方式中，由于没有可以利用的有序索引来取得有序的数据，因此MySQL只能将取得的数据在内存中进行排序，然后再将数据返回给客户端。使用文件排序的方式，对小结果集进行排序会很快，但是如果是对大量的数据排序，速度将会很慢。此外，还有如下注意事项。

- 尽量保证索引列和ORDER BY的列相同，且各列均按相同的方向排序。

- 如果要连接多张表，那么ORDER BY引用的列需要在表连接的顺序的首张表内。

如果不满足以上条件，则不能利用索引进行排序，那么MySQL将使用文件排序，可以用EXPLAIN工具确认查询是否使用了文件排序，文件排序是一个成本比较高的操作，应尽量避免。利用索引来排序同样要遵循最左前缀的规则，前导列（等于确定值）加上排序列（ORDER BY的列）可以组合成最左前缀的也行。比如，对于创建在表table1上的复合索引idx_a_b_c（创建在列a、b、c上）：

```
SELECT * FROM table1 ORDER BY a,b,c;
SELECT * FROM table1 WHERE a=? AND b=? ORDER BY c;
```

以上查询都可以利用有序索引来加速检索数据。

10) 添加冗余索引，需要权衡。

什么是冗余索引？如果已有一个索引（columnA），那么一个新的索引（columnA, columnB）就是冗余索引，因为后面的索引包含了前面索引的所有信息。

冗余索引一般发生在添加索引的时候，有些人可能会选择添加一个新索引（columnA, columnB），而不是更改原来的（columnA）为（columnA, columnB）。一般来说，应该扩展原来的索引，而不是添加新的索引。但在某些情况下，因为扩展索引会导致索引变得非常大，比如原来的索引是创建在一个整型列上的，要是再添加一个很长的字符列，那么索引会变得很大，从而影响性能。这种情况下，可能不得不选择添加新的复合索引，保留原来的索引，这样做的不利之处是增加了索引维护的开销，而且一个新的索引也需要占据内存空间。

3.5.3 索引的错误用法

以下是生产环境中常犯的一些错误，而且由于表结构不易调整，因此往往会导致严重的性能影响。

- 1) 创建了太多的索引或无效的索引。比如在WHERE条件的每个列上都建立单独的索引，当单个索引效率不高的时候，MySQL往往就会选择全表扫描，太多的索引可能会导致索引所占用的磁盘空间比实际数据还大得多。
- 2) 对于复合索引，如果不考虑ORDER BY、GROUP BY这样的一些操作，那么把最具选择性的列放在前面是合适的，复合索引主要用于优化WHERE查找。但如果是排序之类的操作，把最具选择性的列放在前面则不一定最有效，因为避免随机I/O和排序可能才是我们更值得考虑的。
- 3) 忽略了值的分布。某些值只有少量记录，查询对这些值的筛选执行就会很快，而某些值即使经过了索引筛选，满足条件的仍然还有大量的记录，这样索引效果就会很差。一般来说，数据表内值的分布应该尽量均匀，由于MySQL的统计信息不完善，数据分布不均匀很可能会产生很差的执行计划，导致严重的性能问题。
- 4) InnoDB主键过长，导致二级索引过大。主键的选择，一般建议是整型。

以上介绍了索引的使用规则和建议。接下来介绍EXPLAIN工具。互联网应用的开发，索引的调整往往是调优的一个重点，特别是对数据库技术不熟练的团队，在数据量增长后可能会碰到各种性能问题，这通常是因为索引不佳而引起的。EXPLAIN工具的应用不局限于索引的检查，但由于它和索引关系密切，下面将详细介绍下此工具。

3.5.4 如何使用EXPLAIN工具

无论是做研发还是DBA，都有必要学会EXPLAIN工具的使用。使用EXPLAIN工具可以确认执行计划是否良好，查询是否走了合理的索引。不同版本的MySQL优化器各有不同，一些优化规则随着版本的发展可能会有变化，查询的执行计划随着数据的变化也可能会有变化。对于这类情况可以使用EXPLAIN来验证自己的判断。

以下是对EXPLAIN工具的使用说明。我们首先来介绍一下MySQL执行计划的调用方式，然后对执行计划显示的内容进行解读，最后来说一说MySQL执行计划的局限。

1. MySQL执行计划调用方式

我们使用EXPLAIN命令查看执行计划，语法形式类似如下语句。

```
EXPLAIN SELECT...
```

EXPLAIN命令还有如下两种变体。

1) EXPLAIN EXTENDED SELECT.....

以上命令将执行计划“反编译”成SELECT语句，运行SHOW WARNINGS可得到被MySQL优化器优化后的查询语句。

2) EXPLAIN PARTITIONS SELECT.....

以上命令用于分区表的EXPLAIN命令。

2. 执行计划包含的信息及解读

如下是一个显示执行计划的例子。

```

mysql> explain
    > select t1.col12, t2.col1
    > from t1,t12, t2
    >      from t1
    >      where col1 in ('ab','ac')) t1, t2
    > where dt.col12 = t2.col1
    > group by dt.col12, t2.col1
    > order by t2.id;
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | PRIMARY     | <derived2> | ALL  | NULL          | NULL | NULL    | NULL | 285   |
| 1 | PRIMARY     | t1       | ref  | idx_col1_col2 | idx_col1_col2 | 168   | dt.col12 | 1    |
| 2 | DERIVED     | t1       | range | idx_col1_col2_col3 | idx_col1_col2_col3 | 13   | NULL    | 285   |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

该例中EXPLAIN命令的输出信息可以告诉我们MySQL访问了哪些表，以及它是如何访问数据的。里面有很重要的索引使用信息，我们可以据此判断我们的索引是否需要优化。

下面将详细阐述EXPLAIN输出的各项内容。

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra	

(1) id

id包含一组数字，表示查询中执行SELECT子句或操作表的顺序。

如果id相同，则执行顺序由上至下，例如：

```

mysql> explain select t2.*
    > From t1, t2, t3
    > -> where t2.id = t3.id and t1.id = t3.id
    > -> and t1.other_column = '';
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE     | t1       | ref  | PRIMARY_idx_t1 | idx_t1 | 92    | const | 1    |
| 1 | SIMPLE     | t3       | eq_ref | eq_ref | PRIMARY | 4     | test.t1.ID | 1    |
| 1 | SIMPLE     | t2       | eq_ref | PRIMARY | PRIMARY | 4     | test.t1.ID | 1    |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

如果是子查询，id的序号会递增，id值越大则优先级越高，越先被执行。例如：

```

mysql> explain SELECT t2.* 
    > FROM t2
    > WHERE id = (SELECT id
    >      FROM t1
    >      WHERE id = (SELECT t3.id
    >          FROM t3
    >          WHERE t3.other_column = ''));;
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | PRIMARY     | t2       | const | PRIMARY     | PRIMARY | 4     | const | 1    |
| 2 | SUBQUERY    | t1       | const | PRIMARY     | PRIMARY | 4     | const | 1    |
| 3 | SUBQUERY    | t3       | ALL    | NULL        | NULL   | NULL   | NULL  | 1    |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

如果id相同，则可以认为它们是一组，从上往下顺序执行。在所有组中，id值越大，优先级就越高，越先执行。例如：

```

mysql> explain select t2.* from (
    > select t2.id
    > from t3
    > where t3.other_column = '') s1, t2
    > where s1.id = t2.id;
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | PRIMARY     | <derived2> | system | NULL          | NULL | NULL    | NULL | 1    |
| 1 | PRIMARY     | t2       | const | PRIMARY     | PRIMARY | 4     | const | 1    |
| 2 | DERIVED     | t3       | ALL    | NULL          | NULL | NULL    | NULL | 1    |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

(2) select_type

select_type表示查询中每个SELECT子句的类型（是简单还是复杂）。输出结果类似如下：

id	select_type								
1	SIMPLE								
2	PRIMARY								
3	SUBQUERY								
4	DERIVED								
5	UNION								
6	UNION RESULT								

下面是对select_type的详细说明。

·SIMPLE: 查询中不包含子查询或UNION。

·查询中若包含任何复杂的子部分，最外层查询则被标记为PRIMARY。

·在SELECT或WHERE列表中若包含了子查询，则该子查询被标记为SUBQUERY。

·在FROM列表中包含的子查询将被标记为DERIVED（衍生）。

·若第二个SELECT出现在UNION之后，则被标记为UNION；若UNION包含在FROM子句的子查询中，则外层SELECT将被标记为DERIVED。

·从UNION表中获取结果的SELECT将被标记为UNION RESULT。

下面我们通过一个例子来说明查询的类型和执行的顺序。

```
mysql> explain select d1.name, (select id from t3) d2
   -> from (select id, name from t1 where other_column = '') d1
      -> union
         -> (select name, id from t2);
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  1 | PRIMARY    | <derived3> | system | NULL       | NULL | NULL    | NULL |    1 | Using where |
|  3 | DERIVED     | t1        | ALL    | NULL       | NULL | NULL    | NULL |    1 | Using where |
|  2 | SUBQUERY    | t2        | index  | NULL       | PRIMARY | 4       | NULL |    1 | Using index  |
|  4 | UNION       | t2        | ALL    | NULL       | NULL | NULL    | NULL |    1 | Using index  |
| NULL | UNION RESULT | <union1,2> | ALL    | NULL       | NULL | NULL    | NULL | NULL |           |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.01 sec)
```

第一行：id列为1，表示第一个SELECT，select_type列的PRIMARY表示该查询为外层查询，table列被标记为<derived3>，表示查询结果来自于一个衍生表，其中3代表该查询衍生自第3个SELECT查询，即id为3的SELECT。

第二行：id为3，表示该查询的执行次序为2（4→3），是整个查询中第3个SELECT的一部分。因为查询语句包含在FROM子句中，所以为DERIVED。

第三行：SELECT列表中的子查询，select_type为SUBQUERY，为整个查询中的第2个SELECT。

第四行：select_type为UNION，说明第4个SELECT是UNION里的第2个SELECT，最先执行。

第五行：代表从UNION的临时表中读取行的阶段，table列的<union1,4>表示对第1个和第4个SELECT的结果进行UNION操作。

(3) type

type表示MySQL在表中找到所需行的方式，又称“访问类型”，常见的类型如下：

```
+-----+-----+-----+-----+-----+-----+
| ALL  | index | range | ref  | eq_ref | const, system | NULL |
+-----+-----+-----+-----+-----+-----+
```

以上类型，由左至右，由最差到最好。下面我们来详述每种类型。

ALL: Full Table Scan, MySQL将遍历全表以找到匹配的行。如下是一个type为All的例子。

```
mysql> explain select * from t1 where column_without_index = '';
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  1 | SIMPLE     | t1    | ALL  | NULL       | NULL | NULL    | NULL | 516 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

index: Full Index Scan, index与ALL区别为index类型只遍历索引树。如下是一个type为index的例子。

```
mysql> explain select id from t1;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t1 | index | NULL | PRIMARY | 4 | NULL | 516 | Using index |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

range: 索引范围扫描，对索引的扫描开始于某一点，返回匹配值域的行，常见于between、<、>等的查询。如下是两个type为range的例子。

```
mysql> explain SELECT * FROM t1 WHERE id between 38 and 68;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t1 | range | PRIMARY | PRIMARY | 4 | NULL | 31 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
mysql> explain select * from t1 where id in (1, 2, 6);
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t1 | range | PRIMARY | PRIMARY | 4 | NULL | 3 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

一般来说，索引范围扫描要检索的记录更少，因而成本也更低。大量的索引扫描，可能还会导致性能问题。例如，对于如下的两个查询，后一个查询需要检索的记录数就比前一个查询多得多（参考rows列）。

```
mysql> explain select id from t1 where col1 in ('aa','ab','aa') and col2 = 'ac';
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t1 | range | idx_col1_col2 | idx_col1_col2 | 398 | NULL | 3 | |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
mysql> explain select id from t1 where col1 > 'aa' and col2 = 'ac';
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t1 | range | idx_col1_col2 | idx_col1_col2 | 194 | NULL | 299 | |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

ref: 非唯一性索引扫描，将返回匹配某个单独值的所有行。常见于使用非唯一索引或唯一索引的非唯一前缀进行的查找。如下是type为ref的几个例子。

```
mysql> create index idx_col1_col2 on t1(col1,col2);
Query OK, 1000 rows affected (0.15 sec)
Records: 1000  Duplicates: 0  Warnings: 0

mysql> select count(distinct col1) from t1;
+-----+-----+
| count(distinct col1) |
+-----+-----+
| 7 |
+-----+-----+
1 row in set (0.00 sec)

mysql> explain select * from t1 where col1 = 'ac';
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t1 | ref | idx_col1_col2 | idx_col1_col2 | 194 | const | 284 | |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> explain select * from t1, t2 WHERE t1.col1 = t2.col1;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t2 | ALL | NULL | NULL | NULL | NULL | 639 | |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t1 | ref | idx_col1_col2 | idx_col1_col2 | 195 | shared.t2.col1 | 45 | |
+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> explain select * from t1, t2 WHERE t1.col1 = t2.col1 AND t1.col2 = 'ac';
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t2 | ALL | NULL | NULL | NULL | NULL | 639 | |
| 1 | SIMPLE | t1 | ref | idx_col1_col2 | idx_col1_col2 | 398 | shared.t2.col1,const | 45 | |
+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

eq_ref: 唯一性索引扫描，对于每个索引键，表中只有一条记录与之匹配。常见于主键或唯一索引扫描。如下是type为eq_ref的一个例子。

```
mysql> explain select * from t1, t2 where t1.id = t2.id;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t2 | ALL | PRIMARY | PRIMARY | 4 | NULL | 639 | |
| 1 | SIMPLE | t1 | eq_ref | PRIMARY | PRIMARY | 4 | Shared_L2.ID | 1 | |
+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

const、system: 当MySQL对查询的某部分进行优化，并转换为一个常量时，可使用这些类型进行访问。如将主键置于

WHERE列表中，MySQL就能将该查询转换为一个常量。system是const类型的特例，当查询的表只有一行的情况下，即可使用system。如下是type为const和system的一个例子。

```
mysql> explain select * from (select * from t1 where id = 1) d1;
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | PRIMARY | <derived2> | system | NULL | NULL | NULL | NULL | 1 |
| 2 | DERIVED | t1 | const | PRIMARY | PRIMARY | 4 | NULL | 1 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

NULL: MySQL在优化过程中分解语句，执行时甚至不用访问表或索引。如下是type为NULL的一个例子。

```
mysql> explain extended select * from t1 where id = (select min(id) from t2);
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | PRIMARY | t1 | const | PRIMARY | PRIMARY | 4 | const | 1 | 100.00 | |
| 2 | SUBQUERY | NULL | |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set, 1 warning (0.00 sec)

mysql> show warnings;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Note | 1003 | select '1' AS 'ID','hu' AS 'col1','dba' AS 'col2' from 'shared'.`t1` where 1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

(4) possible_keys

possible_keys将指出MySQL能使用哪个索引在表中找到行，查询涉及的字段上若存在索引，则该索引将被列出，但不一定被查询使用。

(5) key

key将显示MySQL在查询中实际使用到的索引，若没有使用索引，则显示为NULL。查询中若使用了覆盖索引，则该索引仅出现在key列表中。如下是使用覆盖索引的一个例子。

```
mysql> explain select col1, col2 from t1;
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t1 | index | NULL | idx_col1_col2 | 898 | NULL | 682 | Using index |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

(6) key_len

key_len表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度。下面我们通过一个例子来说明。

```
mysql> desc t1;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| ID | int(11) | NO | PRI | NULL | auto_increment |
| col1 | char(4) | YES | NULL | NULL | |
| col2 | char(4) | YES | NULL | NULL | |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

上面的t1表col1和col2字段使用的是utf8字符集，utf8字符集的最大字符长度为3个字节，也就是说，它们可能需要12个字节存储一个值。复合索引idx_col1_col2是创建在col1、col2列上的索引。如下两个查询中，第一个查询我们可以看到key_len为13，它只用到了复合索引idx_col1_col2的前半部分信息。第二个查询的key_len为26，它是完整的索引长度，由此可知t1表的索引idx_col1_col2已被充分使用。

```
mysql> explain select * from t1 where col1 = 'ab';
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t1 | ref | idx_col1_col2 | idx_col1_col2 | 13 | const | 143 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> explain select * from t1 where col1 = 'ab' and col2 = 'ac';
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t1 | ref | idx_col1_col2 | idx_col1_col2 | 26 | const,const | 1 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

注意，key_len显示的值为索引字段的最大可能长度，并非实际使用长度，即key_len是根据表定义计算而得的，而不是通过表内检索得出的。

(7) ref

ref表示上述表的连接匹配条件，即哪些列或常量被用于查找索引列上的值。如下的例子中，col1匹配t2表的col1，col2匹配了一个常量，即ac：

```
mysql> explain select * from t1, t2 where t1.col1 = t2.col1 and t1.col2 = 'ac';
+----+-----+-----+-----+-----+-----+-----+
| id | ... | table | type | possible_keys | key | key_len | ref |
+----+-----+-----+-----+-----+-----+-----+
| 1 | ... | t2 | ALL | NULL | NULL | NULL | NULL |
| 1 | ... | t1 | ref | idx_col1_col2 | idx_col1_col2 | 20 | shared.t2.col1,const |
+----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

(8) rows

rows表示MySQL根据表统计信息及索引选用的情况，估算地找到所需的记录所需要读取的行数。如下的例子中，我们可以看到，在创建索引后，执行计划发生改变，所需要读取的行数减少了。

```
mysql> explain select * from t1, t2 where t1.id = t2.id and t2.col1 = 'ac';
+----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
+----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t2 | ALL | PRIMARY | NULL | NULL | NULL |
| 1 | SIMPLE | t1 | eq_ref | PRIMARY | PRIMARY | 4 | Shared.t2.ID |
+----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> create index idx_col1_col2 on t2(col1,col2);
Query OK, 1001 rows affected (0.17 sec)
Records: 1001  Duplicates: 0  Warnings: 0

mysql> explain select * from t1, t2 where t1.id = t2.id and t2.col1 = 'ac';
+----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
+----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t2 | ref | PRIMARY, idx_col1_col2 | idx_col1_col2 | 195 | const |
| 1 | SIMPLE | t1 | eq_ref | PRIMARY | PRIMARY | 4 | Shared.t2.ID |
+----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

(9) Extra

Extra包含不适合在其他列中显示但十分重要的额外信息。它可能包含如下4种信息。

1) Using index。该值表示相应的SELECT操作中使用了覆盖索引。包含满足查询需要的所有数据的索引称为覆盖索引。如下的查询就使用到了覆盖索引。

```
mysql> explain select col2 from t1 where col1 = 'ab';
+----+-----+-----+-----+-----+-----+
| id | ... | possible_keys | key | key_len | ref | rows | Extra |
+----+-----+-----+-----+-----+-----+
| 1 | ... | idx_col1_col2 | idx_col1_col2 | 13 | const | 148 | Using where; Using index |
+----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

2) Using where。该值表示MySQL服务器在存储引擎收到记录后进行“后过滤”（Post-filter）。

如果查询未能使用索引，则Using where的作用只是提醒我们MySQL将用where子句来过滤结果集。如下的查询同时使用到了覆盖索引和过滤。

```
mysql> explain extended select t1.col2 from t1, t2 where t1.col1 = 'ab' and t1.id = t2.id ;
+----+-----+-----+-----+-----+-----+-----+
| id | ... | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+
| 1 | ... | idx_col1_col2 | 13 | const | 143 | 100.00 | Using where; Using index |
| 1 | ... | PRIMARY | 4 | shared,t1.ID | 1 | 100.00 | Using index |
+----+-----+-----+-----+-----+-----+
2 rows in set, 1 warning (0.00 sec)
```

3) Using temporary。该值表示MySQL需要使用临时表来存储结果集，常见于排序和分组查询。

如下是一个使用临时表的例子。

```
mysql> explain select col1 from t1 where col1 in ('ac','ab','aa') group by col2\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
    table: t1
       type: range
possible_keys: idx_col1_col2
      key: idx_col1_col2
  key_len: 13
    ref: NULL
   rows: 509
  Extra: Using where; Using index; Using temporary; Using filesort
1 row in set (0.00 sec)
```

如上查询的EXPLAIN输出中Extra列同时有Using temporary和Using filesort，因此性能可能不佳。如果我们更改了GROUP BY子句，利用索引进行排序，则可以看到EXPLAIN输出里没有了Using temporary和Using filesort，示例如下。

```
mysql> explain select col1 from t1 where col1 in ('ac', 'ab') group by col1, col2\G
***** 1. row *****
      id: 1
  select_type: SIMPLE

    table: t1
       type: range
possible_keys: idx_col1_col2_col3
      key: idx_col1_col2_col3
  key_len: 26
    ref: NULL
   rows: 4
  Extra: Using where; Using index for group-by
1 row in set (0.00 sec)
```

4) Using filesort。Using filesort即文件排序。MySQL中将无法利用索引完成的排序操作称为“文件排序”。如下便是一个使用到了文件排序的例子。

```
mysql> explain select col1 from t1 where col1 = 'ac' order by col2\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
    table: t1
       type: ref
possible_keys: idx_col1_col2_col3
      key: idx_col1_col2_col3
  key_len: 13
    ref: const
   rows: 142
  Extra: Using where; Using index; Using filesort
1 row in set (0.00 sec)
```

如果我们更改查询，利用索引进行排序，则可以优化掉文件排序，例如如下的查询，已经没有了filesort（文件排序）。

```
mysql> explain select col1 from t1 where col1 = 'ac' order by col2, col3\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
    table: t1
       type: ref
possible_keys: idx_col1_col2_col3
      key: idx_col1_col2_col3
  key_len: 13
    ref: const
   rows: 142
  Extra: Using where; Using index
1 row in set (0.00 sec)
```

3.MySQL执行计划的局限

- EXPLAIN不会告诉你关于触发器、存储过程的信息或用户自定义函数对查询的影响情况。
- EXPLAIN不考虑各种Cache。
- EXPLAIN不能显示MySQL在执行查询时所做的优化工作。
- 部分统计信息是估算的，并非精确值。
- MySQL 5.6之前EXPALIN只能解释SELECT操作，其他操作需要重写为SELECT后才能查看执行计划。
- 如果FROM子句里有子查询，那么MySQL可能会执行这个子查询，如果有昂贵的子查询或使用了临时表的视图，那么EXPLAIN其实会有很大的开销。

3.5.5 优化索引的方法学

以上介绍了索引的结构和使用索引的一些规则，随着项目经验的增长，开发人员对于数据库都有一个从不熟悉到熟悉的过程，但由于开发人员的专注领域并不是数据库设计开发，而且不同的数据库产品也有差异，因此导致了部分开发人员对索引产生了一些误解。生产环境中数据库出现性能问题，有80%的原因是索引策略导致的，表结构不易变动，而调整索引或SQL往往可以很快就能解决问题，在开发或上线后，可遵循以下的方法和步骤进行优化。

(1) 有性能测量

在应用程序中记录访问数据库的性能日志，这样就可以对整体的访问吞吐有一个很直观很全面的统计，我们应该优化对数据库操作最频繁、最耗资源的那些SQL，但由于性能统计框架的缺位，大部分中小公司更多地依赖于数据库自身的慢查询日志来定位耗时较长的SQL，由慢查询日志入手也是一个很好的出发点，但可能存在一些滞后，不能及时发现性能问题，MySQL的慢查询日志默认记录查询时间超过1s的查询，4.3节将详细介绍慢查询日志。

(2) 查看执行计划

找到消耗资源最多的查询请求后，可以使用EXPLAIN工具查看其执行计划，检查是否走的是合适的索引。

(3) 优化索引

我们应该熟悉数据量、数据类型等信息及表之间的关系，按照自己的索引经验，调整或增加索引。

(4) 测试验证

如果是线上生产环境，那么请不要在线上环境进行测试验证，除非是非常紧急的情况。应该选择在开发环境中尽量使用和线上环境一样的数据规模，来进行验证测试。

(5) 上线

当确认优化达到了预期的效果后，就可以安排上线了。

有一个错误的观念是定期重建索引，这种方式在早期的传统数据库中用得很多，基于的主要理由是经过长期的生产运行，索引变得越来越不平衡，但是是否需要定期重建索引是有争议的。MySQL在互联网行业一般是OLTP应用，索引重建将导致服务变得不可用，更重要的是，在绝大部分情况下，重建了和没有重建索引，性能上并没有什么区别，唯一可能的场景是在大量删除导入数据后，会导致数据表严重变形。如果需要重建索引，首先要证明，重建索引真的能够大大改善性能，否则建议不要做这种费力又不讨好的事情，数据库索引本来就应该“不好不坏”的状态，不要期望它始终以一种理想的状态在运行。

3.6 ID主键

下面先说明选择主键的注意事项。

- 1) 建议主键是整型。
- 2) 如果表中包含一列能够确保唯一、非空 (NOT NULL)，以及能够用来定位一条记录的字段，就不要因为传统而觉得一定要加上一个自增ID做主键。
- 3) 主键也遵从索引的一些约定，注意联合主键的字段顺序。
- 4) 为主键选择更有意义的名称，如ID这个名称太过笼统，表达的信息可能不准确。

1. 自增ID主键

自增列是MySQL里的一种特殊的整型，我们定义一个列的整型的同时，可以设置它是否为自增的，一个表只能有一个列是自增列，且自增列必然是主键列。自增列的默认起始值是1，默认可以按步长为1进行递增，自增列的增长将受两个MySQL全局参数的影响。

·`auto_increment_offset`: 确定AUTO_INCREMENT列值的起点。

·`auto_increment_increment`: 控制列值增加的间隔，即步长。

也可以单独定义某个表的起始值，如：

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

在复制环境中，设置这两个值可以减少主键冲突，关于这一点以后会在复制章节（第12章）中详述。

使用自增列的原因是唯一标识数据表的某行记录。它们也被用来优化表之间的连接。连接单个列比连接多个列更快，连接整数列比连接其他大多数数据类型也更快。总之，有很多使用它的理由。但也没有必要滥用自增ID，给每个表都设置一个自增ID做主键，有时可能存在另一个从逻辑上来说更加自然的主键。

另外，因为InnoDB引擎的ID主键是聚集索引，从前文可以得知，如果簇索引、数据和主键索引放在一起且是按主键索引进行排序的，那么基于自增主键的单个值查找和小范围查找将是高效的。

研发人员有时倾向于使用字符串做主键，或者使用多个列的联合主键，但需要清楚一个事实：InnoDB的其他索引实际上存储了主键的值，这样做可能会导致索引空间大大增加。

InnoDB选择主键创建簇索引。如果没有主键，就会选取一个唯一非空的索引来替代；如果仍然找不到合适的列，那么将创建一个隐含的主键来创建簇索引。选取一个唯一非空的索引做主键可能不是我们所期待的，一般的解决办法是删除我们不期望的主键（唯一索引），创建一个非空的自增列，再增加这个唯一索引。

例如，由于未定义主键，InnoDB自动把唯一索引`idx_a_b(a,b)`定义为主键了。我们想增加一个自增ID主键，并设置唯一索引`idx_a_b`。`idx_a_b`表示这个索引是建立在a列和b列的复合索引。

```
ALTER TABLE table_name
ADD COLUMN 'id' bigint UNSIGNED NOT NULL AUTO_INCREMENT first,
DROP PRIMARY KEY,
ADD PRIMARY KEY('id') ,
ADD INDEX idx_a_b on table_name(a,b);
```

2.自增ID可以插入指定的值

自增ID还有一个特性，那就是如果插入0值或NULL值，InnoDB会认为没有设定值，然后帮你自增一个值。所以可以利用这个特性生成全局唯一ID、序列。如果数据分片到许多实例、机器上，那么就需要一个全局唯一ID来标识记录了。如下是官方文档推荐的一个创建唯一序列的方法。

创建一个表，用来控制顺序计数器并使其初始化。

```
mysql> CREATE TABLE sequence (id INT NOT NULL);
mysql> INSERT INTO sequence VALUES (0);
```

使用该表产生如下的序列数。

```
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
mysql> SELECT LAST_INSERT_ID();
```

高并发下，`LAST_INSERT_ID`函数可能会有一定的性能问题，但这种方法很简单，一般情况下是可以满足需要的。

3.7 字符集和国际化支持

3.7.1 什么是字符集

字符集（character set）是一套符号和编码。校对规则（collation）是在字符集中用于比较字符的一套规则，即字符集的排序规则。

假设我们有一个字母表使用了4个字母：'A'、'B'、'a'、'b'。现在为每个字母赋予一个数值：'A'=0，'B'=1，'a'=2，'b'=3，字母'A'是一个符号，数字0是'A'的编码，那么这4个字母和它们的编码组合在一起就是一个字符集。我们可以认为字符集是字符的二进制的编码方式，即二进制编码到一套符号的映射。

对于字符集，MySQL能够做如下这些事情。

- 使用多种字符集来存储字符串。
- 使用多种校对规则来比较字符串。
- 在同一台服务器、同一个数据库甚至在同一个表中，使用不同的字符集或校对规则来混合字符串。
- 允许定义任何级别的字符集和校对规则。

可使用SHOW CHARACTER SET语句列出可用的字符集。

```
mysql>SHOW CHARACTER SET;
```

可使用SHOW COLLATION语句列出utf8字符集的校对规则。

```
mysql>SHOW COLLATION LIKE 'utf8%';
```

3.7.2 国际化支持

因为现存编码不能在多语言电脑环境中使用，而且字符数有局限。所以诞生了Unicode（统一码、万国码、国际码、单一码）。Unicode是计算机科学领域里的一项业界标准。它对世界上大部分的文字系统进行了整理、编码，使得电脑可以用更为简单的方式来呈现和处理文字。

一个字符的Unicode编码是确定的，但Unicode的实现方式不同于编码方式。在实际传输过程中，由于不同系统平台的设计不一定都是一致的，且出于节省空间的目的，对Unicode编码的实现方式也有所不同。Unicode的实现方式称为Unicode转换格式（Unicode Transformation Format, UTF）。这其中有一种UTF-8编码，它是一种变长编码，MySQL中经常使用的utf8字符集就是UTF-8编码。UTF-8编码的思想是不同的Unicode字符采用变长字节序列编码：基本拉丁字母、数字和标点符号使用一个字节。大多数的欧洲和中东手写字母适合两个字节序列。韩语、中文和日本象形文字使用三个字节序列。

utf8是MySQL存储Unicode数据的一种可选方法，MySQL还有其他的存储Unicode数据的字符集，这里就不做额外介绍了。



注意 utf8字符集的最大长度是3个字节（中文3个字节，对于英文数字仍然使用一个字节），默认校对（排序）规则为utf8_general_ci（不区分大小写）。如果是CHAR类型，那么可能会导致空间浪费，因为任意字符都需要3个字节来存储。

如果是VARCHAR类型，那么英文、数字、标点符号只需要1个字符来存储即可。

对于utf8，还需要了解这样两个概念：超集、子集。

有字符集A、B。如果B支持的所有字符A都支持，那么字符集A是字符集B的超集。如果A是B的超集，那么字符集B是字符集A的子集。比如，GBK字符集是GB2312字符集的超集，它们又都是ASCII字符集的超集。

3.7.3 字符集设置

字符集设置可以分为两类：一类是创建对象的默认值；另一类是控制server端和client端交互通信的配置。

1. 创建对象的默认值

字符集和校对规则有4个级别的默认设置：服务器级、数据库级、表级和连接级。

使用如下语句列出可用的字符集。

```
mysql> SHOW CHARACTER SET;
```

列出可用的校对规则可使用如下语句。

```
mysql> SHOW COLLATION
```

更低级别的配置会继承更高级别的配置。例如，如果创建一个数据库，不指定字符集，那么它会继承服务器级的默认字符集。

对于生产环境的升级脚本，建议在表级别指定默认的字符集，以避免歧义或继承了错的数据库默认字符集。

2. 控制server端和client端交互通信的配置

绝大部分MySQL客户端都不具备同时支持多种字符集的能力，每次都只能使用一种字符集。客户和服务器之间的字符集转换工作是由以下几个MySQL系统变量来控制的。

- `character_set_server`: MySQL Server默认字符集。
- `character_set_database`: 数据库默认字符集。
- `character_set_client`: MySQL Server假定客户端发送的查询使用的字符集。
- `character_set_connection`: MySQL Server接收客户端发布的查询后，将其转换为`character_set_connection`变量指定的字符集。
- `character_set_result`: MySQL Server把结果集和错误信息转换为`character_set_result`指定的字符集，并发送给客户端。

图3-13说明了字符集的转换过程。

图3-13来自《High Performance MySQL》一书。由图3-13可以知道，当一个客户端和数据库打交道时，客户端、连接、操作系统、数据库、输出结果都有自己的字符集设置，如果字符集不一致，那么就可能需要进行转换，一般情况下，目标字符集应确保是源字符集的超集，以确保转换正常，如果目标字符集不能容纳源字符集的编码或设置错了字符集，那么转换会导致乱码。

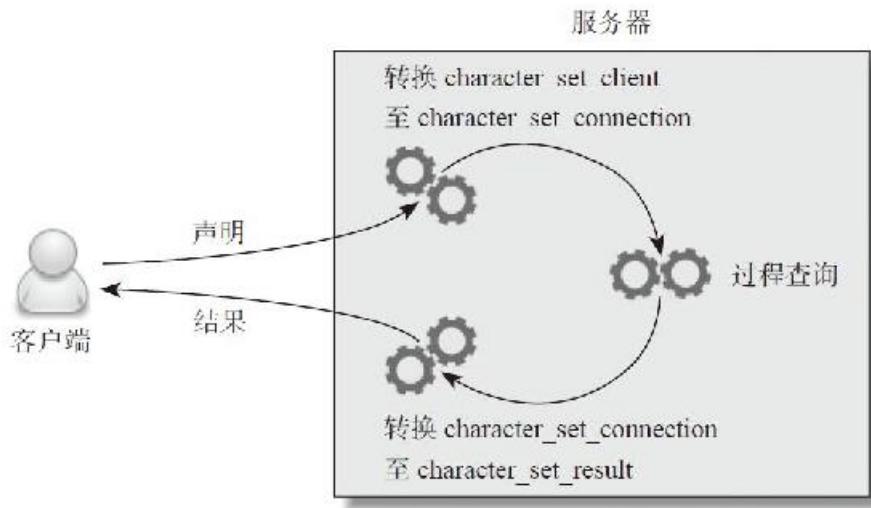


图3-13 字符集的转换过程

以下列举了一些常用的设置字符集的操作。

```
SET NAMES x
```

通过MySQL客户端导入数据时，在使用“mysql>source/path/imp_data.sql”命令的过程中有时可能会出现乱码，这时可能需要先运行SET NAMES x语句设置字符集。

SET NAMES x语句与下面这3个语句是等价的。

```
mysql> SET character_set_client = x;
mysql> SET character_set_connection = x;
mysql> SET character_set_results = x;
```

再来看看SET CHARACTER SET x语句，它等同于下面这3条语句。

```
mysql> SET character_set_client = x;
mysql> SET character_set_results = x;
mysql> SET collation_connection = @@collation_database;
```

有些客户端命令支持“--default-character-set”选项，此选项允许用户连接时设置字符集。它等同于以下这3条语句。

```
mysql> SET character_set_client = x;
mysql> SET character_set_connection = x;
mysql> SET character_set_results = x;
```

如果数据库服务器中有很多数据库使用不同的字符集，且有各种不同语系的客户端，很复杂，那么使用init-connect=SET NAMES binary是一种可以考虑的方式。这个指令的目的是让client与server交互的时候以as-is模式（是什么就是什么，不做任何转换）来传递。

索引可用来排序，但如果指定了用其他的非默认排序规则，那么将不能利用索引进行排序，比如在下面的语句中：

```
EXPLAIN SELECT col_1,col_2
  FROM table_name ORDER BY col_2 COLLATE utf8_bin
```

col_2字段使用的是utf8字符集，且其上有索引，默认的utf8字符集的排序规则是utf8_general_ci，而上面的案例指定的是用utf8_bin进行排序，那么EXPLAIN输出可以看到有filesort（文件排序），即没有利用到索引进行排序。

同理，如果连接两张表使用的连接列不是一样的字符集，那么也不能利用索引，因为必须先执行转换工作，可用EXPLAIN EXTENDED先进行确认。

默认情况下，MySQL的字符集是latin1（ISO_8859_1）。latin1字符集是单字节编码，应用于英文系列，最多能表示的字符范围是0~255（编码范围是0x00~0xFF），其中0x00~0x7F之间和ASCII码完全一致，因此它是向下兼容ASCII的。latin1字符有限，如用来存储中文、日文、韩文、希伯来文等语言时往往会导致乱码，为了避免乱码，支持国际化，个人建议是生产环境都统一使用utf8字符集，除非你有特殊理由。

以下是关于在生产环境中使用utf8字符集的一些说明和注意事项。

1) 为什么生产环境中建议使用utf8字符集？

主要是为了维护和开发都方便。大家都统一使用utf8字符集，将一劳永逸地避免各种乱码问题。一个数据库如果存在各种字符集，就会很容易出错，也会大大提高开发的难度。国际化支持也是使用utf8字符集的一个考虑。

当然，utf8字符集也有弊端，主要就是空间的消耗。比如，对于CHAR(10)，将需要用到30个字节来存放。对于VARCHAR(10)，则是按照字符串的长度来存储的，虽然不存在过多的磁盘空间消耗，但MySQL内部实现的一些数据结构，如临时表需要分配最大可能的长度，也可能导致内存大大增加。更多的空间还意味着更差的I/O性能。

有时，我们可能为了节省空间（如果空间真的是一个需要考虑的因素）而选择其他字符集（如用GBK存储汉字），对于大批量的机器，特定的服务选择特定的字符集，这种情况下所节省的空间也是很可观的，但对于一般的中小型公司，建议统一使用utf8，一劳永逸地解决乱码问题是更明智的选择。

2) 如何判断多字节字符集的字符串长度？

LENGTH()返回值为字符串的长度，单位为字节。一个多字节字符算作多字节。

CHAR_LENGTH()返回值为字符串的长度，长度的单位为字符。一个多字节字符算作一个单字符。

例如：对于一个包含了5个二字节的字符集，LENGTH()返回值为10，而CHAR_LENGTH()的返回值为5。

3) 有时创建索引的时候，可能会提示出错。

MySQL会假定每个字符有3个字节，由于索引长度有限制，那么创建索引的时候，可能会提示下面这样的错误。

```
ERROR 1071 (42000): Specified key was too long; max key length is 1000 bytes
```

这时需要明白，自己创建的是多字节字符集，字节数实际上超过1000了。

4) UTF-8是可变长度的编码，使用1到4个字节来存储。但MySQL 5.1及以前的版本，对UTF-8的支持并不彻底，它的utf8只是3字节字符集，有些文字符号是不能存储的，如emoji表情。

MySQL5.5增加了字符集utf8mb4（4-Byte UTF-8 Unicode Encoding），可以存储一些MySQL utf8不能存储的字符，需要留意的是，设置了utf8mb4字符集后，需要重启MySQL Server才能生效。



小结 本章介绍了一般开发过程中需要了解的数据库知识。理解软件架构的一些概念和数据建模有助于更全面地构建自己的知识体系，从而为研发中大型项目打下基础。SQL是绝大多数IT人员甚至也是部分非IT专业人员的必备技能，在项目实战的过程中，你的SQL技能会越来越熟练。研发是一个很庞大的体系，我们选取PHP数据库开发讲述了一些数据库操作的

知识，希望对其他开发领域也有借鉴作用。本章还讲述了索引、主键、字符集等知识，这些内容是开发过程中最普遍使用的知识。

第4章 开发进阶

本章将介绍一些重中之重的数据库开发知识。在数据库表设计中，范式设计是非常重要的基础理论，因此本章把它放在最前面进行讲解，而这其中又会涉及另一个重要的概念——反范式设计。接下来会讲述MySQL的权限机制及如何固化安全。然后介绍慢查询日志及性能管理的部分理念，并讲述数据库的逻辑设计、物理设计、导入导出数据、事务、锁等知识。最后会提及MySQL的一些非核心特性，并对于这些特性的使用给出一些建议。

4.1 范式和反范式

4.1.1 范式

什么是范式？

范式是数据库规范化的一个手段，是数据库设计中的一系列原理和技术，用于减少数据库中的数据冗余，并增进数据的一致性。

数据规范化通常是将大表分成较小的表，并且定义它们之间的关系。这样做的目的是为了避免冗余存放数据，并确保数据的一致性。添加、删除和修改数据等操作可能需要修改多个表，但只需要修改一个地方即可保证所有表中相关数据的一致性（由于数据没有冗余存放，修改某部分数据一般只需要修改一个表即可）。由于数据分布在多个表之间，因此检索信息可能需要根据表之间的关系联合查询多个表。数据规范化的实质是简单写、复杂读。写入操作比较简单，对于不同的信息，分别修改不同的表即可；而读取数据则相对复杂，检索数据的时候，可能需要编写复杂的SQL来联合查询多个表。

常用的范式有第一、第二、第三范式，通常来说，如果数据库表满足某一个层级的范式，那么它也满足前面所有层级的范式，比如，第三范式肯定满足第一、第二范式。如果一个关系数据库表的设计满足第三范式，通常可认为它是“范式化”的。

那么，这三类范式又分别代表什么含义呢？以下将进行更进一步的阐释。

1. 第一范式

第一范式是指数据库表的每一列（属性）都是不可分割的基本数据项，这就要求数据库的每一列都只能存放单一值，即实体中的某个属性不能有多个值或不能有重复的属性。第一范式（1NF）是对关系模式的基本要求。

图4-1是不满足第一范式的一个例子：“credit_card_transactions（客户信用卡交易）”。

Customer	Transactions		
	Tr. ID	Date	Amount
Jones	12890	14-Oct-2003	-87
	12904	15-Oct-2003	-50
Wilkinson	12898	14-Oct-2003	-21
	12907	15-Oct-2003	-18
Stevens	14920	20-Nov-2003	-70
	15003	27-Nov-2003	-60

图4-1 credit_card_transactions

每个用户（Customer）对应多个交易（Transactions），但这些交易记录被封装在一个复杂的属性Transactions内，这个属性包含多个时间的交易记录，其中Tr.ID列存储的是交易事务ID，Date列的是存储交易时间，Amount列存储的是交易金额，如果要查询某用户某个时间的交易记录，还要解析这个结构，才能找到对应的信息。这样的数据很难在关系型数据库内存储和检索。下面来看下满足第一范式的等价例子，如表4-1所示。

表4-1 满足第一范式的credit_card_transactions表

Customer	Tr. ID	Date	Amount	Customer	Tr. ID	Date	Amount
Jones	12890	14-Oct-03	87	Stevens	12907	15-Oct-03	18
Jones	12904	15-Oct-03	-50	Stevens	14920	20-Nov-03	-70
Wilkins	12898	14-Oct-03	-21	Stevens	15003	27-Nov-03	-60

现在，每一行数据代表一笔单独的信用卡交易记录，这样就可以很方便地进行检索和统计了。

再看表4-2所示的例子。Customer表存放了客户的信息，包括Customer ID（客户ID），First Name（名），Surname（姓），Telephone Number（电话号码）。

表4-2 Customer（客户表）

Customer ID	First Name	Surname	Telephone Number
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659
			555-776-4100
789	Maria	Fernandez	555-808-9633

存储电话记录的列“Telephone Number”里包含了多个值，违反了第一范式。

以下是符合第一范式的设计。

将Customer表分解为两个表：Customer Name（见表4-3）和Customer Telephone Number（见表4-4）。

表4-3 Customer Name（客户姓名）

Customer ID	First Name	Surname	Customer ID	First Name	Surname
123	Robert	Ingram	789	Maria	Fernandez
456	Jane	Wright			

表4-4 Customer Telephone Number（客户电话号码）

Customer ID	Telephone Number	Customer ID	Telephone Number
123	555-861-2025	456	555-776-4100
456	555-403-1659	789	555-808-9633

这样Telephone Number列就不存在多个值了。当然，这样的设计在现实中很少用到。

也可以把多个电话存储为以逗号分隔的字符串，见表4-5。

表4-5 Customer（客户表）

Customer ID	First Name	Surname	Telephone Number
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659, 555-776-4100
789	Maria	Fernandez	555-808-9633

以上的设计，属于一种常用的反范式设计，使用分隔符存储多个值，一般来说，如果应用程序只需要存储和使用，而不需要对单独的项进行修改或检索的话，那就可以存储为以上的形式。

2.第二范式

一个数据表符合第二范式的前提是该数据表符合第一范式。它的规则是要求数据表里的所有数据都要和该数据表的主键有完全相依的关系；如果有哪些数据只和主键的一部分有关的话，就得把它们独立出来变成另一个数据表。如果一个数据表的主键只有单一一个字段的话，那么它就一定符合第二范式。

来看下面的例子，表4-6给出了Employees'Skills（雇员技能）信息。

表4-6 Employees'Skills（雇员技能表）

Employee	Skill	Current Work Location	Employee	Skill	Current Work Location
Brown	Light Cleaning	73 Industrial Way	Jones	Shorthand	114 Main Street
Brown	Typing	73 Industrial Way	Jones	Typing	114 Main Street
Harrison	Light Cleaning	73 Industrial Way	Jones	Whittling	114 Main Street

表4-6的主键是（Employee,Skill），由于当前工作地点（Current Work Location）只取决于主键的部分列（取决于Employee列），显然，它不满足第二范式，Current Work Location列的数据存在重复，且更新数据的时候也可能会忘记更改所有Current Work Location列的信息。要满足第二范式，需要把依赖Employee列的信息独立出来放到另外的表中，见表4-7和表4-8。

表4-7 Employees（雇员表）

Employee	Current Work Location	Employee	Current Work Location
Brown	73 Industrial Way	Jones	114 Main Street
Harrison	73 Industrial Way		

表4-8 Employees'Skills（雇员技能表）

Employee	Skill	Employee	Skill
Brown	Light Cleaning	Jones	Shorthand
Brown	Typing	Jones	Typing
Harrison	Light Cleaning	Jones	Whittling

3.第三范式

第三范式的所有非键属性都只和候选键有相关性，也就是说所有非键属性互相之间应该是无关的。候选键指的是能够唯一标识一笔记录的属性的最小集合，一般我们所说的候选键指的就是主键。

下面是一个关于锦标赛冠军的例子，见表4-9。

表4-9 Tournament Winners（锦标赛冠军表）

Tournament	Year	Winner	Winner Date of Birth
Indiana Invitational	1998	Al Fredrickson	21-Jul-75
Cleveland Open	1999	Bob Alberston	28-Sep-68
Des Moines Masters	1999	Al Fredrickson	21-Jul-75
Indiana Invitational	1999	Chip Masterson	14-Mar-??

上表Tournament列存储锦标赛名称，Year列存储举办日期，Winner列存储冠军姓名。主键是（Tournament,Year），由于冠军

的生日（Winner Date of Birth）是固定的，Winner列可以决定Winner Date of Birth列的信息，Winner和Winner Date of Birth这两个属性都是非键属性，显然违反了第三范式。满足第三范式的表格形式应如表4-10和表4-11所示，这里是将表4-9分解为了两个表。

表4-10 Tournament Winners（锦标赛冠军表）

Tournament	Year	Winner	Tournament	Year	Winner
Indiana Invitational	1998	Al Fredrickson	Des Moines Masters	1999	Al Fredrickson
Cleveland Open	1999	Bob Albertson	Indiana Invitational	1999	Chip Masterson

表4-11 Player Dates of Birth（冠军生日表）

Player	Date of Birth	Player	Date of Birth
Chip Masterson	14 Mar 77	Bob Albertson	28 Sep 68
Al Fredrickson	21-Jul-75		

一般数据库表的设计满足第三范式即可，一些其他的范式，如第四范式、第五范式、DK范式、第六范式，因为使用得很少，这里就不做介绍了。

范式的好处是：使编程相对简单，数据量更小，更适合放入内存，更新更快，只需更新更少的数据。更少的冗余数据意味着更少地需要GROUP、DISTINCT之类的操作。不利之处是查询会变得更加复杂，查询时需要更多连接（JOIN）操作，一些可以复合索引的列由于范式化的需要被分布到了不同的表中，导致索引策略不佳。

4.1.2 反范式

反范式是试图通过增加冗余数据或通过分组数据来优化数据库读取性能的过程。在某些情况下，反范式是解决数据库性能和可伸缩性的极佳策略。

范式化的设计是在不同的有关系的表中存储不同的信息，如果需要查询信息往往需要连接多个表，如果连接的表很多，将会导致很多随机I/O，那么查询可能会非常慢。一般有两种解决方案，一种做法是仍然保持范式化的表设计，但在数据库存储冗余信息来优化查询响应，由数据库来确保冗余副本数据的一致性。例如Oracle的物化视图技术，SQL Server的Indexed View技术。另一种做法是反范式的数据表设计。由于多了冗余数据，因此数据的一致性需要靠数据库约束或应用程序来保证。传统商业数据库一般通过施加数据库约束来确保数据的一致性，而互联网数据库一般靠应用程序来确保数据的一致性。

反范式的好处是减少了连接，因此可以更好地利用索引进行筛选和排序，对于一些查询操作可以提高性能。但也需要清楚一个事实，那就是冗余数据意味着更多的写入，如果冗余的数据量很大，还可能会碰到I/O瓶颈，这会导致性能变得更差，所以需要事先衡量对各个表的更新量和查询量，评估对其他查询的影响，避免引发性能问题。冗余数据也意味着可能要牺牲部分数据的一致性，我们有必要区分不同数据的一致性的优先级，对于重要的、用户比较敏感的数据一定要注意一致性的问题，以免影响用户的体验。

随着开发经验的日渐丰富，做研发的读者通常都会逐渐熟悉范式，创建一个合格的满足第三范式的数据库设计并不会太难；而对于反范式设计，由于不熟悉硬件性能和数据库机制，可能考虑就不是那么周全了。

反范式设计在统计分析、数据仓库等领域使用的比较多，通过冗余数据，增加各种统计表、中间表，数据可以更快地被加载和分析。

以下是一些反范式的例子，一般的方法是冗余或缓存某个表的一些列到另一个表或缓存中。

1) 论坛的消息表forum_message包括如下字段：msg_id、from_uid、to_uid、subject、message、post_time。由于表中只存储了会员id的信息，因此如果要显示发送给某个用户（以to_uid标识）的完整消息，还需要用from_uid去连接会员表，获取会员的姓

名，在高并发的情况下，这样做可能会带来性能问题，常用的解决方案是增加1个冗余字段`from_uname`以避免JOIN。

2) 反范式可以更好地利用索引进行筛选和排序，如上一个例子1) 中，如果需要按照`uname`对消息进行排序，则需要连接会员表，然后按照`uname`进行排序，这样的代价比较高，增加冗余列`uname`，并在其上创建索引，就可以利用索引排序很快地返回结果。对于多个列的筛选排序也可以采用类似的优化思路，例如：

```
select table_a.* from table_a join table_b on table_a.id=table_b.id order by table_a.col_1,table_b.col_2;
```

或者：

```
select table_a.* from table_a join table_b on table_a.id=table_b.id where table_a.col_1=100 order table_b.col_2;
```

这样的SQL语句，由于排序的列不能利用到索引，因此需要创建临时表进行排序，成本比较高。可以考虑在`table_a`表中冗余`col_2`列，并且建立复合索引（`col_1, col_2`）。这样不仅可以不用JOIN两张表，而且还可以利用索引进行排序。

3) 一些程序，需要发送系统信息、推荐信息给用户。一种解决方案是在后台维护一张信息表`a_message`，发布新消息的时候，给`t_message`表的每个用户插入新的消息`id`。用户登录后检查消息表`t_message`是否有新的未读消息，为了节省空间，`t_message`只有信息`id`而没有内容，这在大量用户登录的情况下可能会导致性能问题，因为每次查询都需要JOIN另一个表`a_message`以获取内容。为了以更快的速度加载数据，可以在发布信息的时候把信息内容一并插入到`t_message`表中。

4) 冗余数据也可以放在缓存中，比如表`a`，如果用户名已经缓存在某个缓存产品中了，如`memcached`，那么就可以直接从缓存中获取，而不需要再去JOIN数据库获取用户名了。同样的，对于表`c`，也可以考虑把消息内容放到缓存里。

5) 一些统计操作，比如COUNT、SUM、MAX、MIN等操作，如果计算耗费的资源比较多，可以考虑增加冗余的统计信息，或者增加额外的字段，或者增加额外的表，比如论坛的发帖统计，用户在线人数等。

例如，对于发帖统计，需要统计最近24小时的发帖数量，我们可以每个小时插入一条统计数据到统计表，这样就可以统计最近24小时的数据了，虽然这样做不够准确，但用户一般不会介意这种数据的不准确性。如果需要更精确的统计，可以在前23个小时使用统计值，最近1个小时使用实际值即可。

6) 如果某个用户表的字段比较多（如`uid`、`uname`、`upass`、`email`、`address`、`qq`、`msn`……），数据量很大，超过亿级别，那么为了方便扩展，我们将把数据分片到多个表中，例如，我们对`uid`这个整型字段进行求模运算，把求模运算结果一样的`uid`存放同一张分表中。这时，用户需要以`uname`登录，查询`uid`，以便到分表中去查询数据。因此可以增加一个冗余表，只存储`uname`和`uid`的映射关系。由于这个冗余表仅有两个列，因此虽然数据量很大，但完全可以放在一张表内，也方便加载到内存中进行访问。



提示 要求数据库中的所有表都满足范式是不太现实的，一般生产库中会混合使用范式和反范式。很多应用程序的数据设计，起初都是偏向范式化的，这样做编码会简单方便，但随着流量上涨，数据量增加，往往会碰到一些性能问题，此时就要考虑一些反范式设计。当然，大致估测到以后的流量、数据量，并能够预先考虑反范式设计会更好。建议开发人员首先创建一个完全规范化的设计，然后为了性能原因选择性地对一些表进行反范式化设计。我们要牢记一个准则，设计的数据库应该按照用户可能的访问路径、访问习惯进行设计，而不是严格地按照数据范式来设计。

4.2 权限机制和安全

4.2.1 MySQL访问权限系统

1 概述

MySQL权限系统的主要功能是证实连接到一台给定主机的用户，并且赋予该用户在数据库上的各种权限，一般生产环境中的程序账号只需要SELECT、INSERT、UPDATE和DELETE权限即可。

MySQL根据访问控制列表（ACL）对所有连接、查询和用户尝试执行的其他操作进行安全管理。MySQL将验证用户的3项信息：用户名、密码、主机来源。对通过验证的用户再确认其他的访问权限，以进行访问控制。

权限可以分为两类：系统权限和对象选项。

系统权限允许执行一些特定的功能，如关闭数据库、终止进程、显示数据库列表、查看当前执行的查询等。对象权限是指对一些特殊的对象（表、列、视图、数据库）的访问权限，例如是否允许访问某张表，是否允许在某个库中创建表。

一般不允许直接更改MySQL的权限表，而是通过GRANT和REVOKE语句进行权限的赋予和收回，这也是更安全可靠的办法。

GRANT和REVOKE语句允许系统管理员创建MySQL用户账户、授予权限和撤销权限。授予的权限可以分为多个级别：服务器级别（全局）、数据库级别、表级别、列级别、子程序级别。撤销权限即回收已经存在的权限。

GRANT和REVOKE的基本语法如下所示。

```
GRANT [privileges] ON [objects] TO [user]
GRANT [privileges] ON [objects] TO [user] IDENTIFIED BY [password]
REVOKE [privileges] ON [objects] FROM [user]
```

MySQL为有SUPER权限的用户专门保留了一个额外的连接，因此即使是所有的普通连接都被占用，MySQL root用户仍可以登录并检查服务器的活动。

如果想要限制单个账户允许的连接数量，可以通过设置max_user_connections变量来完成。

MySQL允许对不存在的数据库目标授予权限。这个特性是特意设计的，目的是允许数据库管理员为将在此后被创建的数据库目标预留用户账户和权限。

当在GRANT语句中指定数据库名称时，允许使用“_”和“%”通配符。这意味着，如果想要使用“_”字符作为一个数据库名称的一部分，则应该在GRANT语句中指定它为“_”，例如，“GRANT...ON‘foo_bar’.*TO...。”



注意 SHOW TABLES命令不会显示用户没有权限访问的表。

MySQL的存储过程、触发器、视图都可以提供某种程度的安全性，但不建议采用以上特性来实现安全性。除了尽量给予最小的权限外，建议不要给予过细的权限，MySQL的权限精细控制并不完善，可能会导致维护上的成本增加。



注意 不要重复利用原来的用户名。

不要采取给相同的用户名（但来自于不同的主机）赋予不同权限的方式。这样很容易造成混淆，导致维护的困难，可以另外创建单独的账号。

2. 权限更改何时生效

当mysqld启动时，所有授权表的内容将被读进内存并且从此时开始生效。当服务器注意到授权表被改变了时，现存的客户端连接将会受到如下影响。

- 表和列的权限在客户端的下一次请求时生效。
- 数据库的权限改变在下一个USE db_name命令生效。
- 全局权限的改变和密码改变在下一次客户端连接时生效。

如果使用GRANT、REVOKE或SET PASSWORD命令对授权表进行修改，那么服务器会注意到更改并立即将授权表重新载入内存。

如果手动地修改授权表（使用INSERT、UPDATE或DELETE等），则应该执行mysqladmin flush-privileges或mysqladmin reload告诉服务器再重新装载授权表，否则手动的更改将不会生效，除非重启服务器。

3. 常用的权限

SHOW PRIVILEGES命令可以显示MySQL所支持的权限，如下是一些常用的权限。

- SELECT、INSERT、UPDATE和DELETE权限允许用户在一个数据库现有的表上实施读取、插入、更新和删除记录的操作。这也是一般程序账号所需要的权限。
- SHOW VIEW权限允许用户查看已经创建了的视图。
- ALTER权限允许用户使用ALTER TABLE命令来修改现有数据表的结构。
- CREATE和DROP权限允许用户创建新的数据库和表，或者删除现存的数据库和表。生产环境中一般不赋予程序账号DROP的权限。
- GRANT权限允许用户把自己拥有的权限授予其他的用户。
- FILE权限允许被授予该权限的用户都能读或写MySQL服务器能读写的任何文件。
- SHUTDOWN权限允许用户使用SHUTDOWN命令关掉服务器。可以创建一个用户专门用来关闭服务器。
- PROCESS权限允许用户使用PROCESSLIST命令显示在服务器内执行的进程的信息；使用KILL命令终止服务器进程。用户总是能显示或终止自己的进程，但是，显示或终止其他用户启动的进程则需要PROCESS权限。一些监控工具需要PROCESS权限查看正在执行的命令。

4. 示例

(1) 查看和赋予权限

查看数据库的用户、密码、主机字符串的命令如下。

```
mysql > SELECT user,host,password FROM user;
```

显示某个用户的权限的命令如下。

```
SHOW GRANTS FOR username@'ip_range';
```

赋予某个用户对库db1进行SELECT、INSERT、UPDATE和DELETE的权限的命令如下。

```
GRANT SELECT, INSERT, UPDATE, DELETE ON db1.* TO username@'10.%' IDENTIFIED BY 'your_password';
```

(2) 赋予备份用户权限

赋予备份整个实例权限的命令如下。

```
GRANT LOCK TABLES,RELOAD,SUPER,SELECT,SHOW VIEW,TRIGGER,PROCESS ON *.* TO backup@localhost IDENTIFIED BY 'xxxxxx';
```

赋予远程备份各个库权限的命令如下。

```
GRANT LOCK TABLES,SELECT,RELOAD,SHOW VIEW,trigger ON *.* TO backup@'10.10.10.10' IDENTIFIED BY 'xxxxxx';
```

还有一些常用的权限，如配置复制时，复制用户需要REPLICATION SLAVE权限，查看复制状态需要REPLICATION CLIENT权限。大家可以阅读官方文档来进一步了解具体的权限。

(3) 修改用户密码

可以使用SET PASSWORD命令更改密码。

```
mysql>SET PASSWORD FOR user@'ip_range' = PASSWORD('some password');
```

或者使用GRANT命令重新赋予用户连接密码。

```
mysql>GRANT USAGE ON *.* TO user@'ip_range' IDENTIFIED BY 'some password';
```

或者可以使用如下命令直接修改系统表。

```
shell> mysql -u root  
mysql> UPDATE mysql.user SET PASSWORD=PASSWORD('newpwd')  
      WHERE user='root';  
mysql> FLUSH PRIVILEGES;
```

(4) 强化安全

安装完MySQL之后，一定要移除匿名账号和空密码账号。具体操作请参考2.2节。



注意 MySQL用户主机字符串通配符“%”不包括“localhost”。“localhost”和IP地址“127.0.0.1”并不等同，如果使用“mysql-uroot-h localhost”，则默认会去连接socket文件。如果我们要连接TCP端口，正确的写法应该是“mysql-uroot-h 127.0.0.1”。

4.2.2 强化安全

本节将描述一些常见的需要注意的安全问题，以及一些可以使MySQL安装更加安全的、防止黑客和误操作的措施。

强化安全的目的有如下三点。

- 保护好MySQL主机的安全，同时也需要关注其他能访问数据库的主机的安全。
- 确保MySQL自身的安全，包括生产库和备份，应使用强密码，尽可能分配最小的权限给用户。
- 确保网络、物理的安全，同时也需要关注信息内容的保密。

下面是一些安全的指导原则和注意事项。

- 加强安全意识。比如加密办公电脑、个人笔记本上的重要数据，不要将未加密的数据上传到各种公共云存储中。在不安全的网络环境下，比如一些公共Wi-Fi中，涉及账号的操作可能会泄露你的信息。
- 一般将所有数据库都部署于内网（仅监听内网IP），需要慎重对待跨IDC的数据库同步，MySQL自身并没有很好的方式加密数据传输。
- 开放外网访问的MySQL服务器，需要有相应的访问控制策略，例如通过部署防火墙来限制来源IP。
- 如果条件允许，应该增加网络安全团队进行安全检查和审计。
- 在不安全的网络环境中访问公司或远程维护机器，建议使用VPN。
- 不要让任何人（除了MySQL root账户）访问MySQL数据库中的mysql系统库！
- 用GRANT和REVOKE语句来控制对MySQL的访问。不要授予超过需求的权限。绝对不能为所有主机授权。
- 不要给程序账号授予SUPER权限。
- 生产库上不要留研发人员的账号。
- 隔离生产环境、开发环境和测试环境，不允许研发、测试人员有权限更改生产环境或知道生产环境的账号密码。
- 初始安装后应该移除匿名和空密码账号，可以尝试用‘mysql -u root’，如果你能够成功连接服务器而没有要求/输入任何密码，则说明有问题。
- 不要将纯文本密码保存到数据库中，不要从字典中选择密码，如果你的程序是一个客户端，必须用可读的方式存储密码，那么建议使用可解码的加密办法来存储。一些工具，如telnet、ftp，使用的是明文传输密码，建议不要使用，使用ssh、sftp是更安全的方式。
- 使用更安全的算法加密密码，一些流行算法，如MD5已经被证明是弱加密，不适合用于加密密码。曾经比较流行的散列算法SHA-1也被证明不够安全。推荐的方式是在将密码传入散列函数进行加密之前，将其和一个无意义的字符串拼接在一起，这样即使用户选择了一个在字典中存在的单词作为密码，攻击者也很难使用字典攻击的手段破解密码。
- 试试从Internet上使用工具扫描端口，或者使用shell命令shell>telnet server_host 3306，如果得到连接并得到一些垃圾字符，则端口是打开着的，这种情况应从防火墙或路由器上关闭端口，除非你有足够的理由让它开着。
- 避免SQL注入，不要信任应用程序的用户输入的任何数据。
- 有时候人们会认为如果数据库只包含供公共使用的数据，则不需要保护。这是不正确的。即使允许显示数据库中的任何

记录，也仍然应该保护和防范、拒绝服务攻击。

- 不要向非管理用户授予FILE权限。拥有FILE权限的任何用户都能在拥有mysqld守护进程权限的文件系统里写入一个文件！
- FILE权限也可以被用来读取任何作为运行服务器的Unix用户可读取或访问的文件。使用该权限，可以将任何文件读入数据库表。这可能会被滥用，例如，通过使用LOAD DATA装载“/etc/passwd”进入一个数据库表，然后就能用SELECT显示它。

也可以考虑加密传输HTTPS和SSH tunnel等方案，这些措施将会更安全，但成本比较高，实施起来往往会受制于其他因素。相对来说，从应用层做一些安全措施、在硬件防火墙中设置规则及MySQL权限控制则是更经济、更标准化的做法，总之，需要在安全和方便上达到一个平衡。

研发人员、测试人员也有必要熟悉目前常用的一些攻击手段的原理和预防，如会话(session)劫持、中间人攻击、SQL注入、跨站脚本攻击等。

1.会话劫持

由于HTTP是无状态的，客户端到服务器端并不需要维持一个连接，因此需要有一种关联的手段，基于此，服务器会给新的会话一个标识信息：cookie。在PHP环境中，cookie默认是存储在/tmp下的。生成的用以标识客户信息的cookie一般被称为session id，用户发出请求时，所发送的HTTP请求header内包含了session id的值，可用firebug查看这个值。服务器使用session id来识别是哪个用户提交的请求。session保存的是每个用户的个人数据，一般的Web应用程序会使用session来保存通过验证的用户账号和密码。在转换不同的网页时，如果需要验证用户的身份，就要用session内所保存的账号和密码来比较。

攻击者通过一些手段来获取其他用户session id的攻击就叫session劫持。一个典型的场景是在未加密的Wi-Fi网络中，由于session id在用户的请求内而且是不加密的（未使用HTTPS），通过嗅探工具可以获取到用户的session id，然后可以冒充用户进行各种操作。除了嗅探外，还有一些其他的手段，如跨站脚本攻击、暴力破解、计算等。

如果使用了HTTPS加密传输，那么理论上可以防止嗅探，但实际上，HTTPS在世界范围内远未普及开来，许多网站登录的时候使用了HTTPS，登录成功后仍然返回了HTTP会话，一些网站虽然支持HTTPS，但并不作为默认选项，目前已知的网站中gmail是默认全部使用了HTTPS会话的，但常用的各种社交网站、电商网站大多只是部分采用HTTPS。因为HTTPS无法实现缓存、响应变得缓慢、运营成本高、虚拟主机无法在同一台物理服务器上为多个网站提供服务、和其他不支持HTTPS应用的交互，以上种种因素都制约着HTTPS的普及。

2.中间人攻击

中间人攻击是指攻击者在通信的两端分别创建独立的连接，并交换其所收到的数据，使通信的两端认为他们正在通过一个私密的连接与对方直接对话，但事实上整个会话都被攻击者完全控制（例如，在一个未加密的Wi-Fi无线接入点的中间人攻击者，可以将自己作为一个中间人插入这个网络）。

中间人攻击能够成功的一个前提条件是攻击者能将自己伪装成每一个参与会话的终端，并且不被其他终端识破。大多数的加密协议都专门加入了一些特殊的认证方法以阻止中间人攻击。例如，SSL协议可以验证参与通信的一方或双方使用的证书是否由权威的受信任的数字证书认证机构颁发，并且能执行双向身份认证。

3.跨站脚本攻击

跨站脚本攻击（Cross Site Scripting）是指攻击者利用网站程序对用户输入过滤不足，输入可以显示在页面上对其他用户造成影响的HTML代码，从而盗取用户资料、利用用户身份进行某种动作，或者对访问者进行病毒侵害的一种攻击方式。针对这

种攻击，主要应做好输入数据的验证，对输出数据进行适当的编码，以防止任何已成功注入的脚本在浏览器端运行。

以下将详细介绍SQL注入攻击的原理和预防，这也是DBA需要重点考虑的。

4.2.3 SQL注入

SQL注入（SQL Injection）攻击是发生在应用程序中的数据库层的安全漏洞。简而言之，是在输入的字符串之中注入SQL语句，如果在设计不良的程序中忽略了检查，那么这些注入进去的SQL语句就会被数据库服务器误认为是正常的SQL语句而运行，攻击者就可以执行计划外的命令或访问未被授权的数据。SQL注入已经成为互联网世界Web应用程序的最大风险。我们有必要从开发、测试、上线各个环节对其进行防范。以下将介绍SQL注入的原理及如何预防SQL注入。

SQL注入的原理有如下4点

1) 拼接恶意查询。SQL命令可查询、插入、更新、删除数据，以分号字符分隔不同的命令。

例如：

```
select * from users where user_id = $user_id
```

user_id是传入的参数，如果传入了“1234;delete from users”，那么最终的查询语句会变为：

```
select * from users where user_id = 1234; delete from users
```

如上语句如果执行，则会删除user表的数据。

2) 利用注释执行非法命令。SQL命令中，可以插入注释。

例如：

```
select count(*) as 'num' from game_score where game_id=24411 and platform_id=11 and version=$version and session_id = sessid='d7a157-0f-48b6-98-c35592'
```

如果version包含了恶意的字符串“-1'OR 3 AND SLEEP(500)--”，那么最终查询语句会变成下面这个样子：

```
select count(*) as 'num' from game_score where game_id=24411 and platform_id=11 and version='-' OR 3 AND SLEEP(500)-- ' and session_id = sessid='d7a157-0f-48b6-98-c35592'
```

以上恶意查询只是想耗尽系统资源，SLEEP(500)将导致SQL一直运行，如果添加了修改、删除数据的恶意指令，将会造成更大的破坏。

3) SQL命令对于传入的字符串参数是用单引号引起来的。如果字符串本身包含单引号而没有被处理，则可能会篡改原本的SQL语法的作用。

例如：

```
select * from user_name where user_name = $user_name
```

如果user_name传入的是G'chen，那么最终的查询语句会变成这样：

```
select * from user_name where user_name ='G'chen'
```

以上语句将会出错，这样的语句风险比较小，因为语法错误的SQL语句不会被执行。但也可能恶意产生的SQL语句，没有任何语法错误，并且以一种你不期望的方式运行。

4) 添加一些额外的条件为真值表达式，改变执行行为。

例如：

```
update users set userpass=SHA2('{$userpass}') where user_id={$user_id};
```

如果user_id被传入恶意的字符串“1234 OR TRUE”，最终的SQL语句会变成下面这样：

```
update users set userpass=SHA2('123456') where user_id=1234 OR TRUE;
```

这将更改所有用户的密码。

下面是避免SQL注入的一些方法。

(1) 过滤输入内容，校验字符串

应该在将数据提交到数据库之前，就把用户输入中的不合法字符剔除掉。可以使用编程语言提供的处理函数，如PHP的mysql_real_escape_string()来剔除，或者定义自己的处理函数进行过滤，还可以使用正则表达式匹配安全的字符串。

如果值属于特定的类型或有约定的格式，那么在拼接SQL语句之前就要进行校验，验证其的有效性。比如对于某个传入的值，如果可以确定是整型，那么我们要判断下它是否为整型，不仅在浏览器端（客户端），而且在服务器端也需要进行验证。

(2) 参数化查询

参数化查询目前已被视作是最有效的预防SQL注入攻击的方法。不同于在SQL语句中插入动态内容，查询参数的做法是在准备查询语句的时候，就在对应参数的地方使用参数占位符。然后，在执行这个预先准备好的查询时提供一个参数。

在使用参数化查询的情况下，数据库服务器不会将参数的内容视为SQL指令的一部分来进行处理，而是在数据库完成SQL指令的编译之后，才套用参数运行，因此就算参数中含有破坏性的指令，也不会被数据库所运行。

可以使用MySQLi扩展或pdo扩展来绑定参数实现参数化查询。

如下是一个使用MySQLi扩展绑定参数的示例。

```
<html>
<head>
<title> test parameter query </title>
</head>
<body>
<?php
$host="127.0.0.1";
$port=3306;
$socket="";
$user="garychen";
$password="garychen";
$dbname="employees";
$con = new mysqli($host, $user, $password, $dbname, $port, $socket)
      or die ('Could not connect to the database server' . mysqli_connect_error());
echo 'connect to employees database successfully';
echo "<br />";
echo "select departments table using parameter";
echo "<br />";
$query = "select * from departments where dept_name = ?";
if ($stmt = $con->prepare($query)) {
    $stmt->bind_param("s",$depname);
    $depname="Finance";
    $stmt->execute();
    $stmt->bind_result($field1, $field2);
    while ($stmt->fetch()) {
```

```
    printf("%s, %s\n", $field1, $field2);
    echo "<br />";
}
$stmt->close();
}
$conn->close();
?>
</body>
</html>
```

上例首先是预处理语句if(\$stmt==\$conn->prepare(\$query)), 然后绑定参数使用bind_param()方法, 该方法的语法格式如下所示。

```
bool mysqli_stmt::bind_param ( string $types , mixed &$var1 [, mixed &$... ] )
```

其中, types指定绑定参数的类型, 包含了一个或多个字符。I代表整型, D代表双精度, S代表字符串, B代表BLOB类型, 本例中是S。

但是绑定参数也有如下一些限制。

- 不能让占位符“?”代替一组值, 例如:

```
SELECT * FROM departments WHERE userid IN ( ? );
```

- 不能让占位符“?”代替数据表名或列名, 例如:

```
SELECT * FROM departments ORDER BY ?;
```

- 不能让占位符“?”代替SQL关键字, 例如:

```
SELECT * FROM departments ORDER BY dept_no ?;
```

对于Java、JSP开发的应用, 也可以使用预处理语句加绑定参数的方式来避免SQL注入。

(3) 安全测试、安全审计

除了开发规范, 还需要流程、机制和合适的工具来确保代码的安全。我们应该在开发过程中对代码进行审查, 在测试环节使用工具进行扫描, 上线后定期扫描安全漏洞。通过多个环节的检查, 一般是可以避免SQL注入的。

有些人认为存储过程可以避免SQL注入, 存储过程在传统行业里用得比较多, 对于权限的控制是有一定用处的, 但如果存储过程用到了动态查询, 拼接SQL, 那样一样会存在安全隐患。一些编程框架对于写出更安全的代码也有一定的帮助, 因为它提供了一些处理字符串的函数和使用查询参数的方法, 但同样, 你仍然可以编写出不安全的SQL语句。所以归根到底, 我们需要有良好的编码规范, 并能充分利用参数化查询、字符串处理和参数校验等多种办法来实现安全。

4.3 慢查询日志

慢查询日志可以用来定位执行时间很长的查询，它是我们常用的性能分析工具，通过在开发、测试期间关注慢查询，我们可以尽量避免引入效率很差的查询。以下将介绍慢查询日志的分析策略和常用的工具。

4.3.1 查看慢查询日志

1. 优化策略

性能优化的一个很重要的步骤是识别导致问题的BAD SQL。对于一般的数据库调优，调优人员往往会采用调优TOP 10的策略，如果我们把最“昂贵”的10个查询优化完（更高效地运行它们，例如添加一个索引），那么就会立即看到对整体MySQL的性能的提升。然后就可以重复这一过程，并优化新的前10名的查询。就笔者经验而言，一般一两轮迭代就足够了。以后随着业务发展、用户流量增加，可进行新一轮的调优。

2. 慢查询日志的格式

不同数据库TOP 10基于的标准可能不太一样，商业数据库提供了更完善的成本分析方法，MySQL的慢查询日志比较粗略，主要是基于以下3项基本的信息。

- Query_time:** 查询耗时。
- Rows_examined:** 检查了多少条记录。
- Rows_sent:** 返回了多少行记录（结果集）。

以上3个值可以大致衡量一条查询的成本。

其他信息包括如下几点。

- Time:** 执行SQL的开始时间。
- Lock_time:** 等待table lock的时间，注意InnoDB的行锁等待是不会反应在这里的。
- User@Host:** 执行查询的用户和客户端IP。

以下是一个慢查询日志的例子。

```
# Time:11062210:11:16
# User@Host: rss[rss] @ [12.12.12.12]
#
Query_time:1.637992Lock_time:0.000038Rows_sent:0Rows_examined:101
SET timestamp=1308708676;
select * from rss_doc1 where feed_id=5850 order by doc_id desc limit190,10;
```

其中，**Query_time**、**Rows_examined**、**Rows_sent**这3个值可以大致衡量一条查询的成本。

如果检查了大量记录，而只返回很小的结果集，则往往意味着查询质量不佳。慢查询日志可以用来找到执行时间很长的查询，可以用于优化。但是，检查又长又慢的查询日志会很困难。要想使其变得容易些，可以使用mysqldumpslow命令获得慢查询日志摘要来处理慢查询日志，或者使用更好的第三方工具pt-query-digest。

注意，慢查询日志里的慢查询不一定就是不良SQL，还可能是受其他的查询影响，或者受系统资源限制所导致的慢查询。

比如下面的例子，会话被阻塞了，实际上是一个行锁等待50s超时，然后记录到了慢查询日志里。

```
# Query_time: 50.665866 Lock_time: 0.000102 Rows_sent: 0 Rows_examined: 0
SET timestamp=1339728734;
update tbl_rankings set status=2 where ranking=1;
```

3.如何识别需要关注的SQL

以下是识别需要关注的SQL的步骤。

第一步，确认已经开启了慢查询日志，并记录了合理的阈值。

MySQL可以把慢查询日志记录到数据表内，但更普遍的做法是记录到日志里，然后使用工具来分析。

以下的命令将查看慢查询是否启用了，以及慢查询的日志路径。

```
mysql> show variables like'%query_log%';
-----
show variables like'%query_log%'
-----
+-----+-----+
| Variable_name | Value |
+-----+-----+
| slow_query_log | ON   |
| slow_query_log_file | /path/to/log3304/slowquery.log |
+-----+-----+
2rows in set (0.00sec)
```

MySQL 5.1可以动态打开示例中提到的slow_query_log选项。

如果配置文件或启动参数没有给出file_name值，慢查询日志将默认命名为“主机名-slow.log”，如果给出了文件名，但不是绝对路径名，文件则写入数据目录。

语句执行完成并且所有锁释放后则记入慢查询日志。记录的顺序与执行顺序可以不相同。

我们可以在MySQL客户端下使用命令“SHOW VARIABLES LIKE'%query_time%'”查看全局变量long_query_time。所有执行时间超过long_query_time秒的SQL语句都会被记录到慢查询日志里。

MySQL参数long_query_time默认的2s阈值太大，可能不适用，对于一般的OLTP应用，建议将阈值设置得更小，比如200~500ms。有时手动调整了某变量的值，且需要永久变更，这时则要注意全局变量的值应和配置文件保持一致，配置文件的参数示例如下所示。

```
[mysqld]
slow_query_log=1
slow_query_log_file=/usr/local/mysql/log/slowquery.lo
long_query_time=0.5
```

其中，slow_query_log设置为1表示开启慢查询日志，设置为0则表示关闭慢查询日志。long_query_time的单位为秒，MySQL 5.1.21后可以设置毫秒级的慢查询记录，如设置long_query_time=0.01。

MySQL 5.0慢查询的参数是不一样的，且需要重启后才可以生效，相关的参数为log_slow_queries和slow_launch_time。

另外有一个参数log-queries-not-using-indexes，用于指定如果没有使用到索引或虽然使用了索引但仍然遍历了所有记录，就将其记录下来。默认此选项是关闭的。



注意 对于持久连接（长连接）、连接池这类情况，由于不能重置session会话的变量，因此即使修改了

`long_query_time`的值，也不能马上生效，这会给我们带来一些困扰，不过，使用短连接或使用Percona版本的MySQL可以解决此问题。但对于测试人员或开发人员来说，这点是很方便调整和验证的，重启应用或重连数据库即可解决此问题。

4.3.2 使用工具分析慢查询日志

从前面的内容可知，`Query_time`、`Rows_examined`、`Rows_sent`这3个信息让我们看到了查询需要优化什么。查询时间最长的SQL往往是最需要优化的，如果检查了大量记录（`Rows_examined`），而只返回很小的结果集（`Rows_sent`），往往也意味着存在不良SQL。但在一个高并发的数据库服务上，或者在做压力测试时，如果发现慢查询日志增长得非常快，很难筛选和查找里面的信息，那么在这种情况下，有如下两种选择。

- 调整阈值，先设置为较大的阈值，这样慢查询记录就很少了，等优化得差不多了，再减少阈值，不断进行优化。
- 使用命令/脚本、工具进行分析，如`mysqldumpslow`、`pt-query-digest`等。

第一种方法比较繁琐，建议大家使用第二种方法。如果优化效果比较理想，希望更进一步调优，则可以减低阈值，然后记录更多的慢查询日志，然后继续使用脚本、工具进行分析。

1. 使用操作系统命令分析

可以使用操作系统自带的命令进行一些简单的统计，如`grep`、`awk`、`wc`，但不容易实现更高级的筛选排序。

下面来看个示例，通过如下命令可以看到每秒的慢查询的统计，当检查到有突变时，往往会有异常发生，这时便可以更进一步到具体的慢查询日志里去查找可能的原因。

```
awk '/^# Time:/{print $3, $4, c;c=0}/^# User/{c++}' slowquery.log > /tmp/aaa.log
```

2. mysqldumpslow

`mysqldumpslow`命令是官方自带的，此命令可获得日志中的查询摘要。

以下是`mysqldumpslow`命令的使用示例。

访问时间最长的10个sql语句的命令如下。

```
mysqldumpslow -t10 /path/to/log3304/slowquery.log
```

访问次数最多的10个sql语句的命令如下。

```
mysqldumpslow -s c -t10 /path/to/log3304/slowquery.log
```

访问记录集最多的10个sql语句的命令如下。

```
mysqldumpslow -s r -t10 /path/to/log3304/slowquery.log
```

3.pt-query-digest

有一些第三方分析工具（如mysqlsla、pt-query-digest）比mysqldumpslow更强大，更友好。以下将重点介绍pt-query-digest工具。

pt-query-digest可以生成一份比官方mysqldumpslow可读性好得多的报告。安装也很简单，命令如下。

```
wget www.percona.com/get/pt-query-digest  
chmod u+x pt-query-digest
```

基本语法格式如下所示。

```
pt-query-digest [OPTIONS] [FILES] [DSN]
```

详细的语法介绍，请参考16.2.2节，这里仅给出一些常用的示例。

直接分析慢查询的命令如下。

```
pt-query-digest /path/of/slow.log > slow.rtf
```

分析半个小时内的慢查询的命令如下。

```
pt-query-digest --since 1800s /path/of/slow.log > slow.rtf
```

分析一段时间范围内的慢查询的命令如下。

```
pt-query-digest --since '2014-04-14 22:00:00' --until '2014-04-14 23:00:00' /path/of/slow.log > slow.rtf
```

显示所有分析的查询命令如下。

```
pt-query-digest --limit 100% /path/of/slow.log > slow.rtf
```

其中，“**--limit**”参数默认是“95%:20”，表示显示95%的最差的查询，或者20个最差的查询。

此外，也可以用这个工具来分析二进志日志，以查看我们日常的修改语句是如何分布的，首先需要把二进志日志转换为文本格式。

```
mysqlbinlog mysql-bin.012639 > /tmp/012639.log  
pt-query-digest --type binlog /tmp/012639.log
```

对于以上分析命令，同样可以加上参数筛选信息，如“**--since**”、“**--until**”。

那么，如何查看pt-query-digest报告呢？

以下是一个输出报告，为了节省篇幅，删除了部分信息。

```
# 140.9s user time, 1.4s system time, 57.93M rss, 154.03M vsz  
# Current date: Sun Feb 16 09:16:39 2011
```

解释：执行pt-query-digest工具的时间。

```
# Hostname: db1000
# Files: /usr/lcoal/mysql/data/slowquery.log
# Overall: 304.88k total, 159 unique, 0.22 QPS, 0.15x concurrency _____
```

解释：慢查询次数一共是304.88k，唯一的查询159个。

```
# Time range: 2010-12-01 00:00:01 to 2010-12-17 09:05:17
```

解释：这里记录的是发现第一条慢查询的时间到最后一条慢查询的时间。

Attribute	total	min	max	avg	95%	stddev	median
# Exec time	216112s	500ms	21s	709ms	1s	968ms	552ms
# Lock time	414s	21us	101ms	1ms	626us	7ms	84us
# Rows sent	169.69M	0	213.73k	583.60	97.36	10.75k	9.83
# Rows examine	60.26G	0	866.23k	207.25k	328.61k	70.68k	201.74k
# Query size	120.31M	35	21.07k	413.76	719.66	148.97	363.48

解释分别如下。

·**Exec time:** 执行时间。

·**Lock time:** 表锁的时间。

·**Rows sent:** 返回的结果集记录数。

·**Rows examine:** 实际扫描的记录数。

·**Query size:** 应用和数据库交互的查询文本大小。

# Profile	# Rank	Query ID	Response time	Calls	R/Call	Apdx	V/M	Item
#	#	#	#	#	#	#	#	#
#	1	0x5931CCE8168ECE59	92062.4390	42.6%	168672	0.5458	1.00	0.01 SELECT game_info game_stat
#	2	0xE8691F18411F3DC	23404.4270	10.8%	18602	1.2582	0.60	0.04 SELECT game_info game_stat game_info_2
...								
...								

解释分别如下。

·**Rank:** 所有查询日志分析完毕后，此查询的排序。

·**Query ID:** 查询的标识字符串。

·**Response time:** 总的响应时间，以及总占比。一般小于5%可以不用关注。

·**Calls:** 查询被调用执行的次数。

·**R/Call:** 每次执行的平均响应时间。

·**Apdx:** 应用程序的性能指数得分。（Apdex响应的时间越长，得分越低。）

·**V/M:** 响应时间的方差均值比（变异数对平均数比，变异系数）。可说明样本的分散程度，这个值越大，往往是越值得考虑优化的对象。

·**Item:** 查询的简单显示，包括查询的类型和所涉及的表。

以下将按默认的响应时间进行排序，并列出TOP n条查询。并且pt-query-digest输出了EXPLAIN的语句，以方便我们验证查询计划。

```
# Query 1: 0.12 QPS, 0.07x concurrency, ID 0x5931CCE8168ECE59 at byte 243208985
# This item is included in the report because it matches --limit.
# Scores: Apdex = 1.00 [1.0], V/M = 0.01
# Query_time sparkline: | ^ |
# Time range: 2010-12-01 00:00:01 to 2010-12-17 09:04:53
# Attribute          pct      total      min       max       avg      95%    stddev   median
# ======  ======  ======  ======  ======  ======  ======  ======  ======
# Count              55     168672
# Exec time         42     92062s     500ms      11s     546ms     640ms     77ms     501ms
# Lock time          68      283s      58us     101ms      2ms     690us      8ms     80us
# Rows sent           1      2.04M      10        100     12.67      9.83     14.86     9.83
# Rows examine       54     33.12G   204.96k   208.16k   205.90k   201.74k      0.00   201.74k
# Query size         50     60.64M     376       378     376.97     363.48      0     363.48
# String:
# Hosts
# Users          sd_game
# Query_time distribution
#   1us
#   10us
#   100us
#   1ms
#   10ms
# 100ms ######
#   1s #
# 10s+
# Tables
#   SHOW TABLE STATUS LIKE 'game_info'\G
#   SHOW CREATE TABLE 'game_info'\G
#   SHOW TABLE STATUS LIKE 'game_stat'\G
#   SHOW CREATE TABLE 'game_stat'\G
# EXPLAIN /*!50100 PARTITIONS*/
select ...
```

以上关于SELECT查询的具体文本此处省略。

从pt-query-digest工具中看到的信息里，对于响应时间，不仅需要关注平均值，还需要关注百分比响应，以及关注其的分布情况和离散程度。

对于响应时间的方差均值比，如果该均值比很大，则可能意味着有一些异常值。

慢查询日志里的慢查询不一定就是BAD SQL。可能是受到了其他查询的影响，或者是受系统资源限制所导致的。

有了分析报告，就可以用EXPLAIN工具确认慢查询的执行计划，从而进行调优。通常，80%的问题是因为索引不佳而引起的，添加适当的索引即可。EXPLAIN的使用请参考3.5.4节；SQL的调优请参考第6章。

4.4 应用程序性能管理

4.4.1 为什么需要性能管理

我们知道，一个用户如果要访问网站，往往需要经过许多软硬件设备，现在的大型应用程序架构越来越复杂，可能包含多层架构，拥有各种子系统，如果系统突然变得很慢，而且代码不能告诉你哪里耗时最长，那么你怎样才能找出系统在何处变慢的呢？所以需要对整个项目进行性能管理。

性能管理其实应该在硬件选型和软件编写之前就开始，但是我们的开发工作往往并没有这么做，往往是等到出现性能问题之后才考虑要进行性能管理，这是不合理的。我们应该确定性能目标，并在产品的各个过程中不断进行验证，及时发现软件架构和编码的问题。如果等到项目已经基本完成的情况下才发现性能问题，往往就会难以调整，因为之前确定的软件架构让性能调优变得很难。

影响服务性能的主要因素，从大到小大致是：架构和设计、应用程序、硬件、Web服务器、数据库、操作系统。数据库、Web服务器、主机一般由SA、系统工程师、DBA管理，在长期的实践中，已经积累了很多成熟的工具，也有一些开源的监控软件，但在应用程序领域，研发人员往往会忘记了或不知道如何去监控自己所写的程序的性能。或者即使知道有一些性能收集的方式，一些性能框架，但对于生成何种数据，以及应该如何统计和展现没有足够的意识。

出现这种现象的原因主要是，研发人员往往侧重于功能实现，而忽视了应用程序的可测量性，为了赶进度，在一般的项目中，对于性能的管理，往往也不作为要求。这就导致了很多业务，一旦上线碰到大流量，就会暴露出性能问题，但由于没有做好性能监控，很难进行针对性的调优。其实，让自己写得程序运行得更快、更有效率，往往不是依赖于自己的经验，而是依赖于有一个好的性能分析工具。通过性能分析工具，我们可以知道自己的程序主要耗时在哪里，从而进行专门的优化，一些性能不佳的操作也可以及早发现，而不会带到生产环境中去。

所以，建议在每个新项目中加入性能剖析代码。如果项目已经开发完毕，再来添加性能日志代码，将会非常困难，但在新项目中包含性能记录代码则是很容易的。

以下将详细介绍性能管理的一些知识及一些记录性能日志的例子。

4.4.2 应用性能管理概述

以下定义来自维基百科。

(1) 什么是应用性能管理

在信息技术和系统管理等领域，应用性能管理（APM），是软件应用程序性能和可用性的监控和管理。APM致力于检测和诊断应用性能问题，从而能提供预期的服务水平。

(2) 应用程序性能指标

有两组性能指标，第一组定义了应用程序终端用户的性能体验，一个很好的例子是高峰时刻的平均响应时间。请注意这里有两个组成部分，负载和响应时间。负载是应用程序处理的业务量，如每秒事务数、每秒请求数、每秒PV。响应时间是指在给定的负载下，应用程序响应用户操作的时间。如果没有一定的负载，绝大部分应用程序都运行得足够快，这就是为什么程序员不太可能在开发过程中捕捉到性能问题的原因。

第二组性能指标衡量了在一定负载下应用程序使用的计算资源是否有足够的容量来支持给定的负载，在哪里可能会有性能瓶颈。这些指标的测量为应用建立了一个基于历史经验的性能基线。然后基线可以用来检测性能的变化。性能的变化可以与外部事件相关联，并用于预测应用程序性能的未来变化。

使用APM最常见的领域是Web应用。除了测量用户的响应时间，应用程序组件的响应时间也可以被监控，以协助我们查明延迟的具体原因。

(3) 当前难点

APM已经演变成跨越许多不同的计算平台上的管理应用程序性能的一个概念。它的实现有如下两个挑战。

- 1) 很难通过仪表化的应用程序来监视应用程序性能，尤其是应用程序的内部组件。
- 2) 应用程序可以被虚拟化，这就增加了测量的变化性。分布式、虚拟和基于云的应用程序给应用性能的监控带来了一个独特的挑战，因为大部分关键的系统组件都不再位于同一台主机上。每个功能现在都可能被设计成运行于多个虚拟系统上的一个因特网服务，应用程序本身也很可能会从一个系统迁移到另一个系统上，以实现服务水平目标或应对临时停电。

4.4.3 应用性能管理的关注点

应用程序本身正变得越来越难以管理，因为它们正在走向高度分散、多层次、多元素的构造，在很多情况下它们依赖于应用程序开发框架，如.NET或Java。

对于Web性能管理，我们重点要关注的是终端用户体验监控（主动和被动）、用户自定义事务处理剖析、报告和应用数据分析。

(1) 终端用户体验监控（主动和被动）

测量用户的请求数据然后将响应返回给用户是捕获终端用户体验的一部分。这种测量的结果被称为实时应用监控（又名自上而下的监控），其中有两个组成部分，被动和主动。

被动监控通常是无代理的，比如使用网络端口镜像实现监控。这个解决方案需要考虑的一个关键功能是支持多协议的分析（如XML、SQL、PHP），因为大多数企业已经不仅仅只支持基于Web的应用程序。

主动监控，包含预定义的人工探针和网络机器人，用于报告系统的可用性和业务交易。主动监控是被动监控的一个很好的补充。两种手段配合使用，可以提供可视化的应用健康状况。

(2) 用户自定义事务处理剖析

专注于用户自定义的事务或对于商业团体具有某种意义的URL页面定义。例如，对于一个给定应用程序，如果有200~300个唯一页面，可以把它们分组为8~12个更高层次的类别。这样就可以实现有意义的服务等级协议（SLA）报告，从业务的角度提供应用性能的趋势信息：先从大类开始，逐渐完善它。

(3) 报告和应用数据分析

对于所有的应用程序，提供一套共同的指标来收集和报告信息是很重要的，然后就可以标准化呈现应用程序的性能数据的视图。来自其他工具集收集的原始数据可提高报告的灵活性。这样就可以回答各种各样的性能问题，尽管每个应用程序可能运行在不同的平台上。

注意过多的信息是难以查看的，这就是为什么报告保持简单是很重要的原因，否则它将不被使用。

4.4.4 具体应用

生产环境中常见的方法是记录应用程序的日志。在应用程序中记录性能日志将会更全面，这可以让你跟踪用户从访问应用服务到回传的各个环节。而数据库的性能记录往往只反映了后端数据库的性能记录。DBA常用的诊断工具慢查询日志很粗糙，只记录了超过阈值的慢查询。

通过对日志的分析，可以方便用户了解应用的运行情况，有助于进行容量规划，分配资源，还可以分析得到该应用的健康状况，及时发现问题并快速定位和解决问题；也可以分析用户的操作行为、喜好、地域分布、浏览器类型、操作系统或其他更多信息。我们应该尽可能地记录更多的信息，也就是说，只要愿意，就有能力生成大量的跟踪信息。在这里，我们仅关注下记录性能方面的日志。

不管是使用框架提供的一些功能，还是自己编写记录日志的代码，都要注意如下这些要点。

- 要使用方便，配置简单。
- 要可读性好，方便处理，最好是可以图形化展示，并且趋向实时。
- 不仅要监测整体响应，还要监测每个环节，特别是关键部分的响应时间。

比如对于一个普通的PHP页面，我们可以记录整体的响应时间，页面每个部分的处理时间，也可以记录访问缓存、访问数据库的响应时间，如果有重要的业务逻辑，也要一并记录，通过这些翔实的记录，一旦碰到各种性能问题，我们就可以很方便地定位到出现异常的地方。由于日志的刷新往往很快，因此我们要尽量保持日志紧凑，可以记录到本地，也可以通过网络的方式发送日志到日志服务器。

除了记录日志，日志的解读也很重要，如果可能，最好能够图形化地展示性能、吞吐的变化，这样，我们就可以很直观地通过曲线的变化知道应用的性能是否可能有问题了。一旦出现性能问题，也可以很直观地从图形中得到需要优化的点。

一般记录性能日志不会有什么开销，由于日志是顺序写入的，对I/O的影响也很小。如果真的记录起来成本很昂贵，那么也可以选择某一台应用服务器打开性能记录，或者仅记录一段时间，用来诊断性能问题。

也可以随机抽样，选择记录部分比例的访问的性能记录。

例如：

```
<?php  
$profiling_enabled = rand(0, 100) > 99;  
?>
```

以上代码采样为1%。

这里要介绍一个国外的性能管理服务工具，New Relic公司的性能工具，虽然国内到国外的网络质量不佳，不便直接使用New Relic的服务，但它的思想很值得借鉴。

New Relic是一种提供给公司的SaaS（software-as-a-service）解决方案，可以提供性能监视和分析服务。能够对部署在本地或在云中的Web应用程序进行监控、故障修复、诊断、线程分析及容量计划，它可以监测从浏览器到应用程序到数据库各个环节的性能记录。它还可以从多个角度、实时监测移动设备App的性能，及时发现App的错误。

这样的一个应用性能管理工具，能极大地解决各种性能问题。有兴趣的同学可以试用下，默认的仪表板会显示终端用户和应用服务器的一些指标。

它的基本原理是将工具嵌入到你的应用程序里，剖析它，并发送数据到New Relic的服务器，通过基于Web的界面，让你看到应用程序响应时间的性能记录。

这种SaaS服务，使得在生产环境上时刻记录性能成为可能。而传统的一些性能工具，可能因为消耗的资源巨大，而不能轻易地在生产环境中打开并使用。

PHP也有一些优秀的工具，可以用来剖析的你程序。如Facebook开源出来的xhprof（链接地址：<http://pecl.php.net/package/xhprof>）。

《high performance MySQL》作者开发的一个工具IfP（链接地址：<https://code.google.com/p/instrumentation-for-php/>）。

xhprof对PHP有完善的监控，IfP相对于xhprof来说，对数据库有更详细的测量，它可以自动记录整个页面、数据库和Cache的响应。

4.5 数据库设计

广义的数据库设计包括项目的目标、数据的架构设计、数据库产品的选择、需求收集、数据库逻辑/物理设计、后期维护等多个过程。本书仅阐述数据库设计中DBA关注的两个阶段：逻辑数据库设计阶段和物理数据库设计阶段。以下将介绍设计数据库的一般步骤，我们设计的时候不一定要严格遵循这些步骤，它们比较繁琐，但这些步骤阐明了设计数据库的一般思路，它的方法学值得大家借鉴，尤其是设计很复杂的数据库应用的时候。

4.5.1 逻辑设计

逻辑数据库设计指构建企业所使用的数据模型的过程。它标识了数据库中要描述的重要对象以及这些对象之间的关系。

逻辑数据库设计独立于特定的DBMS和其他的物理考虑事项。

数据库设计人员将根据需求文档，创建与数据库相关的那部分实体关系图（ERD）/类图。这些图形和需求文档相结合，将有助于相关人员更好地理解业务逻辑和实际的表设计。互联网的一些应用往往比较简单，所以经验丰富的研发人员直接设计数据表也是很常见的情况，但是对于复杂的项目，仍然推荐绘制E-R图，如果我们有对逻辑设计的详细描述会更有利于以后程序的开发和维护，也方便DBA与研发设计人员更好地沟通。

逻辑数据库的设计大体可以分为以下这些步骤。

1) 创建并检查ER模型。

此步骤主要是标识实体及实体之间的关系。标识实体的一种方法就是研究用户需求说明里的名词或名词短语。例如员工管理系统里的员工、部门。在线考试系统里的课程、试卷、学员。从用户提供的需求说明中得到的一组实体可能不是唯一的。然而，分析过程的不断迭代必定会引导你选择对完成系统需求来说足够用的实体。

标识关系也可以通过研究需求说明书来实现，需求说明书里的动词或动词短语往往表征了某种关系。大多数情况下，关系都是二元的，例如，员工实体属于某个公司，试卷实体属于某个课程，学员（实体）解答某张试卷（实体）。

接下来就可以标识实体和关系中的属性、主键等信息。比如学员实体包括的属性可能有学员号、姓名、性别、生日等信息。

在确定好实体后，我们再检查实体模型是否能够满足我们的需求。

图4-2是实体关系图的一个例子。

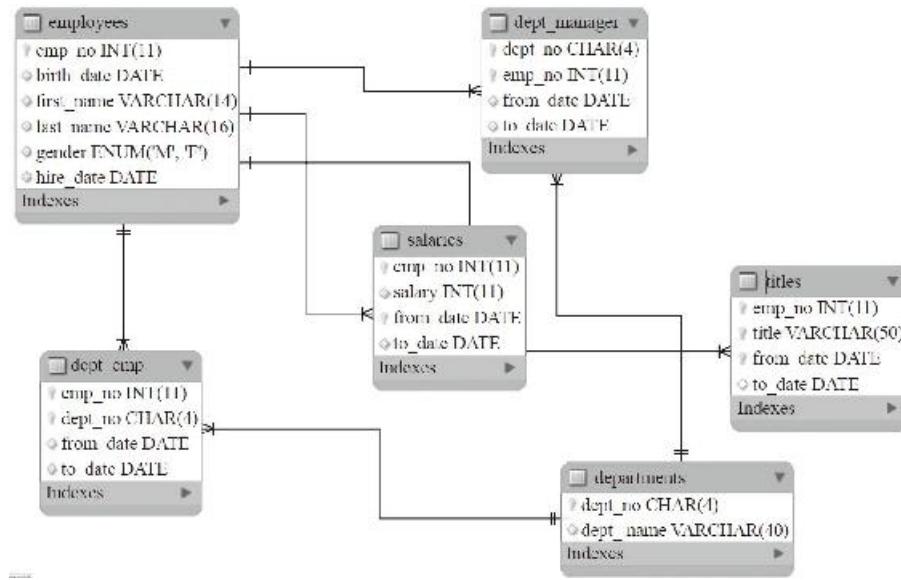


图4-2 实体关系图示例

3.3.7节对这个实体关系图有一些说明。

2) 将ER模型映射为表。

这个步骤的主要目的是为步骤1建立的ER模型产生表的描述。这组表应该代表逻辑数据模型中的实体、关系、属性和约束。产生表的描述后，需要检查表是否满足用户的需求和业务规则。

4.5.2 物理设计

物理数据库设计用于确定逻辑设计如何在目标关系数据库中物理地实现。它描述了基本表、文件组织、用户高效访问数据的索引、相关的完整性约束及安全性限制。

这个阶段允许设计者决定如何实现数据库，因此，物理设计和特定的DBMS有关。

这部分的任务主要是设计表结构。逻辑设计中的实体大部分可以转换成物理设计中的表，但是它们并不一定是一一对应的。

物理设计又可以分为如下几步。

把逻辑设计转换为物理表、分析事务、选择文件组织方式、选择索引（基于最重要的事务），以及适当地进行反范式设计（这么做是为了拥有更好的性能）、列出最终表的详细说明。

1.将逻辑设计转换为物理表

将逻辑设计转换为物理表即用特定的数据库语言来实现逻辑设计过程中产生的表的描述，可以输出的信息有表汇总，如表4-12所示的就是表汇总的模板。

表4-12 表汇总模板

表名	功能说明
表 A	
表 B	
表 C	

2.分析事务

分析事务指的是分析数据库需要满足的用户需求，只有了解了必须要支持的事务的细节，才能做出有意义的物理设计抉择。分析预期的所有事务是极为耗时的，只需研究最重要的那部分事务即可。最活跃的20%的事务往往占据了总的数据访问量的80%。当进行分析时，你会发现这个80/20规则是很有用的方针。

最重要的事务一般是指如下两种事务。

- 经常运行的事务和对性能产生重大影响的事务。

- 业务操作的关键事务。

需要关注的一些细节如下。

- 事务运行的频率？频率信息将标识需要仔细考虑的表。

- 事务的高峰时间？

- 访问记录数比较多的事务。

不用写出全部的SQL语句，但至少应该标识出与SQL语句相连的细节类型，也就是如下这些信息。

- 将要使用的所有查询条件。

- 连接表所需要的列（对查询事务来说）。

- 用于排序的列（对查询事务来说）。

- 用于分组的列（对查询事务来说）。

- 可能使用的内置函数（例如AVG, SUM）。

- 被该事务更新的列。

我们将利用这些信息来确定所需要的索引。

3.选择文件组织方式

选择文件组织方式是指选择表数据的存放方式。

物理设计数据库的目标之一就是以有效的方式存储数据。如果目标DBMS允许，则可以为每个表选择一个最佳的文件组织方式。一般有如下两种方法。

- 1) 保持记录的无序性并且创建所需数目的二级索引。
- 2) 通过指定主键或聚簇索引使表中记录为有序的。这种情况下，应该选择如下的列来排序或聚簇索引记录。
 - 经常用于连接操作的列，因为这样会使连接更有效率。
 - 在表中经常按某列的顺序访问记录的列。

我们一般使用InnoDB（InnoDB主键即聚簇索引），基于主键的唯一查找和小范围查找是最高效的。例如，如果有很频繁的基于USERID的查找，或者对USERID的小范围遍历，那么USERID作为主键就是最高效的方式。因为数据是以USERID为顺序进行存储的。而如果以自增ID为主键，实际的执行过程是需要先按索引列USERID找到索引记录，然后利用存储在索引中的主键值去查找主键，最终定位到记录，这样代价会更高。如果是范围查找，那么虽然索引是有序的，但最终会按照主键值去检索数据，由于主键值并不是连续的，这将产生很多物理随机读。

以上例子仅用于说明问题，实际应用中，对于小范围的索引查找，性能一般不会成为问题。自增主键在一般情况下也会工作得很好。

4.选择索引

设计索引需要平衡性能的提升和维护的成本。

创建你认为是索引的候选列的“意愿表”，然后逐个考虑维护这样的索引的影响。以下是创建索引的一些基本指导原则。

- 1) 不必为小表创建索引。在内存中查询该表会比存储额外的索引结构更加有效。
- 2) 为检索数据时大量使用的列增加二级索引。
- 3) 为经常有如下情况的列添加二级索引。
 - 查询或连接条件
 - ORDER BY
 - GROUP BY
 - 其他操作（如UNION或DISTINCT）
- 4) 考虑是否可以用覆盖索引（covering index）。
- 5) 如果查询将检索表中的大部分记录（例如25%），即使表很大，也不创建索引。这时候，查询整表可能比用索引查询更有效。
- 6) 避免为由长字符串组成的列创建索引。

5.反范式设计



提示 建议先进行规范化的设计，这样将有助于了解系统，但MySQL对于多表连接的支持比较差，也就是优化器比较简单，往往为了性能，我们需要考虑一些反规范化的设计。

反范式的一些方法包括但不限于如下几点。

·合并表。

·冗余列减少连接。

·引入重复组，例如，某公司有5个电话号码，我们不必使用额外的电话表，而是增加5个列telNO1、telNO2、telNO3、telNO4、telNO5（此种情况一般用于重复组的项的数量不多且不易变化）。

·创建统计表。

·水平/垂直分区。



注意 反范式增加了维护数据一致性的成本，因此需要谨慎实施。

6.列出最终表的详细说明

只需要列出重要的表即可。

以下索引是否建立、数据量及数据增长的情况要根据具体的业务需求来确定。

·记录数：记录数，可补充说明未来半年、1年或2年的记录数。

·增长量：单位时间的数据增长量。如果量大可以按每天；如果量不大则可以按每月。

·表字段的区别度：主要是考虑到将来在此字段上建立索引类型选择时作参考，当字段值唯一时可以不考虑；当字段值不唯一时，估算一个区别度，近似即可。例如：如果一个表的NAME字段共有2000个值，其中有1999个不同的值，那么 $1999/2000=0.99$ 越接近1区别度则越高，反之区别度就越低。

·表的并发：根据具体的业务需求预测表的并发访问，或者说明高峰期的并发程度。

最终表的模板如表4-13所示。

表4-13 最终表模板

表名	
主键	
非序字段	
索引字段	
字段名称	数据类型(精度范围) 允许为空 Y/N 唯一 Y/N 区别度 默认值 说明 [字段名称] [数据类型] [Y/N] [Y/N] [高/中/低] [] CREATE TABLE [XXX 表名] { [字段名称][数据类型] [NOT NULL /NULL] ,[字段名称][数据类型] [NOT NULL /NULL] ,[字段名称][数据类型] [NOT NULL /NULL] ,[字段名称][数据类型] [NOT NULL /NULL] ,PRIMARY KEY {[字段名称]}) ENGINE=InnoDB DEFAULT CHARSET=utf8; CREATE INDEX [索引名] ON [XXX 表名]([字段名称])
MySQL 特本	
记录数	[此表的记录数]
增长量	[此表的增长量]
表的并发	[此表的并发程度]
补充说明	[补充说明]

实际设计中，如果表很多，只需要列出最重要、最关键的表设计即可。

4.6 导入导出数据

研发人员往往需要从数据库中导出数据，或者将数据导入到数据库中。一些客户端工具提供了简单方便的功能，让研发人员可以不用去熟悉命令行工具mysql、mysqldump即可进行操作，但客户端工具对于数据的导出导入可能存在兼容性的问题，而原生的命令行工具往往具有更好的兼容性。客户端工具也可能会受到环境的限制而不能使用，所以，研发人员有必要掌握一些常用的命令行操作数据的方式。我们在日常升级操作中，往往也需要提供一些命令让DBA运行，从而把数据导出来给研发、测试人员做二次处理。熟悉导出导入数据的命令也有助于研发、测试人员自己方便地获取数据而不需要通过DBA。一些统计分析脚本也依赖于调用mysql命令行工具实现数据的操作。

MySQL提供了好几种导出导入数据的方法：LOAD DATA、mysqlimport、SELECT...INTO OUTFILE、mysqldump、mysql。其中，mysqldump和mysqlimport是相反的操作，SELECT...INTO OUTFILE和LOAD DATA INFILE是相反的操作。



注意 在使用LOAD DATA或SELECT...INTO OUTFILE命令的时候，要留意操作系统文件的权限。你需要确保MySQL实例进程的拥有者对操作系统文件拥有权限。

4.6.1 规则简介

1.文本文件里的特殊字符处理

LOAD DATA和SELECT...INTO OUTFILE、mysqlimport和mysqldump有一组专门的用来处理文本文件中特殊字符的选项，具体如下所示。

- FIELDS TERMINATED BY'fieldtermstring': 各列（字段）之间用什么字符分隔，默认是tab，一般设置为逗号“,”。
- [OPTIONALLY]ENCLOSED BY'char': 值被什么字符引起来，一般设置为引号""，如果指定了OPTIONALLY，则ENCLOSED BY'char'只对字符串数据类型（比如CHAR、BINARY、TEXT或ENUM）生效。
- ESCAPED BY'escchar': 定义转义字符，默认是“\”。
- LINES TERMINATED BY'linetermstring': 定义行结束符，用于分隔行。

在Windows下需要使用“\r\n”提供一次换行，而在Linux下只需要“\n”就可以了。

2.文本文件的数据格式

所有命令都要求有关的文本文件必须严格遵守一种数据格式，具体如下所示。

- 数值：可以用科学计数法。
- 字符串：字符串里的特殊字符必须加上反斜线字符作为识别标志，以区别于各种分隔符。日期按照2005-12-21格式的字符串来对待，时间值按照23:59:59格式的字符串来对待，时间戳按照20051231235959格式的整数来对待。
- NULL值：假设“\”作为转义前导字符，“\”作为字符串的前后缀标记，那么在导出操作中，NULL值将被表示为\\N；在没有指定转义前导字符的导出操作中，NULL值将被表示为由4个字符构成的字符串。在指定了转移前导字符的操作中，MySQL将把NULL、\\N、'\\N'都解释为NULL值，但'NULL'将被解释为一个字符串'NULL'。

4.6.2 使用mysqldump导出，使用mysql导入

虽然mysqldump速度较慢，但这种方式有最好的兼容性，这也是目前使用最为广泛的备份数据的方式。使用mysqldump导出的一般是SQL文件，也称为转储文件或dump文件，我们可以使用客户端工具mysql执行这个文件，导入数据，示例如下。

1) 导出指定的表。

```
mysqldump test --tables test1 test4 > test1_test4.sql
```

2) 分别导出sql文件和数据文件（数据值以tab分隔）。

```
mysqldump --tab=/home/garychen/tmp test
```

3) 分离导出sql文件和数据文件（定制数据格式，数据值以逗号分隔）

```
mysqldump --tab=/home/garychen/tmp --fields-terminated-by=',' --fields-enclosed-by=''' test
```

4) 导出某个库。

```
mysqldump --complete-insert --force --add-drop-database --insert-ignore --hex-blob --databases test > test_db.sql
```

代码说明如下。

--complete-insert: 导出的dump文件里，每条INSERT语句都包括了列名。

--force: 即使出现错误（如VIEW引用的表已经不存在了），也要继续执行导出操作（mysqldump会打印出错误，注释完VIEW定义后继续后续的数据导出）。

--insert-ignore: 生成的INSERT语句是INSERT IGNORE的形式，如果导入此文件，即使出错了也仍然可以继续导入数据（当作警告）。

例如，使用mysql执行SQL文件，插入与主键冲突的值，如果是INSERT，那么mysql会异常退出，并提示如下错误。

```
ERROR 1062 (23000) at line 28: Duplicate entry '1' for key 1
```

如果是INSERT IGNORE，那么mysql会忽略错误，继续插入后面的值。

例如下面这些语句。

```
INSERT IGNORE INTO 't1' VALUES ('1'),('10'),('11'),('2'),('3'),('4'),('5'),('6'),('7'),('8'),('9');  
INSERT IGNORE INTO 't1' VALUES ('111'),('20'),('21'),('22'),('23'),('4'),('5'),('6'),('7'),('88'),('99');
```

两条INSERT语句，即使有重复键值，也仍然会插入后面的值，因此88、99仍然可以正常插入。

--databases: 类似--tables，后面可以跟多个值。

--compatible=name: 导出的文件和其他数据库更兼容（但不确保），name的值可以是ANSI、MYSQL323、MYSQL40、POSTGRESQL、ORACLE、MSSQL、DB2、MAXDB、NO_KEY_OPTIONS、NO_TABLE_OPTIONS或NO_FIELD_OPTIONS。

5) 导出所有的数据库。

```
mysqldump --all-databases --add-drop-database > db.sql
```

6) 导出xml格式的数据。

```
mysqldump -u root -p --xml mylibrary > /tmp/mylibrary.xml
```

如果有二进制数据，则要使用选项--hex-blob。

InnoDB若想获得一致性的数据库副本，则要启用选项--single-transaction。

mysqldump不能利用通配符导出多个表，表比较多的时候，可以先SELECT出要导出的表，如下语句即可查询到所有的表。

```
select group_concat(table_name SEPARATOR ' ') from information_schema.tables where table_schema ='db_name' and table_name like 'prefix%';
```

或者，可以采用如下方式将表名导出到一个文件。

```
mysql -N information_schema -e "select table_name from tables where table_name like 'prefix_%'" > tbs.txt
```

然后运行如下命令导出数据。

```
mysqldump db 'cat tbs.txt' > dump.sql
```

也可以忽略部分表，加上参数--ignore-table=db_name.tbl_name1、--ignore-table=db_name.tbl_name2。

mysqldump可以把警告和错误追加记录在文件中，加上参数--log-error=file_name即可。

如果使用mysqldump导出数据，可以考虑的优化的方式有如下5种。

·选择I/O活动低的时候。

·I/O分离（数据盘和备份盘I/O分离）。

·输出到管道压缩（gzip）。

·--quick跳过内存缓冲（--opt默认启用）。

·从数据保留策略上想办法，把不需要修改的大量数据放到历史表中，而不是每次都备份。

mysqldump导出的SQL转储文件，可以用如下的形式将数据导入到数据库中。

```
mysql db_name < db_name.sql
```

转储文件（dump文件）里面一般指定了set names utf8，所以我们在导入的时候不再需要指定特殊的字符集。例外的情况是，有一些特殊的场合，SQL文件是以其他的字符集导出的，这个时候导入要注意保持文件的字符集、客户端字符集和连接的字符集的一致性，例如：

```
mysql --default-character-set=charset_name database_name < import_table.sql
```

--default-character-set的意思是，客户端和连接都默认使用charset_name字符集。例如：

```
mysql --default-character-set=gbk < import_table.sql
```

这个文件的字符集是gbk。

如果mysql客户端输出的数据是乱码，那么请检查下客户端、连接的字符集配置。例如，我们使用SSH工具securecrt登录主机，然后使用mysql命令行工具连接MySQL服务器，mysql连接的默认配置可能是latin1，那么此时显示utf8的数据将会是乱码。这种情况下，可以在客户端运行set names utf8，并确认securecrt的字符编码是UTF-8，这样就可以正常显示utf8字符集的数据了。

4.6.3 使用SELECT INTO OUTFILE命令导出数据

如果想要进行SQL级别的表备份，可以使用SELECT INTO OUTFILE命令语句。对于SELECT INTO OUTFILE，输出的文件不能先于输出存在。

示例语句如下所示。

```
SELECT * INTO OUTFILE '/tmp/testfile.txt' FROM exporttable;
SELECT * INTO OUTFILE '/tmp/testfile.txt' FIELDS TERMINATED BY ':' OPTIONALLY ENCLOSED BY '+' ESCAPED BY '!' FROM
exporttable;
SELECT a,b,a+b INTO OUTFILE '/tmp/result.text' FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' LINES TERMINATED BY '\n'
FROM test_table;
```

一般来说，只要导出导入操作中使用的选项完全一致，用SELECT...INTO OUTFILE命令导出的文本文件就可以用LOAD DATA命令导入到数据表里去，不会发生任何变化。

4.6.4 使用LOAD DATA导入数据

SELECT...INTO OUTFILE可以筛选记录，导出表数据到一个文件中，而LOAD DATA INFILE则是相反的操作，是读取这个文件导入表中。

如果MySQL服务器和LOAD DATA命令不在同一台计算机上执行，当想导入本地文件系统的文件时，则需要使用语法变体LOAD DATA..LOCAL INFILE...，也就是说，如果指定LOCAL关键词，则表明从客户主机读文件。如果没指定LOCAL，那么文件必须位于服务器上。

可能是因为字符集设置而导致乱码的问题。LOAD DATA INFILE在某些MySQL的版本上不支持指定导入时的字符集。这时，MySQL将假设导入文件的字符集是character_set_database，这个变量会根据当前数据库指定的字符集而变化；如果没有指定当前数据库，那么它的值将由character_set_server决定。因此如果LOAD DATA INFILE不支持指定字符集，那么在导入前需要确认当前数据库的字符集，如果与当前数据库的字符集不符，则使用SET character_set_database命令进行更改。SET names命令也是可行的，或者直接在LOAD DATA INFILE命令里指定字符集，例如如下语句。

```
mysql> load data infile '/tmp/t0.txt' into table t0 character set gbk fields terminated by ',' enclosed by '"' lines
terminated by '\n' (`name`, `age`, `description`) set update_time=current_timestamp;
```

其他示例如下。

示例1：

```
LOAD DATA INFILE '/path/to/file'
INTO TABLE table_name
FIELDS TERMINATED BY '\t'
ENCLOSED BY '\"'
LINES TERMINATED BY '\n'
```

示例2:

```
LOAD DATA INFILE '/path/to/file' REPLACE
INTO TABLE table_name
FIELDS TERMINATED BY '\t'
ENCLOSED BY '\"'
LINES TERMINATED BY '\n'
```

示例3：导入csv格式的文本文件。csv格式的文件，即逗号分隔的数据文件。首先，生成如下csv文件。

```
mysql> select field_list from table_name into outfile '/home/garychen/tmp/table_name_2.csv' fields terminated by ','
optionally enclosed by '\"' lines terminated by '\n';
```

然后，截断表，清空数据，命令如下。

```
mysql> truncate table table_name;
```

最后，进行验证，可以看到，原来导出的文件，现在可以正常导入到数据表中，语句如下。

```
mysql> load data local infile '/home/garychen/tmp/table_name_2.csv' into table table_name fields terminated by ',' lines
terminated by '\n'(field1,field2,field3);
```

LOAD DATA的优化 相较于普通的mysql命令，LOAD DATA执行SQL文件导入的方式要快得多，一般可以达到每秒几万条记录的插入速度。有时对于大表，我们仍然期望获得更高的导入速度，以下将针对InnoDB和MyISAM表分别叙述如何进行优化。

对于InnoDB的优化，建议的方式如下。

- 将innodb_buffer_pool_size设置得更大些。
- 将innodb_log_file_size设置得更大些，如256MB。
- 设置忽略二级索引的唯一性约束，SET UNIQUE_CHECKS=0。
- 设置忽略外键约束，SET FOREIGN_KEY_CHECKS=0。
- 设置不记录二进制日志，SET sql_log_bin=0。
- 按主键顺序导入数据。由于InnoDB使用了聚集索引，如果是顺序自增ID的导入，那么导入将会更快，我们可以把要导入的文件按照主键顺序先排好序再导入。
- 对于InnoDB引擎的表，可以在导入前，先设置autocommit=0，例如如下语句。

```
truncate table_name;
set autocommit = 0;
load data infile /path/to/file into table table_name...
commit;
```

· 可以将大的数据文件切割为更小的多个文件，例如使用操作系统命令split切割文件，然后再并行导入数据。

对于MyISAM的优化，建议的方式如下。

- 将bulk_insert_tree_size、myisam_sort_buffer_size、key_buffer_size设置得更大些。
- 先禁用key（ALTER TABLE..DISABLE KEYS），然后再导入数据，然后启用key（ALTER TABLE..ENABLE KEYS）。重

新启用key后，可以批量重新创建索引，批量创建索引的效率比在逐笔插入记录时创建索引要高效得多。注意ALTER TABLE... DISABLE KEYS禁用的只是非唯一索引，唯一索引或主键是不能禁用的，除非你先手动移除它。

· 使用LOAD DATA INFILE，tab分隔的文件更容易解析，比其他方式更快。

由于唯一索引（约束）对于我们导入数据的影响比较大，尤其对于大表导入，我们需要留意这一点。不要在大表上创建太多的唯一索引，主键、唯一索引不要包含太多列，否则导入数据将会很慢。

关于优化导入数据的方式，见仁见智，其实一次INSERT插入多条记录，控制每个表的大小（<15GB，确保B-tree索引在内存中），并发导入，批量事务等方式都有好处，但更多的时候也要考虑维护的简单方便。

如果有很多表，那么使用mysqldump会更简单。如果是导入个别大表，而且对于时间有很高的要求，那么LOAD DATA未尝不可。

mysqldump默认的导出文件，其实已经包含了一些优化了，会有禁用key、启用key的操作，而且是一条INSERT语句包括多行记录的。

4.6.5 用mysqlimport工具导入

mysqlimport命令的语法格式如下。

```
mysqlimport databasename tablename.txt
```

示例如下。

```
mysqlimport --local test imptest.txt
```

4.6.6 用mysql程序的批处理模式导出

有时可以考虑使用mysql工具导出数据，特别是远程操作的时候，下面来看几个示例。

示例1： 导出authors表。

```
mysql -u root --password=123456 --batch --default-character-set=utf8 -e "SELECT * FROM authors;" mylibrary > output.txt
```

示例2： 查询结果的纵向显示。

```
mysql -u root --password=123456 --vertical '--execute=SELECT * FROM titles;' mylibrary > test.txt
```

示例3： 生成html表格形式的输出。

```
mysql -u root -p=xxx --html '--execute=SELECT * FROM titles;' \
--default-character-set=latin1 mylibrary > test.html
```

示例4： 用mysql程序生成xml格式的输出。

```
mysql -u root -p -xml --default-character-set=utf8
'-execute=SELECT * FROM titles;' mylibrary > C:\test.xml
```

4.6.7 用split切割文件，加速导入数据

split命令的作用是切割文件，语法格式如下所示。

```
split [OPTION] [INPUT [PREFIX]]
```

如果不加入任何参数，默认情况下是以1000行的大小来分割的。

下面来看个案例，使用split切割导出的数据文件，这些数据文件需要通过PHP脚本解析二次处理后，再插入MySQL数据库，示例如下。

```
split -l 5052000 subs.txt test_split_sub_
```

其中，-l参数指定按多少条记录切割文件。这里将按照每5052000条记录进行切割，生成的文件名以test_split_sub_为前缀，生成的文件名类似如下。

```
test_split_sub_aa test_split_sub_ab test_split_sub_ac ...
```

然后就可以并发执行多个PHP客户端程序来导入数据了。

4.7 事务和锁

4.7.1 概述

我们知道，数据库是一个多用户访问系统，因此需要一种机制来确保当多个用户同时读取和更新数据时，数据不会被破坏或失效，锁就是这样的一种并发控制技术。当一个用户需要修改数据库中的记录时，首先要获取锁，只有这样该用户在锁的持有期间，其他用户就不能对这些记录进行修改了。

不同的数据库产品实现的锁机制各不相同，而锁定的程度也会受到事务隔离级别的影响。不同的数据库产品实现锁的方式各不一样，即使是MySQL，不同版本之间也可能存在差异。本节只是介绍锁的一般表现形式，对于具体的锁定细节，请读者自行参考相关资料并验证。

MySQL Server级别的锁大致有如下两种。

(1) Table locks (表锁)

```
mysql> LOCK TABLES table_name READ;
mysql> SELECT SLEEP(30) FROM table_name LIMIT 1;
```

(2) Global locks (全局锁)

```
mysql> FLUSH TABLES WITH READ LOCK;
Name locks      mysql> RENAME TABLE table_name TO table_name2;
String locks    mysql> SELECT GET_LOCK('my lock', 100);
```

下面将首先简单介绍MyISAM表的锁技术，生产环境中使用MyISAM的场景很少，所以这里只是介绍下基本原理和可能会碰到的问题。然后再着重介绍下InnoDB事务及与事务相关的锁定技术。

4.7.2 MyISAM的表锁

MySQL支持对MyISAM和MEMORY表进行表级锁。

下面来看看表锁定的原理。

对于WRITE，MySQL使用的表锁定方法原理如下。

- 如果在表上没有锁，则在它上面放一个写锁。

- 否则，把锁定请求放在写锁定队列中。

对于READ，MySQL使用的锁定方法原理如下。

- 如果在表上没有写锁定，则把一个读锁定放在它上面。

- 否则，把锁定请求放在读锁定队列中。

当一个锁定被释放时，锁定可先被写锁定队列中的线程得到，然后是读锁定队列中的线程。

这就意味着，如果你在一个表上有很多更新，那么SELECT语句将等待直到没有更多的更新操作为止。

可以通过检查

table_locks_waited

和

table_locks_immediate

状态变量来分析系统上的表锁定争夺。

对于MyISAM引擎的表，如果INSERT语句不会发生冲突，则可以在其他客户正在读取MyISAM表的时候插入行。如果数据文件中不包含空闲块，则不会发生冲突，因为在这种情况下，记录总是插入在数据文件的尾部（从表的中部删除或更新的行可能会导致空洞）。如果有空洞，那么当所有空洞都填入新的数据时，并行的插入就能够重新自动启用。

表锁定将会使很多线程同时从一个表中进行读取操作，但是如果某个线程想要对表进行写操作，那么它必须首先获得独占访问。更新期间，所有其他想要访问该表的线程必须等待，直到更新完成。

如下是需要注意的特殊的表锁机制。

如果一个客户发出了长时间运行的查询（SELECT），而此时，另一个客户想要对同一个表进行更新（UPDATE），那么该客户必须等待直到SELECT完成。如果此时还有一个客户对同一个表也发出了另一个SELECT语句，因为UPDATE比SELECT的优先级高，那么该SELECT语句将会等待直到UPDATE完成，并且它们都要等待第1个SELECT完成。性能问题往往发生在这个步骤。

以上的机制，在很多基于MyISAM引擎表的程序中可能会导致严重的性能问题，比如一些论坛程序。建议的解决方案是设置变量`-low-priority-updates=1`，即可以在系统级别进行设置，以避免SELECT查询线程大量累计。

一些公司采用了MyISAM作为统计库，为了加速，往往在批量更新数据的时候设置了并发，但由于并发更新时频繁的表锁竞争，更新数据的速度反而会下降。可以使用LOCK TABLES来提高速度，因为在一个锁定中进行很多更新比没有锁定的更新要快得多。将表中的内容切分为几个小表也可以有所帮助。

LOCK TABLES的一些表现如下，读者可以自行验证。

·“LOCK TABLES t1 READ;”表示其他会话可读，但不能更新。

·“LOCK TABLES t1 write;”表示其他会话不可读，不可写。

·“UNLOCK TABLES;”表示释放锁。

4.7.3 事务定义和隔离级别

事务是数据库管理系统执行过程中的一逻辑单元，由有限的操作序列构成。

1. 事务的ACID特性

并非任意的对数据库的操作序列都是数据库事务。数据库事务拥有以下4个特性，习惯上被称为ACID特性。

(1) 原子性 (Atomic)

事务作为一个整体被执行，包含在事务中的对数据库的操作要么全部被执行，要么全部都不执行。

比如，InnoDB支持事务，在InnoDB事务内如果执行了一条插入多个值的INSERT语句“`INSERT INTO t VALUES('b1'),('b2'), ('b3'),('b4'),('b5'),('b6');`”只要其中一个值插入失败，那么整个事务就失败了。而对于MyISAM引擎的表，它不支持事务，那么在出错之前的值是可以被正常插入到表中的。

(2) 一致性 (Consistency)

事务应确保数据库的状态从一个一致状态转变为另一个一致状态。一致状态的含义是数据库中的数据应满足约束。

(3) 隔离性 (Isolation)

多个事务并发执行时，一个事务的执行不应影响其他事务的执行。

(4) 持久性 (Durability)

已被提交的事务对数据库的修改应该被永久保存在数据库中。

2. 事务的隔离级别

事务隔离级别越高，越能保证数据的完整性和一致性，但是对并发性能的影响也会越大。MySQL事务包含如下4个隔离级别，按隔离级别从低到高排列如下。

(1) read uncommitted (dirty read)

read uncommitted也称为读未提交，事务可以看到其他事务更改了但还没有提交的数据，即存在脏读的情况。

(2) read committed

read committed也称为读提交，事务可以看到在它执行的时候，其他事务已经提交的数据，已被大部分数据库系统采用。允许不可重复读，但不允许脏读，例如如下语句。

```
begin transaction;
select a from b where c=1;
...          #其他事务更改了这条记录
,并且
commit提交

select a from b where c=1;      #可以看到新的数据, 不可重复读

end
```

(3) repeatable read

repeatable read也称为可重复读。同一个事务内，同一个查询请求，若多次执行，则获得的记录集是相同的，但不能杜绝幻读，示例如下。

```
begin transaction
select a from b where c=1;
...          #其他事务更改了这条记录
,并且
commit
select a from b where c=1;      #仍然看到旧的数据
,可重复读
,但不能杜绝幻读

end
```

发生幻读的场景有，某事务A按某个条件进行查询，此时尚未提交。然后另一个事务成功插入了数据。事务A再次查询时，可能会读取到新插入的数据。

MySQL InnoDB引擎默认使用的是repeatable read（可重复读）。当事务A发出一个一致性读之时，即一个普通的SELECT语句，InnoDB将给事务A一个时间点。如果另一个事务在该时间点被指定之后删除一行并提交，则事务A看不到该行已被删除。插入和更新的处理与此相似。可以通过提交事务来前进时间点，然后进行另一个SELECT。这被称为多版本并发控制（multi-versioned concurrency control）。如果想要查看数据库的最新状态，应该用READ COMMITTED隔离级别或用一个锁定

读“SELECT*FROM t LOCK IN SHARE MODE;”。

为了满足可重复读，事务开启后，对于要查询的数据，需要保留旧的行版本，以便重新查询，这在一些特殊的环境中可能会导致某些问题，比如一些框架，对于任何操作，都要先进入AUTOCOMMIT=0的模式，直到有写入时才会进行COMMIT提交，这可能会导致事务数过多，有时由于框架或编码的不完善，可能会出现长时间不提交的事务，导致UNDO保留的旧的数据记录迟迟不能被删除，还可能导致UNDO空间暴涨。对于这些极端情况，首先应该考虑调整应用，实在没有办法的话，可以考虑将事务的隔离模式更改为read committed。

(4) serializable

serializable也称为序列化，最高级别的锁，它解决了幻读，它将锁施加在所有访问的数据上。

该锁将把普通的SELECT语句默认改成SELECT...LOCK IN SHARE MODE。即为查询语句涉及的数据加上共享锁，阻塞其他事务修改真实数据。

如下的命令语句可查询当前的事务隔离级别。

```
mysql> show variables like '%tx%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| tx_isolation | REPEATABLE-READ |
+-----+-----+
1 row in set (0.00 sec)
```

或者

```
mysql> SELECT @@global.tx_isolation, @@session.tx_isolation;
+-----+-----+
| @@global.tx_isolation | @@session.tx_isolation |
+-----+-----+
| REPEATABLE-READ | REPEATABLE-READ |
+-----+-----+
```

设置事务隔离级别的语法格式如下。

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

在配置文件内修改mysqld节的transaction-isolation参数的方式如下。

```
[mysqld]
transaction-isolation = {READ-UNCOMMITTED | READ-COMMITTED
| REPEATABLE-READ | SERIALIZABLE}
```



注意 如上配置文件transaction-isolation选项的级别名中有连字符，但SET TRANSACTION语句的级别名中则没有连字符。

不建议更改InnoDB的事务隔离级别。一些传统的商业数据库，如Oracle，使用了类似read-commited的隔离级别。但由于绝大部分场景下，MySQL的用户都使用默认的隔离级别repeatable read，此隔离级别下的使用验证会比其他隔离级别完善得多，官方可能也不会对非默认隔离级别进行充分的验证，或者存在不完善支持的行为。

4.7.4 InnoDB的行锁

1.概述

一般来说，我们没有必要针对InnoDB引擎的表使用LOCK TABLES锁定记录。正常情况下，使用InnoDB支持的行锁技术就能够处理绝大部分场景。

行级锁定的优点如下。

- 当在很多线程中访问不同的行时只存在少量锁定冲突。

- 回滚时只有少量的更改。

- 可以长时间锁定单一的行。

数据库的锁定技术往往是基于索引来实现的，InnoDB也不例外。如果我们的SQL语句里面没有利用到索引，那么InnoDB将会执行一个全表扫描，锁定所有的行（不是表锁）。

锁过多的行，增加了锁的竞争，降低了并发率，所以建立索引是很重要的，InnoDB需要索引来过滤（在存储引擎层中）掉那些不需要访问的行。

这里举例说明如下。

```
mysql> SET AUTOCOMMIT=0;
mysql> BEGIN;
mysql> SELECT actor_id FROM sakila.actor WHERE actor_id < 5
    AND actor_id >< 1 FOR UPDATE;
+-----+
| actor_id |
+-----+
| 2 |
| 3 |
| 4 |
+-----+
mysql> EXPLAIN SELECT actor_id FROM sakila.actor
    WHERE actor_id < 5 AND actor_id >< 1 FOR UPDATE;
+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | key | Extra |
+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | actor | range | PRIMARY | Using where; Using index |
+-----+-----+-----+-----+-----+-----+
```

如上案例所示，EXPLAIN输出的执行计划里type为range，即索引范围查找，MySQL会锁定1~4行，而不是2~4行，为什么呢？因为InnoDB存储引擎的优化器会忽略最后一个范围查找之后的条件（即条件actor_id<>1），所以对于查询“SELECT actor_id FROM sakila.actor WHERE actor_id<5 AND actor_id><1 FOR UPDATE;”还锁定了actor_id=1的行。即MySQL执行的计划是InnoDB存储层先进行索引范围查找，扫描了1、2、3、4行的记录，然后才返回给MySQL Server层，Server层再用WHERE条件去过滤掉行1的记录（注意EXPLAIN执行计划里的“Using where”），MySQL Server层并没有告诉InnoDB引擎需要过滤掉行1的记录。

2.几种行锁技术

InnoDB有几种不同类型的行锁技术，如记录锁（record lock）、间隙锁（gap lock），和next-key锁。

记录锁（index-row locking）：这是一个索引记录锁。

它是建立在索引记录上的锁，很多时候，扫描一个表，由于无索引，往往会导致整个表被锁住，建立合适的索引可以防止扫描整个表。

间隙锁：这是施加于索引记录间隙上的锁。

next-key锁：记录锁加间隙锁的组合。也就是说next-key锁技术包含了记录锁和间隙锁。

有时在开发过程中我们会发现，在INSERT的时候会锁定相邻的键。其实这是一个next-key锁技术。MySQL使用这个技术来避免幻读。

当同一查询在不同时间产生不同的结果集时，在事务内发生所谓的幻读。例如，如果SELECT执行两次，但第二次返回第一次未返回的行，则该行为“幻影”行。MySQL默认的是repeatable read，但更进一步，它使用next-key锁来防止发生幻读现象。

例如，对于语句“SELECT*FROM child WHERE id>100 FOR UPDATE;”，如果child表内有id=90、id=102，那么gap就是90–102了，锁住这个gap，才能防止在你的事务执行期间，其他用户插入id=101的记录，造成幻读。当然，你所在的当前事务是允许插入id=101的记录的，这样其实变通实现了唯一性的检查。

如果需要禁用next-key锁，可以设置事务隔离级别为read committed级别，或者设置参数innodb_locks_unsafe_for_binlog=1。

在开发数据库程序的时候必须要清楚的一点，当我们执行数据操作的时候，很可能会导致间隙锁。由于间隙锁锁定的范围比较大，会导致可并发执行的事务数受到限制。

还有一点需要留意的是，next-key锁是为了防止发生幻读，而只有repeatable read及以上隔离级别才能防止幻读，所以在read committed隔离级别下面没有next-key锁这一说法。

3.等待行锁超时

有时我们在慢查询日志中会看到一些很耗时的查询，但单独执行却很快，此时有可能就是因为该查询因等待InnoDB行锁而超时。

如下是生产环境的一个示例。

```
mysql> DESC tbl_rankings;
+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| ranking | int(11) | NO | PRI | NULL | auto_increment |
| rid | int(11) | YES | UNI | NULL | |
| historyRanking | int(11) | YES | | 0 | |
| status | int(11) | YES | | 0 | |
+-----+-----+-----+-----+
4 rows in set(0.00 sec)
mysql> SHOW CREATE TABLE tbl_rankings \G;
***** 1. row *****
Table: tbl_rankings
Create Table: CREATE TABLE `tbl_rankings` (
  `ranking` int(11) NOT NULL AUTO_INCREMENT COMMENT '排名,自增',
  `rid` int(11) DEFAULT NULL COMMENT '角色
id,唯一约束',
  `historyRanking` int(11) DEFAULT '0' COMMENT '历史排名',
  `status` int(11) DEFAULT '0',
  PRIMARY KEY (`ranking`),
  UNIQUE KEY `ridIndex` (`rid`)
) ENGINE=InnoDB AUTO_INCREMENT=209315 DEFAULT CHARSET=utf8 COMMENT='龙虎榜核心表'
1 row in set (0.00 sec)
mysql> select * from tbl_rankings limit 30;
+-----+-----+-----+-----+
| ranking | rid | historyRanking | status |
+-----+-----+-----+-----+
| 1 | 551915 | 24 | 0 |
| 2 | 350149 | 9 | 0 |
| 3 | 229709 | 35 | 0 |

```

表tbl_rankings的rid列上创建了唯一索引。

下面新建两个会话，分别执行如下的操作。

会话1执行命令“update tbl_rankings set rid=0 where ranking=1;”，此时会锁住ranking=1的记录里的rid值索引（原来的

rid=551915)。该会话不允许其他的会话设置rid=551915，同样的，也不允许其他会话设置rid=0。

会话2运行命令“update tbl_rankings set rid=551915 where ranking=2;”，此时会话2会被阻塞，并且一直等待。在50s后超时并在慢查询日志里记录超时信息。

```
# Time: 120615 10:45:35
# User@Host: root[root] @ localhost []
# Query_time: 50.727669 Lock_time: 0.000107 Rows_sent: 0 Rows_examined: 1
SET timestamp=1339728335;
update tbl_rankings set rid=551915 where ranking=2;
```

可以看到索引上有锁。

```
show innodb status \G;
mysql thread id 2211, query id 2708 localhost root Updating
update tbl_rankings set rid=551915 where ranking=2
----- TRX HAS BEEN WAITING 8 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 5254 n bits 1192 index `ridIndex` of table `momo`.`tbl_rankings` trx id 0 7722 lock mode S
waiting
Record lock, heap no 176 PHYSICAL RECORD: n_fields 2; compact format; info bits 32
 0: len 4; hex 80086beb; asc k ;; 1: len 4; hex 80000001; asc ;
```

4.MVCC简要介绍

单纯靠行级别的锁，是不可能实现好的并发性的，MySQL InnoDB还需要配合MVCC（Multiversion Concurrency Control）技术来提供高并发访问。在很多情况下MVCC可以不需要使用锁，即可实现更新数据时的无阻塞读。

在常用的事务隔离级别read committed和repeatable read级，都应用了MVCC技术。

InnoDB官方建议的默认的事务隔离级别是可重复读（repeatable read），意思是在同一个事务内，对于同一个查询请求，多次执行，获得的记录集是相同的。这样，事务内的查询会看到一致性的数据，而不管它执行了多久，这一般是通过保存数据的快照来实现的。MVCC会保存某个时间点上的数据快照。这就意味着事务可以看到一个一致的数据视图，不管它们还需要运行多久。这同时也意味着不同的事务在同一个时间点看到的同一个表的数据可能是不同的。

具体的更详细的介绍，请参考官方文档。

4.8 死锁

死锁是指两个或两个以上的事务在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法进行下去。我们可以用图4-3来说明死锁的形成。

图4-3中，进程P1、P2都需要申请额外的资源，P1持有资源R2，需要申请资源R1，P2持有资源R1，需要申请资源R2，此时就会形成一个闭环，两个进程都无法继续运行。

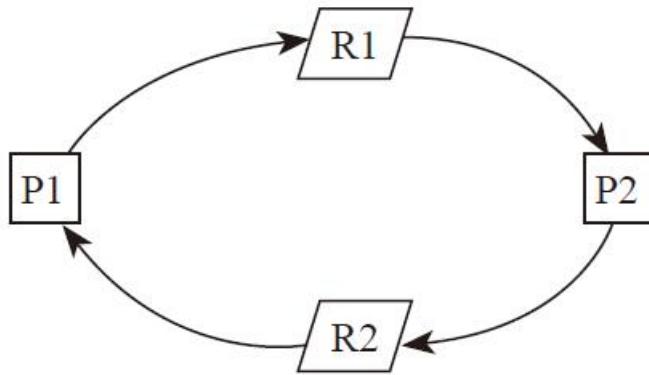


图4-3 死锁图

理论上，产生死锁有4个必要条件。

- 禁止抢占（no preemption）

- 持有和等待（hold and wait）

- 互斥（mutual exclusion）

- 循环等待（circular waiting）

预防死锁就是至少破坏这4个条件中的一项，即破坏“禁止抢占”、“持有等待”、“资源互斥”或“循环等待”。

实践中，处理死锁的方法大致分为两种。既可以检测死锁并进行修复，也可以对事务进行管理，使死锁永远都不可能形成。当存在死锁时，对该状态进行修复以使所有涉及的事务都能继续执行通常是不可能的。因此，至少其中的一个事务必须终止并重新开始。InnoDB会自动检测死锁。据官方文档可知，目前InnoDB处理死锁的机制是：发现有循环等待的现象，立即回退（rollback）开销更小的事务，也就是插入、修改、删除了更少记录的事务。

对于MySQL死锁的解决，通常有如下方法。

- 经常提交你的事务。小事更多地倾向于冲突。

- 以固定的顺序访问你的表和行。这样事务就会形成定义良好的查询并且没有死锁。

- 将精心选定的索引添加到你的表中。这样你的查询就只需要扫描更少的索引记录，并且因此也可以设置更少的锁定。

- 不要把无关的操作放到事务里面。

- 在并发比较高的系统中，不要显式加锁，特别是在事务里显式加锁。如SELECT...FOR UPDATE语句，如果是在事务里（运行了START TRANSACTION或设置了autocommit等于0），那么就会锁定所查找到的记录。

·尽量按照主键/索引去查找记录，范围查找增加了锁冲突的可能性，也不要利用数据库做一些额外的计算工作。比如有些读者会用到“SELECT...WHERE...ORDER BY RAND();”这样的语句，由于类似这样的语句用不到索引，因此将导致整个表的数据都被锁住。

·优化SQL和表设计，减少同时占用太多资源的情况。比如说，减少连接的表，将复杂SQL分解为多个简单的SQL。

4.9 其他特性

4.9.1 临时表

临时表指的是CREATE TEMPORARY TABLE命令创建的临时的表，临时表只对当前连接可见，对其他连接不可见，结束连接或中断，数据表（数据）将丢失。也就是说，在短连接的情况下，断开连接后，这个表就自动删除了。如果是长连接的话，则需要自己先初始化下表。

我们常使用临时表来存储一些中间结果集，如果需要执行一个很耗资源的查询或需要多次操作大表，那么把中间结果或小的子集放到一个临时表里，可能会有助于加速查询。

创建了临时表之后，如果运行SHOW TABLES、SHOW OPEN TABLES、SHOW TABLE STATUS命令及在INFORMATION_SCHEMA库中都将看不到临时表，这不是Bug，而是设计就是如此。

临时表支持多种存储引擎，如HEAP、MyISAM、InnoDB，当设置ENGINE=HEAP时，就会具有内存表的属性，即表的大小超过max_heap_table_size时就会报错。我们需要注意的是，在已有的内存表上设置该变量是没有效果的，除非用CREATE TABLE、ALTER TABLE、TRUNCATE TABLE等语句重新创建表。当然，重启也是可以生效的。

MySQL临时表也有一些限制。比如不能用RENAME来重命名一个临时表，可以用ALTER TABLE来代替。比如，在同一个查询语句中，你只能查找一次临时表。临时表的详细使用方法和相关限制请参考官方文档。

4.9.2 分区表

分区表是商业数据库的一项高级技术，MySQL从5.1版开始也支持分区表，分区表技术允许按照设置的规则，跨文件系统分配单个表的多个部分。实际上，表的不同部分在不同的位置被存储为单独的表。用户所选择的、实现数据分割的规则被称为分区函数，在MySQL中它可以是模数，或者是简单地匹配一个连续的数值区间或数值列表，或者是一个内部HASH函数，或者是一个线性HASH函数。

以笔者使用分区表的经验来看，分区表一直不太成熟，据说在MySQL 5.6以后才趋向成熟稳定，所以，不要轻易将分区表应用于生产环境。

如下命令将确定MySQL是否支持分区。

```
mysql> SHOW VARIABLES LIKE '%partition%';
+ Variable_name      + Value +
| have_partition_engine | YES |
```

可使用EXPLAIN命令查看是否过滤掉了不需要查询的分区，如“mysql>EXPLAIN PARTITIONS SELECT*FROM trb1\G”。

MySQL 5.1有如下的一些分区类型，RANGE分区、LIST分区、HASH分区、KEY分区和子分区。常用的存储引擎，如InnoDB、MyISAM、MEMORY都支持分区表。

RANGE分区的表是通过如下这种方式进行分区的，基于一个连续区间的列值，把多行分配给分区，例如某个时间段的值属于某个分区，某个数值范围的值应该属于某个分区。

MySQL中的LIST分区在很多方面都类似于RANGE分区。和按照RANGE进行分区的方式一样，每个分区都必须明确定义。它们的主要区别在于，LIST分区中每个分区的定义和选择是基于值列表的，而RANGE分区是从属于一个连续区间值的集合

的。

HASH分区是基于用户定义的表达式的返回值选择分区。它主要用来确保数据在预先确定了数目的分区中是平均分布的。在RANGE分区和LIST分区中，必须明确指定一个给定的列值或列值集合应该保存在哪个分区中；而在HASH分区中，MySQL将自动完成这些工作，你所要做的只是为将要被散列的列值指定一个列值或表达式，以及指定被分区的表将要被分割成的分区数量。

按照KEY进行分区类似于按照HASH进行分区，除了HASH分区使用的是用户自定义的表达式，而KEY分区的散列函数是由MySQL服务器提供的。

子分区是分区表中每个分区的再次分割。

以下是一些MySQL 5.1分区表的操作示例。

创建一个RANGE分区表，语句如下。

```
CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)
PARTITION BY RANGE( YEAR(purchased) ) (
    PARTITION p0 VALUES LESS THAN (1990),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (2000),
    PARTITION p3 VALUES LESS THAN (2005)
);
```

RANGE分区和LIST分区的操作示例如下。

(1) 删除分区（需要DROP权限）

```
ALTER TABLE tr DROP PARTITION p2;
```

如果需要调整分区，但不想丢失数据，那么可以重整分区。

```
ALTER TABLE ... REORGANIZE PARTITION;
```

(2) 增加分区

对于RANGE分区，只能从分区列表的最高端开始增加。

例如，对于如下的表使用ALTER TABLE...ADD PARTITION命令添加分区。

```
CREATE TABLE members (
    id INT,
    fname VARCHAR(25),
    lname VARCHAR(25),
    dob DATE
)
PARTITION BY RANGE( YEAR(dob) ) (
    PARTITION p0 VALUES LESS THAN (1970),
    PARTITION p1 VALUES LESS THAN (1980),
    PARTITION p2 VALUES LESS THAN (1990)
);
#增加一个分区。
ALTER TABLE members ADD PARTITION (PARTITION p3 VALUES LESS THAN (2000));
```

如果要加入1960分区则会报错。

```
mysql> ALTER TABLE members
      ADD PARTITION (
          PARTITION n VALUES LESS THAN (1960));  #报错
```

可以增加多个分区，例如：

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    hired DATE NOT NULL
)
PARTITION BY RANGE( YEAR(hired) ) (
    PARTITION p1 VALUES LESS THAN (1991),
    PARTITION p2 VALUES LESS THAN (1996),
    PARTITION p3 VALUES LESS THAN (2001),
    PARTITION p4 VALUES LESS THAN (2005)
);
ALTER TABLE employees ADD PARTITION (
    PARTITION p5 VALUES LESS THAN (2010),
    PARTITION p6 VALUES LESS THAN MAXVALUE
);
```

(3) 调整分区

如果想要调整分区，比如在分区列表中加入一个分区，或者忘记增加分区了，所有的数据都落入了最后一个分区，这时想重新定义最后的分区，那么你可以使用重整分区的功能。

```
ALTER TABLE members
REORGANIZE PARTITION p0 INTO (
    PARTITION n0 VALUES LESS THAN (1960),
    PARTITION n1 VALUES LESS THAN (1970)
);
```

(4) 合并分区

还可以合并分区，注意，对于RANGE分区，合并的分区必须是相邻的分区。

```
ALTER TABLE members REORGANIZE PARTITION s0,s1 INTO (
    PARTITION p0 VALUES LESS THAN (1970)
);
ALTER TABLE members REORGANIZE PARTITION p0,p1,p2,p3 INTO (
    PARTITION m0 VALUES LESS THAN (1980),
    PARTITION m1 VALUES LESS THAN (2000)
);
```

对于LIST分区，如果新加分区中的元素和旧的分区有冲突，那么可以先添加分区（只有没有冲突的元素），然后重整分区。

```
ALTER TABLE tt ADD PARTITION (PARTITION np VALUES IN (4, 8));
ALTER TABLE tt REORGANIZE PARTITION p1,np INTO (
    PARTITION p1 VALUES IN (6, 18),
    PARTITION np VALUES IN (4, 8, 12)
);
```

(5) 重建分区 (rebuilding partition)

相当于删除所有的数据，再INSERT所有的数据，整理碎片可用，语句如下。

```
ALTER TABLE t1 REBUILD PARTITION p0, p1;
```

(6) 优化分区 (optimizing partition)

如果某个分区中删除了大量数据，或者频繁修改了表（有可变字段），那么可以考虑优化该分区，语句如下。

```
ALTER TABLE t1 OPTIMIZE PARTITION p0, p1;
```

(7) 分析分区 (analyzing partition)

如下这个命令将分析分区的key分布信息。

```
ALTER TABLE t1 ANALYZE PARTITION p3;
```

(8) 检查分区 (checking partition)

检查表，如果坏了，则用REPAIR命令修复，语句如下。

```
ALTER TABLE trb3 CHECK PARTITION p1;
```

(9) 修复分区 (repairing partition)

修复分区的语句如下。

```
ALTER TABLE t1 REPAIR PARTITION p0,p1;
```

如果需要对所有分区进行操作，那么可加入**All**关键字，语句如下。

```
mysql> ALTER TABLE hotspace_0 ANALYZE PARTITION ALL;
```

MySQL 5.1 RANGE分区有如下一些注意事项。

- 同一个分区表中的所有分区必须使用同一个存储引擎，并且存储引擎要和主表的存储引擎保持一致。
- 有MAXVALUE值之后，直接加分区是不可行的。
- RANGE的分区方式在加分区的时候，只能从最大值的后面添加，而在最大值的前面不可以添加。
- 分区键必须包含在主键里面。

如上列了一些常用的分区表操作，主要是基于MySQL 5.1的版本，MySQL分区表的技术在不断发生改变，而且不同版本的变化也比较大，一些限制和弱点不断地在新的版本中取消或完善，如果大家要使用分区表，建议参考官方文档，采用合适的方法。

分区包括如下一些优点。

- 与单个磁盘或文件系统分区相比，可以存储更多的数据。表分区物理上被存储为单独的表，所以可以把分区存储到不同的磁盘或文件系统中。在现实生产环境中，这样使用还是比较少见的。选择分区表更常见的是基于业务的需要，是否能够更高效地查询数据和维护数据。
- 对于那些已经失去了保存意义的数据，通常可以通过删除与那些数据有关的分区，很容易地删除掉那些数据。
- 一些查询可以得到极大的优化，这主要是借助于满足一个给定WHERE语句的数据可以只保存在一个或多个分区内，这样在查找时就不用再查找剩余的其他分区了。

分区表也有如下一些不足之处。

MySQL的分区表不像Oracle那么灵活和成熟可靠，也不像Oracle那样可以有全局的索引，MySQL的索引对于每个表来说都是单独的。这样如果有跨越多个分区的查找，那么效率可能就会有问题。

一般来说，系统设计人员在碰到一些有“分区”特征的数据时，可能就会倾向于分区，比如一些按时间记录的流水账，这种想法本身并没有错，但是需要明白的是，分区表不能跨越MySQL的实例，也就是说不能超过单机，扩展性仍然有限，而且由于分区表的不成熟，可能会给整个系统带来隐患。这里有一些通用的建议。

- 1) 只有大表才可能需要分区，几百万笔记录的表并不算大，对于一些高配置的数据库主机，几千万甚至上亿条数据的表也不算大。
- 2) 分区数不能过多，很难想象大于500的分区数。
- 3) 查询的时候，不要跨越多个分区，建议最多跨越1~2个分区。
- 4) 索引的列应该是分区的列，或者有其他条件限制的分区，否则访问所有分区上面的索引进行查找，开销会比较大。

笔者个人不推荐在生产环境中使用分区表，基于的理由如下。

- 1) 就目前的生产环境来说，分区表还只是一项不是很成熟的技术：据官方发布的Bug升级记录可知，5.1、5.5长期以来修复了很多Bug。虽然Oracle公司也在不断完善分区表，官方宣称在MySQL 5.6已经成熟了很多，但如果要使用分区表，仍然建议事先经过充分的测试和验证。
- 2) 目前已知的官方5.1版本的内存分配机制有一定的问题，有内存碎片，笔者曾经发现在生产环境里使用了分区表的实例，内存会不断上升。
- 3) MySQL分区表的管理性、可维护性还存在一些问题。如果数据不能单独分布在一两个有限的分区内，那么查询性能往往更差。因为扫描多个分区将比扫描原来的一张表慢得多。
- 4) 使用分区表往往需要更多的技术考虑，需要更多的经验，且不一定适合未来的业务需求。
- 5) 一般从应用层分表是很成熟的技术，各种大型项目中更多的是从应用层分片数据。

4.9.3 存储过程、触发器、外键

1. 存储过程/函数

MySQL在MySQL 5.0版之后支持存储过程。

存储程序和函数是用CREATE PROCEDURE和CREATE FUNCTION语句创建的子程序。

(1) 存储过程的使用

由于存储过程包含多个语句，因此需要在MySQL客户端使用另外的分隔符，语句如下。

```
DELIMITER //
DELIMITER $$ 
CREATE PROCEDURE p1 () SELECT * FROM t; //
```

声明的变量，如果没有DEFAULT子句，那么变量的值默认为NULL，如下例中a变量的默认值即为NULL。

```
CREATE PROCEDURE p10 () 
BEGIN
DECLARE a, b INT DEFAULT 5;
INSERT INTO t VALUES (a);
SELECT s1 * a FROM t WHERE s1 >= b;
END; //
```

作用域BEGIN...END之内的声明离开作用域就失效了，例如如下的语句。

```
mysql> DELIMITER //  
mysql>  
CREATE PROCEDURE p11 ()  
BEGIN  
    DECLARE x1 CHAR(5) DEFAULT 'outer';  
    BEGIN  
        DECLARE x1 CHAR(5) DEFAULT 'inner';  
        SELECT x1;  
    END;  
    SELECT x1;  
END; //  
mysql> DELIMITER ;  
mysql> call p11();
```

显示的值将是outer。

存储过程的name不区分大小写，可以使用database_name.procedure_name来调用。

存储过程支持常见的控制体结构，比如IF语句、WHEN条件分支语句、WHILE...DO循环语句。

IF语句的示例如下。

```
CREATE PROCEDURE p12 (IN parameter1 INT)  
BEGIN  
DECLARE variable1 INT;  
SET variable1 = parameter1 + 1;  
IF variable1 = 0 THEN  
INSERT INTO t VALUES (17);  
END IF;  
IF parameter1 = 0 THEN  
UPDATE t SET s1 = s1 + 1;  
ELSE  
UPDATE t SET s1 = s1 + 2;  
END IF;  
END; //
```

CASE...WHEN语句的示例如下，满足条件值后执行相应的分支语句。

```
CREATE PROCEDURE p13 (IN parameter1 INT)  
BEGIN  
DECLARE variable1 INT;  
SET variable1 = parameter1 + 1;  
CASE variable1  
WHEN 0 THEN INSERT INTO t VALUES (17);  
WHEN 1 THEN INSERT INTO t VALUES (18);  
ELSE INSERT INTO t VALUES (19);  
END CASE;  
END; //
```

WHILE...DO语句的示例如下，满足条件后执行相应的循环体。

```
CREATE PROCEDURE p14 ()  
BEGIN  
DECLARE v INT;  
SET v = 0;  
WHILE v < 5 DO  
INSERT INTO t VALUES (v);  
SET v = v + 1;  
END WHILE;  
END; //
```

REPEAT...UNTIL语句的示例如下，首先执行循环体，再判断条件，即至少执行相应的一次。

```
CREATE PROCEDURE p15 ()  
BEGIN  
DECLARE v INT;  
SET v = 0;  
REPEAT  
INSERT INTO t VALUES (v);  
SET v = v + 1;  
UNTIL v >= 5          #注意后面没有分号。
```

```
END REPEAT;
END; //
```

此外，还支持标号，示例如下。

```
CREATE PROCEDURE p16 ()
BEGIN
    DECLARE v INT;
    SET v = 0;
    loop_label: LOOP
        INSERT INTO t VALUES (v);
        SET v = v + 1;
        IF v >= 5 THEN
            LEAVE loop_label;
        END IF;
    END LOOP;
END; //
```

以下是综合上述控制体的一个示例。

```
CREATE PROCEDURE p21
(IN parameter_1 INT, OUT parameter_2 INT)
LANGUAGE SQL DETERMINISTIC SQL SECURITY INVOKER
BEGIN
    DECLARE v INT;
    start_label: LOOP
        IF v = v THEN LEAVE start_label;
        ELSE ITERATE start_label;
        END IF;
    END LOOP start_label;
    REPEAT
        WHILE 1 = 0 DO BEGIN END;
        END WHILE;
    UNTIL v = v END REPEAT;
END; //
```

SQL SECURITY特征可以用来指定子程序是用创建子程序者的许可权限来执行，还是使用调用者的许可权限来执行。默认值是DEFINER。

·SQL SECURITY DEFINER: 按创建存储过程的用户的许可权限来执行。

·SQL SECURITY INVOKE: 按调用者的许可权限来执行。

不能在存储过程中再执行一些更改存储过程的操作，比如CREATE PROCEDURE、ALTER PROCEDURE等。

如下是一个创建存储过程的例子。

```
mysql> delimiter //
mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
BEGIN
    SELECT COUNT(*) INTO param1 FROM t;
END
//  
mysql> delimiter ;
mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @a;
+-----+
| @a   |
+-----+
| 3    |
+-----+
1 row in set (0.00 sec)
```

再来看下面这个例子。

```
CREATE PROCEDURE procedure1          /* name */
(IN parameter1 INTEGER)           /* parameters */
BEGIN                            /* start of block */
DECLARE variable1 CHAR(10);       /* variables */
IF parameter1 = 17 THEN          /* start of IF */
    SET variable1 = 'birds';      /* assignment */
ELSE
    SET variable1 = 'beasts';     /* assignment */
END IF;                          /* end of IF */
```

```
INSERT INTO table1 VALUES (variable1);      /* statement */
END                                         /* end of block */
```

存储过程的实际定义是存放在系统表mysql.proc中的，所以查看或备份存储过程也可以针对这个表来进行。

创建存储过程的时候可以保存sql_mode。如下示例将演示ansi模式。

```
mysql> set sql_mode='ansi';
Query OK, 0 rows affected (0.00 sec)
mysql> select 'a'||'b';
+-----+
| 'a'||'b' | 在
ansi模式下才可行。
+-----+
| ab     |
+-----+
1 row in set (0.00 sec)
mysql> set sql_mode='';
Query OK, 0 rows affected (0.00 sec)
mysql> select 'a'||'b';
+-----+
| 'a'||'b' | 
+-----+
|      0   |
+-----+
1 row in set, 2 warnings (0.00 sec)
mysql> show warnings;
+-----+-----+-----+
| Level | Code | Message
+-----+-----+-----+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'a' |
| Warning | 1292 | Truncated incorrect DOUBLE value: 'b' |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

ansi模式包含一些组合，比如REAL_AS_FLOAT、PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、ANSI等。

下面将实际创建一个存储过程，并查看是否保存了sql_mode，命令如下。

```
mysql> set sql_mode='ansi' //'
mysql> create procedure p3()select'a'||'b'//'
mysql> set sql_mode=''//'
mysql> call p3()//'
```

可以看到，在创建存储过程的时候，存储过程定义中保存了sql_mode的。所以虽然后来又设置了sql_mode，但是存储过程不会受到影响。我们可以运行命令SHOW CREATE PROCEDURE procedure_name来查看创建存储过程的代码，里面有sql_mode的信息。

以上对于存储过程的介绍比较粗略，由于篇幅所限，且MySQL存储过程并非必须要掌握的知识，因此这里仅列举一些代码，未做详细说明，大家如果有兴趣深入学习和编写存储过程，请参考官方文档。

(2) 对于复制的影响

CREATE PROCEDURE、CREATE FUNCTION、ALTER PROCEDURE和ALTER FUNCTION语句都将被写进二进制日志，CALL、DROP PROCEDURE和DROP FUNCTION也一样。

存储子程序（存储过程/函数）在复制中引发了很多问题，如果应用了存储过程，则复制可能就是不可靠的了。笔者认为主要原因在于它不是核心的功能但又足够复杂。对于一项大多数人都不使用的特性，如果你要使用，那么使用的时候一定要慎重。

(3) DETERMINISTIC定义

生产中，如果要创建存储过程/函数，往往需要添加DETERMINISTIC定义，否则可能会报错，我们需要在BEGIN关键字之前添加DETERMINISTIC，例如如下语句。

```
CREATE PROCEDURE procedure1
(IN parameter1 INTEGER)
/* name */
/* parameters */
DETERMINISTIC
BEGIN
```

如果程序或线程总是对同样的输入参数产生同样的结果，则可认为它是“确定的”，否则就是“非确定”的。如果既没有给定 DETERMINISTIC也没有给定NOT DETERMINISTIC， 默认的就是NOT DETERMINISTIC。

加上DETERMINISTIC关键字的目的是，确保我们的存储过程/函数不会导致复制不可靠。如果一个存储函数在一个诸如 SELECT这样不修改数据的语句内被调用，即使函数本身更改了数据，函数的执行也不会被写进二进制日志里。这个记录日志的行为会潜在地导致问题。

假设函数myfunc()定义如下。

```
CREATE FUNCTION myfunc () RETURNS INT
BEGIN
INSERT INTO t (i) VALUES (1);
RETURN 0;
```

按照上面的定义，下面的语句将修改表t，因为myfunc()修改表t，但是语句不会被写进二进制日志，因为它是一个SELECT 语句。

```
SELECT myfunc();
```

默认地，要想让一个CREATE PROCEDURE或CREATE FUNCTION语句被接受，那么必须明白地指定DETERMINISTIC、NO SQL或READS SQL DATA三者中的一个，否则会产生错误。

- DETERMINISTIC：确定的。
- NO SQL：没有SQL语句，当然也不会修改数据。
- READS SQL DATA：只是读取数据，当然也不会修改数据。

注意，子程序本身的评估是基于创建者的“诚实度”的，MySQL不会检查被声明为确定性的子程序是否不包含产生非确定性结果的语句。

我们也可以设置全局变量“SET GLOBAL log_bin_trust_routine_creators=1;”，这样就可以不用添加DETERMINISTIC关键字了。官方文档的解释是：若启用了二进制记录，则该变量适用。它控制是否可以信任程序的作者不会创建向二进制日志写入不安全事件的程序。如果设置为0（默认情况下），则不允许用户创建或修改保存的程序，除非他们不仅拥有CREATE ROUTINE或ALTER ROUTINE的权限还拥有SUPER的权限。设置为0还强制限制程序必须用DETERMINISTIC、READS SQL DATA或NO SQL 三者中的一个进行声明。如果将变量设置为1，那么MySQL不会对保存程序的创建强加限制。

（4）游标功能

存储过程和函数内均支持游标（cursor），其语法格式如下。

```
DECLARE cursor-name CURSOR FOR SELECT ...;
OPEN cursor-name;
FETCH cursor-name INTO variable [, variable];
CLOSE cursor-name;
```

示例如下。

```
CREATE PROCEDURE curdemo()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE a CHAR(16);
    DECLARE b,c INT;
    DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
    DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
    OPEN cur1;
    OPEN cur2;
    REPEAT
        FETCH cur1 INTO a, b;
        FETCH cur2 INTO c;
        IF NOT done THEN
            IF b < c THEN
                INSERT INTO test.t3 VALUES (a,b);
            ELSE
                INSERT INTO test.t3 VALUES (a,c);
            END IF;
        END IF;
    UNTIL done END REPEAT;
    CLOSE cur1;
    CLOSE cur2;
END
```

(5) 错误异常处理

语法格式如下。

```
DECLARE
{ EXIT | CONTINUE }
HANDLER FOR
{ error-number | { SQLSTATE error-string } | condition }
SQL statement
```

这个语句指定了每个可以处理一个或多个条件的处理程序。如果产生一个或多个条件，则指定的语句将被执行。对于一个CONTINUE处理程序，当前子程序的执行将在执行处理程序的语句之后继续。对于EXIT处理程序，当前的BEGIN...END复合语句的执行将被终止。

下面给出了一个示例。

```
mysql> CREATE TABLE test.t (s1 int,primary key (s1));
Query OK, 0 rows affected (0.00 sec)
mysql> delimiter //
mysql> CREATE PROCEDURE handlertdemo ()
-> BEGIN
->     DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
->     SET @x = 1;
->     INSERT INTO test.t VALUES (1);
->     SET @x = 2;
->     INSERT INTO test.t VALUES (1);
->     SET @x = 3;
-> END;
-> //
Query OK, 0 rows affected (0.00 sec)
mysql> CALL handlertdemo()//
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @x//
```

	@x
1	3

```
1 row in set (0.00 sec)
```

其中，SQLSTATE23000'是重复键的错误消息。

可以注意到，@x是3，这表明了MySQL被执行到了程序的末尾。如果“DECLARE CONTINUE HANDLER FOR SQLSTATE'23000'SET@x2=1;”这一行不存在，那么第二个INSERT因PRIMARY KEY强制而失败之后，MySQL会采取默认(EXIT)路径，并且SELECT@x会返回2。

异常处理中需要注意的是，不一定是当前的语句/游标会触发错误，程序体的其他部分也可能触发异常处理，使程序以一种我们不期望的方式来运行。例如对于SQLSTATE:02000(ER_SP_FETCH_NO_DATA)找不到数据。我们知道对于“SELECT...FROM table_name WHERE...”语句，可能会触发这个条件，但是如果SELECT...INTO语句查找不到记录，其实也会触发

SQLSTATE:02000。

2.触发器

对触发器的支持，使得InnoDB也具有了商业数据库的功能。但就笔者个人的使用经验而言，InnoDB触发器离传统商业数据库的成熟度还比较遥远。

下面给出了一个简单的示例，在该示例中，针对INSERT语句，将触发程序和表关联了起来。其作用相当于累加器，能够将插入表中某一列的值累加起来。

在下面的语句中，创建了一个表，并为该表创建了一个触发程序。

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
      FOR EACH ROW SET @sum = @sum + NEW.amount;
```

CREATE TRIGGER语句创建了与账户表相关的、名为ins_sum的触发程序。它还包括一些子句，这些子句指定了触发程序激活的时间、触发程序事件，以及激活触发程序时要做些什么，关键字BEFORE指明了触发程序的动作时间。在本例中，将在将每一行插入表之前激活触发程序。如果需要在事件发生后激活触发程序，则需要指定关键字AFTER。关键字INSERT指明了激活触发程序的事件。在本例中，INSERT语句将导致触发程序的激活。同样也可以为DELETE和UPDATE语句创建触发程序。跟在FOR EACH ROW后面的语句定义了每次激活触发程序时将要执行的程序，对于受触发语句影响的每一行执行一次。在本例中，触发的语句是简单的SET语句，负责将插入amount列的值累加起来。该语句将列引用为NEW.amount，意思是“将要插入到新行的amount列的值”。要想使用触发程序，将累加器变量设置为0，执行INSERT语句，然后查看变量的值，语句如下。

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----+
| Total amount inserted |
+-----+
| 1852.48                |
+-----+
```

在本例中，执行了INSERT语句后，@sum的值是14.98+1937.50-100，或1852.48。

要想销毁触发程序，可使用DROP TRIGGER语句。

```
mysql> DROP TRIGGER test.ins_sum;
```

3.外键

InnoDB支持外键约束。InnoDB定义外键约束的语法格式如下所示。

```
[CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
  REFERENCES tbl_name (index_col_name,...)
  [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
  [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

例如，如下是一个通过单列外键联系起的父表和子表，语句如下。

```
CREATE TABLE parent(id INT NOT NULL,
                     PRIMARY KEY (id)
) TYPE=INNODB;
CREATE TABLE child(id INT, parent_id INT,
                  INDEX par_ind (parent_id),
                  FOREIGN KEY (parent_id) REFERENCES parent(id)
                  ON DELETE CASCADE
```

```
) TYPE=INNODB;
```

InnoDB支持使用ALTER TABLE来移除外键。

```
ALTER TABLE yourtablename DROP FOREIGN KEY fk_symbol;
```

要使得重新导入有外键关系的表变得更容易操作，那么mysqldump会自动在dump输出文件中包含一个语句设置FOREIGN_KEY_CHECKS为0。这就避免了在dump文件被重新装载之时，因为约束而导入失败。我们也可以手动设置这个变量，语句如下。

```
mysql> SET FOREIGN_KEY_CHECKS = 0;
mysql> SOURCE dump_file_name;
mysql> SET FOREIGN_KEY_CHECKS = 1;
```

InnoDB不允许删除一个被FOREIGN KEY表约束引用的表，除非设置了SET FOREIGN_KEY_CHECKS=0。

外键约束使得程序员更不容易将不一致性引入数据库，而且设计合适的外键也有助于以文档方式记录表间的关系。但请记住，这些好处是以数据库服务器为执行必要的检查而花费额外的开销为代价的。服务器进行额外的检查会影响性能，对于某些应用程序而言，该特性并不受欢迎，应尽量避免（出于该原因，在一些主要的商业应用程序中，在应用程序级别上均实施了外键逻辑）。

查询外键信息的语句如下。

```
select CONSTRAINT_SCHEMA,CONSTRAINT_NAME,TABLE_NAME,COLUMN_NAME,REFERENCED_TABLE_SCHEMA,REFERENCED_TABLE_NAME,
REFERENCED_COLUMN_NAME
from information_schema.KEY_COLUMN_USAGE where referenced_table_schema is not null ;
```

使用外键应注意如下一些要点。

- 不存在服务器端外键关联检查时，应用程序本身必须处理这类关联事宜。
- 从具有外键的表中删除记录时，在缺少ON DELETE的情况下，一种解决方式是为应用程序增加恰当的DELETE语句。实际上，它与使用外键同样快，而且移植性更好。

4.建议

传统的观点认为，如果有一个重复执行的任务，而且这个任务需要检查、循环重复执行多条语句，但实际上不需要交互，那么我们使用存储过程会更高效，这样将不存在在客户端和服务器之间来回地传递信息。存储过程/触发器往往还是已经编译好了的，所以也会更快。在商业数据库实现比较完善的存储过程/触发器后，存储过程、外键及触发器这些特性，在传统行业也获得了大量的应用。

许多复杂的业务逻辑用存储过程来实现，还可以保证安全、进行权限控制、集中控制业务逻辑，客户端也可以大大简化。如果业务逻辑发生变更，只需要修改下存储过程即可，而不需要繁琐地升级大量的客户端，而且数据库服务器往往更强劲，执行得也更快更有效率，网络通信来回往返传输的开销也可以节省。所以，当数据量还没有大到RDBMS处理不了的时候，可以考虑使用存储过程，毕竟这是花了钱购买的，充分利用商业数据库的潜能往往可以获得比较好的收益。

但是，即使存储过程、外键、触发器有这么多的好处，在现实中却也存在许多问题，特别是互联网行业，使用的是开源免费的MySQL数据库，在当前海量数据的环境下，关系型数据库本身都需要NoSQL的补充，存储过程的使用也就受到了约束。随着业务规模的扩大，数据库会逐渐成为系统的瓶颈，而在客户端和数据库中间增加应用服务器（应用层）来实现业务逻辑，

用应用服务器（客户端）来确保数据的完整性和一致性，是伸缩性更好的方案。

如今的计算模式也已经和以前有了很大的不同，特别是在互联网环境中，相对廉价的PC服务器集群的大量应用，硬盘容量更大，价格更低，更倾向水平扩展，而没有必要把负荷都堆积到中心的数据库服务器之上。所以对于互联网应用，存储过程、外键及触发器这些特性也已不再凸显其重要性，许多项目基本不用。

而且对于大多数的程序员来讲，他们更熟悉语言框架，数据库更多的只是作为一个存储数据的容器。这也影响到了存储过程的使用。就现状而言，存储过程只能存在于较少的业务场景中。

下面就从不同的角度来分析下存储过程/触发器。

(1) 安全

理论上来说，业务逻辑和各种约束越靠近数据库就会越安全，也能最大化地充分利用数据库。但对于互联网行业的应用来说，一般没有那么高的数据安全性，也不需要很强的数据完整性和一致性，如果确实有非常严苛的数据一致性的需求，那么可以专门实现一个“数据访问层”，其他的应用都将通过它来访问数据库。

(2) 性能和扩展性

对于单线程而言，据官方资料表示，存储过程有20%的性能提升，但应用很少是单线程的，随着连接数的不断增加，存储过程对比直接的SQL并不见得有什么性能上的提升。此时系统的性能提升已经让位于多线程并发管理，而且随着连接数的继续增加，存储过程的性能可能还会降低。

MySQL的触发器只支持行级别（`for each row`）一种方式，对于大数据量表的处理，这种方式将会很无效。触发器没有WHEN条件，不能控制何时触发，可能会造成性能瓶颈，无谓地消耗资源。

外键对并发性能的影响比较大，因为每次修改数据都需要去另外一个表检查数据，需要获取额外的锁（以确保事务完成之前，父表的记录不会被删除），高并发的环境下很容易出现性能问题。而级联更新删除之类的特性也比我们正常执行批量更新删除之类的操作要慢得多（级联删除、更新是one by one的）。所以更好的办法是在应用层实现外键约束。

在应用层实现业务逻辑的网络通信的成本可能高了点，但这只是一个相对的概念，在距离很遥远的情况下，客户端和服务端通信的成本比较大，这个时候存储过程更显优势，但Web服务器和数据库服务器一般位于同一个集群的内网中，网络交互很快、很稳定，成本也很低。很多真正能提高效率的终极办法是使用缓存而不是在数据库中进行运算，靠数据库预编译或减少网络流量那点优化就可以了，那也说明性能要求原本就不高。

数据库实现存储过程、触发器和外键，很大一个背景就是数据库服务器很强大，传统行业一般是昂贵小型机，有非常强劲的处理能力，配备的是Oracle等商业产品，业务需求相对稳定，需要充分利用数据库的能力而不仅仅将它当作一个数据的容器。而互联网行业一般使用的是MySQL数据库，相对廉价的PC，业务增长的不确定性，甚至是爆炸式的增长，如果数据架构不足，数据库很可能会成为整个系统的瓶颈，数据库的资源一般会比较紧张（服务器和人），扩展性不强，成本更昂贵；而Web服务器相对来说更便宜，更容易水平扩展，把业务逻辑放到Web服务器上去实现可以保证系统有更好的伸缩性。

(3) 迁移

如果需要在不同的数据库产品之间进行迁移，虽然有一些文档和各种各样的迁移方案供我们选择，但存储过程、触发器的迁移却是一个难题。往往需要投入巨大的精力进行开发和测试，所以如果有数据库解耦的需求，就不应该使用存储过程。

(4) 升级、维护、诊断、调优

实际上，从设计的角度来看，逻辑封装很重要，不是存储过程那一点的封装，而是整个业务逻辑的封装。如果把业务逻辑分散在程序代码和存储过程两部分中，那么它实际上是业务碎片化，不利于表述业务逻辑，会造成后期阅读和维护的困难。

如果使用存储过程，往往会降低上线、升级的效率，DBA和研发需要高度协调。以前一般是分离的，或者升级代码，或者升级数据库结构，而现在则需要升级存储在数据库服务器上的代码，但DBA往往并不熟悉业务逻辑。

升级失败很难马上恢复，而且影响面太大。而升级Web服务器，可以逐台进行升级。一般情况下是可以做到逐台升级而不会导致异常的。

对于业务非常繁忙的系统，升级存储过程可能会导致系统出现异常，因为要升级的存储过程可能正被频繁访问，或者应用系统足够复杂，存储过程互相调用，因此升级单个存储过程需要特别小心，以免影响整个系统。

开发、测试环境和生产环境很可能会不一致，从而导致开发环境的存储过程、触发器需要经过修改，才能升级到生产环境，因为存储过程、视图和触发器附加了一些与生产环境不一致的信息。

存储过程、触发器备份恢复不方便。

MySQL不能临时禁用或启用触发器，因为这点在做数据迁移时，修复会比较麻烦，需要临时删除触发器，可能还会影响到生产环境。

由于存储过程或触发器不易测试，或者未做充分测试，一旦升级失败就可能会导致数据错误，因为已经事先删除了存储过程或触发器。

不易分析存储过程或触发器的性能，因为不能通过慢查询日志去分析存储过程或触发器的具体执行情况。慢查询日志里只记录了“call procedure_name();”这样简单的信息。

触发器可能会导致死锁。

以上只是列举一些问题，具体的使用过程中，MySQL的存储过程和触发器离商业产品的距离还比较远。MySQL的很多Bug都涉及了存储过程和触发器。存储过程对于复制的支持也不太好。

(5) 开发

存储过程并不是一种结构化良好的语言，对于习惯于面向对象编程的人而言，存储过程更加难以理解。代码的可读性和可维护性在工程上是很重要的，从这点来说，存储过程并不适合工程化的需要。

存储过程和触发器的调试都比较困难，也没有什么好的工具和方法。

一个表中同类型的触发器只能建立一个，这就可能会导致代码逻辑很复杂，不易阅读和维护，因为需要把很多不相关的逻辑都写在一个触发器代码内。

存储过程也有诸多限制，具体请参考官方文档<http://dev.mysql.com/doc/refman/5.1/en/stored-program-restrictions.html#stored-routines-trigger-restrictions>

如果没有完善的、一致的文档，开发人员往往会不熟悉（遗漏）数据库上的存储过程。

存储过程比较简单，功能也很有限，而程序代码却可以实现更多的功能，实现更复杂的业务逻辑。因此，笔者建议，一定要慎用存储过程，业务逻辑不要放在存储过程中。不要使用触发器。在良好的业务系统中应该尽量抛弃存储过程和触发器之类的东西。不要用外键，在高并发的情况下，外键会降低并发性，外键自身的维护性和管理性也欠佳。

4.9.4 视图

MySQL 5.0版以上的MySQL服务器提供了视图功能（包括可更新视图）。如下命令将创建一个视图。

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

注意视图（VIEW）并不会保存任何数据，查询视图返回的结果都是来自于基表存储的数据。视图一般不会用来提升性能，而是用来简化部分开发，进行权限限制。

表和视图将共享数据库中相同的名称空间，因此，数据库不能包含具有相同名称的表和视图。视图必须具有唯一的列名，不得有重复，就像基表那样。默认情况下，由SELECT语句检索的列名将用作视图列名。

可使用多种SELECT语句创建视图。视图能够引用基表或其他视图，还能使用联合、UNION和子查询。

视图可以简化一些操作，比如隐藏基表的复杂性，进行一些安全控制（基于列的权限控制），但如果使用不当，很可能会带来性能问题。

我们需要了解视图实现的机制。对于包含视图的SQL，优化器进行优化的机制有两种：MERGE和TEMPTABLE。

·TEMPTABLE：创建一个临时表，把视图的结果集放到临时表中，然后SQL操作这个临时表。

·MERGE：重写SQL，合并视图的SQL，这种方法更智能。

例如，新建一个视图test.v。

```
CREATE VIEW test.v AS SELECT * FROM t where a=1;
```

对于视图的查询语句：

```
select a,b,c from test.v where b=2;
```

第一种临时表的方式类似如下语句。

```
CREATE TEMPORARY TABLE TMP_a AS SELECT * FROM t where a=1;
select * from TMP_a where b=2;
```

这种方式必须先查出所有视图的数据，然后才能基于这个视图的数据进行查找。显然可能会有性能问题。同时，外部的WHERE条件也不能传递到内部视图的限制中，临时表上没有索引。

而用第二种方式，优化后的SQL类似如下语句。

```
SELECT * FROM t where a=1 and b=2;
```

MySQL将尽量使用第二种合并SQL的方式，但在很多情况下，由于研发人员编写的查询采用临时表的方式，因而导致性能很差。可以用EXPLAIN命令来确认，如果EXPLAIN的select_type输出显示DERIVED（查询结果来自一个衍生表），那么查询使用的是临时表的方式，示例如下。

```
root@localhost test>CREATE VIEW test_garychen_v AS SELECT * FROM test_garychen GROUP BY STAT_TIME;
root@localhost test>EXPLAIN SELECT * FROM test_garychen_v;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | PRIMARY | <derived2> | ALL | NULL | NULL | NULL | NULL | 110 |   |
| 2 | DERIVED | test_garychen | ALL | NULL | NULL | NULL | NULL | 5000 | Using temporary; Using filesort |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

使用临时表的方式，性能可能会变得很差，视图也没有索引，外部的**WHERE**条件也不会传递到内部的视图（类似于**UNION ALL**）中。如果两个视图相连接，那将无法利用索引，可能会导致严重的性能问题。所以需要小心编写查询，以免使用到临时表的机制。也就是说，视图里应尽量避免使用**GROUP BY**、**ORDER BY**、**DISTINCT**、聚集函数、**UNION**和子查询。

一些复杂的视图，若使用EXPLAIN命令显示执行计划，将会执行得很慢，因为EXPLAIN会实际执行和物化派生表（derived table）。

视图里隐藏了很多细节，研发人员可能会觉得这个表很简单，但实际上底层是很复杂的查询。如果认为这个视图很简单，那么可能将它当作一个简单查询频繁调用而不自知，从而导致性能问题。在生产环境中我们还发现，高并发的情况下，查询优化器将在planing和statistics阶段花费大量时间，甚至导致MySQL服务器停滞，所以即使使用的是merge的算法，也仍然可能导致严重的性能问题。



小结 本章介绍了范式和反范式，反范式对于开发中的大型项目很重要，我们有必要在项目中不断积累这方面的经验。关于慢查询日志，它不仅是DBA常用的工具，研发人员一样也需要熟练掌握。由于现实开发项目中存在一个很大的问题，缺少性能管理，本章也介绍了性能管理的一些概念和方法。总之，如果要开发高质量的项目，一定要深刻理解数据库，对于数据库的逻辑设计、物理设计、事务、锁等内容都需要深入理解。本章最后介绍了一些非核心的MySQL特性，对于非核心的MySQL特性的使用，一定要慎重对待。

第5章 开发技巧

本章将介绍一些和数据库相关的开发技巧。由于开发领域很广，这里只选取部分比较常见的小技巧。

5.1 存储树形数据

有时我们需要保存一些树形的数据结构，比如组织架构、话题讨论、知识管理、商品分类，这些数据存在一种递归关系，很多研发人员想到的第一个解决方案往往是记录每个节点的父节点，例如以下的评论表。

```
CREATE TABLE comments (
    comment_id int(10) NOT NULL,
    parent_id int(10) DEFAULT NULL,
    comment text NOT NULL,
    PRIMARY KEY (comment_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

实际数据类似于表5-1所示。

表5-1 comment表的实际数据（记录了父节点信息）

comment_id	parent_id	comment
1	0	这本书不错
2	1	此书作者和译者的视野颇为广阔；在思想上更大胆
3	1	我在犹豫要不要买
4	2	说得有道理
5	3	值得购买，内容比较契合目前的数据发展潮流
6	5	封面和纸张设计、做工感觉有点粗糙了
7	0	在当下这个大数据时代，这本书一定要看

(续)

comment_id	parent_id	comment
8	3	该书气味之大，装帧之差实属罕见
9	7	这本书必须要看
10	7	比较实用，实践性比较强

如果采用这样的结构，当一篇帖子回复讨论的内容很多的时候，就需要编写复杂的代码递归检索很多记录，查询的效率就会很低。如果数据量不大、讨论内容相对固定，数据的层次较少，那么采用这样的结构就会是简单的、清晰的，这种情况下此结构还是合适的；但如果数据量很大，查询就会变得很复杂。下面介绍两种更通用，扩展性更好的解决方案：路径枚举和闭包表。

(1) 路径枚举

对于如表5-1所示的表结构，可以增加一个字段path，用于记录节点的所有祖先信息。记录的方式是把所有的祖先信息组织成一个字符串。类似于表5-2所示的形式。

表5-2 comment表的数据（记录了父节点及祖先信息）

comment_id	parent_id	path	comment
1	0	1/	这本书不错
2	1	1/2	此书作者和译者的视野颇为广阔；在思想上更大胆
3	1	1/3	我在犹豫要不要买
4	2	1/2/4	说得有道理
5	3	1/3/5	值得购买，内容比较契合目前的数据发展潮流
6	5	1/3/5/6	封面和纸张设计、做工感觉有点粗糙了
7	0	7/	在当下这个大数据时代，这本书一定要看
8	3	1/3/8	该书气味之大，装帧之差实属罕见
9	7	7/9	这本书必须要看
10	7	7/10	比较实用，实践性比较强

因为路径(path)字段包含了该节点的所有祖先信息，所以可以轻易地获取某个节点的所有祖先节点，可以用程序先获取path字符串，然后再使用切割字符串的函数处理得到所有的祖先节点。

如果要查找某个节点的所有后代，例如查找comment_id等于3的所有后代，可以使用如下的查询语句。

```
SELECT * FROM comments WHERE path LIKE  
1/3/_%  
;
```

如果要查找下一层子节点，可以使用如下的查询语句

```
SELECT * FROM comments WHERE path REGEXP ^  
^1/3/[0-9]+/$  
;
```

插入操作也比较简单，只需要复制一份父节点的路径，并将新节点的ID值（comment_id）添加到路径末尾就可以了。

枚举路径的方式使得查询子树和祖先都变得更加简单，查看分隔符即可知道节点的层次，虽然冗余存储了一些数据，应用程序需要额外增加代码以确保路径信息的正确性，但这种设计的扩展性更好，更能适应未来数据的不断增长。表5-2中，仍然保留了parent_id列，是为了使一些操作更加方便，也可以用来校验路径信息是否正确。

(2) 闭包表

闭包表也是一种通用的方案，它需要额外增加一张表，用于记录节点之间的关系。它不仅记录了节点之间的父子关系，也记录了树中所有节点之间的关系。

使用如下命令语句新建表path

```
CREATE TABLE path (  
ancestor int(11) NOT NULL,  
descendant int(11) NOT NULL,  
PRIMARY KEY (ancestor,descendant)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

ancestor表示祖先，descendant表示后代，存储的是comment_id值。

数据类似于表5-3所示。

表5-3 path表的数据（记录了所有节点之间的关系）

ancestor	descendant	ancestor	descendant	ancestor	descendant	ancestor	descendant
1	4	2	4	4	4	7	9
1	2	3	3	3	3	7	10
1	6	3	5	5	6	8	8
1	8	3	6	6	6	9	9
2	2	3	8	7	7	10	10

有了如表5-3所示的完整的节点间关系，查找后代节点、祖先节点也变得更容易，比如，如果要统计comment_id等于3的所有后代（不包括其自身），可以直接搜索path表祖先是3的记录即可得到，搜索语句如下。

```
SELECT COUNT(*) FROM path WHERE ancestor=3 AND descendant <> 3;
```

为了更方便地查询直接父节点/子节点，可以增加一个path_length字段以表示深度，节点的自我引用path_length等于0，到它的直接子节点的path_length等于1，再下一层为2，以此类推。

如上所述的数据结构，新增了一个表，用于存储节点之间的信息，是一种典型的“以空间换时间”的方案，而且一个节点可以属于多棵树。相对于路径枚举，闭包表的节点关系更容易维护。其他的操作如删除、插入等这里不再赘述，有兴趣的读者可以在网上查找“闭包表”的相关案例深入学习。

5.2 转换字符集

如果我们要修改某个表的字符集，比如A表的字符集原来是gbk，现在要将其修改为utf8，一般有以下3种方法。

(1) 直接在mysql命令行下完成

步骤如下

1) 建立一个临时表B，字段类型和A一致，但字符集是utf8，即表定义中DEFAULT CHARSET=utf8。

2) INSERT INTO B SELECT*FROM A;

3) DROP TABLE A;

4) RENAME B TO A;

(2) 使用mysqldump工具完成

首先导出数据，默认mysqldump导出的dump转储文件为utf8编码的文件，有删除表、创建表的语句。然后修改dump转储文件，将创建表语句里的表或列的字符集定义修改为utf8。最后重新导入此文件即可。

(3) 使用ICONV命令转换文件编码

步骤如下

1) 以gbk字符集导出数据，不导出表定义。

```
mysqldump -t -uroot -p database_name table_name1 table_name2 --default-character-set=gbk > a_gbk.sql
```

2) 使用iconv命令转换文件编码，将其转换为utf8编码。

```
iconv -f gbk -t utf-8 a_gbk.sql > a_utf8.sql
```

3) 修改文件中的相关字符集设置。

```
sed -i ''  
s/SET NAMES gbk/SET NAMES utf8/  
a_utf8.sql
```

4) 删除旧表（table_name1, table_name2），新建表（table_name1, table_name2），注意新建的表应该是utf8字符集。

5) 使用修改过的文件导入数据。

```
mysql -uroot -p database_name --default-character-set=utf8 < a_utf8.sql
```

在早期的数据库版本中还会有一种特殊情况，由于研发人员缺乏经验，选择了错误的数据库编码，采用latin1编码存储了中文数据。因为特殊的字符集设置，比如客户端（character_set_client）、连接（character_set_connection）、结果（character_set_results）的编码都是latin1编码，这样从程序到数据库就不会做任何转换，而将中文编码（例如gbk）以latin1编码的方式进行存储。也就是说，把每个汉字当成两个latin1字符进行存储，而且数据库发送结果的时候也是按照latin1的方式进行发送，而我们的页面接收到数据之后则是以中文的编码方式进行显示，因此能正常地显示。但是，这毕竟是一种错误的设置，数据存在重大隐患，解决方案是将latin1字符集转换为gbk字符集或utf8字符集。如下是具体的转换步骤。

(1) latin1转gbk

1) 导出数据库

```
mysqldump --default-character-set=latin1 -h xxx.xxx.xxx.xxx -u root -P 3306 -pxxxxxxxx db_name table_name > /usr/home/garychen/table_name.sql
```

2) 修改table_name.sql

将`/*!40101 SET NAMES latin1*/;`改为`/*!40101 SET NAMES gbk*/;`

将`DEFAULT CHARSET=latin1;`改为`DEFAULT CHARSET=gbk;`



注意 不同版本的`mysqldump`修改时可能稍有出入，建议实际修改时再确认下。

3) 导入数据库

```
mysql -uroot -pxxxxxxxx db_name < table_name.sql
```

(2) latin1转utf8

1) 导出数据库，同上面的例子。

2) 转换编码

```
iconv -t utf-8 -f gbk -c table_name.sql > table_name_u8.sql
```



注意 用`latin1`保存中文原本就是错误的做法，文件中存储的是错误的`latin1`编码，但实际上正确的`gbk`编码，所以这里输入编码（-f）应为`gbk`。

3) 修改table_name_u8.sql，使用`vi`或`sed`命令把`latin1`都改为`utf8`。

4) 导入数据库

```
mysql -uroot -pxxxxxxxx db_name < table_name_u8.sql
```

5.3 处理重复值

表或结果集有时会包含重复记录，需要采用某种方法标识这些重复的记录并移除它们，以下示例将说明如何预防重复值，以及如果存在重复记录时应如何移除它们。

(1) 防止表中出现重复的记录

可以使用主键或唯一索引来防止出现重复的记录。例如，下表person_tbl允许出现first_name和last_name组合相同的记录。

```
CREATE TABLE person_tbl
(    first_name CHAR(20),      last_name CHAR(20),      sex CHAR(10));
```

可以设置(last_name, first_name)为主键，以确保不出现重复记录，语句如下。

```
CREATE TABLE person_tbl
(    first_name CHAR(20) NOT NULL,    last_name CHAR(20) NOT NULL,    sex CHAR(10),    PRIMARY KEY (last_name, first_name));
```

也可以设置唯一索引，来强制记录是唯一的，语句如下。

```
CREATE TABLE person_tbl
(    first_name CHAR(20) NOT NULL,    last_name CHAR(20) NOT NULL,    sex CHAR(10)    UNIQUE (last_name, first_name));
```

对于可能出现重复的记录，我们可以考虑使用INSERT IGNORE语句。如果插入的记录并没有和现存的记录发生冲突，则正常插入之；如果有重复冲突，那么INSERT IGNORE将会告诉MySQL丢弃这条记录，且不报错。如下面这个例子。

```
mysql> INSERT IGNORE INTO person_tbl (last_name, first_name)
VALUES ('Jay', 'Thomas');
mysql> INSERT IGNORE INTO person_tbl (last_name, first_name)
VALUES ('Jay', 'Thomas');
```

还可以考虑采用REPLACE语句，如果记录是新的，那么它等同于INSERT。如果插入的是一个重复的记录，那么新记录将会替换旧的记录。

```
mysql> REPLACE INTO person_tbl (last_name, first_name)
VALUES ('Ajay', 'Kumar');
Query OK, 1 row affected (0.00 sec)
mysql> REPLACE INTO person_tbl (last_name, first_name)
VALUES ('Ajay', 'Kumar');
Query OK, 2 rows affected (0.00 sec)
```

综上所述，对于重复的记录，INSERT IGNORE仍然保留着现在的记录，丢弃新插入的记录。而REPLACE语句则会使用新的记录覆盖掉旧的记录。

(2) 统计和识别重复值

如下语句将查询和计算表person_tbl中(last_name, first_name)组合有重复的记录的数量。

```
mysql> SELECT COUNT(*) AS repetitions, last_name, first_name
FROM person_tbl
GROUP BY last_name, first_name
HAVING repetitions > 1;
```

(3) 从结果集中消除重复记录

使用DISTINCT关键字即可从结果集中消除重复记录。

```
mysql> SELECT DISTINCT last_name, first_name  
      FROM person_tbl  
     ORDER BY last_name;
```

或者，也可以使用GROUP BY子句。

```
mysql> SELECT last_name, first_name  
      FROM person_tbl  
     GROUP BY (last_name, first_name);
```

(4) 删除表中的重复记录

```
mysql> CREATE TABLE tmp SELECT last_name, first_name, sex  
           FROM person_tbl;  
        GROUP BY (last_name, first_name);  
mysql> DROP TABLE person_tbl;  
Mysql> ALTER TABLE tmp RENAME TO person_tbl;
```

还有一个不为人知的技巧，可以直接在一个有重复记录的表上加上主键或唯一索引，可使用ALTER IGNORE语句，命令如下。

```
mysql> ALTER IGNORE TABLE person_tbl  
    ADD PRIMARY KEY (last_name, first_name);
```

可以使用如上的方法消除重复记录，并且确保以后都有唯一约束。

也可以采用如下的方式，直接删除重复数据，如下语句将删除name相同的数据，其中id是主键。

```
DELETE t1 FROM table1 AS t1 JOIN table1 AS t2 ON t1.id>t2.id AND t1.name=t2.name;
```

5.4 分页算法

下面来看如下这个查询语句。

```
mysql> SELECT col_1,col_2 FROM profiles WHERE here sex='M' ORDER BY rating limit 10;
```

如果没有索引，以上查询将会变得很慢，即使有了索引，也不一定会变快。程序的展示页面可能是分页显示，如果有人点击的是中间的某个页面，类似如下的查询。

```
mysql> SELECT col_1,col_2 FROM profiles WHERE sex='M' ORDER BY rating limit 100000, 10;
```

这种查询，无论如何索引，效率都会奇差，因为大偏距 (high offset) 值的查询，会花费大部分时间来扫描大量数据，而这些数据最终都会被丢弃；这种情况下，更好的办法是限制用户所看到的页，比如只提供最新的几页、上一页、下一页，因为没有什么用户会去关注第10000页的内容。

或者换一个思路，用户点击1000页或10000页这个行为很稀少，那么根本没有必要做得很准确，自己根据数据库的数据估算总的页数，构建连接即可，有一些误差是可以接受的。

另一个办法是使用覆盖索引 (covering index)。以下示例中的表已经在 (sex, rating) 上创建了索引，id是主键。

```
mysql> SELECT col_1,col_2 FROM profiles INNER JOIN
      (SELECT id FROM profiles
       WHERE x.sex='M' ORDER BY rating) AS x USING id;
```

以上语句中的SELECT子查询 (SELECT id.....) 可以利用到覆盖索引，由于覆盖索引一般已被加载到内存，因此这种方式的排序效率会高许多。在一定的数据量下，性能尚可。

5.5 处理NULL值

对于SQL新手，NULL值的概念常常会造成混淆，他们常认为NULL与空字符串“”是相同的，然而事实并非如此。例如，下述语句就是完全不同的。

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ('');
```

这两条语句均会将值插入phone（电话）列，但第1条语句插入的是NULL值，第2条语句插入的是空字符串。第1条语句的含义可被解释为“电话号码未知”，而第2条语句的含义可被解释为“该人员没有电话，因此没有电话号码”。

在SQL中，NULL值与任何其他值的比较（即使是NULL）永远都不会为“真”。

为了进行NULL处理，可使用IS NULL和IS NOT NULL操作符。如，

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
```

使用LOAD DATA INFILE读取数据时，对于空的或丢失的列，将用空字符串“”来更新它们。如果希望在列中具有NULL值，应在数据文件中使用\N。

使用DISTINCT、GROUP BY或ORDER BY时，所有NULL值将被视为是等同的。

使用ORDER BY时，首先将显示NULL值，如果指定了DESC按降序排列，那么NULL值将在最后面显示。

对于聚合（累计）函数，如COUNT()、MIN()和SUM()，将忽略NULL值。对此的例外是COUNT(*)，它将计数行而不是单独的列值。例如，下述语句会产生两个计数。首先计数表中的行数，其次计数age列中的非NULL值的数目：

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

对于某些列类型，MySQL将对NULL值进行特殊处理。如果将NULL值插入TIMESTAMP列，那么将插入当前日期和时间。如果将NULL值插入具有AUTO_INCREMENT属性的整数列，那么将插入序列中的下一个编号。

NULL值的存取，可能导致程序异常，我们有很多方法可用来友好地显示NULL值。

(1) 使用CASE语句

```
SELECT
CASE
    WHEN SUM(size) IS NULL THEN 0
    ELSE SUM(size)
END
INTO @l_sum_vol FROM table_a ;
```

(2) 使用COALESCE函数

```
SELECT COALESCE( sum(size) , 0 ) FROM table_a
```

COALESCE(value,...)函数：返回值为列表当中的第一个非NULL值，在没有非NULL值的情况下返回值将为NULL。

(3) 使用IFNULL函数

```
SELECT SUM (ifnull(size,0)) FROM table_a;
```

IFNULL(expr1, expr2)函数：假如expr1不为NULL，则IFNULL () 的返回值为expr1；否则其的返回值为expr2。

IFNULL () 的返回值是数字还是字符串取决于其所使用的语境。

(4) 使用IF函数

```
SELECT SUM (IF (size is null, 0, size)) AS totalsize FROM table_a;
```

IF(expr1, expr2, expr3): 如果expr1是TRUE，则IF()的返回值为expr2；否则返回值为expr3。 IF()的返回值是数字还是字符串视其所在的语境而定。

NULL值可能会导致MySQL的优化变得复杂，所以，一般建议字段应尽量避免使用NULL值。

5.6 存储URL地址

在互联网应用中，存储和检索域名或长的URL地址是很常见的。那么对于此类数据的存取，又有哪些技巧呢。

对于存储域名，可按照字符颠倒的方式进行存储，这样做可方便索引。

如：

```
com.fabulab.marcomacaco  
com.fabulapps.kiko  
com.fandora9.angryvirus
```

存储URL值，一般推荐的做法是对URL值做一个散列，散列值最好是整型，然后存储这个散列值，并在其上创建索引。如下示例将对域名进行散列。

```
SELECT CONV (RIGHT(MD5(`  
http://www.mysql.com/`  
) , 16), 16, 10) AS HASH64;
```

新建一个字段url_hash，用于保存类型为整型的散列值。以后这样查询即可。

```
SELECT id FROM url WHERE  
url_hash=CONV(RIGHT(MD5('http://www.mysql.com/'), 16), 16, 10)  
AND url='http://www.mysql.com';
```

散列函数可以用程序来实现，以减少在MySQL侧的运算。

5.7 归档历史数据

随着项目的发展，将历史数据从日常使用的数据中删除，或将其移动到归档历史表中是比较常见的需求。对过期数据的查询很少时，这样做可以提高性能，而且也不用对程序做大的变动。还可以把过期的历史数据放到其他性能较差的实例、机器上，以便更好地利用资源。

另一种比较常见的方式，是按照时间分表，比如按月份、按日来分表存储数据。这种方式比较容易区分数据，方便维护。但要留意如果有跨越多个表的查询，效率可能会比较差，需要综合考虑平衡分表的粒度。

笔者不建议使用MySQL自身的特性实现归档，比如分区表。一般来说，把归档操作的逻辑放到程序处，可以更方便后期的维护。

归档数据可以通过定时执行的守护执行，也可以使用一些特定的归档工具来归档数据。还有，需要确保这种大数据量的操作不会影响到正常的生产。

5.8 使用数据库存储图片

如果使用文件系统或分布式文件系统存储图片，那么文件和数据库的信息可能难以保持一致，也不好回滚，不好统一进行备份，尤其是在多机房的环境下。为了简化开发和架构，也可以考虑使用数据库来存储图片。MySQL BLOB类型(MEDIUMBLOB，最大支持16MB的数据)对于绝大部分图片来说都足够了，我们可以使用LOAD_FILE()方法读取一个文件，然后将内容保存到BLOB列中。

由于数据库毕竟不适合于存储大量的图片，如果存储大量图片的话，仍然建议使用文件系统或分布式文件系统。分布式文件系统配合缓存、CDN技术，往往是海量图片存储系统的优选方案。

5.9 多表UPDATE

MySQL可以基于多表查询更新数据。如下是MySQL多表UPDATE在实践中的几种不同写法。对于多表的UPDATE操作需要慎重，建议在更新之前，先使用SELECT语句查询验证下要更新的数据与自己期望的是否一致。

假定我们有两张表，一张表为product表，存放产品信息，其中有产品价格列price；另外一张表是product_price表，要将product_price表中的价格字段price更新为product表中价格字段price的80%。

在MySQL中我们有几种手段可以做到这一点，一种是“UPDATE table1 t1,table2,...,table n”的方式。

```
UPDATE product p, product_price pp
SET pp.price = p.price * 0.8
WHERE p.productId = pp.productId
```

另外一种方法是使用INNER JOIN然后更新。

```
UPDATE product p
INNER JOIN product_price pp
ON p.productId = pp.productId
SET pp.price = p.price * 0.8
```

也可以使用LEFT JOIN来做多表UPDATE，如果product_price表中没有产品价格记录的话，将product表的isDeleted字段设置为1，SQL语句如下。

```
UPDATE product p
LEFT JOIN product_price pp
ON p.productId = pp.productId
SET p.deleted = 1
WHERE pp.productId IS NULL
```

另外，上面的几个例子都是在两张表之间做关联，但是只更新一张表中的记录，其实MySQL是可以同时更新两张表的，如下查询就同时修改了两个表。

```
UPDATE product p
INNER JOIN product_price pp
ON p.productId = pp.productId
SET pp.price = p.price * 0.8,
p.dateUpdate = CURDATE ()
```

两张表做关联，同时更新了product_price表的price字段和product表的dateUpdate两个字段。

5.10 生成全局唯一ID

由于分布式数据库的部署，多个节点之间为了避免数据冲突，需要有一个全局唯一的ID进行标识，一些NoSQL数据库从设计之初，就考虑了ID的不重复，而MySQL在这方面仍然需要借助一些特殊的手段来生成全局唯一的ID。可以考虑如下这些方式。

- 1) 利用数据库自身的特性，在数据库启动参数里配置auto_increment_increment和auto_increment_offset，不过我们不推荐这种方式，因为这会导致数据库的维护成本上升。
- 2) 配置一个单独的服务生成全局ID，可以是MySQL，也可以是NoSQL产品，甚至可以构建自己的专门用来生成唯一ID的服务，为了提高效率，还可以批量获取唯一的ID序列。
- 3) 另外一种方式是，通过函数、程序算法或字段组合生成唯一ID，这种方式，可能会产生冲突，但是可以将这个冲突的概率做到非常小，我们更推荐使用这种方式。

5.11 使用SQL生成升级SQL

可以使用SQL去生成升级的SQL文件，如使用CONCAT函数拼接生成SQL语句。例如，批量删除前缀为“prefix”的表，命令如下。

```
SELECT CONCAT ('drop table ',table_name,';') INTO OUTFILE '/tmp/drop_table.sql' FROM information_schema.tables WHERE table_name LIKE 'prefix%' AND table_schema='db_name';
```



小结 我们使用一些技巧、方法，是为了更方便、更高效地使用数据库。部分技巧的使用和具体数据库无关；部分技巧的使用，需要深入了解数据库。在我们撰写代码的过程中，应该经常问自己，自己对于数据的操作，是否优雅、高效、可扩展。在这样的理念的引导下，我们将会变得越来越有技巧。

第6章 查询优化

查询优化是研发人员比较关注也是疑问较多的领域。本章首先为读者介绍常用的优化策略、MySQL的优化器、连接机制，然后介绍各种语句的优化，在阅读本章之前，需要先对EXPLAIN命令，索引知识有必要的了解。

研发人员应该掌握并且熟悉优化技巧，某种意义上，因为研发人员熟悉业务逻辑，因此应该比DBA更加擅长于对SQL的优化。现实中，各种技术之间的界限变得越来越模糊，不同背景的IT从业人员之间的交流也越来越频繁，本书将属于优化的大部分内容都放在开发篇，是因为优化的重心将会越来越向前推移到研发团队，DBA也需要了解开发，需要融入整个研发体系中去。

6.1 基础知识

6.1.1 查询优化的常用策略

一般常用的查询优化策略有优化数据访问、重写SQL、重新设计表、添加索引4种。下面将分别介绍这4种优化策略。

(1) 优化数据访问

应该尽量减少对数据的访问。一般有如下两个需要考虑的地方：应用程序应减少对数据库的数据访问，数据库应减少实际扫描的记录数。

例如，如果应用程序可以缓存数据，就可以不需要从数据库中直接读取数据。

例如，如果应用程序只需要几个列的数据，就没有必要把所有列的数据全部读取出来，应该尽可能地避免“`SELECT*FROM table_name`”这样的语句。

例如，有时我们在慢查询日志里会看到`Rows_examined`这一项的值很高，而实际上，并不需要扫描大量的数据，这种情况下添加索引或增加筛选条件都可以极大地减少记录扫描的行数。

类似的例子还有很多，这里就不一一列举了。

(2) 重写SQL

由于复杂查询严重降低了并发性，因此为了让程序更适于扩展，我们可以把复杂的查询分解为多个简单的查询。一般来说多个简单查询的总成本是小于一个复杂查询的。

对于需要进行大量数据的操作，可以分批执行，以减少对生产系统产生的影响，从而缓解复制超时。

由于MySQL连接（JOIN）严重降低了并发性，对于高并发，高性能的服务，应该尽量避免连接太多表，如果可能，对于一些严重影响性能的SQL，建议程序在应用层就实现部分连接的功能。这样的好处是：可以更方便、更高效地缓存数据，方便迁移表到另外的机器，扩展性也更好。

(3) 重新设计库表

有些情况下，我们即使是重写SQL或添加索引也是解决不了问题的，这个时候可能要考虑更改表结构的设计。比如，可以增加一个缓存表，暂存统计数据，或者可以增加冗余列，以减少连接。优化的主要方向是进行反范式设计，反范式的设计请参考4.1节。

(4) 添加索引

生产环境中的性能问题，可能80%的都是索引的问题，所以优化好索引，就已经有了一个好的开始。索引的具体优化，请参考3.5节。

6.1.2 优化器介绍

查询优化器的任务是发现执行SQL查询的最佳方案。“好”方案和“坏”方案之间性能的差别可能会很大。大多数查询优化器，包括MySQL的查询优化器，总是或多或少地在所有可能的查询评估方案中搜索最佳方案。不同版本优化器的优化算法可能

也会不同，随着MySQL版本的进化，优化器也变得越来越强大和智能，去除了一些限制，改进了一些算法。本书主要关注的是MySQL 5.1版本的优化方式，MySQL 5.5、5.6、5.7版本目前都已经有了GA版本，相关的改进，建议大家参考官方文档。如果不確定优化器的优化方式，可以使用EXPLAIN语句验证之。

1.优化器的不足

MySQL优化器也有很多不足之处，它不一定能保证选择的执行计划就是最优的。

- 数据的统计信息有可能是错误的，对于复杂的查询，数据库可能会执行错误的执行计划，从而导致严重的性能问题。
- MySQL优化器的优化是基于简单的成本评估进行的，总是会选择成本更小的执行计划，其对成本衡量的标准是读取的随机块的数量，但是，本质上成本往往包括了诸多因素，CPU、内存、数据是否在缓存中，都是需要考虑到的因素，这样往往会导致MySQL计算得出的成本最小的执行计划不一定是响应最快的。
- 优化器不会考虑并发的情况，而实际的数据库执行，并发处理则是复杂的，资源的争用可能会导致性能问题。
- 一些商业数据库在执行的过程中会对各种优化结果的执行情况进行统计评估，以便自动改进后续的执行优化状况，而MySQL目前还没有这些功能。

2.优化器加提示

有时我们需要告诉优化器，让它按我们的意图生成执行计划，但是，加提示（hint）的方式不到万不得已，建议不要使用。一般来说，复杂的SQL走了错误的执行计划的时候才可能需要使用到提示，我们应该尽量让MySQL的优化器去决定执行计划。否则，将会增加MySQL的维护成本，你也可能需要更多的额外工作。随着时间的演变，我们选择的提示所依据的外部条件很可能已经发生了变化，比如说数据量、数据分布发生了变化，如果你仍然使用旧的提示，可能会导致MySQL承担过多的工作。而且，在MySQL升级了新版本后，你也应用不了新的优化技术。

比较常用的加提示的方式有如下6种。

(1) 使用索引 (USE INDEX)

USE INDEX(index_list)将告诉MySQL使用我们指定的索引去检索记录。index_list是索引名列表，以逗号分隔。注意，我们这里设置的是索引名或索引名列表，而不是索引基于的字段名，主键名为PRIMARY。可以使用SHOW INDEX FROM table_name命令显示表上的索引名。

下面的例子表示建议使用索引名为col1_index或col2_index的索引检索表。

```
SELECT * FROM table1 USE INDEX (col1_index,col2_index)
WHERE col1=1 AND col2=2 AND col3=3;
```

(2) 不使用索引 (IGNORE INDEX)

IGNORE INDEX(index_list)将建议MySQL不使用指定的索引。如果我们用EXPLAIN命令查看执行计划，发现走了错误的索引，那么可以使用IGNORE INDEX来避免继续使用错误的索引。如下的例子表示建议MySQL不使用索引名为col3_index的索引。

```
SELECT * FROM table1 IGNORE INDEX (col3_index)
WHERE col1=1 AND col2=2 AND col3=3;
```

(3) 强制使用索引 (FORCE INDEX)

有时我们使用USE INDEX指定了索引，但MySQL优化器仍然选择不使用我们指定的索引，这时可以考虑使用FORCE INDEX提示。

注意，USE INDEX、IGNORE INDEX和FORCE INDEX这些提示方式只会影响MySQL在表中检索记录或连接要使用的索引，它们并不会影响ORDER BY或GROUP BY子句对于索引的选择。

(4) 不使用查询缓冲 (SQL_NO_CACHE)

SQL_NO_CACHE提示MySQL对指定的查询关闭查询缓冲机制。有时为了验证一条SQL语句实际执行的时间，我们可以临时加上SQL_NO_CACHE，以免被查询缓冲给误导了。对于一些不期望被缓存的SQL，比如夜间的报表查询，可以通过设置SQL_NO_CACHE来让MySQL查询缓冲更高效地工作。

(5) 使用查询缓冲 (SQL_CACHE)

有时我们将查询缓冲设置为显式模式 (explicit mode, query_cache_type=2)，也就是说，除非指明了SQL需要缓存，否则MySQL是不考虑缓存它的，我们使用SQL_CACHE来指定哪些查询需要被缓存。

(6) STRAIGHT_JOIN

这个提示将告诉MySQL按照FROM子句描述的表的顺序进行连接。如果用EXPLAIN命令进行检查，确认了MySQL没有按照最优的顺序进行表的连接，那就可以使用这个提示，告诉MySQL按照我们指定的顺序进行连接。不建议自己指定连接顺序，可以尝试重写SQL，看看MySQL是否能够选择更好的执行计划，也可以尝试分析表（运行ANALYZE TABLE命令）以更新索引统计信息，STRAIGHT_JOIN应该是最不得已时才做的选择。

6.1.3 MySQL的连接机制

MySQL中的JOIN（连接）这个术语泛指一切查询，而不是传统术语中的定义：“两个表之间的JOIN”。MySQL的查询优化器最重要的部分就是连接优化器，由它来决定多个表连接的次序。其他的查询语句都相应地向JOIN靠拢：单表查询将被当作JOIN的特例，子查询也将被尽量转换为JOIN查询。

MySQL一般使用的是“Nested Loop Join”，即嵌套连接。图6-1是《High Performance MySQL》一书中对嵌套连接的说明图，col3为连接列，连接两个表tbl1、tbl2的步骤类似如图6-1所示。

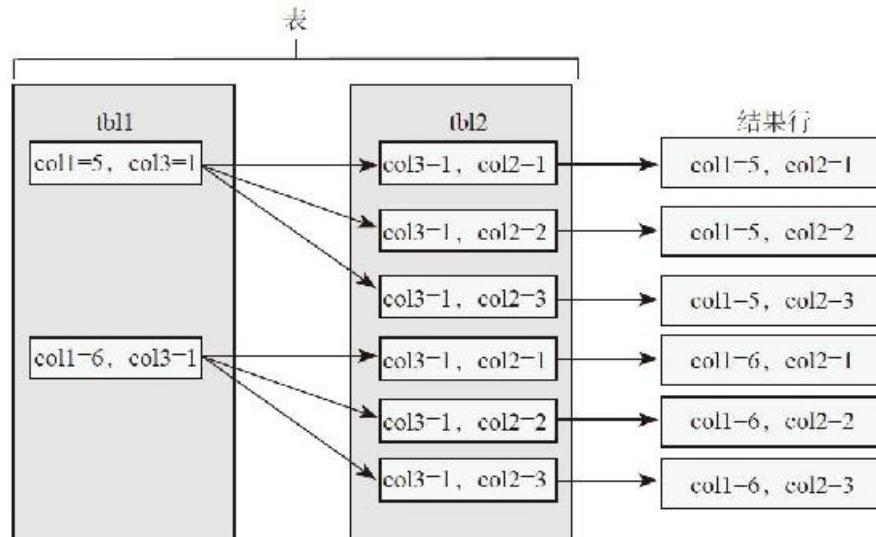


图6-1 嵌套连接

如图6-1所示，嵌套连接遍历tbl1表，对于tbl1表中的每一行记录，都将去tbl2表中探测，看是否有满足条件的记录。上述执行步骤，可以简单地描述成如下语句，实际上，MySQL对如下算法做了一些改进。

```
For each tuple r in tbl1 do
    For each tuple s in tbl2 do
        If r and s satisfy the join condition
            Then output the tuple <r,s>
```

我们称外部的tbl1表为驱动表或外部表，内部的tbl2表为内部表。这种算法的成本与外部表行数乘以内部表行数的乘积是成正比例的。如果嵌套的层次比较多，也就是说连接了很多表，那么成本将是昂贵的。如果两个表进行连接，MySQL优化器一般会选择更小的表或更小子集（满足查询条件的记录行数少）的表作为驱动表。为什么要这么做呢？由上面的代码可知，随着驱动表（外部表）行数的增加，成本会增加得很快，选择更小的外部表或更小子集的外部表，是为了尽量减少嵌套连接的循环次数，而且，内部表一般在连接列有索引，索引一般常驻于内存中，这样可以保证很快完成连接。

因此，MySQL应该尽量避免连接太多表。在现实的生产环境中，这个问题很普遍，研发人员往往低估了连接太多表所带来的负面影响。

6.2 各种语句优化

6.2.1 连接的优化

由于连接的成本比较高，因此对于高并发的应用，应该尽量减少有连接的查询，连接的表的个数不能太多，连接的表建议控制在4个以内。互联网应用比较常见的一种情况是，在数据量比较小的时候，连接的开销不大，这个时候一般不会有性能问题，但当数据量变大之后，连接的低效率问题就暴露出来了，成为整个系统的瓶颈所在。所以对于数据库应用的设计，最好在早期就确定未来可能会影响性能的一些查询，进行反范式设计减少连接的表，或者考虑在应用层进行连接。

优化连接的一些要点如下。

- 1) ON、USING子句中的列确认有索引。如果优化器选择了连接的顺序为B、A，那么我们只需要在A表的列上创建索引即可。例如，对于查询“SELECT B.* , A.* FROM B JOIN A ON B.col1=A.col2;”语句MySQL会全表扫描B表，对B表的每一行记录探测A表的记录（利用A表col2列上的索引）。
- 2) 最好是能转化为INNER JOIN, LEFT JOIN的成本比INNER JOIN高很多。
- 3) 使用EXPLAIN检查连接，留意EXPLAIN输出的rows列，如果rows列太高，比如几千，上万，那么就需要考虑是否索引不佳或连接表的顺序不当。
- 4) 反范式设计，这样可以减少连接表的个数，加快存取数据的速度。
- 5) 考虑在应用层实现连接。

对于一些复杂的连接查询，更值得推荐的做法是将它分解为几个简单的查询，可以先执行查询以获得一个较小的结果集，然后再遍历此结果集，最后根据一定的条件去获取完整的数据，这样做往往是更高效的，因为我们把数据分离了，更不容易发生变化，更方便缓存数据，数据也可以按照设计的需要从缓存或数据库中进行获取。例如，对于如下的查询：

```
SELECT a.* FROM a WHERE a.id IN (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17);
```

如果id=1~15的记录已经被存储在缓存（如Memcached）中了，那么我们只需要到数据库查询“SELECT a.* FROM a WHERE a.id=16”和“SELECT a.* FROM a WHERE a.id=17”了。而且，把IN列表分解为等值查找，往往可以提高性能。

- 6) 一些应用可能需要访问不同的数据库实例，这种情况下，在应用层实现连接将是更好的选择。

6.2.2 GROUP BY、DISTINCT、ORDER BY语句优化

GROUP BY、DISTINCT、ORDER BY这几类子句比较类似，GROUP BY默认也是要进行ORDER BY排序的，笔者在本书中把它们归为一类，优化的思路也是类似的。可以考虑的优化方式如下。

- 尽量对较少的行进行排序。
- 如果连接了多张表，ORDER BY的列应该属于连接顺序的第一张表。
- 利用索引排序，如果不能利用索引排序，那么EXPLAIN查询语句将会看到有filesort。
- GROUP BY、ORDER BY语句参考的列应该尽量在一个表中，如果不在同一个表中，那么可以考虑冗余一些列，或者合并

表。

·需要保证索引列和ORDER BY的列相同，且各列均按相同的方向进行排序。

·增加sort_buffer_size。

sort_buffer_size是为每个排序线程分配的缓冲区的大小。增加该值可以加快ORDER BY或GROUP BY操作。但是，这是为每个客户端分配的缓冲区，因此不要将全局变量设置为较大的值，因为每个需要排序的连接都会分配sort_buffer_size大小的内存。

·增加read_rnd_buffer_size。

当按照排序后的顺序读取行时，通过该缓冲区读取行，从而避免搜索硬盘。将该变量设置为较大的值可以大大改进ORDER BY的性能。但是，这是为每个客户端分配的缓冲区，因此你不应将全局变量设置为较大的值。相反，只用为需要运行大查询的客户端更改会话变量即可。

·改变tmpdir变量指向基于内存的文件系统或其他更快的磁盘。

如果MySQL服务器正作为复制从服务器被使用，那么不应将“--tmpdir”设置为指向基于内存的文件系统的目录，或者当服务器主机重启时将要被清空的目录。因为，对于复制从服务器，需要在机器重启时仍然保留一些临时文件，以便能够复制临时表或执行LOAD DATA INFILE操作。如果在服务器重启时丢失了临时文件目录下的文件，那么复制将会失败。

·指定ORDER BY NULL。

默认情况下，MySQL将排序所有GROUP BY的查询，如果想要避免排序结果所产生的消耗，可以指定ORDER BY NULL。

例如：

```
SELECT count(*) cnt, cluster_id FROM stat GROUP BY cluster_id ORDER BY NULL LIMIT 10;
```

·优化GROUP BY WITH ROLLUP。

GROUP BY WITH ROLLUP可以方便地获得整体分组的聚合信息（super aggregation），但如果存在性能问题，可以考虑在应用层实现这个功能，这样往往会更高效，伸缩性也更佳。

·使用非GROUP BY的列来代替GROUP BY的列。

比如，原来是“GROUP BY xx_name,yy_name”，如果GROUP BY xx_id可以得到一样的结果，那么使用GROUP BY xx_id也是可行的。

·可以考虑使用Sphinx等产品来优化GROUP BY语句，一般来说，它可以有更好的可扩展性和更佳的性能。

6.2.3 优化子查询

由于子查询的可读性比较好，所以有些研发人员习惯于编写子查询，特别是刚接触数据库编程的新手。但子查询往往也是性能杀手，在生产环境中，子查询是最常见的导致性能问题的症结所在。

对于数据库来说，在绝大部分情况下，连接会比子查询更快。使用连接的方式，MySQL优化器一般可以生成更佳的执行计划，可以预先装载数据，更高效地处理查询。而子查询往往需要运行重复的查询，子查询生成的临时表上也没有索引，因此效率会更低。

一些商业数据库已经可以智能地识别子查询，转化子查询为连接查询，或者转化连接为子查询。这种情况下，编写子查询也许是更好的方式，毕竟更符合人的思考方式，也能避免因为重复记录的匹配导致连接结果集的异常。但MySQL对于子查询的优化一直不佳，就目前的研发实践来说，子查询应尽量改写成JOIN的写法。如果我们不能确定是否要使用连接的方式，那么可以使用EXPLAIN语法查看语句具体的执行计划。

如下是一个带子查询的语句。

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN ( SELECT column1 FROM t2);
```

普通的子查询一般都可以转化为连接的方式。上面的例子可以转化为如下的写法。

```
SELECT DISTINCT t1.column1 FROM t1, t2 WHERE t1.column1 = t2.column1;
```

如下的两个查询是等价的。

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

还可以改写成如下LEFT JOIN的形式。

```
SELECT table1.*
  FROM table1 LEFT JOIN table2 ON table1.id=table2.id
 WHERE table2.id IS NULL;
```

下面再举一些例子。

把子句从子查询的外部转移到内部。例如，把此查询：

```
SELECT * FROM t1
WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

转化成如下的写法：

```
SELECT * FROM t1
WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

将此查询：

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

转化成如下的写法：

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

将此查询：

```
SELECT * FROM t1
WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
AND t2.column2=t1.column2);
```

转化成如下的写法，使用行子查询来代替关联子查询：

```
SELECT * FROM t1
```

```
WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

对于只返回一行的无关联子查询，IN的速度慢于“=”。

将此查询：

```
SELECT * FROM t1 WHERE t1.col_name  
IN (SELECT a FROM t2 WHERE b = some_const);
```

转化成如下的写法：

```
SELECT * FROM t1 WHERE t1.col_name  
= (SELECT a FROM t2 WHERE b = some_const);
```

MySQL优化器这些年来一直都在改进，MySQL后续版本对于子查询也有了更多改进。读者可以参考如下链接：

<http://dev.mysql.com/doc/refman/5.6/en/subquery-optimization.html>

<http://dev.mysql.com/doc/refman/5.7/en/subquery-optimization.html>

6.2.4 优化limit子句

Web应用经常需要对查询的结果进行分页，分页算法经常需要用到“LIMIT offset, row_count ORDER BY col_id”之类的语句。一旦offset的值很大，效率就会很差，因为MySQL必须检索大量的记录（offset+row_count），然后丢弃大部分记录。

可供考虑的优化办法有如下4点。

1) 限制页数，只显示前几页，超过了一定的页数后，直接显示“更多（more）”，一般来说，对于N页之后的结果，用户一般不会关心。

2) 要避免设置offset值，也就是避免丢弃记录。

例如以下的例子，按照id排序（id列上有索引），通过增加一个定位的列“id>990”，可以避免设置offset的值。

```
SELECT id, name, address, phone  
FROM customers  
WHERE id > 990  
ORDER BY id LIMIT 10;
```

也可以使用条件限制要排序的结果集，如可以这样使用。

```
WHERE date_time BETWEEN  
2014-04-01 00:00:00  
AND  
2014-04-02 00:00:00  
ORDER BY id
```

对条件值可以进行估算，对于几百上千页的检索，往往不需要很精确。也可以专门增加冗余的列来定位记录，比如如下的查询，有一个page列，指定记录所在的页，代价是在修改数据的时候需要维护这个列的数据，如下面的查询。

```
SELECT id, name, address, phone  
FROM customers  
WHERE page = 100  
ORDER BY name;
```

3) 使用Sphinx。

4) 使用INNER JOIN。

以下的例子中，先按照索引排序获取到id值，然后再使用JOIN补充其他列的数据。customers表的主键列是id列，name列上有索引，由于“SELECT id FROM customers...”可以用到覆盖索引，所以效率尚可。

```
SELECT id, name, address, phone
FROM customers
INNER JOIN (
    SELECT id
    FROM customers
    ORDER BY name
    LIMIT 999,10)
AS my_results USING(id);
```

6.2.5 优化IN列表

对于IN列表，MySQL会排序IN列表里的值，并使用二分查找（Binary Search）的方式去定位数据。

把IN子句改写成OR的形式并不能提高性能。以笔者个人的经验，IN列表不宜过长，最好不要超过200。对于高并发的业务，小于几十为佳。

如果能够将其转化为多个等于的查询，那么这种方式会更优。例如如下这个查询。

```
SELECT * FROM table_a WHERE id IN (SELECT id FROM table_b);
```

我们可以先查询SELECT id FROM table_b，然后把获取到的id值，逐个地和“SELECT*FROM table_a”进行拼接，转化为“SELECT id FROM table_a WHERE id=?”的形式。这个操作用程序来实现其实是很简单的。

6.2.6 优化UNION

UNION语句默认是移除重复记录的，需要用到排序操作，如果结果集很大，成本将会很高，所以，建议尽量使用UNION ALL语句。对于UNION多个分表的场景，应尽可能地在数据库分表的时候，就确定各个分表的数据是唯一的，这样就无须使用UNION来去除重复的记录了。

另外，查询语句外层的WHERE条件，并不会应用到每个单独的UNION子句内，所以，应在每一个UNION子句中添加上WHERE条件，从而尽可能地限制检索的记录数。

6.2.7 优化带有BLOB、TEXT类型字段的查询

由于MySQL的内存临时表不支持BLOB、TEXT类型，如果包含BLOB或TEXT类型列的查询需要用到临时表，就会使用基于磁盘的临时表，性能将会急剧降低。所以，编写查询语句时，如果没有必要包含BLOB、TEXT列，就不要写入查询条件。

规避BLOB、TEXT列的办法有如下两种。

1) 使用SUBSTRING()函数。

2) 设置MySQL变量tmpdir，把临时表存放在基于内存的文件系统中。如Linux下的tmpfs。可以设置多个临时表的路径（用分号分隔），MySQL将使用轮询的方式。

优化的办法有如下3种。

- 1) 如果必须使用，可以考虑拆分表，把BLOB、TEXT字段分离到单独的表。
- 2) 如果有许多大字段，可以考虑合并这些字段到一个字段，存储一个大的200KB比存储20个10KB更高效。
- 3) 考虑使用COMPRESS()，或者在应用层进行压缩，再存储到BLOB字段中。



注意 如果BLOB列很大，可能需要增大`innodb_log_file_size`（MySQL错误日志内可能会提示事务日志小了）。

6.2.8 filesort的优化

有时我们使用EXPLAIN工具，可以看到查询计划的输出中的Extra列有`filesort`。`filesort`往往意味着你没有利用到索引进行排序。`filesort`的字面意思可能会导致混淆，它和文件排序没有任何关系，可以理解为不能利用索引实现排序。

排序一个带JOIN（连接）的查询，如果ORDER BY子句参考的是JOIN顺序里的第一张表的列且不能利用索引进行排序，那么MySQL会对这个表进行文件排序（`filesort`），EXPLAIN输出中的Extra列就有`filesort`。如果排序的列来自于其他的表，且需要临时文件来帮助排序，那么EXPLAIN输出的Extra列就有“Using temporary;Using filesort”字样。对于MySQL 5.1，如果有LIMIT子句，那么是在`filesort`之后执行LIMIT的，这样做效率可能会很差，因为需要排序过多的记录。

1.两种`filesort`算法

MySQL有两种`filesort`算法：`two-pass`和`single-pass`。

(1) `two-pass`

这是旧的算法。列长度之和超过`max_length_for_sort_data`字节时就使用这个算法，其原理是：先按照WHERE筛选条件读取数据行，并存储每行的排序字段和行指针到排序缓冲（`sort buffer`）。如果排序缓冲大小不够，就在内存中运行一个快速排序（`quick sort`）操作，把排序结果存储到一个临时文件里，用一个指针指向这个已经排序好了的块。然后继续读取数据，直到所有行都读取完毕为止。这是第一次读取记录。

然后合并如上的临时文件，进行排序。

然后依据排序结果再去读取所需要的数据，读入行缓冲（`row buffer`，由`read_rnd_buffer_size`参数设定其大小）。这是第二次读取记录。

以上第一次读取记录时，可以按照索引排序或表扫描，可以做到顺序读取。但第二次读取记录时，虽然排序字段是有序的，行缓冲里存储的行指针是有序的，但所指向的物理记录需要随机读，所以这个算法可能会带来很多随机读，从而导致效率不佳。

优点： 排序的数据量较小，一般在内存中即可完成。

缺点： 需要读取记录两次，第二次读取时，可能会产生许多随机I/O，成本可能会比较高。

(2) `single-pass`

MySQL一般使用这种算法。其原理是：按筛选条件，把SQL中涉及的字段全部读入排序缓冲中，然后依据排序字段进行排序，如果排序缓冲不够，则会将临时排序结果写入到一个临时文件中，最后合并临时排序文件，直接返回已经排序好的结果集。

优点：不需要读取记录两次，相对于two-pass，可以减少I/O开销。

缺点：由于要读入所有字段，排序缓冲可能不够，需要额外的临时文件协助进行排序，导致增加额外的I/O成本。

2.相关参数的设置和优化

相关参数如下。

max_length_for_sort_data: 如果各列长度之和（包括选择列、排序列）超过了max_length_for_sort_data字节，那么就使用two-pass算法。如果排序BLOB、TEXT字段，使用的也是two-pass算法，那么这个值设置得太高会导致系统I/O上升，CPU下降，建议不要将max_length_for_sort_data设置得太高。

max_sort_length: 如果排序BLOB、TEXT字段，则仅排序前max_sort_length个字节。

可以考虑的优化方向如下。

·加大sort_buffer_size。

一般情况下使用默认的single-pass算法即可。可以考虑加大sort_buffer_size以减少I/O。

需要留意的是字段长度之和不要超过max_length_for_sort_data，只查询所需要的列，注意列的类型、长度。MySQL目前读取和计算列的长度是按照定义的最大的度进行的，所以在设计表结构的时候，不要将VARCHAR类型的字段设置得过大，虽然对于VARCHAR类型来说，在物理磁盘中的实际存储可以做到紧凑，但在排序的时候，是会分配最大定义的长度的，有时排序阶段所产生的临时文件甚至比原始表还要大。MySQL 5.7版本在这方面做了一些优化，有兴趣的同学可以参考<http://dev.MySQL.com/doc/refman/5.7/en/order-by-optimization.html>

·对于two-pass算法，可以考虑增大read_rnd_buffer_size，但由于这个全局变量是对所有连接都生效的，因此建议只在会话级别进行设置，以加速一些特殊的大操作。

·在操作系统层面，优化临时文件的读写。

6.2.9 优化SQL_CALC_FOUND_ROWS

建议不要使用SQL_CALC_FOUND_ROWS这个提示，虽然它可以让开发过程变得简单一些，但并没有减少数据库做的事情。例如以下这个查询。

```
SELECT SQL_CALC_FOUND_ROWS col_name FROM table_name where ... LIMIT N
```

它使用LIMIT子句限制了返回记录数，但实际上数据库仍然需要扫描大量记录以找到符合查询条件的所有记录。这是一个成本昂贵的操作。如果实在需要使用的话，建议使用独立的语句SELECT COUNT (*)，这样做将会更高效。

一些统计值，如果可以缓存的话，那么缓存之更好。现实应用中，有时并没有必要显示记录的总数，或者不要求精确性，这时我们应该尽量减少这种消耗资源的查询。

6.2.10 优化临时表

如果不能利用索引排序，那么我们在MySQL中可能需要创建一个临时表用于排序。MySQL的临时表分为“内存临时表”和“磁盘临时表”，其中，内存临时表使用MySQL的MEMORY存储引擎。磁盘临时表使用MySQL的MyISAM存储引擎；一般情况下，MySQL会先创建内存临时表，但当内存临时表超过配置参数指定的值后，MySQL会将内存临时表导出到磁盘临时表。

触发以下条件，会创建临时表。

- ORDER BY子句和GROUP BY子句引用的列不一样。
- 在连接查询中，ORDER BY或GROUP BY使用的列不是连接顺序中的第一个表。
- ORDER BY中使用了DISTINCT关键字。

通过EXPLAIN的Extra列可以查看是否用到了临时表：“Using temporary”表示使用了临时表。

如果查询创建了临时表（in-memory table）来排序或检索结果集，分配的内存大于tmp_table_size与max_heap_table_size参数之间的最小值，那么内存临时表就会转换为磁盘临时表（on-disk table），MySQL会在磁盘上创建磁盘临时表，这样会可能导致I/O瓶颈，进而影响性能。

- tmp_table_size：指定系统创建的内存临时表的最大大小。
- max_heap_table_size：指定用户创建的内存表的最大大小。

SHOW FULL PROCESSLIST命令输出的state列为“Converting heap to MyISAM”时表明临时表大于我们所设置的参数值，此时将会产生磁盘临时表，但是数据库执行查询往往很快，“Converting heap to MyISAM”这个状态不一定能及时被看到，我们需要关注Created_tmp_tables和Created_tmp_disk_tables这两个变量的变化。由于MySQL慢查询日志里没有使用临时表的信息，这就给我们诊断性能问题带来了一些不便，第三方的版本如Percona Server，在慢查询里可以有更详细的信息，将会记录临时表使用的情况，从而有助于我们诊断和调优。

如下情况也可能会导致使用到磁盘临时表。

- 表中有BLOB或TEXT字段。
- 使用UNION或UNION ALL时，SELECT子句中包含了大于512字节的列。

使用临时表一般意味着性能会比较低，特别是使用磁盘临时表时，性能将会更慢，因此我们在实际应用中应该尽量避免临时表的使用。

常见的避免临时表的方法有如下3点。

- 创建索引：在ORDER BY或GROUP BY的列上创建索引。
- 分拆长的列：一般情况下，TEXT、BLOB，大于512字节的字符串，基本上都是为了显示信息，而不会用于查询条件，因此设计表的时候，可以考虑将这些列分离到另外一张表中。
- 不需要用DISTINCT时就没必要用DISTINCT，能用UNION ALL就不要用UNION。

6.3 OLAP业务优化

由于MySQL对于复杂SQL的优化不佳，所以对于一些OLAP的应用需要格外小心，在前期就做好一些针对性的设计，以尽量避免数据量剧增后碰到性能问题。关于SQL查询、索引优化，这里就不再赘述了。下面主要说下OLAP类型的业务需要考虑的一些要点。

(1) 使用冗余数据

有时最好的办法是在表中保存冗余的数据，虽然这些冗余数据有时也可以由其他的列推断得出。冗余数据可以让查询执行得更快。比如，我们可以增加一个专门的计数表或计数字段，实时更新计数信息。比如，大表之间的连接操作很耗时，增加冗余字段则可以有效地减少连接的表的个数。

(2) 计算复用，使用缓存表

我们可以使用缓存表存储一些结果，这里所说的“缓存表”，意思是这些值在逻辑上是冗余的，可以从原始表中获取到，但显然从原始表中获取数据更慢。

(3) 预计算

预先对一些常用的大查询生成汇总表。我们需要有这样一个意识，如果你需要处理大量数据，一般需要昂贵的计算成本。所以预算往往值得考虑的好方法。我们可以把查询结果存储到独立的汇总表中，或者可以把相关联的表的一些字段存放在一个独立的新表中，基于这个新的汇总表去做统计。

当我们使用缓存表和汇总表时，我们要做出决定：是实时更新数据还是定期更新，这依赖于你的应用。

当我们实时或定期重建缓存表、汇总表的时候，我们需要数据在操作的时间范围内仍然可用。我们可以采用一种“影子表”的方法，即建立一个临时表，在建立好之后，通过原子性地重命名表的操作，实现切换。

如下是重命名表，实现表切换的一个例子。

```
mysql> DROP TABLE IF EXISTS my_summary_new, my_summary_old;
mysql> CREATE TABLE my_summary_new LIKE my_summary;
mysql> RENAME TABLE my_summary TO my_summary_old, my_summary_new TO my_summary;
```

我们保留my_summary_old表，用于以后进行回滚，可以一直保留到下次操作。

以上的方式，增加了写操作和维护的工作，但要想获得更高的性能，往往是需要付出一定代价的。通过这些方法可以极大地加速读操作，虽然要承担写操作更慢的代价。

(4) 统计框架的改善

需要将一个复杂的查询任务放在一个SQL查询中来完成，往往会导致性能问题，使用这种方式最常见的原因是你正在使用一个编程框架或一个可视化组件库直接和数据源相连，然后在程序里直接展示数据，简单的商务智能和报表工具都属于这一分类。

一些报表框架，如果表设计不佳，那么随着数据量的增加，数据库将越来越力不从心，难以适应复杂的查询。组件或报表工具通常假设单个查询仅仅只用来完成一个简单的任务。但它鼓励你去设计更庞大的查询来生成报告中的所有数据。如果你使用某个这样的报表程序，就可能被迫去写一个复杂的SQL查询，而没有机会写代码操作结果集。

如果报表需求太复杂，不能用单个SQL查询来完成，那么更好的方案可能就是生成多个报表、增加一些限制条件。有时我

们想从多个维度进行各种组合得出报表，但是，表的设计往往限制了这种可能性，反而会导致复杂的查询，甚至导致发布查询后，长时间无法得到响应。报表研发人员可以和用户沟通，限制一些查询的使用，引导用户培养一些能够更快查询数据的习惯，让用户能够自己综合分析一些报表而不是完全借助计算机系统。如果你的老板不喜欢这样的解决方案，那么要提醒他报表的复杂度和生成报表所花的时间是成正比的。



小结 一般MySQL的优化有两个方向，一个是让SQL语句执行得更快，一个是让SQL语句做更少的事。比如，我们可以升级硬件让SQL跑得更快。或者，我们可以把小批量数据的排序交由应用程序去执行，MySQL不做排序计算。类似的方法有很多，但基本不外乎这两个方向。

MySQL的查询优化器比较简单，没有商业数据库那么强大和智能，我们应该理解MySQL的优化器限制，按优化器能理解的方式编写SQL。对于大流量的业务，应该尽量保持MySQL查询的简单性，以保证尽可能地支持更高的并发。现实中，对于数据库流量很大的业务，数据库往往已经退化为一个存储数据的容器，只利用它最高效的核心的特性。

第7章 研发规范

本章将为读者解读一份研发规范。为了更好地协同工作和确保所开发的应用尽可能的稳定、高效，建立一套数据库相关的研发规范是很有必要的，虽然研发规范的确立和推广是一项很耗时的工作，但所取得的收益也是长久的，它可以让研发人员更高效地使用数据库，可以让新的研发人员尽快融入研发体系，还可以极大地减少DBA和研发团队、测试团队的沟通成本。

如果DBA需要建立研发规范，建议和研发团队一起沟通确定，因为标准的实行和落地，需要考虑到现有的一些框架、语言和习惯，而旧有的力量往往是强大的。不同的人背景不一样，看待事务的标准也不一样，自然就会有理解上的不一致，我们应该尽可能地求得最大的共识。

以下将列举下一些研发规范，主要包括命名约定、索引、表设计、SQL语句、升级/部署脚本规范、数据架构建议这几个部分，以供读者参考。这些规范中有些并不是绝对要遵循的，需要依据现实情况进行权衡取舍。

7.1 命名约定

对于命名，并没有很严苛的规定，但在同一个应用中，建议风格统一。

以下是一些通用的法则，可能有互相冲突的地方，请读者自行衡量取舍。

·命名应有意义，以使用方便记忆、描述性强的可读性名称为第一准则，应尽量避免使用缩写或代码来命名。传统的使用缩写或代码的方法是出于一些历史原因，比如希望节省空间、尽快加载数据等，但随着硬件的快速发展，目前来说这么做的意义不大。

·数据库、表都用小写（尽量不要使用除下划线、小写英文字母之外的其他字符，如果要用下划线，应该尽量保持一致的风格）。

·索引的命名以`idx_`为前缀。

·命名不要过长（应尽量少于25个字符）。

·不要使用保留字。

·注意字段类型的一致性、命名的一致性，同一个字段在不同的表中也应是相同的类型或长度。

·如果同一个数据库下有不同的应用模块，则可以考虑对表名用不同的前缀标识。

·备份表时加上时间标识。

·新建库必须提供库名，库的命名规则必须契合所属业务的特点，新建库必须说明需要授权的用户，若要新建用户，则必须提供用户名，用户名命名规则要契合业务。

7.2 索引

- 建议索引中的字段数量不要超过5个。
- 单张表的索引数量建议控制在5个以内。
- 唯一键和主键不要重复。
- 索引字段的顺序需要考虑字段唯一值的个数，选择性高的字段一般放在前面。
- `ORDER BY`、`GROUP BY`、`DISTINCT`的字段需要放在复合索引的后面，也就是说，复合索引的前面部分用于等值查询，后面的部分用于排序。
- 使用`EXPLAIN`判断SQL语句是否合理使用了索引，尽量避免Extra列出现`Using File Sort, Using Temporary`。
- `UPDATE`、`DELETE`语句需要根据`WHERE`条件添加索引。
- 建议不要使用“`like%value`”的形式，因为MySQL仅支持最左前缀索引。
- 对长度过长的`VARCHAR`字段（比如网页地址）建立索引时，需要增加散列字段，对`VARCHAR`使用散列算法时，散列后的字段最好是整型，然后对该字段建立索引。
- 存储域名地址时，可以考虑采用反向存储的方法，比如把`news.sohu.com`存储为`com.sohu.news`，方便在其上构建索引和进行统计。
- 合理地创建复合索引，复合索引(`a,b,c`)可以用于“`where a=?`”、“`where a=?and b=?`”、“`where a=?and b=?and c=?`”等形式，但对于“`where a=?`”的查询，可能会比仅仅在`a`列上创建单列索引查询要慢，因此需要在空间和效率上达成平衡。
- 合理地利用覆盖索引。由于覆盖索引一般常驻于内存中，因此可以大大提高查询速度。
- 把范围条件放到复合索引的最后，`WHERE`条件中的范围条件（`BETWEEN`、`<`、`<=`、`>`、`>=`）会导致后面的条件使用不了索引。

7.3 表设计

·如果没有特殊的情况，建议选择InnoDB引擎。

·每个表都应该有主键，可选择自增字段，或者整型字段。使用UNSIGNED整型可以增加取值的范围。例外的情况是，一些应用会频繁地基于某些字段进行检索，设计人员可能会认为这些字段/字段组合更适合做主键，因为它们更自然、更高效。

·尽量将字段设置成NOT NULL。如果没有特殊的理由，建议将字段定义为NOT NULL。如果将字段设置成一个空字符串或设置成0值并没有什么不同，都不会影响到应用逻辑，那么就可以将这个字段设置为NOT NULL。NULL值的存储需要额外的空间，且会导致比较运算更为复杂，这会使优化器更难以优化SQL。当然，是否设置为NULL更应取决于你的业务逻辑，如果你确实需要，那么就设置它允许NULL值，NULL值虽然会导致比较运算更加复杂，但这比因为定义了NOT NULL默认值而导致应用逻辑出现异常要好。

·使用更短小的列，比如短整型。整型列的执行速度往往更快。

·考虑使用垂直分区。比如，我们可以把大字段或使用不频繁的字段分离到另外的表中，这样做可以减少表的大小，让表执行得更快。我们还可以把一个频繁更新的字段放到另外的表中，因为频繁更新的字段会导致MySQL Query Cache里相关的结果集频繁失效，可能会影响性能。需要留意的一点是，垂直分区的目的是为了优化性能，但如果将字段分离到了分离表后，又经常需要建立连接，那可能就会得不偿失了，所以，我们要确保分离的表不会经常进行连接，这时，用程序进行连接是一个可以考虑的办法。

·存储精确浮点数时必须使用DECIMAL替代FLOAT和DOUBLE。

·建议使用UNSIGNED类型存储非负值。

·建议使用INT UNSIGNED存储IPV4。可以使用INET_ATON()、INET_NTOA()函数进行转换，PHP里也有类似的函数如ip2long()、long2ip()。

·整形定义中不添加显示长度的值，比如使用INT，而不是INT(4)。

·建议不要使用ENUM类型。

·尽可能不要使用TEXT、BLOB类型。

·在VARCHAR(N)中，N表示的是字符数而不是字节数，比如VARCHAR(255)，最大可存储255个汉字。需要根据实际的宽度来选择N。此外，N应尽可能地小，因为在MySQL的一个表中，所有的VARCHAR字段的最大长度是65535个字节，进行排序和创建临时表一类的内存操作时，会使用N的长度申请内存（对于这一点，MySQL 5.7后有了改进）。

·字符集建议选择UTF-8。

·存储年时使用YEAR类型。

·存储日期时使用DATE类型。

·存储时间时（精确到秒）建议使用TIMESTAMP类型，因为TIMESTAMP使用的是4字节，DATETIME使用的是8个字节。

·不要在数据库中使用VARBINARY或BLOB存储图片及文件等。MySQL并不适合大量存储这种类型的文件。

·JOIN（连接）字段在不同表中的类型和命名要一致。

·如果变更表结构可能会影响性能，则需要通知DBA审核。

7.4 SQL语句

执行一些大的DELETE、UPDATE、INSERT操作时要慎重，特别是对于业务繁忙的系统，要尽量避免对线上业务产生影响。长时间的锁表，可能会导致线上部分查询被阻塞，甚至导致Web应用服务器宕机。解决的方案是，尽可能早地释放资源，尽可能把大操作切割为小的操作，比如使用LIMIT子句限制每次操作的记录数，也可以利用一些日期字段，基于更小粒度的时间范围进行操作。

我们也可以基于自增字段ID分批分段删除数据，如下的例子，是一个定时删除线上数据的脚本，interval变量用于设置每次循环删除的记录数，i变量用于控制循环的次数。由于在删除记录的同时，可能也插入了记录，因此设置为最后一次删除的记录数小于500（\$delRow-le 500）时，退出循环。

```
interval=200000
i=1
while [ $i -lt 100 ]
do
    delRow=`mysql db_name 2>>$logFile <<EOF
        set @minMid=(select min(id) from table_name);
        delete from table_name where id < @minMid + $interval + 500 and date_time <
2014-10-10 00:00:00;
    ;
        select ROW_COUNT();
EOF`
    if [ $? -ne 0 ] ; then
        echo "delete table_name failed"
        tee -a $logFile
        exit 1
    fi
    echo "$i round: delete $delRow rows"
    if [ $delRow -le 500 ] ; then
        break
    fi
    sleep 1
    i=$((i + 1))
done
```

由于篇幅有限，笔者对以上代码做了适当简化。大家可以将上述代码自行改造为适合自己的形式。下面列举一些相应的法则。

- 不要使用ORDER BY RAND()。
- 避免使用SELECT*语句，SELECT语句只用于获取需要的字段。
- 使用预编译语句（prepared statement），可以提高性能并且防范SQL注入攻击。
- 分割大操作。
- SQL语句中IN包含的值不应过多，建议少于100。
- 一般情况下在UPDATE、DELETE语句中不要使用LIMIT。
- WHERE条件语句中必须使用合适的类型，避免MySQL进行隐式类型转化。
- INSERT语句必须显式地指明字段名称，不要使用INSERT INTO table()。
- 避免在SQL语句中进行数学运算或函数运算，避免将业务逻辑和数据存储耦合在一起。
- INSERT语句如果使用批量提交（如INSERT INTO table VALUES(),0,0.....），那么VALUES的个数不应过多。一次性提交过多的记录，会导致线上I/O紧张，出现慢查询。

- 避免使用存储过程、触发器、函数等，这些特性会将业务逻辑和数据库耦合在一起，并且MySQL的存储过程、触发器和函数中可能会存在一些Bug。
 - 应尽量避免使用连接（JOIN），连接的表也不宜过多。
 - 应使用合理的SQL语句以减少与数据库的交互次数。
 - 建议使用合理的分页技术以提高操作的效率。
- 如果性能没有问题，则只在主库上执行后台查询或统计功能。如果必须在从库上执行大的查询，那么应该先通知DBA增加专门用于生产查询的从库。

7.5 SQL脚本

·SQL脚本必须去除^M符号。Windows系统中，每行的结尾是“<回车><换行>”，即“\r\n”；Mac系统里，每行的结尾是“<回车>”，即“\r”。Unix/Linux系统里，每行的结尾是换行CR，即“\n”。三个系统行的结尾各不相同，这会导致的一个直接后果是，Unix/Mac系统下的文件在Windows里打开时，所有的文字会变成一行；而Windows里的文件在Unix/Mac下打开，在每行的结尾可能会多出一个^M符号。而在SQL脚本中，必须要将此符号去除。

·对于存储过程或触发器，升级脚本里应该正确设置分隔符（DELIMITER）。

·对于函数，需要确认DETERMINISTIC。

·如果没有特殊需要，应该一律使用InnoDB引擎和utf8字符集。升级脚本应尽量做到方便回滚、可重复执行。

·必须保证注释的有效性（注：MySQL注释可以使用“--”、“#”或“/* */”，其中“--”后面跟内容时一定要有空格，由于“--”这种注释方法经常导致出错，建议统一使用“#”进行注释）。

·对一个表的表结构的变更，应合并为一条SQL实现。

·SQL文件必须是UTF-8无BOM格式的文件。对于存在非英文字符的升级文件，可以用file命令确认它是否为一个UTF-8编码的文件。例如：

```
[linux1]$ file upgrade.sql
upgrade.sql: UTF-8 Unicode text, with very long lines
```

需要留意的是，英语字母的utf8编码和ASCII编码是一样的。对于一个全英文字母的文件，file命令不会指明这是一个UTF-8编码的文件。file命令对于GBK等字符集可能也会识别不佳。

对于开发和测试环境，建议制订严格的规范，让大家都使用UTF-8编码的文件。可以使用enca、iconv等命令批量转换文件。

iconv的命令格式为：iconv-fencoding-t encoding infile

iconv-l可列出所支持的字符集。

如下命令将转换GBK字符集的aaa.txt文件为utf8字符集的bbb.txt。

```
iconv -fgbk -t utf-8 aaa.txt > bbb.txt
```

一些编辑工具可以轻易地转换文件格式，图7-1展示了notepad++转换编码的菜单项。



图7-1 notepad++转换文件的编码为UTF-8无BOM格式

·一些初始化数据的操作，也可以用mysqldump导出测试/开发环境数据，然后提交给DBA升级生产环境数据库。mysqldump可以保持最佳的兼容性。而其他的客户端工具导出的文件则可能存在一些异常或不兼容的情况。

·导出导入数据时需要注意MySQL Server和客户端工具的版本。

由于一般软件都是向后兼容的，因此在高低版本间导出导入数据时，如果大版本是一致的，比如，都是MySQL 5.1，一般是不会有什麼问题的。但如果大版本不一致，则可能存在兼容性的问题，如从MySQL 5.0导入到5.1，或者从MySQL 5.1导入到5.0，请尽量遵循以下原则。

从MySQL Server低版本导入数据到MySQL Server高版本时，应该直接以高版本的mysqldump导出，然后导入高版本的MySQL Server中，当然，以低版本的mysqldump导出可能也行。

从MySQL Server高版本导入数据到MySQL Server低版本，应该以低版本的mysqldump导出，然后再导入低版本的MySQL Server。

7.6 数据架构的建议

- 每张表的数据量控制在5000万以下。
- 推荐使用CRC32求余（或者类似的算术算法）进行分表，表名后缀使用数字，数字必须从0开始并等宽，比如散100张表，后缀则是从00-99。
- 使用时间分表，表名后缀必须使用固定的格式，比如按日分表为user_20110101。

7.7 开发环境、测试环境的配置参数建议

假设我们统一字符集为utf8，统一默认引擎为InnoDB，那么建议默认的配置文件my.cnf如下，这份配置文件没有进行关注性能方面的调整，大家可以对照自己的环境修改或增加适当的参数。

```
[client]
port          = 3306
socket        = /tmp/mysql.sock
default-character-set = utf8
[mysqld]
character-set-server = utf8
port          = 3306
socket        = /tmp/mysql.sock
user          = mysql
skip-external-locking
max_connections=3000
max_connect_errors=3000
thread_cache_size = 300
skip-name-resolve
server-id      = 1
binlog_format=mixed
expire_logs_days = 8
sync_binlog=60
innodb_log_file_size = 256M
default-storage-engine=innodb
[mysqldump]
quick
max_allowed_packet = 16M
[mysql]
no-auto-rehash
# Remove the next comment character if you are not familiar with SQL
#safe-updates
default-character-set = utf8
```

7.8 数据规划表

数据库是一项比较紧缺的资源，往往需要进行数据规划和资源申请。表7-1是一个申请资源的范本表，可以作为研发团队提交给DBA进行申请资源之用。由于互联网业务的变化可能会很快，往往难以准确地估计数据量和业务量的增长速度，所以，对于这两项可以要求不必非常准确，但最好不要有数量级的估算错误，你规划得越准确，后续的运维成本就越低，调整的代价就越小。

表7-1 申请资源范本表

类型	数据	类型	数据
insert 事务 / 天:		数据重要程度:	
update、delete 事务 / 天:		数据敏感程度:	
select 次数 / 天:		数据保留时长:	
峰值事务增比幅度:		预计三个月后的数据文件大小:	
长查询事务:		预计一年以后的数据文件大小:	
部署并发连接数:			

其中，峰值事务增比幅度=最高峰值事务/平均事务。

对于长查询事务，因为数据库不是很擅长同时处理批量大事务和实时短事务，因此对于线上的繁忙生产系统，一般是不允许有很多长查询存在的，以免影响线上业务。如果有统计类的分析业务，则建议尽早规划，将统计数据分离到其他的数据库实例。

关于数据文件的大小，建议使用真实的数据进行估算。如输入30万条数据，然后使用如下查询验证数据大小。

```
select sum(data_length+index_length) from information_schema.tables where table_schema='db_name' and table_name='table_name';
```

由以上结果可以估算出100万数据的大小。

7.9 其他规范

- 批量导入、导出数据时DBA需要进行审查，并在执行过程中观察服务。
- 批量更新数据时，如执行UPDATE、DELETE操作，DBA也要进行审查，并在执行过程中观察服务。
- 产品出现非数据库平台运维导致的问题和故障时，请及时通知DBA，以便于维护服务的稳定性。
- 业务部门推广活动，请提前通知DBA进行服务和访问评估。
- 如果业务部门出现人为误操作而导致数据丢失，则需要恢复数据，请在第一时间通知DBA，并提供数据丢失的准确时间，误操作语句等重要线索。



小结 规范的根本目的是为了帮助开发、释放人的潜力，提高生产力，而不是约束人，让人失去发挥的空间。标准的建设任重而道远，在制定的过程中，前期宜宽不宜紧，逐渐收集信息，提高规范的适应性，最终是可以达到一个平衡的。友好的规范既能保证运维的安全、便捷，也能让研发、测试团队的工作更加高效。它还应该是一个知识的集聚地，让接触规范的人尽快变得训练有素。

第三部分 测试篇

测试需要掌握的知识面很广泛，本篇的关注点是数据库的性能测试和压力测试，对于其他领域的测试，由于涉猎不多，笔者就不做叙述了。DBA的工作职责之一就是评估软硬件，这往往是一项很耗时的工作，本书将分两个章节为读者介绍数据库的性能、压力测试所需要掌握的理论知识，并提供一个简单的基准测试模型以供大家参考。这部分内容对于大部分中小型公司来说应该够用了。

第8章 测试基础

本章将为读者介绍测试数据库要掌握的一些概念、步骤和注意事项。很多时候，我们在做架构设计时会拿不定主意，而这是源于对软硬件的极限不是很清楚，通过测试所获得的经验将为我们进行决策提供依据。在做测试之前，我们需要知道应该如何测试，以及为什么要这样测试。随着经验的增长，我们将越来越擅长于数据库软硬件的测试，还可以根据自己的需求灵活地进行测试工作。

8.1 基础概念

数据库性能测试一般是指通过运行测试程序来衡量硬件或软件（编译器、数据库等）在不同架构下的性能。测试的含义很广，包括数据流各个环节的测试，本书如果不加以特别说明，指的就是数据库的性能测试或压力测试。在现实生产环境中，对于性能测试或压力测试并没有进行清晰地划分，本书也不会分别加以论述，我们可以认为压力测试是性能测试的一个特例。

衡量数据库性能的主要指标包括事务吞吐率和响应时间，同样，测试的时候也主要是考虑这两个指标。事务吞吐率是指数据库操作的速率，即每秒能完成多少事务，由于MySQL InnoDB默认的模式是自动提交，所以也可以近似地将其看作每秒查询数。响应时间指的是响应请求的总耗时，包括等待时间、执行时间及传输数据的时间。现实中，我们往往过于看重事务吞吐率，而忽略了响应时间，在生产环境下，应该意识到，合理的响应时间范围内的事务吞吐率才有意义。因为，如果没有稳定、良好的用户体验，事务吞吐率再高也没有什么意义。

8.2 性能测试的目的

我们可能会出于不同的目的对数据库主机的性能进行测试，具体测试内容如下。

- 建立自己的基准指标，也就是基准测试。
- 在采购服务器时，可能需要测试不同软硬件组合配置下的数据库性能，以选取性价比较高的那个方案。
- 对比不同系统参数或数据库参数配置下的数据库性能。
- 对比不同的数据库产品。
- 对比数据库不同版本之间的差异。
- 对一些新特性进行试用和验证。
- 对一些操作系统补丁和数据库补丁进行验证。
- 对比不同的操作系统、文件系统和库的差异。

如果都是比较成熟的数据库产品，那将很难证明在所有指标上，一个产品完胜另外一个产品，产品的设计哲学往往决定了它的优势和劣势，或者说安全、效率、价格、稳定这些因素往往不可兼得。所以我们进行测试的目的不是要证明存在一个完美的产品，而是在损失可以接受的范围内，进行合理地软硬件配置。

比如，插入数据的速度慢一些往往无关紧要，如果可以有更高的压缩率、更高的存储效率的话，那么比较低的插入速度是可以接受的。

对于数据库产品来说，除了传统的性能指标之外，还需要考虑一些非常重要的影响现实决策的因素。比如灾难恢复、存储效率、对于复杂业务逻辑的支持、对于其他数据库产品的兼容程度等，这些内容在测试篇中不会加以阐述，在性能调优与架构篇中会详细讲述这部分的内容，以帮助大家了解如何选择一个适合自己业务的数据库产品。

8.3 基准测试

基准测试是我们依据自身的软硬件配置所做的一个数据库性能测试，它能够尽可能地覆盖生产中的一般场景。随着软硬件的升级换代，基准测试的相应指标可能也需要做一些改变。

很多软硬件厂商官方测试结果的数据指标都非常好，但这些往往都是不可信的，它们可能经过了特殊的调整来适应基准测试软件，从而回避了自身的不足之处，他们更多是希望展示自己的产品在性能测试中的亮点，因此这些测试结果不太可能适用于真实的世界。不同的公司有不同的业务特点，所以我们有必要建立自己的测试基准，保存自己的历史测试数据，以便衡量不同的主机、软件、架构及不同时期的性能数据。虽然很难实现完全适合自己的业务模型，但至少能提供一个相对可靠的模型，可以用作采购机器、选择数据库产品、启用数据库新特性等的依据。

我们可以依据基准测试的数据来猜测系统大概还有多少性能余量，但由于测试工具存在一定的局限性，因此很难用它来模拟真实场景，所以需要谨慎对待基准测试的数据。

我们应该对于系统的可扩展性、不同数据量下的性能吞吐有一个大概的认识，预先判断瓶颈点可能会出现在哪里。这些认识和判断往往依赖于经验的累计，随着经验的增长，你自然而然会具备一些意识，这个时候，就可以有针对性地进行测试了，可以更有效地利用基准测试数据了。而且，一旦我们产生某个想法，就能知道应该改变哪些软硬件配置来验证自己的想法。

一个好的基准测试，应满足如下的一些要素。

(1) 有现实意义

基准测试需要具有现实意义，工作负荷、样本数据、系统配置应该和我们测试的目的相关，这样才更有实际意义。

(2) 具有可观察性、易理解、文档化

基准测试必须充分文档化，其他人在阅读文档时能够知道你的软硬件环境配置是如何进行测试的，可能还要附上你的配置文件。并不是所有的人都专业，如果你不说明自己的系统、软件版本、负载等信息，其他人在其他系统上可能会得到不同的测试结果。测试结果往往要在一定的上下文中才有意义，比如一个数据库的I/O测试可能需要包含如下信息：负载是什么样的？使用了什么软硬件、什么测试工具进行测试？数据库是什么版本？测试环境是如何部署的？数据文件多大？数据写入频率如何？数据文件磁盘空间占比如何？使用何种方式写入数据文件？使用何种方式写日志文件？使用独立表空间还是共享表空间？使用的是什么文件系统？使用的是什么I/O调度算法？磁盘阵列是什么RAID级别？有带电池的RAID卡吗？信息记录得越详细越好，不仅方便自己以后参考，也方便其他人对比不同配置下的测试结果。

(3) 可运行且具有可重复性

基准测试是可以重复进行得到类似结果的。所以务必要减少干扰因素，尽可能让其他人可以按照你文档描述的步骤得到一样的结果。当然，也不能忽视干扰的存在，比如定时守护，其他用户的操作等。对于云上的环境来说，由于对其他用户不可见，因此相对于传统的主机环境来说，更加难以确认干扰。基于此，我们需要熟悉数据流的各个环节，比如负载均衡设备、Web服务器、数据库服务器、应用服务器、存储设备等，将这些环节映射到我们的模型中，可以帮助我们发现一些之前被忽略的干扰源。

(4) 收集足够的信息

基准测试应该尽可能地收集信息，比如内存占用、I/O性能、CPU性能等。收集尽可能多的信息总是一件好事，因为这样做有利于分析问题和发现问题。

(5) 有分析结果

要对基准测试结果进行分析、看和我们预期的是否一样，和经验常识是否一致。不能只提供数据而不提供分析结果。

(6) 要对基准测试结果进行解释和说明

我们应该说明测试结果中的一些异常状况，比如是否有错误、异常或干扰，如果有一些不可理解的地方，也请描述出来，也许有经验的其他人员可以帮助你进行分析。如果和我们预期的不一致，那么也有可能是我们的测试方法有问题，或者被其他的因素干扰了。总之，如果测试结果有很多不能解读的地方，那么建议不要在公开场合发布。此外，因为基准测试很耗时，所以有时会让初级工程师进行基准测试，然后让高级工程师查看性能记录，并分析和解释基准测试数据。这样做并不好，最好是进行基准测试的时候就能够实时查看。

8.4 性能/基准测试的步骤

性能测试需要合理的计划和有条理的步骤，不能随意得出结论，性能测试的大概步骤如下。

- 1) 明确测试目的。
- 2) 设计测试模型。
- 3) 准备测试集群环境。
- 4) 准备压力测试工具或编写压力测试脚本。
- 5) 明确性能指标并加以监控。
- 6) 根据2)设计的测试模型准备测试数据。
- 7) 测试执行。
- 8) 测试分析。

第9章会详述如何进行测试。

8.5 测试的注意事项

1) 需要明白，干扰是必然存在的。

干扰是必然存在的，比如定时守护、其他用户的操作等。性能测试所处的环境可能是不干净的，即使你认为很干净了，仍然可能有你所不知道的因素影响测试结果。干扰的来源可能不那么清晰，如果你需要仔细研究系统性能，你就需要确定它。数据流的各个环节，如负载均衡设备、Web服务器、数据库服务器、应用服务器、存储设备都可能存在干扰，而有些环节你不能忽略。对于一些云上的环境，由于你和其他用户共享资源，其他用户的活动也可能会影响到你，而你在一个客户环境内，是很难知道物理系统的资源竞争的。

现在的应用环境，往往包含了多个组件，如负载均衡软硬件设备、Web服务器、数据库服务器、存储系统等。有一个足够真实的模拟环境，可以及早发现干扰的源头。各个组件对照物理环境独立部署，互不影响，可以更好地确保测试结果的可靠性。

2) 性能/压力测试，往往需要时间预热，需要不那么平均分布的数据。

笔者见过很多不完善的测试报告，可能是测试者为了赶进度，测试时间比较短，这点可以理解，因为大家的时间都很紧张。但实际上，我们的测试是需要足够多的时间的，要有足够多的时间进行预热，当一些热点数据加载到内存中时，数据才可能更符合实际生产情况。现在流行的测试工具或方法，往往是对平均分布的数据进行测试，但真实的负载往往是不均匀的，可能某部分数据比较“热”，某部分数据则基本没有被访问，或者基于某些索引值只有少量结果，而另一些索引值则会检索到大量的记录，所以如果我们的真实数据确实存在比较突出的数据不均匀现象，那么测试的时候最好也让数据变得不均匀。在真实环境中，数据往往也有“碎片”，很多性能测试，往往就是直接装载数据，然后马上开始进行测试，但实际上，应该尽量采取一些操作，让数据变得不那么“整齐”，比如在INSERT、UPDATE或DELETE数据的时候按随机的key顺序进行操作，有“碎片”的数据应该是正常的，应该模拟出这种效果。

3) 性能/压力测试，需要真实的数据。数据量不够大，往往难以反映真实的瓶颈所在。

4) 模拟真实的环境总是困难的，从真实环境引流是一个可以考虑的策略。

5) 测试程序应该是多线程的。如果是单线程的话，则需要多个实例来运行，以提高吞吐率。

6) 测试需要和各方都进行信息沟通，在充分了解软件的情况下再设计测试场景。



小结 本章叙述了性能测试需要掌握的一些基础知识和方法学，它是一项繁琐又耗时的工作，甚至有时会给你带来挫败感。掌握足够多的理论，有其他领域的知识储备，才能够选择正确的测试策略，设计良好的测试步骤，从而达到测试的目的。测试之后的文档整理工作也很重要，虽然不那么有趣，但是如果要发布你的测试，让别人知道你的成果，你就应该认真对待它。

第9章 测试实践

在第8章中介绍了测试所需要的理论知识，本章将为读者讲述实际的测试过程。实际测试一般包括硬件测试、MySQL基准测试及应用服务压力测试，下面将分别讲述这三方面的内容。此外，测试工具的选择也很重要，本章将为读者介绍两个常用的工具sysbench和mysqlslap。

9.1 硬件测试

9.1.1 概述

有时我们出于一些原因，需要进行硬件的测试。比如，软件架构很复杂，难以模拟，这时我们可以大致测量一些硬件指标，建立比较基本的性能和容量模型。比如，在升级硬件的时候，往往不会选择升级所有硬件，而是更着重于首先升级系统紧缺的资源，例如I/O，那么这时就需要专门针对不同的硬件配置，来测试I/O的提升效果。再比如，硬件厂商往往夸大其词，这时我们就需要运用自己认为可靠的工具去实际验证，确认新的硬件在一些关键指标上是否有大幅度的提升。

现实中，硬件和数据库的测试工具并没有划分得很清晰，一些数据库测试工具，本身就可以对各种硬件资源进行压力测试。比如sysbench，既可以测试数据库，又可以用来测试CPU、内存等硬件资源。

本书将主要关注Linux下的软硬件测试。

一些需要测量的硬件有：内存、CPU、磁盘、网卡、网络等。

网上也有很多优秀的开源测试工具，这里仅列出一些常用的测试工具。

- 内存测试的工具有sysbench、stream、RamSpeed、stress等。
- CPU测试的工具有sysbench、cpuburn、stress等。
- 磁盘测试的工具有sysbench、iozone等。

9.1.2 CPU测试

sysbench命令通过进行素数运算来测试CPU的性能。cpu-max-prime选项指定了最大的素数为20000，如下：

```
sysbench --test=cpu --cpu-max-prime=20000 run
```

对于CPU的测试，我们要重点关注三个指标：上下文切换（context switch）、运行队列（run queue）和使用率（utilization）。

(1) 上下文切换

在操作系统中，若要将CPU切换到另一个进程，需要保存当前进程的状态并恢复另一个进程的状态：即将当前运行任务转为就绪（或者挂起、删除）状态，让另一个被选定的就绪任务成为当前任务。上下文切换包括保存当前任务的运行环境，恢复将要运行任务的运行环境等。过多的上下文切换会给系统造成很大的开销。

(2) 运行队列

当Linux内核要寻找一个新的进程在CPU上运行时，需要考虑处于可运行状态的进程，运行队列容纳了系统中所有可运行的进程。理想情况下，调度器会让队列中的进程不断运行，如果CPU过载，就会出现调度器跟不上系统的情况，从而导致可运行的进程填满队列。队列越大，程序执行的时间就越长。“load”用于表示正在等待运行的队列长度，top命令可以让我们看到在一分钟、5分钟和15分钟内CPU运行队列的大小。这个值越大则表明系统负荷越大。

(3) 使用率

CPU使用率可分为以下几个部分。

·**User Time**: 执行用户进程的时间占全部时间的百分比，通常是期望这个值越高越好。

·**System Time**: CPU内核运行及中断的时间占全部时间的百分比，通常是希望这个值越低越好，系统CPU占用率过高时，通常表明系统的某部分存在瓶颈。

·**Wait I/O**: I/O等待的CPU时间占全部时间的百分比，如果I/O等待过高，那么说明系统中存在I/O瓶颈。

·**Idle**: CPU处于Idle状态的时间占全部时间的百分比。

以下是一些很普遍的CPU性能要求，供大家参考。

·对于CPU的每一个核来说运行队列不要超过3，例如，如果是双核CPU就不要超过6。

·如果CPU正处于满负荷运行状态，那么使用率应该符合下列分布。

User Time: 65%~70%

System Time: 30%~35%

Idle: 0%~5%

·对于上下文切换，要结合CPU使用率来看，如果CPU使用率满足上述分布，那么大量的上下文切换也是可以接受的。常用的监视上下文切换的工具有：vmstat、top、dstat和mpstat。

通过禁用或启用CPU核，可以进行不同CPU核数的性能和压力测试。

一般来说，对于数据库，比如MySQL，一条查询使用一颗CPU核，MySQL还不具备一个查询可以在多颗CPU核中并行运行的能力。数据库操作中如果有大量的内存读，比如读取索引、读取InnoDB buffer里的数据，那么往往会展现出CPU瓶颈，内存复制也是如此。

9.1.3 内存测试

使用sysbench测试内存的命令如下。

```
sysbench --test=memory --memory-block-size=8K --memory-total-size=4G run
```

上述参数指定了本次测试的整个过程是在内存中传输4GB的数据量，每个块（block）的大小为8KB。

9.1.4 I/O测试

1. 普通磁盘阵列测试

(1) 使用hdparm

磁盘性能测试可采用hdparm命令，对于上线的服务器，为了简便，可用自带的命令hdparm初步判断磁盘的性能，确定工作是否正常。如果要更可靠地验证磁盘、RAID性能，建议使用专门的测试工具，如iozone或sysbench。

hdparm的使用示例如下。

如下命令可查看某SATA硬盘的设置。

```
hdparm /dev/sda
/dev/sda:
IO_support      = 0 (default 16-bit)
readonly        = 0 (off)
readahead       = 256 (on)
geometry        = 60801/255/63, sectors = 976773168, start = 0
```

解释：geometry=60801【柱面数】/255【磁头数】/63【扇区数】，sectors=976773168【总扇区数】，start=0【起始扇区数】。

如下命令可查看SSD的设置。

```
hdparm /dev/sdc
/dev/sdc:
IO_support      = 0 (default 16-bit)
readonly        = 0 (off)
readahead       = 256 (on)
geometry        = 36481/255/63, sectors = 586072368, start = 0
```

以下命令用于检测硬盘的读取速率（buffered disk reads），需要多运行几次，以便更准确。

```
hdparm -t /dev/sda
/dev/sda:
Timing buffered disk reads: 426 MB in 3.00 seconds = 141.82 MB/sec
```

以下命令用于检测硬盘快取时的读取速率（cached reads），需要多运行几次，以便更准确。

```
hdparm -T /dev/sda
/dev/sda:
Timing cached reads: 22096 MB in 2.00 seconds = 11070.65 MB/sec
```

其中的参数解释如下。

·**-t**: 衡量顺序读取的能力，不经过操作系统缓存。

·**-T**: 衡量系统的吞吐性能，未访问底层的物理设备，直接从操作系统缓存里读取数据。

(2) 使用dd

通过dd命令可实现用指定大小的块复制一个文件，并在复制的同时进行指定的转换。

dd命令的语法格式如下。

```
dd if=<source> of=<target> bs=<byte size> skip=<blocks> seek=<blocks> count=<blocks> conv=<conversion>
```

以下仅以RHEL 5.4的dd为例来说明参数，其他平台所支持的参数可能不一样。

·**if=<source>**: 指定源文件。默认为标准输入。

·**of=<target>**: 指定目标文件。默认为标准输出。

·**bs=<bytes>**: 同时设置读入/输出的块大小为bytes个字节。也可以指定其他单位，如KB、MB等。

·**skip=<blocks>**: 从输入文件的开头跳过blocks个块后再开始复制。

- seek=<blocks>**: 从输出文件的开头跳过blocks个块后再开始复制。
- count=<blocks>**: 仅复制blocks个块。
- conv=<conversion>**: 用指定的参数转换文件，参数如下，可组合使用，中间用逗号分隔。
 - ascii**: 转换ebcdic为ascii。
 - ebcdic**: 转换ascii为ebcdic。
 - ibm**: 转换ascii为alternate ebcdic。
 - block**: 使每一行的长度都为cbs，不足部分用空格填充。
 - unblock**: 使每一行的长度都为cbs，不足部分用空格填充。
 - lcase**: 把大写字符转换为小写字符。
 - ucase**: 把小写字符转换为大写字符。
 - swab**: 交换输入的每对字节。
 - noerror**: 如果发生错误，程序也将继续运行。
 - notrunc**: 不截断输出文件。
 - sync**: 填充每个块到指定字节，不足部分用空（NUL）字符补齐。
- dd命令还有一组参数oflag和iflag，用于控制源文件和目标文件的读写方式。

```
iflag=flag[,flag]...
oflag=flag[,flag]...
```

- 标志（flag）可选的值如下。
 - append**: 以追加（append）模式写数据。这个标志仅适用于输出（写文件）。如果和of=file联合使用，则需要同时设置conv=notrunc。
 - dsync**: 采用同步I/O读写数据，确保数据刷新到了磁盘。
 - sync**: 与上者类似，但同时也对元数据生效，在测试数据库机器的磁盘性能时，为了得到更真实的性能数据，建议使用sync或dsync的方式读写数据。
 - direct**: 使用direct I/O操作数据，可以避免操作系统缓存的影响，即读或写文件时越过操作系统的读写缓存。如果指定oflag=direct，则写文件时会忽略缓存的影响；而如果指定iflag=direct，则读文件时会忽略缓存的影响。
 - fullblock**: 为输入积累完整块（仅针对iflag）。
 - nonblock**: 采用非阻塞I/O模式。
 - nofollow**: 不跟随链接文件，即忽略链接文件指向的文件。当从标准输入读取或写入到标准输出时，不要使用此选项。
 - noctt**: 不根据文件的指派控制终端。

例1： 测试磁盘写能力，利用操作系统写缓存。

```
dd if=/dev/zero of=blah.out bs=1M count=2000
```

或者：

```
time (dd if=/dev/zero of=blah.out bs=1M count=2000 ; sync )
```

因为`/dev/zero`是一个伪设备，它只会产生空字符流，对它不会产生I/O，所以，I/O都集中在of文件中，of文件只用于写，所以这个命令相当于测试磁盘的写能力。

`dd`默认的方式不包括“同步（sync）”命令。也就是说，`dd`命令完成之前并没有让系统真正把文件写到磁盘上。脏数据可能还在操作系统的缓存里，并没有刷新到磁盘上。如果内存比较大，我们往往可以看到磁盘的写能力很高。

如果以上命令在一块普通sata硬盘上执行，结果如下，可以看到，每秒写入828MB，显然磁盘的写能力远没有这么高。

```
dd if=/dev/zero of=blah.out bs=1M count=2000
2000+0 records in
2000+0 records out
2097152000 bytes (2.1 GB) copied, 2.53394 seconds, 828 MB/s
```

我们可以更改系统参数，把可用内存降低到很小，或者使用参数`oflag=sync`来确保已将数据刷新到了磁盘。

例2： 使用`oflag=sync`模式测试磁盘写能力，对于数据库机器，一般建议这样测试磁盘。

```
time dd if=/dev/zero of=blah.out oflag=sync bs=1M count=2000
```

对比例1的测试结果，本次测试的写入速度大大降低，这才是比较真实的写入速度。

```
time dd if=/dev/zero of=blah.out oflag=sync bs=1M count=2000
2000+0 records in
2000+0 records out
2097152000 bytes (2.1 GB) copied, 27.861 seconds, 75.3 MB/s
real    0m27.913s
user    0m0.003s
sys     0m3.204s
```

一般来说，随着bs值的增加，吞吐往往会更高。

例3： 测试磁盘读能力。

```
time dd if=/dev/sdb1 of=/dev/null bs=8k
```

因为`/dev/sdb1`是一个物理分区，对它的读取会产生I/O，而`/dev/null`是伪设备，相当于黑洞，of到该设备不会产生I/O，所以，这个命令的I/O只发生在`/dev/sdb1`上，也相当于测试磁盘的读能力。

例4： 测试同时读写的能力。

```
time dd if=/dev/sdb1 of=test1.dbf bs=8k
```

这个命令下，一个是物理分区，一个是实际的文件，对它们的读写都会产生I/O（对`/dev/sdb1`是读，对`test1.dbf`是写），假设它们都在一个磁盘中，这个命令就相当于测试磁盘同时读写的能力。

可以另开一个会话（session），运行命令“`kill -s USR1 pid`”，用于显示`dd`进程的I/O统计，`pid`为正在执行`dd`命令的进程id，命

令如下。

```
$ dd if=/dev/zero of=/dev/null& pid=$!
$ kill -USR1 $pid; sleep 1; kill $pid
```

2.SSD测试

对于SSD，也可以使用dd进行测试和验证，但对于数据库负载，为了更好地模拟负载，建议使用更智能的工具进行测试，如sysbench，sysbench的具体使用方法，请参考9.2.2节。

如下是一些SSD的测试建议。

·测试要尽可能避免缓存（cache）的影响。任何磁盘产品，在碰到I/O瓶颈的时候，都不可能会很快，这个时候才是真正的磁盘的性能。如果我们在测试一些数据库产品的时候，发现有非常高的吞吐率，那么就要思考一下，是不是有什么其他因素影响了测试结果，网上的很多测评就是陷入了这样一个误区，忽略了缓存的影响，这是一个致命的错误。

·要注意碎片、空间占用的影响。随着使用时间的增长，SSD可能有碎片，空间占用率也可能上升，这个时候性能就可能会下降。一般来说，针对SSD的测试需要较长时间，因为可能有垃圾回收（Garbage Collection, GC）机制的影响，需要将其考虑进去，笔者个人偏向于每种测试模型的测试时长均大于1小时。而且，你需要测试不同空间占用比下的性能，现实中，一般如果数据库占用了90%以上的磁盘空间，则必须要考虑扩容，所以对于企业级应用，可以测试一下几乎写满（>90%）情况下的性能，此时的性能更有参考价值。

·RAID卡、I/O控制器、缓存等因素也会影响到SSD的性能。需要留意这一点，高端RAID卡和中低端RAID卡的性能有很大的差别。

·可测试单盘性能，如果需要做RAID，那么可测试一下RAID 5和RAID1+0。出于节省成本和空间的考虑，有些人使用RAID 5。虽然理论上RAID 5的性能会比较差，但RAID卡厂商一般专门提供了优化RAID 5的技术，而且很多情况下I/O也不再成为瓶颈了。如果单盘空间足够大且成本合适的话，那么不做RAID，直接测试单盘性能也是可以的。

·不仅要测试吞吐率，还需要测试响应的稳定性，以反映真实的环境。

·验证案例是否覆盖了所有的情况，而且要使用成熟稳定的工具来完成，以免破坏数据。

·SSD对于温度会比较敏感，需要留意温度的影响。

9.1.5 网络测试

一般网卡出故障的可能性非常低，所以我们可以不用过多地进行验证测试，在系统部署后验证即可，可用ethtool命令验证网卡，也可以使用专门的网络评测工具进行验证，还可以通过网络传输文件来验证，比如，直接ftp一个大文件到ftp服务器以验证网络传输是否正常。

9.2 MySQL测试

9.2.1 概述

MySQL测试的范围很广，我们出于不同的目的进行测试，最常见的是性能基准测试。有时还需要做一些其他测试，比如，验证MySQL的复制特性，在高并发的压力下，不断破坏从库（模拟宕机、磁盘空间满等情况），来查看复制能否顺利进行。比如，通过测试宕机下的灾难恢复性，来衡量灾难恢复所需要的时间等。本节将仅叙述MySQL的性能测试。

影响性能测试的因素较多，除了MySQL自身之外，文件系统、操作系统块大小、应用访问模式、RAID阵列条带大小等诸多因素都对性能有或多或少的影响。本章不会对所有类型都进行测试，但会说明测试的方法及注意事项，读者可以按照自己拟定的方法和步骤测试验证不同软硬件设置下的MySQL性能。

9.2.2 常用测试工具的介绍和使用

MySQL的测试工具，推荐用sysbench。虽然hammerora、super-mark、tpc-c等一些其他工具也很强大，但sysbench的文件I/O测试与InnoDB的行为很相似，针对MySQL也有比较完善的测试模型，还可以方便地修改lua脚本，以实现更强大、更灵活的测试功能。其实，设计sysbench的初衷就是为了衡量MySQL的性能，而很多其他工具，对于MySQL的支持往往只是一个选项，功能还不够强大，难以模拟真实的数据库负载。MySQL自带的mysqlslap也是一个不错的工具，它是从5.1.4版开始的一个MySQL官方提供的压力测试工具，可通过模拟多个并发客户端访问MySQL来执行压力测试。

这两个工具可以满足大部分情况下的性能测试和压力测试。sysbench可以自定义lua脚本，开发人员可以编写适合自己业务逻辑的lua脚本。当然也可以使用其他高级语言编写测试工具，这样会更灵活，更接近实际业务数据库操作。

1.sysbench的使用

目前sysbench主要支持MySQL、PostgreSQL、Oracle这3种数据库。

它主要包括以下几种方式的测试。

·Fileio: 文件I/O测试。

·Cpu: CPU性能测试。

·Memory: 内存性能测试。

·Threads: 线程性能测试。

·Mutex: Mutex性能测试。

·Oltp: OLTP测试，MySQL一般会选择此种测试类型。

(1) 安装

首先，从<https://github.com/akopytov/sysbench>下载源码包，单击Download Zip。然后，按照如下步骤进行安装。

```
unzip sysbench-0.5.zip
cd sysbench-0.5
./autogen.sh
./configure --with-mysql-includes=/usr/local/mysql/include --with-mysql-libs=/usr/local/mysql/lib
make
make install
```

用如上的参数进行编译的话，需要确保你的MySQL lib目录下有对应的库文件，如果没有，则可以下载devel或share包来进行安装。也可以下载MySQL的二进制安装包解压到/usr/local/mysql下。

(2) 开始测试

在sysbench--test=memory命令后添加help可以查看帮助。

```
sysbench --test=memory help
```

一些参数解析如下。

·--percentile 95%: 响应时间，也就是删除5%的响应时间最长的请求，然后从剩余的请求中选取最大的响应时间值。

·--max-time: 运行时间限制，单位是秒。

--num-threads: 线程数。

--max-requests: 查询数限制。

下面来举例说明。

1) CPU性能测试。

```
sysbench --test=cpu --cpu-max-prime=20000 run
```

CPU测试主要是进行素数的运算，在上面的例子中，指定了最大的素数为20000，也可以根据机器CPU的性能来适当调整数值。

如下命令，执行20s就输出了，而不会等待命令执行完。

```
sysbench --test=cpu --cpu-max-prime=20000 run --max-time=20
```

2) 线程测试。

```
sysbench --test=threads --num-threads=64 --thread-yields=1000 --thread-locks=8 run
```

3) 磁盘I/O性能测试。

```
sysbench --test=fileio --num-threads=16 --file-total-size=12G --file-test-mode=rndrw prepare  
sysbench --test=fileio --num-threads=16 --file-total-size=12G --file-test-mode=rndrw run  
sysbench --test=fileio --num-threads=16 --file-total-size=12G --file-test-mode=rndrw cleanup
```

上述代码分为3个步骤，第一条命令初始化文件，第二条命令执行测试，第三条命令清理文件。`--num-threads`参数指定了最大创建16个线程，`--file-total-size`参数指定创建文档的总大小为12GB，`--file-test-mode`指定文档的读写模式为随机读写。

磁盘I/O性能测试是进行数据库基准测试时要着重加以研究的。我们需要衡量各种因素，比如操作类型、读写的频率、I/O大小、是随机读写还是顺序读写、写的类型是异步还是同步、并发线程情况、操作系统缓存状态及文件系统有哪些调优等因素。

文件测试类型（`file-test-mode`）有如下几种。

`seqwr`: 顺序写。

`seqrewr`: 顺序重写（`rewrite`）。

`seqrd`: 顺序读。

`rndrd`: 随机读。

`rndwr`: 随机写。

`rndrw`: 随机读写。

4) 内存测试。

```
sysbench --test=memory --memory-block-size=8K --memory-total-size=4G run
```

上述参数指定了本次测试的整个过程是在内存中传输4GB的数据量，每个块（block）的大小为8KB。

5) OLTP测试。

在测试之前请预先创建数据库，并给予测试用户足够的权限。

```
mysql > create database sbtest;  
mysql > grant all privileges on sbtest.* to test@localhost  
identified by 'test';  
;
```

如下例子演示了多线程如何测试MySQL。

首先初始化数据。

```
sysbench --test=./sysbench/tests/db/oltp.lua --mysql-table-engine=innodb --oltp-tables-count=256 --oltp-table-size=1000000 --mysql-user=test --mysql-password=test --mysql-socket=/tmp/mysql.sock prepare
```

上述参数指定了本次测试的表存储引擎类型为InnoDB，指定了表的最大记录数为1000000，初始化生成256个表。测试OLTP时，可以自己先创建数据库sbtest，或者自己用参数--mysql-db来指定其他数据库。

然后进行实际测试，测试模型是OLTP，并发8个线程，执行1个小时，如下：

```
sysbench --test=./sysbench/tests/db/oltp.lua --oltp-tables-count=256 --oltp-table-size=1000000 --mysql-user=test --mysql-password=test --mysql-socket=/tmp/mysql.sock --max-time=3600 --max-requests=0 --num-threads=8 --report-interval=10 run
```

其中，--report-interval=10表示每10s就输出一次数据，输出格式类似如下。

```
[ 10s] threads: 2, tps: 290.39, reads: 4065.82, writes: 1161.58, response time: 8.65ms (95%), errors: 0.00, reconnects: 0.00  
[ 20s] threads: 2, tps: 270.90, reads: 3795.10, writes: 1083.80, response time: 10.14ms (95%), errors: 0.00, reconnects: 0.00  
[ 30s] threads: 2, tps: 277.40, reads: 3883.50, writes: 1109.40, response time: 9.82ms (95%), errors: 0.00, reconnects: 0.00  
[ 40s] threads: 2, tps: 273.50, reads: 3828.09, writes: 1094.00, response time: 9.93ms (95%), errors: 0.00, reconnects: 0.00
```

测试完成后，清理数据。

```
sysbench --test=./sysbench/tests/db/oltp.lua --oltp-tables-count=256 --oltp-table-size=1000000 --mysql-user=test --mysql-password=test --mysql-socket=/tmp/mysql.sock cleanup
```

2.mysqlslap的使用

mysqlslap是MySQL 5.1.4及以后版本自带的一个用于实现负载性能测试和压力测试的工具。它可以模拟多个客户端对数据库进行施压，并生成报告以衡量数据库的一些指标。

其工作原理可分为如下三个步骤。

- 1) 首先生成测试数据，即创建表，导入数据。这个步骤将使用单个客户端连接执行。
- 2) 然后运行性能测试，可以使用单线程或多线程。
- 3) 最后清理测试数据。这个步骤将使用单个客户端连接执行。

一些使用示例如下所示。参数的具体说明请参考官方文档。

- 1) 分别并发10个线程或100个线程进行混合测试。

```
mysqlslap --uroot --engine=innodb --auto-generate-sql --auto-generate-sql-unique-query-number=100 --auto-generate-sql-unique-write-number=100 --auto-generate-sql-write-number=1000 --create-schema=test --auto-generate-sql-load-type=mixed --concurrency=10,100 --number-of-queries=1000 --iterations=1 --number-char-cols=1 --number-int-cols=8 --auto-generate-sql-secondary-indexes=1 --debug-info --verbose
```

以上命令的大致步骤是，首先生成1000条数据（--number-of-queries=1000），然后进行混合测试（--auto-generate-sql-load-type=mixed，SELECT操作和INSERT操作大致各占一半），此时数据会不断增长。

然后，先使用10个并发线程进行测试，再用100个并发线程进行测试（--concurrency=10,100），进行新的并发测试前会清理和初始化测试数据。

需要留意的是，自动生成的SELECT语句是全表扫描，语句如下。

```
SELECT intcol1,intcol2,intcol3,intcol4,intcol5,intcol6,intcol7,intcol8,charcol1 FROM t1
```

INSERT语句类似如下。

```
INSERT? INTO t1 VALUES (uuid(),389111603,476395693,1231278962,1952007439,1880139043,1004384052,914532...)
```

- 2) 测试基于主键查找的性能。

```
time mysqlslap -uroot --engine=innodb --auto-generate-sql-load-type=key --auto-generate-sql --auto-generate-sql-write-number=100000 --auto-generate-sql-guid-primary --number-char-cols=10 --number-int-cols=10 --concurrency=10,100 --number-of-queries=5000
```

以上命令的大致步骤是，首先创建一个表，使用单线程初始化插入100000条记录，然后并发10条线程执行基于主键的查询。接着删除库表，再初始化插入100000条记录，然后并发100条线程执行基于主键的查询。基于主键的查询可能被缓存，所以有必要生成不同的SELECT语句。

- 3) 生成一张两千万条记录的表，进行混合型负荷测试（SELECT+INSERT），语句如下。

```
mysqlslap -uroot -p --engine=innodb --auto-generate-sql --auto-generate-sql-write-number=20000000 --auto-generate-sql-add-autoincrement --auto-generate-sql-secondary-indexes=2 --concurrency=50 --number-of-queries=1000000 --number-char-cols=3 --number-int-cols=2 --debug-info
```

以上命令的大致步骤是，初始化记录时使用自增主键（--auto-generate-sql-add-autoincrement）并发50个线程进行查询，一共执行100万个查询，也就是说平均每个线程大概执行2万个查询，如果有自增ID，那么SELECT语句是基于自增ID的，这样更能反映生产环境实际情况。

对于如上的命令，一台普通的数据库服务器（SAS硬盘三块：SAS 15K 300G*3，做成了RAID 5），初始化过程中大概会插入2000万条记录，到达1500万条记录的时候，INSERT速率大概可以达到6000~7000条记录每秒。iostat命令显示每秒写入20MB~30MB的数据。

插入2000万条记录，初始化完成后，开始并发50条线程进行混合测试，同时有INSERT和SELECT操作。大概每秒执行INSERT操作600次，SELECT操作500次。

9.2.3 MySQL基准测试模型

1. 指引

基准测试的目的之一是在平时做好数据准备，记录标准的数据库软硬件配置下的性能数据，以便在未来更改数据库配置或调整升级数据库主机时有一个参照。很多人在新机器上线后不想做基准测试，认为基准测试繁琐、耗时，而且难以度量。如果不是标准化的采购、安装、部署，就很难说主机配置得完全正确，所以，如果有一个工具可以预先针对CPU、磁盘、网络、数据库进行压力测试，验证硬件是否已经正确配置，就省事多了。我们还可以把基准测试的历史数据记录下来，在以后进行选购或升级软硬件的时候，再重新进行基准测试，从而验证软硬件的升级效果。

基准测试的一个重要环节就是基准测试模型，我们需要一个相对简单、高效，实现起来成本比较低的模型。想用测试模型来真实反映现实的生产环境是很难做到的，但是我们可以按照数据库的理论和实践，设置一些比较能够反映生产负荷的输入条件，评估测试的输出指标，从而建立可以用来衡量数据库架构、软硬件配置的基准模型。

以下将介绍下具体的思路和设计方法。

- 当我们选择硬件的时候，需要考虑到各项成本，对于项目风险、开发成本和维护成本比较难以衡量，而计算机性能相对来说是更好地限定和比较的，所以可以考虑建立一个MySQL的基准测试模型。

- 性能测试很难模拟真实环境的负荷，一般使用比较简单的模型，因为真实环境下的负荷具有不确定、变化较大、复杂且难以理解等特点，故而难以得出结论，不容易对比。

- 单个产品的基准测试，主要用于对比版本和衡量软硬件调整的效果，对于整个应用系统的测试没有太大的参考意义，应用系统自己的基准测试模型会比单个MySQL测试模型更全面更准确。

- 明确目标后，再进行基准测试，才能更好地选择工具和测试方法。

- 如果是SSD，建议文件的最大空间不超过磁盘空间的85%，以避免SSD空间占比可能带来的性能下降。

- 除了关注吞吐率（TPS），还需要关注响应时间（response time）。

- 需要留意并发性（concurrency），如MySQL Server同时运行的线程数（threads_running）和伸缩性（scalability）等。假如我们增加了一倍线程，那么好的伸缩性就意味着系统的吞吐也可以线性地增加一倍，或者说当我们增加了一倍的硬件资源，那么系统的吞吐也可以翻倍。

- 基准测试需运行足够长的时间。关于数据预热所需要的时间，一般查看吞吐量的曲线图就可以大致判断出来，很多情况下，可能需要运行半个小时以上才会稳定下来。

- 尽量确保测试结果在同样的配置下可以重现。

- 衡量MySQL性能需要考虑诸多因素，包括但不限于以下这些因素。

- 硬件：CPU速度、CPU架构、CPU个数、CPU核数、总线速度、内存访问速度、设备I/O性能、RAID卡、磁盘条带、块大小、网络设备。

- 操作系统：原生API性能、线程、锁、内存、I/O调度算法。

- 客户端连接次数。

- 数据库服务器处理任务的线程个数。

- 数据库设计。

- 数据量。

- 应用类型。

- 数据访问模式：一般来说我们的应用热点数据较小，读远大于写。如果你的应用热点数据比较大，访问各种数据比较分散，分布比较均匀，那么这种测试更考验了数据库的原始性能。

- 数据库版本：社区版、企业版还是第三方分支版本。

·引擎。

·数据库配置。比如NUMA策略、页块大小（page size）、是独立表空间还是共享表空间、顺序访问和随机访问文件的分布、InnoDB buffer pool的大小及其他一些影响重大的参数。

·重要的参数修改，每次尽可能少更改点参数，一次更改太多的参数不容易判断问题所在。

2.模型简介

以下是一个测试MySQL数据库的简单模型。

```
## MySQL基准测试模型介绍。
## 组合以下不同条件
# 测试类型、线程数、表个数、表记录数
# (大小)
# 进行测试

## =====开始测试
=====
## 测试逻辑
:
## 对每种测试类型
## 对各种并发线程数

## 对指定的表个数
## 对不同表大小
## prepare
sysbench 测试
## ,默认测试
1200s
## cleanup
## =====结束测试
=====
```

测试类型一般选择oltp。对于单个表，将按顺序执行如下操作。

- 1) 几个基于主键的查询。
- 2) 主键范围查找。
- 3) 主键范围查找+聚合函数。
- 4) 主键范围查找+文件排序。
- 5) 主键范围查找+临时表+文件排序。
- 6) 更新操作（基于主键查询）。
- 7) 删除操作（基于主键查询）。
- 8) 插入操作。
- 9) 提交。

由于sysbench官方版本的oltp模型里没有复杂的查询连接（JOIN）操作，但两三个表的连接又是比较普遍的，因此可以考虑更改下官方的oltp测试模型，以反映真实的生产负荷。

3.测试脚本示例

如下是一个按照上面的模型编写的脚本示例，这里将叙述它所实现的功能，以及分析几个测试脚本的输出结果。测试脚本的代码请到www.db1110.com上下载。

测试脚本可实现如下功能。

- 1) 收集操作系统、硬件信息和MySQL脚本信息。如下是收集到的一些信息。

```
=====Host=====
Hostname | xxxx
Release | Red Hat Enterprise Linux Server release 5.4 (Tikanga)
Processors | physical = 2, cores = 8, virtual = 16, hyperthreading = yes
Models | 16xIntel(R) Xeon(R) CPU E5520 @ 2.27GHz
Total | 15.6G
# RAID Controller #####
Controller | No RAID controller detected
# Disk Schedulers And Queue Size #####
sda | [cfg] 128
sdb | [cfg] 128
=====MySQL=====
# Report On Port 3306 #####
User | root@localhost
```

```

Hostname | xxxxxxxx
Version   | 5.1.58-log MySQL Community Server (GPL)
Built On  | unknown-linux-gnu x86_64
Databases | 5
# Table cache #####
Size      | 4096
# InnoDB #####
Version  | default
Buffer Pool Size | 2.5G
File Per Table | OFF
Page Size | 16k
Log File Size | 2 * 360.0M = 720.0M
Flush Log At Commit | 2
Thread Concurrency | 16
Txn Isolation Level | READ-COMMITTED
sync_binlog | 20
innodb_max_dirty_pages_pct      = 50

```

- 2) 可以在脚本中调整MySQL的参数设置。
- 3) 收集操作系统的性能信息，在测试期间的内存、CPU、磁盘等各种信息都应该收集起来，不管目前有没有用，收集尽可能多的信息会方便自己以后的分析。
- 4) 能使用sysbench的oltp模型进行测试，并且能够生成图形。

以下是性能测试脚本的运行结果，我们将对这些测试完成后生成的图形做一些分析。

例1：图9-1是不同线程（thread）数量的时候，事务吞吐率（tps）的变化图，数据小于InnoDB缓冲池（0.7382*buffer）。

图9-1中所示的MySQL实例有256张表，每张表有3万记录，数据小于InnoDB缓冲区。随着线程数量的增加，事务吞吐率会缓慢下降，这说明等待的时间在增长，从而导致事务吞吐率下降。

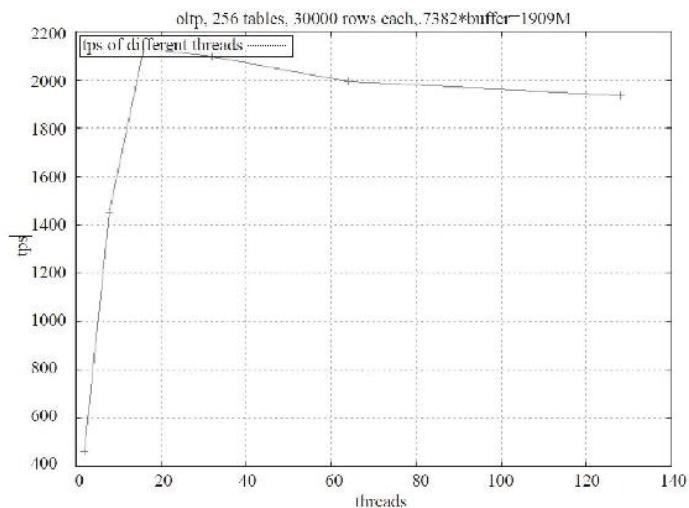


图9-1 数据小于缓冲，线程数改变时事务吞吐率的变化

对于线程数量不断增加的数据库测试，性能吞吐率曲线最开始往往是线性增长的，但终将到达一个拐点，可能到达拐点的时候，是因为某项资源出现了瓶颈，对资源的竞争将开始影响到性能。比如图9-1的压力测试中，随着线程数量的增加，吞吐率也在增加，但如果存在过多的线程，由于CPU资源不够，将导致频繁的上下文切换，从而导致延时增加。如果是CPU资源的瓶颈，那么性能下降得不是很快；但如果是内存瓶颈，则属于另外一种情况，性能可能会急剧变差；磁盘I/O瓶颈同样可能导致急剧的性能变差，请参见下面例2的图。

例2：图9-2和图9-1类似，描述了不同线程数下事务吞吐率的变化，不过数据占用空间远大于InnoDB缓冲池（4.8573*buffer）。可以看到事务吞吐率已经大大下降了。

例3：图9-3描述了不同数据量下，事务吞吐率的变化。在8个线程、256张表的情况下，随着数据量的增长，事务吞吐率逐渐下降。随着数据的不断增长，3万记录也逐渐增长到5万、10万、20、30万，事务吞吐率则从1500tps下降到了400 tps。

由此可以证明，热点数据能否缓存在内存中，对事务的吞吐率影响是很大的。

例4：图9-4由多个子图构成，分别展示了事务吞吐、读、写、响应时间随时间变化的曲线。数据库实例的数据空间占用小于InnoDB缓冲（0.7382*buffer）。

对于我们来说，更有效的数据是性能趋于稳定后的数据，大家做测试的时候，一定要确认自己是否已经运行了足够长时间，可以简单地通过查看事务吞吐率是否趋于稳定来衡量时间的长短。

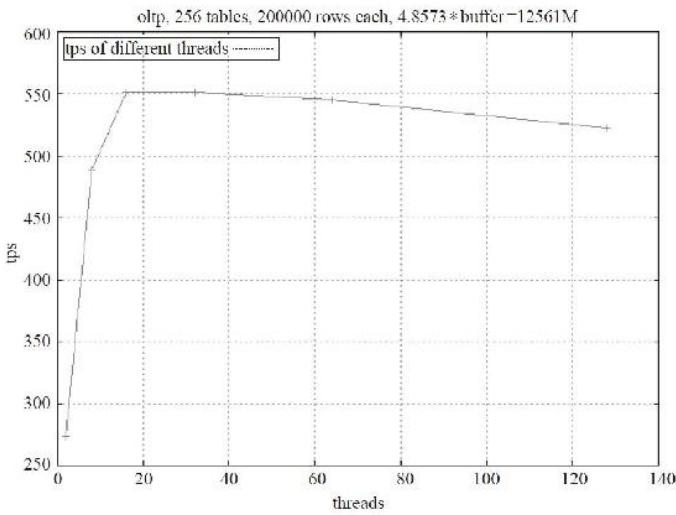


图9-2 数据远大于缓冲，线程数改变时事务吞吐率的变化

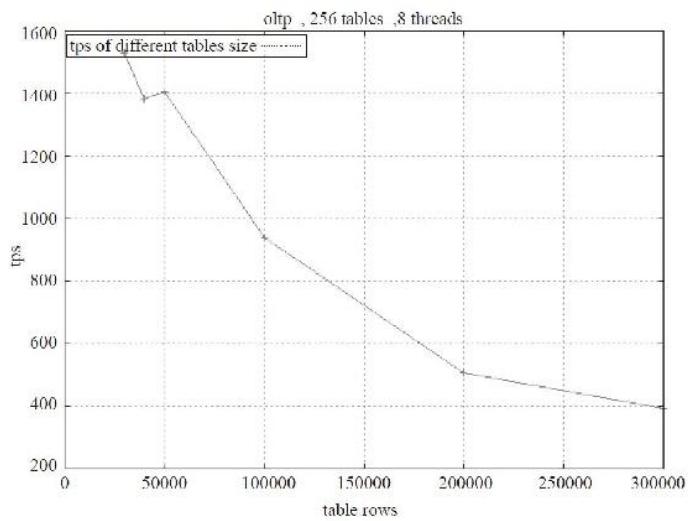


图9-3 不同数据量下，事务吞吐率的变化

例5： 图9-5说明了磁盘的I/O瓶颈可能会导致性能急剧变差。

可以看到在数据远大于InnoDB缓冲 ($4.8573 * \text{buffer}$) 的时候，事务吞吐率下降了很多。且随着时间的增长，有时会突然出现性能急剧下降的情况（见图9-5中长的“毛刺”）。这主要是MySQL刷新数据的机制不够完善所导致的。高并发读写的数据库负载很可能会出现此种情况，且MySQL 5.1很难避免出现这种情况。必须承认这是一个固有的缺陷，需要想办法尽可能地避免。MySQL 5.6对此有一定的改善。

4.基准测试的不足及注意事项

·不容易测试系统的其他特性：稳定性、安全性、扩展性、可用性、灾难恢复性等。

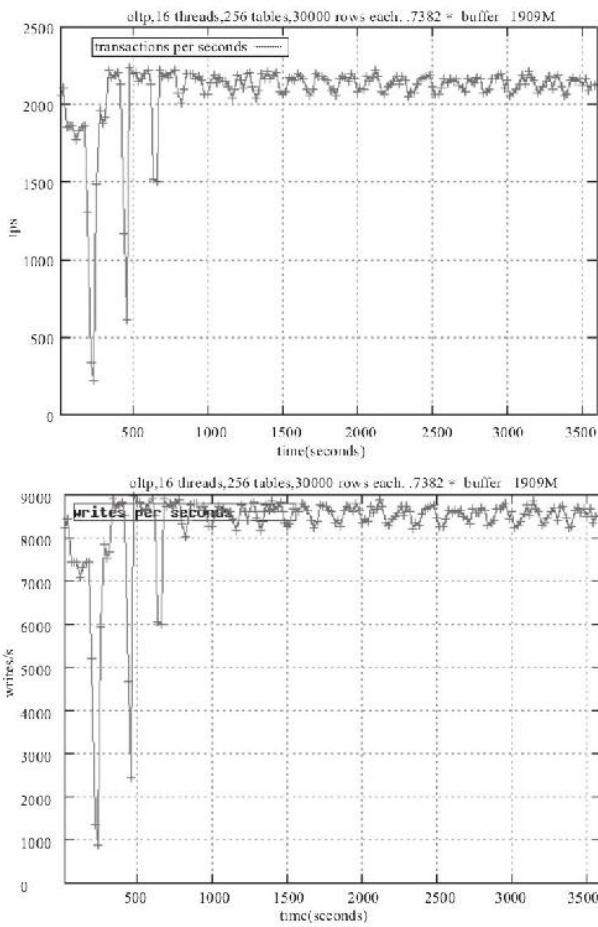


图9-4 数据小于InnoDB缓冲池时，事务吞吐率、读、写、响应时间随时间的变化

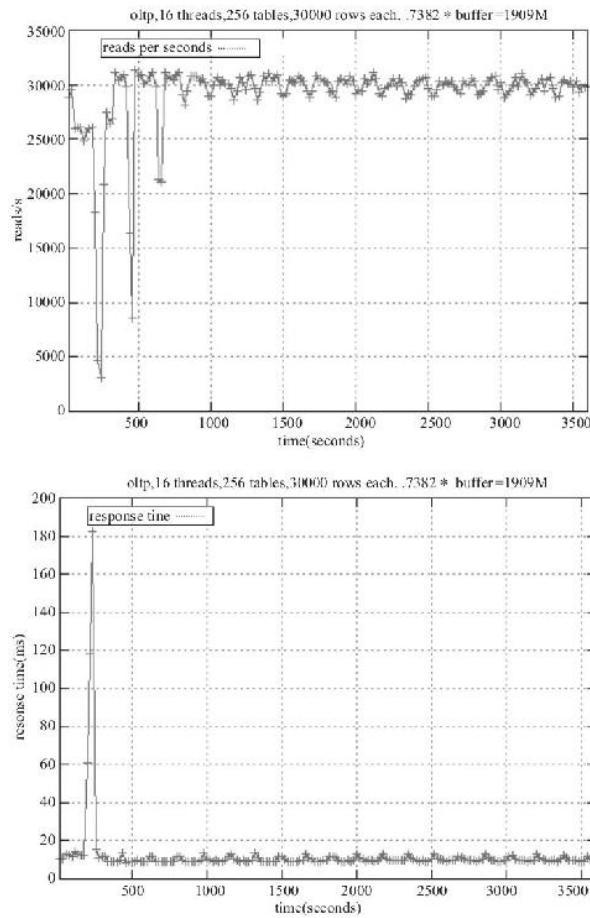


图9-4 （续）

·没有计算成本（磁盘、内存等），但即使计算了成本，也可能被厂商欺骗，厂商会使用最优的配置，以尽可能低的成本支撑更大的吞吐。

·没有衡量能源消耗。

·没有考虑到复杂的网络环境。

·用户考虑的是这个产品能够满足何种服务品质协议（service-level agreement），比如说99.99%的时间是可用的，而基准测试更多考虑的是能够达到的平均分数。一般测试报告列出的可能是80%~90%的系统资源使用率下的数据，没有考虑系统资源严重瓶颈，而现实中往往并非如此，在系统资源出现严重瓶颈时，性能、安全性、稳定性可能急剧下降。

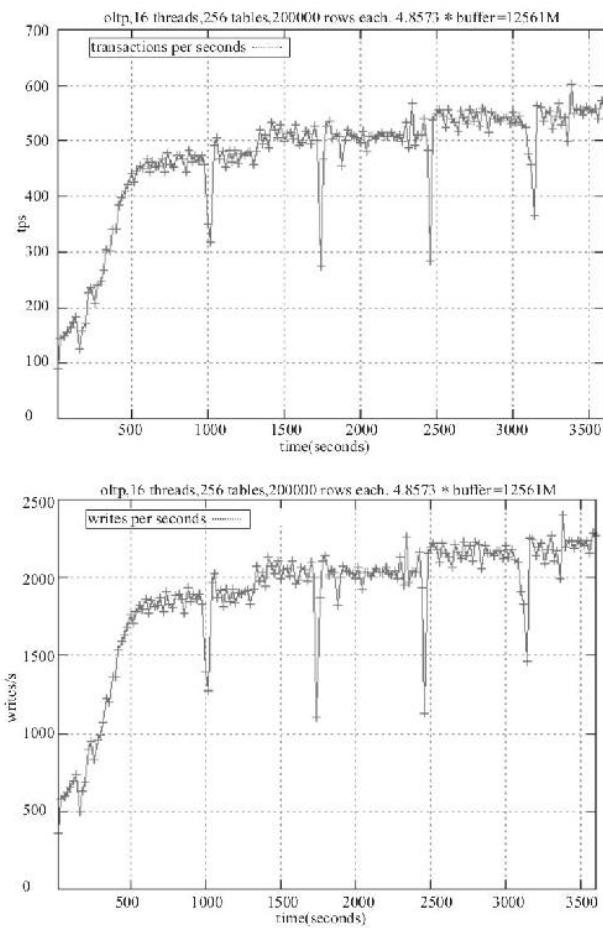


图9-5 数据远大于InnoDB缓冲时，事务吞吐率、读、写、响应时间随时间的变化

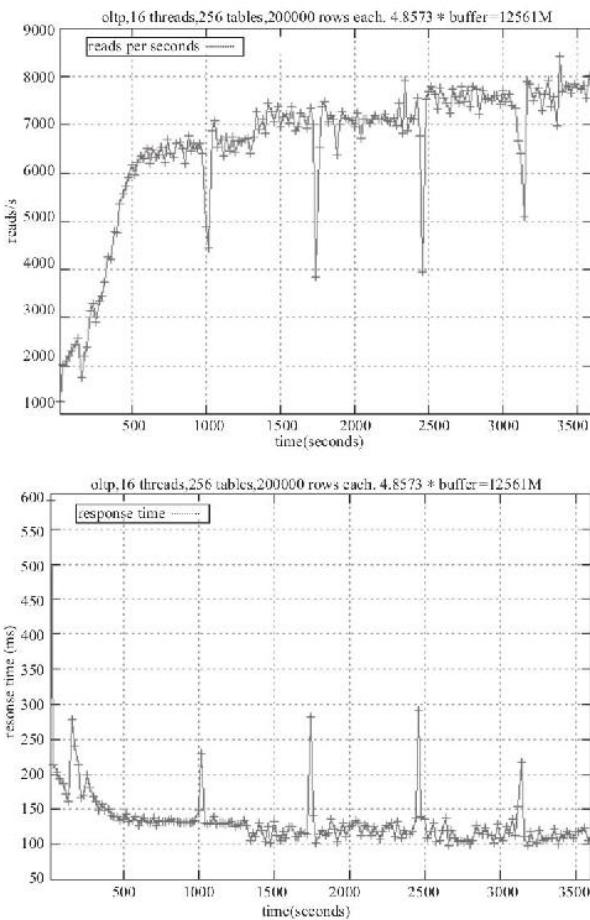


图9-5 (续)

·一般来说，新版本的功能更强，优化器更复杂，更擅长处理复杂的查询。在高并发和大数据集下会更具备优势。

·可能很难模拟复杂的实际访问。

·针对不同问题设计负载，比如，是模拟计算密集型（CPU-bound）的负载还是I/O密集型（I/O-bound）的负载呢？

·向上扩展（scale-up）可以提高吞吐，但存在一个最佳配置，超过这个配置后，吞吐不会再有明显增加。即使增加了CPU、增大了内存，增加了更快速的硬盘，往往还是得不到更高的性能，瓶颈点在于数据库自身的等待而不是硬件，MySQL数据库难以充分利用硬件资源，所以生产环境更倾向于多实例部署，倾向于不那么强劲的服务器主机。

·研发、测试人员往往不擅长做数据库的压力测试，可能存在的问题包括不熟悉硬件，测试时MySQL Server的参数并没有经过优化，没有使用足够多的真实可靠的数据，没有运行足够长时间的测试计划以预热数据等。如果可能，应该提供生产环境的真实数据供研发、测试人员测试，可以考虑的一个方法是重放日志来模拟负载，但必须也要模拟多线程并发的场景。

·生产环境数据库和Web服务器一般分布在不同的主机上，严格来说，测试工具部署在非数据库机器上更好，但是，测试工具、客户端和数据库在同一台机器上做测试一般也是可行的。

9.3 应用数据库性能测试

数据库的性能测试可能是由其他团队来实施，而不是由DBA来完成的，由于关注的重心不一样，使用的工具不一样，可能分析的结果也会有偏差，但只要我们了解了测试的本质，熟悉了数据库，就可以阅读其他部门和团队所做的数据库测试报告，从而得出自己认可的结果。

一般研发、测试人员可能会专门使用一些Web测试工具，如Apache自带的工具ab，包括http_load、Webbench、Siege、JMeter等。这些工具会调用一些Web页面程序，对数据库间接进行施压，如果选择的测试模型和步骤合适，往往比直接使用工具对数据库进行压力测试更合理。

应用负载往往是复杂的，很难去模拟，即使是企业级的测试工具也难以重现生产环境，有兴趣的读者可以去试用一下tcpdump这个工具，它可以把生产环境的流量复制到测试环境中，对于测试数据库的性能很有帮助，尤其是测试只读的MySQL从库。



小结 本章介绍了MySQL的一个基准测试模型，比较简单，大家可以在上面加入自己的想法。随着读者MySQL水平的提高，会逐渐意识到性能测试的重要性。如果需要提高自身的软硬件架构功力，就一定要多做性能测试，通过性能测试，可以对数据架构有更深刻的理解。由于数据库性能测试很耗时、繁琐，因此尽可能使用自动化测试。数据库性能测试的方法和结果也应该分享给其他团队，让他们把握当下软硬件的能力和极限，减少沟通成本，避免错误决策。

第四部分 运维篇

首先来了解一下数据库的定义，数据库是高效的、可靠的、易用的、安全的多用户存储引擎，我们可以通过它访问大量的持久化数据。我们管理和维护数据库，本质上也是要确保如上的特性，尽可能地保证数据库的高效、可靠、易用、安全、高并发和高吞吐。

比如，对于安全，我们要尽量避免因各种软件、硬件、操作错误而导致的数据丢失或损毁。对于高并发，也要求我们在访问控制、并发控制上做适当的设置和调优。数据库系统也应该是易用的，应尽可能地做到对应用程序透明，研发人员不用去关心具体的物理存储对于应用程序的影响。数据存储在磁盘上的方式和布局应与程序认为的逻辑结构无关。数据库系统应该是高效的，比如能够处理高并发的请求，能够处理复杂的查询，或者能够计算大量的数据。MySQL处理复杂查询的能力目前还不太好，对大数据的分析处理也不是强项，但对于互联网的OLTP应用，如果设置、调优得当，得到较高的吞吐率其实并不是一件难事。此外，数据库也应该是可靠的、高可用的，数据库运维很重要的一个指标就是服务的可用性，如果不能提供持续稳定的服务，那么其他指标再好也没有用。

运维篇将首先介绍数据库运维的一些基础知识，接着再介绍各种维护任务所需要的知识和技能，如监控、复制、升级、迁移、备份和恢复。然后通过一些案例给读者讲述一些维护技巧及如何处理问题。数据库运维从来都不仅仅是一个技术问题，本篇最后将讲述规模化运维管理的一些原则、经验总结和认知。

第10章 基础知识

笔者在此假设本书的读者是熟悉Unix或Linux操作系统的，至少会进行一般的操作，这本书不会讲述操作系统的学习和脚本语言的撰写，如果你是一个初学者，那么建议先阅读一些入门图书，比如《Unix&Linux大学教程》、《鸟哥的Linux私房菜》、《Linux命令行与Shell脚本编程大全》等。读者还需要搭建自己的学习和测试环境，配备了基础的学习环境并懂得构建自己的测试环境后，才可以通过实践不断拓宽、深化自己的知识体系。如果你现在仍然没有一个适宜的学习环境，那么建议你尽快搭建一套LAMP或LNMP环境。基础环境的部署和使用将有助于你快速熟悉操作系统和数据库。

本章将主要讲述和MySQL相关的一些基础知识。包括与MySQL相关的数据库文件及参数设置，最后也会简要介绍下MySQL的灾难恢复过程。

10.1 文件和I/O管理

10.1.1 MySQL日志文件

如表10-1所示，MySQL有几个不同的日志文件，可以帮助你了解mysqld（MySQL Server的主程序）内部发生的事情。

表10-1 MySQL的日志文件及功能

日志文件	记录文件中的信息类型
错误日志	记录启动、运行或停止mysqld时出现的问题
通用日志	记录建立的客户端连接和执行的语句
二进制日志	记录更改数据的所有语句，还用于复制
慢查询日志	记录执行时间超过long_query_time秒的所有查询

默认情况下，所有日志均创建于mysqld数据目录中。通过刷新日志，可以强制mysqld关闭和重新打开日志文件（或者在某些情况下切换到一个新的日志中）。当你执行一个flush logs语句或执行mysqladmin flush-logs或mysqladmin refresh时，会使得日志刷新。下面将分别叙述各种日志文件。

1.错误日志

错误日志文件包含了mysqld启动或停止时，以及服务器在运行过程中发生任何严重错误时的相关信息。可以用--log-error=file_name选项来指定mysqld保存错误日志文件的位置。如果没有给定file_name值，mysqld将使用错误日志名host_name.err，并在数据目录中写入日志文件。

需要留意的是，对于MySQL 5.1，当我们使用flush logs命令刷新日志时，错误日志会被清空，并生成一个备份的错误日志，这种情况下，往往只能看到最近的错误日志，这可能会致使我们不能及时发现问题。

我们可以使用工具实时监控错误日志，比如swatch，或者自己编写脚本检查错误日志。也可以发送MySQL错误日志到系统日志服务Syslog，这样，我们就可以利用一些日志分析工具集中分析和处理错误信息。

2.通用日志

如果想要知道mysqld内部发生了什么，你应该用--log=file_name或--log-bin=file_name选项启动它。如果没有给定file_name的值，那么默认名就是host_name.log。所有连接和语句都将被记录到日志文件中。如果怀疑在客户端发生了错误并且想要确切地知道该客户端发送给mysqld的语句，那么该日志可能会非常有用。生产环境中，下线一个业务的时候，也可以打开这个日志，检查是否仍然有流量过来访问。

mysqld按照它接收的顺序将语句记录到查询日志。这个顺序可能与执行的顺序不同。

如下命令可查询通用日志的路径。

```
mysql> show variables like '%gene%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| general_log | OFF   |
| general_log_file | /path/to/loggeneral.log |
+-----+
2 rows in set (0.00 sec)
```

可以使用命令SET GLOBAL general_log ON打开通用日志记录。由于通用日志记录了所有的查询，所以一定要记得关闭它，否则，在一个生产繁忙的系统中，通用日志在几小时之内可能就会塞满磁盘。

3.二进制日志

二进制日志包含了所有更新了数据或已经潜在更新了数据的语句。语句以“事件”(event)的形式保存，它描述了数据的更改信息。二进制日志还包含了每个更新数据库的语句的执行时间信息，但它不包含没有修改任何数据的语句。如果想要记录所有的语句（例如，为了识别有问题的查询），那么我们应该使用通用日志。二进制日志的主要目的是恢复数据，因为二进制日志包含备份后进行的所有更新。

二进制日志还用于在主复制服务器上记录所有将要发送给从服务器的语句。

如果未给出二进制日志的文件名，那么默认名为主机名-bin。如果给出了文件名，但没有包含路径，那么文件将被写入数据目录。建议最好指定一个文件名，语句如下。

```
log-bin =/path/to/logmysql-bin
```

mysqld将在每个二进制日志名的后面添加一个数字扩展名。每次要启动服务器或刷新日志时，该数字将会增加。如果当前的日志大小达到了max_binlog_size参数设置的值，那么mysqld会自动创建新的二进制日志。

mysqld还将创建一个二进制日志索引文件，其中包含了所有使用二进制日志文件的文件名。默认情况下该索引文件与二进制日志文件的文件名相同，扩展名为“.index”。当mysqld正在运行时，不可手动编辑该文件，这样做可能会使mysqld发生异常。

我们可以使用mysql连接数据库，运行SHOW BINARY LOGS命令查看当前有哪些二进制文件，还可以用RESET MASTER语句删除所有的二进制日志文件，或者用PURGE BINARY LOGS命令只删除部分二进制文件。如下的例子将删除历史二进制日志，一直到mysql-bin.000005这个文件为止。

```
mysql> purge binary logs to mysql-bin.000005;
;
```

具有SUPER权限的客户端可以通过SET sql_log_bin=0语句禁止将自己的语句记入二进制记录中。这在某些情况下很有用，比如进行数据库的主主切换时，再或者进行数据库的版本升级时。

我们可以用mysqlbinlog工具检查二进制日志文件。如果想要重新处理日志上的语句，那么这个工具将会很有用。例如，可以用二进制日志更新MySQL数据库，方法如下。

```
shell> mysqlbinlog log-file | mysql -h host -P port
```

默认情况下，并不是每次写入时都会将二进制日志与硬盘同步。因此如果操作系统或机器（不仅仅是MySQL服务器）发生崩溃，那么二进制日志中最后的语句有可能就会丢失。要想防止这种情况的发生，可以设置sync_binlog全局变量为N（1是最安全的值，但也是最慢的），使二进制日志在每N次二进制日志写入后就与硬盘同步一次。

下面来简单介绍下二进制日志的格式。

MySQL有两种记录命令的形式，一种是语句级(binlog_format=statement)，一种是行级(binlog_format=row)。建议将记录命令的形式设置为混合模式(binlog_format=mixed)，这在大部分情况下是适用的，它在一般情况下将使用语句记录日志，但在一些特殊情况下，就会临时更改为行级记录的形式，以便得到更健壮的复制特性。

(1) 语句级(statement-based)

基于语句级的日志记录里包含了原始执行的SQL语句（这会让DBA的维护更方便），还有其他信息，如执行语句的线程ID，语句执行时的时间戳，执行所耗时长等。

(2) 行级(row-based)

如果是行级格式的日志，那么它所记录的事件信息包含了行的更改信息而不是原始的SQL语句，这样可能会让DBA觉得不方便。通过mysqlbinlog默认看到的都是一些经过base-64编码的信息，mysqlbinlog加参数-verbose（或-v），将会生成带注释的语句，如果连续两次使用这个参数（如-v-v），则会生成字段的类型、长度、是否为NULL等属性信息。

一般而言，行级日志更健壮，而语句级的日志如果应用了MySQL的一些额外特性，比如存储过程、触发器，则可能会导致复制异常。所以，如果使用的是语句级的复制，那么请务必保持数据库应用的简单性，只用到基本的核心特性即可。

以下将简单介绍下mysqlbinlog解析出来的二进制日志，主要有如下几项。

#at 141: 事件的起始点。

#1003099:28:36 server id 123 end_log_pos 245: 语句执行的时间，对于复制，这个时间会传输到从库。server id是产生这个事件的MySQL实例的server id参数值。
end_log_pos指下一个事件的开始点，其实也就是这个事件的终点+1。

Query thread_id=3350 exec_time=11 error_code=0: thread_id指执行这个SQL的线程id。exec_time在主从库中有不同的含义，在主库中，等于执行这个事件所花费的时间；在从库中，等于这个事件结束执行的时间点减去在主库上开始执行的时间点，这个差异可以表征主从之间的滞后程度。error_code为错误状态，等于0时表示状态正常。

4. 慢查询日志

当参数slow_query_log=1时，mysqld将记录一个执行时间超过long_query_time秒的所有SQL语句的日志文件。

如果没有给出慢查询文件名，则默认为主机名，后缀为“-slow.log”。如果给出了文件名，但不是绝对路径名时，文件将会写入数据目录。

执行完语句并且释放完所有锁后即可记入慢查询日志。记录顺序与执行顺序可以不相同。

慢查询日志可以用来找到执行时间很长的查询，可以用于优化。但是，检查又长又慢的查询日志会很困难。要想让检查变得容易些，可以使用mysqldumpslow命令或pt-query-digest获得日志中显示的查询摘要来处理慢查询日志。慢查询日志的详细介绍和相关命令的使用请参考4.3节。

5. 日志文件维护

MySQL服务器可以创建各种不同的日志文件，从而可以很容易地查看所进行的操作。但是，必须要定期清理这些文件，以确保日志文件不会占用太多的硬盘空间。至于错误日志文件，一般情况下不会变得很大；慢查询日志在慢查询很多的情况下可能会变得很大，这时可能需要手动处理或编写脚本进行处理；对于二进制日志文件，可以设置合适的过期策略，如expire_logs_days=10，该语句的意思是设置过期日期为10天。expire_logs_days设置会在运行flush logs命令后触发删除过期的日志，注意，不要用操作系统下的rm命令删除日志，这可能会导致你执行日志清理的命令失败，你可能需要手动编辑文件hostname-bin.index来反映实际的文件列表。虽然MySQL 5.1可以设置日志过期策略，但仍然存在一个可能，对于生产繁忙的系统，二进制日志可能会塞满磁盘，MySQL 5.6可以设置保留的二进制日志文件大小，以免磁盘空间过满，这在一定程度上改善了日志的保留策略。

10.1.2 InnoDB数据文件和日志文件

1. 概述

先来简单看下数据库数据目录下的一些文件。假设数据目录为/usr/lib/mysql/data，此目录下可能有如下这些文件。

(1) db.opt

数据库的结构定义和设置。

(2) *.frm

数据表的结构定义。

(3) *.MYD

MyISAM表数据。

(4) *.MYI

MyISAM索引数据。

(5) ibdata*

InnoDB表空间数据文件。

如果将innodb_file_per_table设置为1，那么InnoDB数据表可以各自存储为一个文件，称为独立表空间。如果innodb_file_per_table等于0，那么InnoDB数据表则可以统一存放在一个共享表空间里。默认innodb_file_per_table等于0，即InnoDB将使用共享表空间的方式，所有的数据都会存储在类似ibdata*这样的文件内。

(6) `ib_logfile*`

InnoDB日志数据。

(7) `*.idb`

InnoDB数据和索引（当将`innodb_file_per_table`设置为1，即为独立表空间的方式）。

(8) `*.trg`

触发器。

以下将主要讨论InnoDB表空间数据文件和它的日志文件。

如果你指定了无InnoDB配置选项，那么MySQL将在MySQL数据目录下创建一个名为`ibdata1`的10MB大小的自动扩展数据文件，以及两个名为`ib_logfile0`和`ib_logfile1`的5MB大小的日志文件。对于一般的生产负荷来说，这种配置太小了，可能会导致性能问题，所以需要手动设置大小。笔者建议日志文件应大于256MB，数据文件初始可以分配1GB到5GB，并设置为自动扩展，这样的配置在一般情况下已经够用了，相关的配置项设置如下。

```
innodb_data_file_path = ibdata1:1000M:autoextend  
innodb_log_file_size = 256M
```

`innodb_data_file_path`的值应该为一个或多个数据文件规格的列表。如果要命名一个以上的数据文件，请用分号“;”分隔它们。其语法格式为：

`innodb_data_file_path=datafile_spec1[;datafile_spec2]...`

例如：

```
innodb_data_file_path=ibdata1:5000M;ibdata2:5000M:autoextend
```

其中，`autoextend`属性和后面跟着的属性只能被用于`innodb_data_file_path`行里的最后一个数据文件。

如果对最后的数据文件指定`autoextend`选项，那么当数据文件耗尽表空间中的自由空间时，InnoDB就会扩展这个数据文件，扩展的幅度默认是每次8MB。

2. 独立表空间的原理和设置

共享表空间的使用很简单，维护方便，同时它也是MySQL默认的配置，所以在生产中得到了广泛的应用，但它也存在一些劣势，使用共享表空间比较明显的缺点是，不能快速回收删除大表的空间，I/O操作可能会消耗更多的资源等待。而独立表空间是很多DBA推荐使用的方式，它刚好在这两点上弥补了共享表空间的不足。使用独立表空间，可以在它自己的文件中存储每个InnoDB表和它的索引，这种情况下，每个表都有它自己的表空间。

可以向`my.cnf`的`[mysqld]`节中添加下面的语句来允许使用独立表空间，重启MySQL实例（MySQL Server）即可生效。

```
[mysqld]  
innodb_file_per_table
```

重启实例之后，InnoDB将会把每个新创建的表存储到数据库目录下的文件`tbl_name.ibd`中。这类似于MyISAM存储引擎所做的，但MyISAM是把表分成数据文件`tbl_name.MYD`和索引文件`tbl_name.MYI`。对于InnoDB，数据和索引则会被一起存放到`.ibd`文件中。不过`tbl_name.frm`文件照旧会被创建。

如果从`my.cnf`文件里删除了`innodb_file_per_table`行，并重启了实例，那么InnoDB将会在共享的表空间文件里再次创建表。也就是说，`innodb_file_per_table`只会影响表的创建。如果用这个选项启动实例，那么新表将会被`.ibd`文件创建，但是你仍然能够访问共享表空间中的表。如果删掉了这个选项，那么新表将在共享表空间内被创建，但是你仍然可以访问用独立表空间创建的任何表。

即使使用了独立表空间，也仍然有一部分共享数据需要存放在共享表空间内，所以`idata*`文件仍然存在。

你不能像对待MyISAM一样，在数据目录之间随意地移动`.ibd`文件。这是因为表定义是被存放在InnoDB共享表空间内的，而且InnoDB必须保持事务ID和事务日志顺序号的一致性。

如果某个数据文件变得很大，比如上百GB，这时你可能想要另外增加一个数据文件；或者磁盘已满，这时你想要把其他数据添加到另一个硬盘上，那么这时可以手动添加一个数据文件。

3. InnoDB增加数据文件

手动增加一个数据文件时需要重启MySQL实例，我们可以计算出最后一个文件的大小（针对按MB计算的大小取整，即字节数除以10242，再四舍五入），然后修改配置文件，把`innodb_data_file_path`参数指定的最后一个文件大小设置为该值，并在其后继续追加新的数据文件。

解决方案具体如下。

当你要添加一个新文件名到`innodb_data_file_path`参数指定的文件名列表时，请确保它并不存在。当你重启实例时，InnoDB会创建并初始化这个文件。

如果最后一个数据文件是用关键字`autoextend`定义的，那么在编辑`my.cnf`文件时必须考虑最后一个数据文件已经增长到多大了。你需要获取这个数据文件的大小，四舍五入，使其最接近 $1024*1024$ bytes的乘积（即1MB），然后在`innodb_data_file_path`中明确指定大致的尺寸。然后添加另一个数据文件。记住，只有`innodb_data_file_path`里的最后一个数据文件才可以被指定为自动扩展。

如下是一个修改数据文件大小的示例。

首先关闭实例，查看最后一个数据文件的大小。如下是Linux操作系统命令的输出。

```
-rw-rw--- 1 mysql mysql 10829692928 Mar 10 10:27 ibdata4
```

然后计算最后一个数据文件的大小。

$10829692928/1024/1024=10328$ MB（四舍五入）

那么对原配置文件：

```
innodb_data_file_path = ibdata1:4000M;ibdata2:4000M;ibdata3:4000M;ibdata4:4000M:autoextend
```

做如下修改，增加一个数据文件`ibdata5`，初始值为8000MB，可自动扩展。

```
innodb_data_file_path = ibdata1:4000M;ibdata2:4000M;ibdata3:4000M;ibdata4:10328M;ibdata5:8000M:autoextend
```

最后，重新启动实例，MySQL Server会自动创建`ibdata5`。

4.改变InnoDB事务日志大小

不要试图通过直接更改配置文件来设置InnoDB事务日志的大小，这会导致不能启动数据库。如果想要改变InnoDB事务日志文件的数量和大小，那么必须要停止MySQL实例，并确定它被无错误地关闭了。随后复制旧日志文件到一个安全的地方作为备份，万一出错还可以恢复，然后从日志文件目录删除所有的旧日志文件，之后编辑`my.cnf`改变日志文件配置，并再次启动MySQL实例。`mysqld`在启动之时会发现没有日志文件，然后告诉你它正在创建一个新的日志文件。

更改InnoDB事务日志大小的具体步骤如下。

- 1) 干净关闭MySQL。
- 2) 使用`mv`命令移走旧的InnoDB事务日志。
- 3) 修改配置文件，更改`innodb_log_file_size`。
- 4) 启动MySQL。

注意，在旧版本的MySQL中，所有事务日志大小的总和不能超过4GB。MySQL 5.6将总大小的限制扩展到了512GB。

5.InnoDB的undo区域

`undo`区域也称为`undo`空间、`undo`表空间，是InnoDB设计的一个特殊存储区域，它保存了被活动事务更改的数据的副本（前像），如果另一个事务需要查看原来的数据（例如，满足一致性读），那么可以从`undo`区域中获得未被更改的数据。默认情况下，`undo`区域也是在InnoDB共享表空间内。MySQL的更高版本（MySQL 5.6及以上）也提供了该选项，可以把`undo`空间放到独立的表空间里，这样就可以把`undo`表空间放到其他更快的磁盘设备上，进行专门的优化。

如果`undo`暴涨可能会把共享表空间撑大。出现这种情况，可能是因为写负载很大，比如执行了大量的删除和修改操作，但在生产环境中，更可能出现的一种情况是存在长时间未提交的事务。

如果一个事务长时间未提交，而我们默认使用的是`repeatable read`事务隔离级别，那么InnoDB不会去清理旧的行版本（old row versions），因为未提交的事务仍然需要看到它。当这个事务一直保持打开而不提交，就可能会导致大量旧的版本数据无法删除，从而导致`undo`暴涨。将事务的隔离级别更改为`read committed`可以解决此问题。但根本的处理措施还是检查代码，找到未提交的事务。

通过命令`SHOW INNODB STATUS`的输出，可以看到当前有多少没有被清理的记录。对比下面的`Purge done for trx`和`Trx id counter`，如果差异很大，则可能是因为大量事务所导致，也可能是操作大量数据的个别事务所导致的。

```
-----  
TRANSACTIONS  
-----  
Trx id counter 0 80157601  
Purge done for trx: n:o:<0 80154573 undo n:o:<0 0
```

对于写操作很频繁的应用，InnoDB清理线程的速度可能会跟不上，从而导致undo表空间越来越大，可以通过设置`innodb_max_purge_lag`参数，来避免InnoDB表空间的过分增大。InnoDB事务系统维持了一个事务列表，该列表记录被UPDATE或DELETE操作标志为删除的索引记录。这个列表的长度为`purge_lag`。当`purge_lag`超过`innodb_max_purge_lag`之时，每个INSERT、UPDATE和DELETE操作都将被延迟一定的时间，比如我们可以将其设置为100万。即允许有100万条未清理的记录，在达到100万的阈值后，就会触发延迟其他的查询操作。

简而言之，undo里保存了数据的前像，它可以满足一致性查询，同时，在灾难恢复过程中，它也扮演了重要的角色，它的主要功能是在灾难恢复过程中回滚那些没有提交的变更。灾难恢复的具体过程请参考10.2节。

10.1.3 临时文件

MySQL使用环境变量TMPDIR的值作为保存临时文件的目录路径名。如果未设置TMPDIR，那么MySQL将使用系统的默认值，通常为/tmp、/var/tmp或/usr/tmp。如果包含临时文件目录的文件系统过小，则可以对mysqld使用“--tmpdir”选项，在具有足够空间的文件系统内指定1个目录，或者修改配置文件内的参数tmpdir。

在MySQL 5.1中，“--tmpdir”选项可被设置为多个路径的列表，以循环的方式使用。在Unix平台上，路径可用冒号字符“:”隔开，在Windows、NetWare和OS/2平台上，路径可用分号字符“;”隔开。注意，为了有效地分布负载，这些路径应位于不同的物理磁盘上，而不是位于相同磁盘的不同分区中。

如果MySQL服务器正作为复制从服务器使用，那么不应将“--tmpdir”设置为指向基于内存的文件系统的目录，或者当服务器主机重启时将要清空的目录。对于复制从服务器，需要在机器重启时仍保留一些临时文件，以便能够复制临时表或执行LOAD DATA INFILE操作，如果在服务器重启时丢失了临时文件目录下的文件，那么复制将会失败。

MySQL会以隐含的方式创建所有的临时文件。这样，就能确保在中止mysqld时会删除所有的临时文件。使用隐含文件的缺点在于，在临时文件目录所在的位置中，看不到占用了文件系统的大临时文件。

进行排序时（ORDER BY或GROUP BY），MySQL通常会使用1个或多个临时文件。对于大数据量的排序，临时空间可能会超过/tmp空间，此时，执行查询将会失败，MySQL错误日志里会出现错误记录“sort abort”。解决方案是优化查询或把临时目录设置到另一个空间足够大的分区中。

对于某些SELECT查询，MySQL还会创建临时SQL表，它们有sql_*形式的名称。ALTER TABLE会在与原始表目录相同的目录下创建临时表。

10.1.4 MySQL套接字文件

服务器用来与本地客户端进行通信的Linux套接字文件（也称为socket文件），其默认位置是/tmp/mysql.sock。此文件位于/tmp目录下可能会导致一些问题，原因在于，在某些版本的Linux上，任何人都能删除/tmp目录下的文件。在Linux系统下，系统会自动删除/tmp目录下的一些文件，但并不会删除socket文件。但某些没有经验的系统管理员可能配置了定时任务去删除/tmp目录下的文件，很可能连socket文件也会被删除，这将导致MySQL无法通过socket文件的方式进行登录。由于现在的服务器一般都很强劲，多实例的配置也很普遍，建议不要将socket文件集中放在/tmp目录下，最好是放在单独的实例自身的目录中。我们可以在全局配置文件中指定socket文件路径。例如，将下述行置于文件/etc/my.cnf中。

```
[mysqld]
socket=/path/to/socket
[client]
socket=/path/to/socket
```

如果你不放心socket文件，那么可以保留默认的root的其他登录方式，默认的root账号可以通过socket文件或127.0.0.1进行登录。建议保留127.0.0.1的root登录账号，以防socket文件被异常清除。

10.2 MySQL如何进行灾难恢复

MySQL的灾难恢复类似于其他传统数据库的灾难恢复。

MySQL靠预写式日志（Write-Ahead Logging，WAL）来保证持久性，也就是说，数据文件不会马上写入脏数据，而是会先写日志。InnoDB的脏数据是存在于`innodb_buffer_pool`里的，它会按一定的机制批量刷新到磁盘，这样做可以提高吞吐率。

我们把上面这种日志称为redo日志，即InnoDB的事务日志。如果突然断电了，那么InnoDB是不能保证数据已经写入磁盘的，数据库重启后，MySQL需要知道当时执行的操作是成功了还是部分成功或失败了这时，只要使用了预写式日志，程序就可以检查redo日志，并将突然断电时计划执行的操作内容跟实际上执行的操作内容进行比较。在这个比较的基础上，MySQL就可以决定是撤销已做的操作还是继续完成相应的操作，或者是保持原样。这就是灾难恢复的过程。

由于MySQL知道宕机时有哪些日志是还没有被实际写入到数据文件的，所以它会找到事务日志的某个点，把这个点之后的日志运行一遍，这个时候就会产生一个新的问题，虽然把所有日志都执行了一遍，但有一些更改并没有被提交，需要回滚。我们配合undo日志（在undo区域内）可以确定哪些变更需要回滚的，然后回滚那些没有提交的日志，简单地说，灾难恢复过程可以分为redo（重做）和undo（回退）两个步骤。

由上可知，InnoDB事务日志在很大程度上决定了数据的安全性，事务日志的持久性决定了灾难恢复后最多丢失了多少记录？事务日志都是顺序写入的，因此可以设置参数来调整commit（事务提交）时写入事务日志的频率。MySQL的事务日志刷新可能会出现如下3种情况。

(1) innodb_flush_log_at_trx=1

每次commit时都写入磁盘。这样理论上我们只会丢失一个事务。

(2) innodb_flush_log_at_trx=2

每次commit时，写日志只缓冲（buffer）到操作系统缓存，但不刷新到磁盘，InnoDB会每秒刷新一次日志，所以宕机丢失的是最近1秒的事务。生产环境中建议使用此配置。

(3) innodb_flush_log_at_trx=0

每秒把日志缓冲区的内容写到日志文件，并且刷新到磁盘，但commit时什么也不做。

数据文件的写操作，可能会将块写坏，MySQL设计了一个数据存储区域双写缓冲（double write buffer），InnoDB使用双写缓冲来确保数据的安全，避免损坏块。双写缓冲是InnoDB表空间的一个特殊的区域，主要用于写入页的备份，并且是顺序写入。当InnoDB刷新数据（从InnoDB缓冲池到磁盘）时，首先写入双写缓冲，然后写入实际数据文件。这样即可确保所有写操作的原子性和持久性。

崩溃重启后，InnoDB会检查每个块（page）的校验和，判断块是否损坏，如果写入双写缓冲的是坏块，那么显然没有写入实际数据文件，就要用实际数据文件的块来恢复双写缓冲，如果写入了双写缓冲，但是数据文件写的是坏块，那么就用双写缓冲的块来重写数据文件。这样的机制虽然提供了安全保障，但也增加了I/O。

对于读操作，InnoDB通过页校验码来保证数据的存取，每页在内存中都先算好一个校验值，放在文件头部，写入的时候先写校验值，读的时候也会校验一下校验值。

通过如上描述的预写式日志机制和双写缓冲区域，MySQL提供了极佳的灾难恢复性。MySQL的稳定版本很少会因为主机断电等硬件故障而导致数据损坏。

10.3 变量设置、配置文件和主要参数

10.3.1 概述

很多人都喜欢研究各种参数配置文件，然后给自己的生产环境加上很多参数。笔者的建议是，可以去研究它，测试它，但是在生产环境中，你应该在确定某个选项能解决特定的性能问题时，才去设置它，否则你应该尽量保持简单。配置文件添加了过多的参数可能会导致混淆，维护性可能会变差，后来接手的DBA往往会上问，为什么要这么设置。实际的数据库产品中，很多参数只有在特定的上下文中才有意义，时过境迁，一些参数可能反而会成为性能问题的根源所在。所以建议让生产环境的配置文件尽可能地保持简单，在确定需要时，才去设置相应的参数。

另外，数据库配置文件所起的作用有限。系统的性能更多地取决于物理部署和架构，取决于数据库设计、索引和SQL质量等。设置好正确的基本参数之后，最好就不用再去关注它，应该花费更多的时间在库表设计、索引和查询优化上。

官方的安装包内有附带的示例配置文件，但不建议使用。里面的一些设置不太符合生产实践，可能会有误导，而且这些配置也过时了，不适合现在的硬件和负载，也不适合互联网公司流量比较大的业务。

本章稍后会给出一份比较简单的配置文件，大家可以去对比下，然后检验下你的生产环境设置得是否合理。注意，适合生产环境的才是最佳的，而任何建议的参考配置文件，往往是不可能覆盖到各种应用类型的，仅仅是为你的决策提供一个参照物。所以，仍然建议以自己的生产配置为准。

10.3.2 如何设置参数、变量

配置文件内的参数需要尽量保持一样的书写风格，要么都是用下划线（如slow_query_log_file）要么都使用中线（slow-query-log-file）。

配置文件内的参数有些是影响全局的，有些是会话（session）级别的，即我们也可以在独立的连接内进行设置。

sort_buffer_size可用于设置全局和会话级，如下：

```
SET sort_buffer_size = <value>; #设置会话级。  
SET GLOBAL sort_buffer_size = <value>; #设置全局。  
set sort_buffer_size =default; #恢复默认值。
```

生产中尽量不要使用32位系统，32位系统的机器有内存寻址的限制，不能突破二点几GB的限制。如果一定要使用，那么配置参数的时候，注意不要设置得过高，内存参数如果设置得太高，可能会导致32位的MySQL实例崩溃。

我们可以在SET命令中使用表达式，即，SET sort_buffer_size=10*1024*1024，但配置文件不允许使用表达式。

有时我们需要临时设置会话变量，执行操作，然后恢复原来的设置，通行的办法如下所示。

```
SET @saved<unique_variable_name> := @@session.sort_buffer_size;
SET @@session.sort_buffer_size := <value>;
-- Execute the query...
SET @@session.sort_buffer_size := @saved<unique_variable_name>;
```

有时我们需要临时调整一些参数或变量，来验证自己的一些想法，但在此过程中需要注意以下两点。

- 1) 调整参数需要有一个基准，调整参数后，我们需要衡量调整的结果。最好是有一套监控系统来收集实例的运行状态，这样可以方便我们进行对比。
- 2) 应尽量小步调整参数，一次不要调整太多参数，调整太多参数会比较危险，也会使我们无法明确到底是哪些参数调整后有效果。

随着对生产环境的日渐熟悉，我们总能找到一套适合自己生产环境的配置。

10.3.3 配置文件的读取顺序

在Unix中，MySQL程序从表10-2所示的文件中读取启动选项。

表10-2 读取启动项的文件

文件名	目的
/etc/my.cnf	全局选项
(续)	
文件名	目的
\$MYSQL_HOME/my.cnf	服务器相关选项
defaults-extra-file	用--default-extra-file=path指定的文件。如果有在该文件的话
~/.my.cnf	用户相关选项

其中，\$MYSQL_HOME是一个环境变量，包含与服务器相关的my.cnf文件驻留的目录路径。

如果未设置\$MYSQL_HOME，并且DATADIR中有一个my.cnf文件，而BASEDIR中没有my.cnf文件，那么mysqld_safe将会把\$MYSQL_HOME设置为DATADIR。如果未设置\$MYSQL_HOME并且在DATADIR中没有my.cnf，则mysqld_safe将\$MYSQL_HOME设置为BASEDIR。也就是说，数据目录内的配置文件和安装目录下的配置文件都可能生效。

典型情况下二进制的安装目录为/usr/local/mysql/data，源代码的安装目录为/usr/local/var。请注意这是配置时指定的数据目录的位置，而不是mysqld启动时用--datadir指定的。运行时使用--datadir对寻找选项文件的服务器没有效果，因为服务器在处理命令行参数之前就寻找这些选项了。

MySQL按照上述顺序寻找选项文件，如果存在多个选项文件，那么文件中指定的后读取的选项要优先于文件中指定的先读取的选项。所以理论上在datadir或basedir内放置一个my.cnf即可。

在Unix平台上，MySQL忽略了人人可写的配置文件。这是特意设置的，它其实是一个安全措施。

MySQL默认加载配置文件的先后顺序也可以通过应用如下命令来得知。

```
$ which mysqld
/usr/local/mysql/bin/mysqld
/usr/local/mysql/bin/mysqld --verbose --help | grep -A 1 .
Default options:
Default options are read from the following files in the given order:
/etc/my.cnf /etc/mysql/my.cnf /usr/local/mysql/etc/my.cnf -
./.my.cnf
```

通过以上命令可以知道加载配置文件的顺序。



注意 不要在生产环境中运行，因为会真的启动mysqld程序。

虽然官方文档中说明了配置文件的读取顺序，可是该顺序不一定可靠。建议读者不要依赖于官方文档所说明的顺序来部署自己的多个MySQL配置文件。对于生产环境的部署，建议仅存在并加载一个配置文件，而不要配置多个配置文件。有些人除了配置文件，还喜欢在命令行内也设置一些参数，这样容易导致混淆，维护性也会变差，最终将丢失你所做的变更。

10.3.4 环境变量、配置文件、命令行选项的优先级

MySQL程序首先会检查环境变量，然后检查选项文件，最后再来检查命令行以确定给出了哪些选项。如果多次指定一个选项，那么最后出现的选项占先。这说明环境变量具有最低的优先级，命令行选项具有最高的优先级。

可以在选项文件中指定程序选项的默认值来让MySQL程序处理各个选项。不需要在每次运行程序时都输入选项，但可以根据需要通过命令行选项来覆盖默认值。

10.3.5 配置文件详述

配置文件分成了很多节，MySQL程序通常会读取命名和自己名字一样的节。比如如下的配置文件。

```
[client]
port = 3306
socket = /path/to/tmp//3306/mysql.sock
default-character-set = utf8
```

客户端工具，如mysql、mysqldump会读取client这一节的配置。

default-character-set指程序和MySQL服务器进行通信时所使用的字符集。这个字符集应该和输入窗口（Windows）或控制台窗口（Unix/Linux）里默认使用的字符集一致。

再来看一个配置文件：

```
[mysqld]
character-set-server = utf8
port = 3306
socket = /path/to/tmp//3306/mysql.sock
user = mysql
skip-external-locking
datadir =/path/to/data/3306
log-error =/path/to/log3306/mysqld.err
pid-file = /path/to/tmp//3306/mysql.pid
#init_connect=
set autocommit=0

#init_connect=
set names utf8

#read-only
```

mysqld服务会读取这一节的配置。

init_connect这个参数可以在客户端连接进来的时候执行一些初始化操作，如记录连接IP，但不会对Super用户起作用。

对于my.cnf配置文件，可以添加一些基本设置，如下是一个例子。

```
expire_logs_days=10;
max_connect_errors=5000;
max_connections=2048;
slow_query_log=on;
long_query_time=0.5;
skip_name_resolve
```

下面对其中的参数做一些简单的介绍。

(1) max_connect_errors

将此值设置得足够大会更好，推荐值是5000。如果一台尝试连接数据库的主机失败的次数超过了此阈值，那么这个主机会被MySQL Server阻止访问，必须在MySQL Server上运行FLUSH HOSTS才能解除此限制。

(2) skip_name_resolve

必须设置此项，因MySQL的DNS解析可能会导致严重的性能问题。注意设置了此项之后，MySQL权限表将使用IP来统一标识主机，而不能使用主机名来标识了。

(3) sync_binlog

默认情况下，并不是每次写入时都会将二进制日志与硬盘同步。因此如果操作系统或机器（不仅仅是MySQL实例）发生崩溃，那么有可能二进制日志中最后的语句会丢失。要想防止出现这种情况，可以使用sync_binlog全局变量（1是最安全的值，但也是最慢的），使二进制日志在每N次写入后与硬盘同步一次。

待sync_binlog个记录写入二进制日志后，MySQL服务器会将该二进制日志同步到硬盘上。请注意如果是autocommit模式，那么每执行一个语句便会向二进制日志写入一次，否则每个事务执行完才写入一次。sync_binlog的默认值是0，表示不与硬盘同步。值为1是最安全的选择，因为崩溃时最多丢掉二进制日志中的一个语句/事务；但是，这也是最慢的选择（除非硬盘有电池备份缓存，使同步工作较快）。建议配置范围为8~20。

10.3.6 配置文件示例

最终的一份简单的配置文件示例如下（MySQL 5.1）。

```
[mysqld]
# GENERAL
datadir      = /var/lib/mysql
socket       = /var/lib/mysql/mysql.sock
pid_file    = /var/lib/mysql/mysql.pid
user         = mysql
port         = 3306
```

```
storage_engine = InnoDB
sync_binlog = 20
# INNODB
innodb_buffer_pool_size      = <value>
innodb_log_file_size         = <value>
innodb_file_per_table        = 1
innodb_flush_method          = O_DIRECT
# MyISAM
myisam-recover=default      默认自动修复
key_buffer_size               = <value>
# LOGGING
log_error                     = /var/lib/mysql/mysql-error.log
log_slow_queries              = /var/lib/mysql/mysql-slow.log
long_query_time               = <value>
# OTHER
skip_name_resolve             =
expire_logs_days              = <value>
max_connect_errors            = <value>
tmp_table_size                 = 32M
max_heap_table_size           = 32M
query_cache_type              = 0
query_cache_size               = 0
max_connections                = <value>
max_connections_size          = <value>
thread_cache_size             = <value>
table_cache_size               = 65535
open_files_limit               = 65535
[client]
socket                         = /var/lib/mysql/mysql.sock
port                           = 3306
```

10.4 MySQL Query Cache和优化器

MySQL Query Cache内缓存了我们提交的SQL语句的结果集及相关信息，有助于加速查询响应。一般不需要考虑Query Cache带来的额外开销，除非是写操作很频繁的应用。

工作原理

当MySQL运行查询语句时，首先会检查是否命中缓存，如果命中那么此时会增加Qcache_hits状态变量的值，并返回结果集给客户端。

如果在缓存中找不到此语句的缓存，则进入如下步骤。

1) MySQL解析器将分解查询语句，并建立一棵“解析树”，解析器会使用MySQL的语法解析并验证查询语句的语法是否正确，是否符合规范，当然各种符号也包含在检查范围之内。

2) 预处理器检查“解析树”中的表和列是否存在，列的别名是否混淆，并进行相关权限的检查。

3) 如果前面两步都通过了检验，那么再进行如下步骤。

步骤1：优化器对“解析树”进行优化，生成执行成本最低的执行计划。

步骤2：执行此计划，存储查询结果。

步骤3：返回结果集给客户端。

Query Cache默认是关闭的，临时禁用Query Cache的办法是设置query_cache_size为0，注意FLUSH QUERY CACHE命令并不会清空缓存。清除缓存的命令是RESET QUERY CACHE。

查看相关参数的语句为mysql>show variables like '%query_cache%';

查看相关状态变量的语句为mysql>show global status like '%Qcache%';

至于是否可以禁用Query Cache，对此我们要谨慎些，如果命中率不高，比如才70%~80%，那么关闭Query Cache一般不会有太大的问题，但如果Query Cache有98%~99%，那么关闭Query Cache可能会导致比较大的冲击，要仔细评估因为缓存失效而可能对数据库造成的冲击。

任何不是从缓存块中取得数据的查询语句都称为“缓存错失（cache miss）”，造成缓存错失的原因有以下几种。

1) 所发送的查询语句是不可缓存的，查询语句不可缓存的原因主要有两种：一是语句包含了不确定的值；二是所得到的结果集太大而无法将它保存到缓存中。这两种原因造成的结果都会增加Qcache_not_cached变量的值，可以通过查看这个变量的值来检查查询语句的缓存情况。

2) 所发送的查询语句之前没有发送过，所以也不会有什么缓存存在。

3) 所发送的查询语句的结果集是之前存在于缓存中的，但由于内存不足，MySQL不得不将之前的一些缓存清除掉，以腾出空间来放置其他新的缓存结果。

4) 数据的变更也会引发缓存的失效。如果是数据的变更引起的缓存失效，那么可以通过查看Com_*变量的值来确认有多少查询语句更改了数据，这些变量包括Com_update、Com_delete等。

Query Cache有如下一些要点需要注意。

·SQL语句在Query Cache中是通过散列映射表来查找的，大小写、空格等差异都会导致不同的散列结果，所以开发人员应该有一致的代码规范，以保证SQL语句

风格一致。

·Query Cache不会缓存子查询。

·如果Query Cache结果集中相关的对象发生了变化，那么这个结果集就会被失效。比如某张表修改了数据，那么Query Cache内所有涉及这张表的结果集都会失效。需要注意的是，长时间运行的事务，会降低Query Cache的效率。因为如果InnoDB事务内的一条语句更改了表，那么MySQL会让Query Cache与这个表相关的Cache都失效掉。直到这个事务提交之后，才可以重新缓存这个表的结果集。

·Query Cache分配内存的时候，每次至少要分配query_cache_min_res_unit大小的内存块，Query Cache并不需要等待所有的结果集在Cache内全部生成后才发送给客户端。因为失效等原因，实际上生产环境结果集所需要的Query Cache并不是很大，一般256MB就足够了。

·对于写操作很频繁的应用，可以考虑禁用Query Cache。

·留意碎片（fragmentation）的原因是，如果每次都分配较大的内存（query_cache_min_res_unit较大），那么更容易导致碎片化；如果每次分配较小的内存（query_cache_min_res_unit较小），则需要更频繁的分配，所以需要在内存的浪费和CPU的成本之间做一个取舍。我们可以计算下平均查询大小（Query Size）。公式为：Query Size=(query_cache_size-Qcache_free_memory)/Qcache_queries_in_cache，通过平均查询的大小来大致确定一个合适的query_cache_min_res_unit应该设置为多大。

·如果Qcache_lowmem_prunes比较大，而Qcache_free_blocks也比较大，那么可能是碎片比较严重，导致了查询缓冲被大量剔除。

·我们不太好衡量开启了Query Cache是否真的有帮助。最简单的办法是衡量缓冲命中率，公式为Qcache_hits/(Qcache_hits+Com_select)，如果缓冲命中率比较高，那么它就是有效的。但即使不高（如20%~30%），也不一定意味着低效，我们关注的是提高特定查询的访问速度而不是只关注命中率这个指标相对查询来说，将结果集存储到Query Cache比结果集失效的成本更低。如果一个系统中，大部分都是复杂的查询，那么用Query Cache将是一个很好的选择。

·如果Qcache_not_cached比较小，但有大量缓存未命中，那么可能会有很多失效的操作，或者MySQL没有预热数据，或者重复的查询很少。Qcache_inserts在预热数据后，应该比Com_select小得多。

·可监控一下Qcache_lowmem_prunes，确定是否因为内存不够而剔除了结果集。Query Cache的效率比较高的时候，Qcache_inserts应该比Com_select小得多。

如果查询结果没有被缓存，那么，MySQL将解析查询（Parse），通过优化器（Optimizer）生成执行计划，然后运行执行计划获取数据。MySQL优化器生成的执行计划，在很大程度上决定了其性能，随着新版本的发布，MySQL优化器越来越智能，但它仍然存在很多限制，DBA和研发人员需要熟悉所使用的MySQL版本的优化器规则，充分利用优化器，撰写高质量的SQL。

让优化器工作得更好，本质上就是进行查询优化，具体可参考第6章“查询优化”。

10.5 SHOW INNODB STATUS解析

SHOW ENGINE INNODB STATUS是一种常用的工具，但运行这个命令的输出却不容易阅读。

我们可以通过创建一些InnoDB监控表（注意必须是InnoDB引擎的表），来启用性能监控输出，输出InnoDB的各种信息，默认输出至MySQL错误日志。

如下命令将创建InnoDB标准监视器，即SHOW INNODB STATUS输出。

```
CREATE TABLE innodb_monitor (a INT) ENGINE=InnoDB;
```

如下命令将创建表空间监视器，以输出共享表空间的信息。对独立表空间来说，它不适用，如果关闭了数据文件的自动扩展，那么通过这个监控，可以监视数据文件是否需要扩展。

```
CREATE TABLE innodb_tablespace_monitor (a INT) ENGINE = InnoDB;
```

如下命令将开启表监控器，会输出系统中所有InnoDB表的一些结构和内部信息。

```
CREATE TABLE innodb_table_monitor (a INT) ENGINE = InnoDB;
```

如下命令将开启InnoDB锁监控器，它的输出结果和标准监视器基本类似，但会有更多关于锁的信息。

```
CREATE TABLE innodb_lock_monitor(a INT) ENGINE = InnoDB;
```

创建表只是发出一个命令给InnoDB引擎，同理，删除表也是发送一个停止监控的命令给InnoDB引擎。所以MySQL在重启后是不会自动启动InnoDB监控的。

以下将对InnoDB进行标准监控，也就是运行SHOW ENGINE INNODB STATUS，对其输出做一些解析，其他监控器（如对于表空间的监控）可参考官方文档。

SHOW INNODB STATUS命令的输出信息不太方便进行脚本解析，而且输出信息里有很多平均值，不太好估算我们自己指定范围的统计结果，SHOW GLOBAL STATUS命令也有很多InnoDB的输出信息，使用SHOW GLOBAL STATUS会更好估算一些，也会更易于监控系统性能。

创建这些表之后，MySQL就会输出各种内部结构和性能信息到MySQL错误日志，对于InnoDB标准监视器，大概是每隔15s输出一次。笔者个人很少启用各种性能监控，一般是在做诊断的时候，直接运行命令，例如：

```
SHOW ENGINE INNODB STATUS \G
```

具体的输出解析如下。

```
***** 1. row *****
Status:
=====
100206 21:51:18 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 26 seconds
```

以上输出结果为最近26s的统计。如果是前1~2s的统计那么结果将不太可信。我们需要确保至少有20~30s的统计，否则结果会不太准确，还需要重新运行这个命令。

SHOW ENGINE INNODB STATUS的输出主要包含以下几个部分，这里以MySQL 5.1/5.5为例来进行讲述，其他版本与此类似。

- Background Thread
- Semaphores
- Latest Foreign Key Error
- Latest Detect Deadlock
- File I/O
- Insert Buffer and Adaptive Hash Index
- Log
- Buffer Pool and Memory
- Row Operations
- Transactions

(1) 信号量 (Semaphores)

下面是信号量相关信息。

```
SEMAPHORES
-----
OS WAIT ARRAY INFO: reservation count 13569, signal count 11421
--Thread 1152170336 has waited at ./../include/buf0buf.ic line 630 for 0.00 seconds the semaphore:
Mutex at 0x2a957858b8 created file buf0buf.c line 517, lock var 0
waiters flag 0
wait is ending
--Thread 114709792 has waited at ./../include/buf0buf.ic line 630 for 0.00 seconds the semaphore:
Mutex at 0x2a957858b8 created file buf0buf.c line 517, lock var 0
waiters flag 0
wait is ending
Mutex spin waits 5672442, rounds 3899888, OS waits 4719
RW-shared spins 5920, OS waits 2918; RW-excl spins 3463, OS waits 3163
```

解析：信号量（SEMAPHORES）节包含两部分信息，一部分信息是当前的操作系统等待（OS WAIT ARRAY INFO），在高并发的环境下，我们可能会看到这部分信息，因为InnoDB自旋等待超过了阈值，就会触发操作系统等待，如果等待通过自旋能够解决，那么这些信息就不会显示了。

通过检查这部分信息，可以大致判断负载的热点在哪里，由于输出行只包含了一些文件名，因此还需要有一些源码的知识，才能判断出现等待的真实原因。

另一部分信息是事件统计（event counter），reservation count和signal count的值表征了InnoDB需要OS WAIT的频率。我们也可以使用操作系统命令，如vmstat，通过检查上下文切换（context switch）的频率来确认OS WAIT的严重程度。

我们还需要了解一些操作系统进程调度的知识，如果进程不能获取锁（mutex可以理解为一种轻量级的锁），则CPU会自旋（spin），也就是CPU空转，以等待资源，此时并不需要进行上下文切换这种高成本的操作，也许CPU空转一些时间片，就可以获取到资源，但如果自旋超过了一定的次数，仍然无法获得资源，那么进程就需要切换到睡眠状态进行等待（OS WAIT），大量的OS WAIT意味着资源竞争很厉害，将造成很高的上下文切换频率。如果每秒有几万次的OS WAIT，那么很可能系统中存在性能问题。

大量的spin waits和spin rounds，意味着CPU在空转而没有实际做事，这会消耗大量的CPU资源，所以有时我们看到系统的CPU利用率很高，但也许并不是真正地在做事，而是CPU正在空转等待资源。通过调整innodb_sync_spin_loops参数，可以在CPU资源消耗和上下文切换之间找到平衡点。

(2) 死锁

下面是一个系统的死锁信息。

```
-----  
LATEST DETECTED DEADLOCK-----  
100206 14:46:39  
*** (1) TRANSACTION:  
TRANSACTION 0 353348573, ACTIVE 0 sec, process no 22381, OS thread id 823933856 inserting  
mysql tables in use 1, locked 1  
LOCK WAIT 3 lock struct(s), heap size 320, 2 row lock(s), undo log entries 1  
MySQL thread id 3176551, query id 27696260 del40 10.12.14.181 ooes_rss update  
insert into ooes_fav(id,name,uid,ctime,wapflag,url,parent_id,type) values('1'  
'  
'  
'  
7080277,  
'  
1265438796,  
'  
1265438796,  
'  
'  
'  
'  
2'  
)  
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:  
RECORD LOCKS space id 0 page no 1484846 n bits 144 index `uid` of table `ooes_rss`.`ooes_fav` trx id 0 353348573 lock_mode X insert intention waiting  
Record lock, heap no 1 PHYSICAL RECORD: n_fields 1; compact format; info bits 0  
0: len 8; hex 73757072656d756d; asc supremum;;  
*** (2) TRANSACTION:  
TRANSACTION 0 353348572, ACTIVE 0 sec, process no 22381, OS thread id 894077856 inserting, thread declared inside InnoDB 500  
mysql tables in use 1, locked 1  
7 lock struct(s), heap size 1024, 103 row lock(s), undo log entries 101 #这个事务更大  
MySQL thread id 3176549, query id 27696261 del40 10.12.14.180 ooes_rss update  
*** (2) HOLDS THE LOCK(S): #Note -  
InnoDB only prints information about few of the locks which transaction is holding.  
RECORD LOCKS space id 0 page no 1484846 n bits 72 index `uid` of table `ooes_rss`.`ooes_fav` trx id 0 353348572 lock_mode X  
Record lock, heap no 1 PHYSICAL RECORD: n_fields 1; compact format; info bits 0  
0: len 8; hex 73757072656d756d; asc supremum;;  
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:  
RECORD LOCKS space id 0 page no 1484846 n bits 144 index `uid` of table `ooes_rss`.`ooes_fav` trx id 0 353348572 lock_mode X insert intention waiting  
Record lock, heap no 1 PHYSICAL RECORD: n_fields 1; compact format; info bits 0  
0: len 8; hex 73757072656d756d; asc supremum;;  
*** WE ROLL BACK TRANSACTION (1)
```

解析：这段信息展示了是哪些事务导致了死锁、死锁过程中它们的状态、它们持有的锁、要等待的锁、回退到哪个事务（据官方文档可知，MySQL会回滚成本较小的事务，比如更新更少的行）等内容。由输出的最后一行可以得知，回退到了事务1。需要留意的是，这里只显示了部分持有的锁，只显示了事务中最近的语句，而实际上占据资源的可能是事务中前面的语句。在一些简单情况下，可以通过SHOW ENGINE INNODB STATUS的输出确认导致死锁的原因；在复杂的情况下，则需要打开通用日志，检查具体各个事务是如何互相等待资源从而导致死锁的。

MySQL 5.6可以通过参数innodb_print_all_deadlocks将死锁信息打印到错误日志中。

(3) 外键冲突

以下为外键冲突信息，开发人员需要注意。

```
-----  
LATEST FOREIGN KEY ERROR-----  
060717 4:29:00 Transaction:  
TRANSACTION 0 336342767, ACTIVE 0 sec, process no 3946, OS thread id 1151088992 inserting, thread declared inside InnoDB 500  
mysql tables in use 1, locked 1  
3 lock struct(s), heap size 368, undo log entries 1  
MySQL thread id 9697561, query id 188161264 localhost root update  
insert into child values(2,2)  
Foreign key constraint fails for table `test/child`:  
'CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`) REFERENCES `parent` (`id`) ON DELETE CASCADE'  
Trying to add in child table, in index `par_idx` tuple:  
DATA TUPLE: 2 fields;  
0: len 4; hex 80000002; asc    ; 1: len 6; hex 000000000401; asc      ;  
But in parent table `test/parent`, in index `PRIMARY`,  
the closest match we can find is record:  
PHYSICAL RECORD: n_fields 3; 1-byte off 0; info bits 0  
0: len 4; hex 80000001; asc    ; 1: len 6; hex 0000140c2d8f; asc -  
;; 2: len 7; hex 80009c40050084; asc
```

(4) 事务信息

```
-----  
TRANSACTIONS  
-----  
Trx id counter 0 80157601  
Purge done for trx  
s n:o < 0 80154573 undo n:o < 0 0  
History list length 6  
Total number of lock structs in row lock hash table 0  
LIST OF TRANSACTIONS FOR EACH SESSION:  
--TRANSACTION 0 0, not started, process no 3396, OS thread id 1152440672  
MySQL thread id 8080, query id 728900 localhost root  
show innodb status  
--TRANSACTION 0 80157600, ACTIVE 4 sec, process no 3396, OS thread id 1148250464, thread declared inside InnoDB 442  
mysql tables in use 1, locked 0  
MySQL thread id 8079, query id 728899 localhost root Sending data  
select sql_calc_found_rows * from b limit 5  
Trx read view will not see trx with id >= 0 80157601, sees < 0 80157597  
--TRANSACTION 0 80157599, ACTIVE 5 sec, process no 3396, OS thread id 1150142816 fetching rows, thread declared inside InnoDB 166  
mysql tables in use 1, locked 0  
MySQL thread id 8078, query id 728898 localhost root Sending data  
select sql_calc_found_rows * from b limit 5  
Trx read view will not see trx with id >= 0 80157600, sees < 0 80157596
```

```
--TRANSACTION 0 80157598, ACTIVE 7 sec, process no 3396, OS thread id 1147980128 fetching rows, thread declared inside InnoDB 114
mysql tables in use 1, locked 0
MySQL thread id 8077, query id 728897 localhost root Sending data
select sql_calc_found_rows * from b limit 5
Trx read view will not see trx with id >= 0 80157599, sees < 0 80157595
--TRANSACTION 0 80157597, ACTIVE 7 sec, process no 3396, OS thread id 1152305504 fetching rows, thread declared inside InnoDB 400
mysql tables in use 1, locked 0
MySQL thread id 8076, query id 728896 localhost root Sending data
select sql_calc_found_rows * from b limit 5
Trx read view will not see trx with id >= 0 80157598, sees < 0 80157594
```

解析：事务列表可能会很长，所以对于存在大量并发事务的系统，SHOW ENGINE INNOD STATUS会截去部分内容，只显示部分事务。

具体输出参数及其解析如下所示。

·Trx id counter...: 当前事务号，每创建一个新事务，这个值就会递增。

·Purge done for trx's no...: 最近一次进行线程清理的事务号，事务如果过期，则可以被清除，清除的标准是这些事务已经提交，且不会再被其他的事务所需要。

我们可以检查当前事务号和最近一次清理线程所清理的事务号的差异，例如，0（64位）80154573（32位）与0（64位）80157601（32位），如果差异很大，则可能有大量未被清理的事务，或者少量事务更新了大量数据。

事务应该被及时提交。长时间未提交的事务可能会阻塞清理操作，耗尽资源，不过对于Web访问，一般都是很小的事务，这点不太可能会成为问题。

事务更新记录时，将在UNDO中保存记录的前像。UNDO记录保存在InnoDB的共享表空间内。

如果事务未提交，或者其他用户需要查询UNDO记录以获得一致性读，此时是不能清理这部分事务的。大量未清理的事务，可能会导致UNDO空间暴涨，在紧急情况下，我们可以设置innodb_max_purge_lag参数来延缓新事务的更新，不过这个参数要慎用，因为它会降低性能，治标不治本。

下面来举个例子说明一下这个参数。如果你的InnoDB表空间可以忍受100M未清理的行，也就是平均每个事务大概影响1K的行，那么你可以设置这个值为100000（100M/1K）。

·undo no: Purge操作正在处理的UNDO日志记录号。

·History list length 6: 在UNDO空间内未被清理的事务数量，在事务更新数据的时候该值会增加，在事务清理后该值会减少。

·Total number of lock structs in row lock hash table 0: 行锁哈希表（row lock hash table）中的锁结构（lock struct）的数量，该值不同于被锁定的行，因为通常会有多个行对应一个锁结构。

·LIST OF TRANSACTIONS FOR EACH SESSION:

--TRANSACTION 00, not started, process no 3396, OS thread id 1152440672: 每个事务都有两个状态，即not started或active。在生产系统中，同时运行的线程一般最多只有几个，所以大部分事务都是not started。

需要留意的是，即使连接的状态是sleep，事务也可能是active的，因为事务可能是多语句的，在生产环境中可以发现，一些长时间sleep的异常线程可能会持有着资源不释放，从而导致整个系统出现异常。

InnoDB有一个参数为innodb_thread_concurrency，用来控制并发执行的线程数。InnoDB试着在其内部控制操作系统线程的数量，使其少于或等于这个参数给出的限制。如果SHOW INNODB STATUS显示有很多线程在等待（waiting in InnoDB queue或sleeping before joining InnoDB queue）进入队列，那么往往是有性能上的问题，导致系统挂死。MySQL让等待的线程睡眠，从避免太多线程并发竞争，如果你的计算机有多个处理器和磁盘，则可以试着将这个值调整得更大以更好地利用计算机的资源。一个推荐的值是采用系统上处理器和磁盘的个数之和。



注意 MySQL的配置里还有一个thread_concurrency参数，建议设置为CPU数的2倍大小。此变量仅仅影响Solaris系统。在Solaris中，mysqld用该值调用thr_setconcurrency()函数。该函数使得应用程序可以向线程系统提供需要同时运行的、期望的线程数目。此外，其实innodb_thread_concurrency这个参数才会影响到所有的平台。

·mysql tables in use 1,locked 0: 访问的表数目，锁定的表数目。一般的操作是不会锁表的，InnoDB支持行级锁，所以locked一般等于0，除非是进行ALTER TABLE、LOCK TABLE之类的操作。

·MySQL thread id 52111305: SHOW PROCESSLIST命令输出中的id列。

(5) I/O信息

以下是IO helper threads的状态。

```
-----  
FILE I/O  
-----  
I/O thread 0 state: waiting for i/o request (insert buffer thread)  
I/O thread 1 state: waiting for i/o request (log thread)  
I/O thread 2 state: waiting for i/o request (read thread)  
I/O thread 3 state: waiting for i/o request (write thread)
```

这4个线程（Unix/Linux下总是4个）的作用分别是insert buffer merges、asynchronous log flushes、read-ahead和flushing of dirty buffers。

当前看到它们的状态都是waiting for i/o request。

```
Pending normal aio reads: 0, aio writes: 0,  
ibuf aio reads: 0, log i/o:  
s: 0, sync i/o:  
s: 0  
Pending flushes (fsync) log: 0; buffer pool: 0  
6845394 OS file reads, 209547550 OS file writes, 1051178 OS fsyncs  
7.27 reads/s, 16384 avg bytes/read, 256.68 writes/s, 1.88 fsyncs/s
```

如果以上Pending为非零值，则可能存在I/O瓶颈。

对于随机I/O，因InnoDB的I/O最小单元（page size）=16KB。所以为16384 avg bytes/read，对于全表扫描（full table scan）、索引范围扫描（index scan），这个avg bytes/read会大得多。

(6) INSERT BUFFER AND ADAPTIVE HASH INDEX

MySQL并没有提供手段对以下结构进行调优。

```
-----  
INSERT BUFFER AND ADAPTIVE HASH INDEX-----  
Ibuf: size 1, free list len 0, seg size 2,
```

这里ibuf即Insert buffer，虽然英文中说的是“buffer”，但实际上这是分配在InnoDB表空间中的一块区域，它可以和其他数据块一样，缓存在InnoDB缓冲池里，Insert buffer可以减少I/O，因为它可以合并对索引叶节点的更改操作。

(7) LOG

下面将讲述InnoDB的log子系统。

```
-----  
LOG-  
Log sequence number 449 61757582  
Log flushed up to 449 61751106  
Last checkpoint at 448 4209429402  
0 pending log writes, 0 pending chkp writes  
201992232 log i/o:  
s done, 250.14 log i/o:  
s/second
```

其中的输出参数及其解析具体如下。

·Log sequence number 44961757582：表空间创建后写入log buffer的字节数，这个值可以用来衡量日志的写入速度。通过采样Log sequence number的输出，可以获取每秒写入的日志量，如果我们要设置InnoDB事务日志的大小，那么能保持连续写入日志30~60分钟为佳。

·Log flushed up to 44961751106：最近刷新（flush）数据的位置。

由此可以计算还有多少未刷新到日志文件（logfile）的数据。如果这些数据大于innodb_log_buffer_size的30%，那么就要考虑是否应增加日志缓冲（log buffer）了。

·Last checkpoint at 4484209429402：最近一次检查点的位置。

·0 pending log writes, 0 pending chkp writes：pending如果大于0，则可能有I/O瓶颈。

·201992232 log i/o's done, 250.14 log i/o's/second：这些输出衡量了我们的log I/O。

(8) BUFFER POOL AND MEMORY

以下是InnoDB缓冲池的信息。

```
-----  
BUFFER POOL AND MEMORY  
-----  
Total memory allocated 4648979546; in additional pool allocated 16773888  
Buffer pool size 262144  
Free buffers 0  
Database pages 258053  
Modified db pages 37491  
Pending reads 0  
Pending writes: LRU 0, flush list 0, single page 0  
Pages read 57973114, created 251137, written 10761167  
9.79 reads/s, 0.31 creates/s, 6.00 writes/s  
Buffer pool hit rate 999 / 1000
```

需要说明的是“Buffer pool hit rate”的参考价值不是很大。即使有很高的命中率，也可能有大量的物理磁盘读写。

(9) ROW OPERATIONS

以下是行操作信息。

```
-----  
ROW OPERATIONS  
0 queries inside InnoDB, 0 queries in queue  
1 read views open inside InnoDB  
Main thread process no. 10099, id 88021936, state: waiting for server activity  
Number of rows inserted 143, updated 3000041, deleted 0, read 24865563  
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
```

我们可以由以上信息获知各种查询的大概频率，需要留意的是如果“0 queries in queue”不为0，则是有查询需要等待，可能意味着系统忙，你需要做进一步的诊断。



小结 本章介绍了MySQL运维所需要了解的各种数据库文件及MySQL如何进行灾难恢复。你必须了解各种文件的作用和机制，避免在操作系统下对数据库文件误操作。本章还介绍了数据库的参数设置与配置文件，MySQL的配置不应该经常变动，你应该使用大多数人建议的配置，根据自己的生产环境做适当调整即可。最后介绍了查询缓冲和MySQL优化器，我们要熟悉这些主要的组件。此外，还讲述了如何阅读SHOW INNODB STATUS\G命令的输出。

其他的一些基础知识已在开发篇中进行了介绍，比如索引设计、查询优化。读者也应该熟悉这些内容。

第11章 MySQL的监控

为什么我们需要监控呢？因为如果没有了监控，那么我们的服务可用性就无从度量，我们也无法及时地发现问题和处理问题。一个完善的监控体系，不仅需要进行实时的监控，也需要分析历史的监控数据，以便掌握性能和容量趋势的变化，从而为产品、架构人员提供决策的依据。

本章将为读者讲述针对MySQL所提供的监控方法，然后，再来探讨下数据库监控的友好呈现，也就是数据的可视化技术。

11.1 非数据库的监控

11.1.1 开源监控工具/平台

一个完整的监控体系，要求能够监控各种非数据库的资源，如操作系统、硬件、网络等。目前比较流行的方式是部署一些集中监控工具，当你需要维护越来越多的机器，特别是基于云的部署时，一个集中式的监控产品就会变得很重要了，你可以从该监控界面上，查看所有设备的使用情况。

对于集中式的监控产品，一般需要在被监控的服务器中部署一个代理服务（agent）来收集数据，如Ganglia、Nagios等，或者通过一些系统服务收集信息，比如snmp。广泛使用的一些平台有Zabbix、Nagios、Ganglia、Cacti，读者可以自行阅读相关图书，学习如何使用它们，本书将只关注数据库的性能监控和故障发现。

有时我们希望能够开发出自己的数据库监控平台，自己编写脚本、工具来收集信息。这样会更有针对性。不过笔者建议读者使用市面上流行的监控工具或平台，很多监控平台都有MySQL相关的监控插件，我们需要做的只是少量的二次开发工作。完全重新开发一个监控平台的成本往往比较高，需要综合权衡是否有必要投入人力去实现，有时，在一些开源软件上做二次开发，是更经济的方式。

11.1.2 编写程序来收集信息

如果我们想要自己编写脚本收集信息，那么，对于操作系统的信息收集一般有两种方式。

第一种，使用工具收集。可以使用vmstat、dstat、iostat、sar、netstat这些工具/命令实时观察操作系统。

第二种，直接使用接口，比如读取/proc/伪文件系统数据这种方式，/proc是许多可视化工具获取信息的来源，一些工具，如ps、top、pmap其实都是读取/proc下的数据。使用这种方式有一个风险，那就是它们可能没有工具那么通用，在系统升级后，数据格式可能会有变化，你需要调整处理数据的程序。

在Linux系统中，/proc是一个伪文件系统（启动时动态生成的文件系统），它是访问内核统计数据的一个接口，由于/proc不是一个真正的文件系统，因此它并不占用存储空间，而是占用有限的内存。/proc下面有很多以进程id命名的目录，目录下的很多文件记录了进程的使用信息。/proc目录下也有一些系统级别的统计信息，如/proc/meminfo就记录了内存活动和统计信息。

自行编写程序进行监控需要考虑到如下一些要点。

1) Linux的I/O是比较难监控的，如果在一台主机之上有多个应用，那么判断I/O负载重的业务有哪些将会很困难。安装iotop之类的工具可以更快地定位到I/O负载重的进程之上，但iotop之类的工具需要新的内核支持。

2) 由于SSD的大量使用，因此还需要增加对SSD的监控，常用的方式是使用smartctl命令进行监控。

3) 有时我们需要模拟业务访问。人的行为是复杂的，复杂的业务系统更是充满了变数，模拟人的行为是一件困难且富有挑战性的事情，我们应该模拟尽量真实

的访问，这样才能得到真实的反馈，从而衡量服务是否真的健壮、可靠、体验良好。

- 4) 要注意收集信息的频率，粒度太大了可能不能及时发现问题。
- 5) 要防止积压收集信息的程序任务。
- 6) 要确保报警通知到人，还要确保邮件服务、短信等通道的正常。

11.2 数据库的监控

11.2.1 数据库服务的基本监控方式

一般数据库的监控包括探测数据库主库的可用性、复制状态监控、数据库的性能监控、数据库的故障发现等。

对于数据库主库的监控，可以在主库上创建一张监控表，使用监控程序定期去读写这张表中的数据，以判断数据库是否可以正常提供服务。

对于数据库从库的监控，由于从库一般都是只读的，因此只需要定期查询从库上监控表的数据即可。

对于复制状态的监控，由于主库有定期更新的监控表，因此可以认为它也是一张心跳表，表里的数据带有时间戳信息，主库监控表（心跳表）每分钟被UPDATE一次，去从库中查询对应的记录，就可以依据记录内的时间戳信息来确认延时了多少。这里需要说明的是，MySQL自身的SHOW SLAVE STATUS\G显示的延时时间可能是不准确的，所以，推荐使用心跳表的方式。

需要注意的是，每次信息收集的时间间隔不能太大，否则会难以发现和诊断问题。比如磁盘I/O数据每10分钟才去收集一次，数据库性能每隔几分钟才去收集一次，就不是一个好的选择。

数据库的性能监控主要依靠于收集MySQL的一些状态变量，也就是SHOW GLOBAL STATUS的输出。

数据库的故障发现涉及的内容包括：分析MySQL的查询响应、错误日志，以及监控是否可以读写数据库。

11.2.2 应该收集的信息和收集方法

我们收集的信息主要包括MySQL的运行状态、程序性能日志、慢查询日志、状态变量、数据量、数据占用空间等。

1.MySQL的参数及运行状态

以下代码可查看MySQL实例的参数及运行状态。

```
SHOW VARIABLES LIKE '%parameter%';
SHOW FULL PROCESSLIST;
SHOW INNODB STATUS \G;
```

以下是对一个SHOW PROCESSLIST的解析。可以看到不同状态下线程的比例。

```
mysql -uroot -p -S /path/to/tmp/3306/mysql.sock -e 'SHOW PROCESSLIST\G' | grep State: | sort | uniq -c | sort -rn
```

下面来解释一些常用的状态。

·**Sleep**: 线程正在等待来自客户端的新查询。

·**Query**: 线程正在执行查询，或者正在发送结果给客户端。

·**Locked**: 线程正在等待表锁。

·**Analyzing**和**statistics**: 线程正在获取存储引擎的统计数据和优化查询。

·**Copying to tmp table[on disk]**: 线程正在处理查询，复制数据到临时表中。如果后面有“on disk”字样，则表明MySQL正在将内存临时表转换为磁盘临时表。

·**Sorting result**: 线程正在排序结果集。

·**Sending data**: 这个状态有多种可能，可能是内部各步骤之间传递数据，生成结果集；或者是将结果集返回给客户端。

大多数状态对应的操作都非常快。如果一个线程停留在一个给定的状态好几秒，那么它可能是有问题的，需要进一步查明。

下面来查看InnoDB的一些统计数据，命令如下所示。

```
SHOW INNODB STATUS \G;
```

如下命令可查看SQL的执行频率，实时显示当前各种SQL的执行频率等信息，该命令摘录自网上。

```
mysqladmin -uroot -p -r -i 2 extended-status |awk -F "!" 'BEGIN { count=0; } { if($2 ~ /Variable_name/ && ++count%15 == 1){print "-----|-----|--- MySQL\nCommand Status ---|---- InnoDB row operation ----|--- Buffer Pool Read --"; print "---Time---|---QPS---|select insert update delete| read inserted updated deleted| logical physical";} else if ($2 ~ /Queries/){queries=$3;} else if ($2 ~ /Com_select/){com_select=$3;} else if ($2 ~ /Com_insert/){com_insert=$3;} else if ($2 ~ /Com_update/){com_update=$3;} else if ($2 ~ /Com_delete/){com_delete=$3;} else if ($2 ~ /InnoDB_rows_read/){innodb_rows_read=$3;} else if ($2 ~ /InnoDB_rows_deleted/){innodb_rows_deleted=$3;} else if ($2 ~ /InnoDB_rows_inserted/){innodb_rows_inserted=$3;} else if ($2 ~ /InnoDB_rows_updated/){innodb_rows_updated=$3;} else if ($2 ~ /InnoDB_buffer_pool_read_requests/){innodb_buffer_pool_read_requests=$3;} else if ($2 ~ /InnoDB_buffer_pool_reads/){innodb_buffer_pool_reads=$3;} else if ($2 ~ /InnoDB_buffer_pool_writes/){innodb_buffer_pool_writes=$3;} else if ($2 ~ /Uptime/ && count >= 2){ printf("%H:%M:%S",strftime("%H:%M:%S",queries);printf("|%6d %6d %6d",com_select,com_insert,com_update,com_delete);printf("|%8d %7d %7d",innodb_rows_read,innodb_rows_inserted,innodb_rows_updated,innodb_rows_deleted); printf("|%10d %11d\\n",innodb_lor,innodb_phr);}}'
```

如果需要做进一步的分析，也可以用tcpdump配合pt-query-digest工具来获取更多的信息，它所生成的报告不仅包括SQL的计数，还包括SQL的耗时及其他成本信息。

首先，在root用户下执行如下命令。

```
nohup tcpdump -i eth1 port 3306 -s 65535 -x -nn -q -ttt > db1000_sql_new.log &
```

然后在mysql用户下执行如下命令。

```
./pt-query-digest --type=tcpdump --watch-server 11.11.11.11:3306 db1000_sql_new.log > app_db.rtf
```

对于以上生产报告，可以发送邮件给DBA阅读，或者将其过滤后存放在数据库中。

SQL的统计最好在应用层收集信息，这是笔者推荐的方式，SQL的很多统计，结合应用才能易于理解，才能更好地评判是否应该进行优化，大致方法如下。

1) 直接记录SQL到日志，统计日志中各种查询的比例。

2) 根据Web服务器日志，例如根据一天中高峰期一个小时的日志，统计涉及某些功能（SQL）页面的日志在总的日志中所占的比例。

2.性能日志

这里所说的性能日志，一般是指程序性能日志。很多公司并没有考虑性能日志，主要是出于开发的成本考虑。但一个良好的、完善的服务程序，应该包含自诊断的信息，以协助诊断问题。

我们应先查看整个应用的性能表现，从总体上来分析。一般来说，程序的性能日志是最容易诊断出性能瓶颈的，性能日志也可以用图形的方式展示出应用的性能变化趋势，方便作为以后扩容的依据。

例如，对于一个PHP程序，应该收集的信息主要有如下几点。

·合计执行时间（页面执行时间）。

其他各部分时间相加应等于合计执行时间，差别不能过大，否则就要研究是否哪部分操作未做记录。

·每个查询的执行时间。

·打开的连接。

·对外部服务的调用。

·可能消耗资源较大的数据库操作。

如果性能日志足够详细，那么就可以快速地定位性能的瓶颈所在了，从而判断是否真的是MySQL导致了性能问题，是否访问MySQL耗费了绝大部分的页面时间。进行压力测试的时候，可以定位伸缩性存在问题的环节。关于性能日志更详细的信息请参考4.4节。

3.慢查询日志

除了SHOW GLOBAL STATUS和SHOW PROCESSLIST之外，还可以检查慢查询日志，一般推荐优先采用前面两种方式检查系统，慢查询的检查又耗时又复杂。

MySQL提供了两类日志：通用日志（general log）和慢查询日志（slow log）。通用日志记录了接收的所有查询。这个日志有助于判断读写的比例，看MySQL的主要工作是什么？但打开通用日志需要注意日志的空间消耗，可能还需要考虑轮询切割日志。一般情况下没必要启用通用日志。这里仅分析慢查询日志。

慢查询日志记录了慢查询情况，我们可以把捕捉到的日志二次处理后发送给研发人员进行优化，MySQL 5.1及以上版本可以动态启用慢查询日志，MySQL 5.0则需要重启后才能生效。

需要设置的参数如下。

MySQL 5.0需要设置`log_slow_queries`和`slow_launch_time`。

MySQL 5.1需要设置`slow_query_log`和`long_query_time`，这里不需要再使用`slow_launch_time`这个参数了，因为这个参数不能设置到毫秒级。MySQL 5.1.21后可以进行毫秒级的慢查询记录，例如，设置`long_query_time=0.01`。

有一个参数`log_queries_not_using_indexes`，也可以协助分析，不过小表无须建立索引速度也很快，这样的情况下，使用该参数可能会导致产生大量的日志记录。因此建议忽略这个参数，不予设置。

有一些补丁或MySQL分支，如Percona Server，可以显示出更翔实的慢查询信息，这样就有助于我们探查到底是什么原因导致的查询慢，因为官方版本中慢查询日志默认的输出信息都比较粗略，并没有告诉我们查询为什么会变慢。

某个SQL出现在慢查询日志里，并不意味着这就是一个质量差的SQL，也并不表示现在或未来这个查询很慢，也许你手动执行它，会非常之快。有诸多因素会影响到SQL的响应：如锁表、数据或索引初次使用时未被缓存、磁盘I/O紧张、内存泄露等。现实中，如果一个查询平时运行得很快，但在发现性能问题时被记入慢查询日志，可能是因为其他查询占用了大量的系统资源，被阻塞而导致的。

对于慢查询日志的分析可以使用MySQL自带的`mysqldumpslow`来实现，还有一些比`mysqldumpslow`更强大的分析工具，如`pt-query-digest`。

对于慢查询日志，可以关注执行时间过长的查询，或者执行次数过多的查询，或者结果集过大的查询。

通过如下命令，可以看到每秒的慢查询统计，以方便绘图。当检查到有突变时，往往这个时候会有异常发生，可以更进一步到具体的慢查询日志中去查找可能的原因。

```
awk '/^# Time:{print $3, $4, c;c=0}/{User/(c++)' slowquery.log > /tmp/aaa.log
```

4.状态变量

如`SHOW GLOBAL STATUS`、`SHOW SESSION STATUS`和`SHOW PROFILE`

查看全局状态变量的命令如下。

```
SHOW GLOBAL STATUS LIKE '%parameter%';
```

查看吞吐率时，可多次运行下面的命令，检查增量。

```
SHOW GLOBAL STATUS LIKE '%question%';
;
SHOW GLOBAL STATUS LIKE '%Com_%';
;
```

也可以使用如下命令检查多个系统状态变量的变化。

```
SHOW GLOBAL STATUS WHERE Variable_name LIKE 'Com_select' OR Variable_name LIKE 'Com_insert' OR variable_name LIKE 'com_update' OR variable_name LIKE 'com_delete' OR variable_name LIKE 'Qcache_hits';
```

使用`mysqladmin`命令可监视状态变量的变化，注意如下命令中添加了参数`-r`。

```
mysqladmin -uroot -p extended-status -r -i 10 |egrep "Com_select|Com_insert|Com_delete|Com_update|Qcache_hits|Handler_write|Handler_read"
```

`mysqladmin`命令的另一个示例如下。

```
mysqladmin -uroot -p -il | awk '
/Quries/{q=$4 qp=$4}
/Threads_connected/{tc=$4}
/Threads_running/{printf "%5d %5d %5d\n",
q,
tc,
$4}'
```

也可以使用监控工具的一些插件来监控状态变量的变化，如使用Cacti的MySQL插件。Cacti有丰富的模板支持，可以近乎实时地监察MySQL的运行状态。它主要也是用于获取`SHOW GLOBAL STATUS`的信息。

关于查询读写比率的计算，可以大致采用如下的公式进行计算。

```
(SELECT + Qcache_hits) / (INSERT + UPDATE + REPLACE + DELETE)
```

相应状态变量可以查询以“com_”或以“handler_”为前缀的一些变量。

在SHOW GLOBAL STATUS中，我们需要关注的主要有以下几个计数器：handler、temporary files、command、wait。

有时我们需要单独分析一些查询的成本，需要先手动清除状态变量（运行命令FLUSH STATUS），然后再运行查询，最后重新运行SHOW SESSION STATUS，从此来查看查询所耗费的成本。

SHOW SESSION STATUS，顾名思义，显示的是当前会话的状态变量，它不受其他进程的影响。

以下示例显示了执行一个SQL后会话的Select%状态的变化，为了节省空间，这里没有列出所有状态的值。

```
flush status;
mysql> SELECT SQL_NO_CACHE ... from ...
mysql> show session status like 'Select%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Select_full_join | 0 |
| Select_full_range_join |
| Select_range | 0 |
| Select_range_check | 0 |
| Select_scan | 2 |
+-----+-----+
```

·**Select_full_join**: 全表扫描连接的次数，如果该值比较高，那么可能是没有正确地创建索引。

·**Select_full_range_join**: 在引用的表中使用范围查找的连接数量。

·**Select_scan**: 执行了全表扫描的数量。

如下命令将检查存储引擎操作。

```
mysql> SHOW SESSION STATUS LIKE 'Handler%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Handler_commit | 0 |
| Handler_delete | 0 |
| Handler_discover | 0 |
| Handler_prepare | 0 |
| Handler_read_first | 1 |
| Handler_read_key | 5665 |
| Handler_read_next | 5662 |
| Handler_read_prev | 0 | DESC.
| Handler_read_rnd | 200 |
| Handler_read_rnd_next | 207 |
| Handler_rollback | 0 |
| Handler_savepoint | 0 |
| Handler_savepoint rollback | 0 |
| Handler_update | 5262 |
| Handler_write | 219 |
```

·**Handler_read_first**: 索引中第一条被读的次数。如果较高，则代表服务器正在执行大量全索引扫描。

·**Handler_read_key**: 根据键读取一行记录的请求数。如果该值较高，则说明查询和表的索引是正确的。

·**Handler_read_next**: 按照键顺序读下一行的请求数。如果你使用范围约束或执行索引扫描来查询索引列，那么该值会增加。

·**Handler_read_prev**: 按照键顺序读取前一行的请求数。

·**Handler_read_md**: 根据固定位置读取一行的请求数。如果你正执行大量的查询并且需要对结果进行排序，那么该值会较高。如果使用了大量的需要MySQL扫描整个表的查询语句，或者连接没有正确地使用键，那么该值也会较高。

·**Handler_read_md_next**: 在数据文件中读取下一行的请求数。如果你正在进行大量的表扫描，那么该值会较高。通常情况下，该值高说明你的表索引不正确，或者写入的查询没有利用索引。

·**Handler_update**: 在表内插入一行的请求数。以上示例中**Handler_update**计数器的值比较高，是因为MySQL的GROUP BY、ORDER BY操作会先把表写入一个临时表，扫描后进行排序，然后进行输出。

·**Sort_merge_passes**: 排序算法已经执行了合并的数量。如果这个变量值较大，则应该考虑增加**sort_buffer_size**系统变量的值。

如下命令将检查sort相关的统计。

```
mysql> SHOW SESSION STATUS LIKE 'Sort%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Sort_merge_passes | 0 | | Sort_range | 0 |
| Sort_rows | 200 |
| Sort_scan | 1 |
+-----+-----+
```

·**Sort_rows**: 已经排序的行数。

·Sort_scan: 通过扫描表完成排序的数量。

如下命令将查看临时表的创建情况。

```
mysql> SHOW SESSION STATUS LIKE 'Created%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Created_tmp_disk_tables | 0 |
| Created_tmp_files | 0 |
| Created_tmp_tables | 5 |
```

·Created_tmp_disk_tables: 如果持续增加, 那么可能是有性能问题。

以上输出可能仍然会受到内部操作的影响, 建议多运行几次查询, 从而得到一个比较可靠的增量。笔者将在后续章节里更详细地解释一些状态变量的含义。

另外还有一个简单易用的方法, 使用SHOW PROFILE。该功能默认是关闭的, 但是会话级别可以开启这个功能。开启它可以让MySQL收集在执行语句的时候所使用的资源和耗时。

下面的示例将使用SET profiling=1开启这个功能。

```
root@localhost test>SHOW VARIABLES LIKE '%profil%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| profiling | OFF |
| profiling_history_size | 15 |
+-----+-----+
2 rows in set (0.00 sec)
root@localhost test>SET profiling = 1;
Query OK, 0 rows affected (0.00 sec)
root@localhost test>SELECT COUNT(*) FROM testad;
+-----+
| count(*) |
+-----+
1 |
+-----+
1 row in set (0.00 sec)
root@localhost test>SHOW PROFILES \G;
***** 1. row *****
Query_ID: 1
Duration: 0.00015100
Query: select count(*) from testad
1 row in set (0.00 sec)
ERROR:
No query specified
root@localhost test>SHOW PROFILES;
+-----+-----+
| Query_ID | Duration | Query |
+-----+-----+
| 1 | 0.00015100 | select count(*) from testad |
| 2 | 0.00017100 | select count(*) from testac |
+-----+-----+
```

如果SHOW PROFILE后不加参数, 则显示最近的查询统计。Status栏位与SHOW FULL PROCESSLIST的Status栏位相同。

```
root@localhost test>SHOW PROFILE;
+-----+-----+
| Status | Duration |
+-----+-----+
| starting | 0.000031 |
| checking query cache for query | 0.000035 |
| Opening tables | 0.000011 |
| System lock | 0.000004 |
| Table lock | 0.000019 |
| init | 0.000010 |
| optimizing | 0.000006 |
| executing | 0.000009 |
| end | 0.000003 |
| query end | 0.000002 |
| freeing items | 0.000011 |
| storing result in query cache | 0.000006 |
| logging slow query | 0.000002 |
| cleaning up | 0.000002 |
+-----+-----+
14 rows in set (0.01 sec)
mysql> SHOW PROFILE CPU FOR QUERY 1;
```

如上介绍了SHOW SESSION STATUS及SHOW PROFILE命令, 笔者很少使用它们, 一般来说, 使用SHOW GLOBAL STATUS命令检查状态变量即可。

5.MySQL实例的数据增长

我们需要获取MySQL实例的数据增长情况, 以便提前进行扩容, MySQL的information_schema库记录了各个库、表的数据量大小, 可以据此统计实例的数据增长情况, 以及各个库, 甚至各个表的数据增长情况, 研发人员通过判断表的数据量增长趋势及数据库的操作频率, 大致判断应用的数据库流量的特点, 从而更有针对性地进行数据库的应用优化。

需要注意的一点是, SHOW TABLE STATUS命令可以查看表的很多信息, 但InnoDB引擎表的统计信息可能不是很准确, 尤其是在表特别大的时候。

数据表的大小可根据information_schema.tables表中Data_length和Index_length列的和大致统计。

当我们使用共享表空间的时候, 有时希望能够合理分配每个数据文件的大小, 还可能需要知道数据文件的空闲空间还有多少。这时, 可以启动Tablespace Monitor, 通过日志输出收集表空间的信息, 通过计算使用的块和空闲的块来判断表空间的空闲空间还有多少, 以及数据增长的趋势。但此种方式不易操作, 分析也较复杂, 所以更合适的办法是简单查询MySQL自带的统计表, 据此进行估算。

在实际生产环境中，我们可以定期查询INFORMATION_SCHEMA信息数据库，把收集的数据库大小插入监控数据库，在收集的信息的基础上进行空间趋势分析。

下面的查询将检查数据库argls下面的所有基础表的信息。

```
SELECT TABLE_SCHEMA, TABLE_NAME, TABLE_TYPE, ENGINE, TABLE_ROWS ,DATA_LENGTH, INDEX_LENGTH, DATA_FREE FROM tables WHERE TABLE_SCHEMA='argls' AND TABLE_TYPE='BASE TABLE';
```

下面的查询将统计数据库argls的大小。

```
SELECT SUM(DATA_LENGTH),SUM(INDEX_LENGTH),SUM(DATA_FREE) FROM tables WHERE TABLE_SCHEMA='argls' AND TABLE_TYPE='BASE TABLE';
SUM(DATA_LENGTH) | SUM(INDEX_LENGTH) | SUM(DATA_FREE)
| 35503304092 | 5593716736 | 42949672960 |
```

关于DATA_FREE列，没有太大的参考意义此处不做讲解。

11.2.3 MySQL需要关注的参数及状态变量

以下的一些状态变量，是监控系统需要着重关注的，由于篇幅所限，这里并没有列出所有值得关注的状态变量。

(1) open_files_limit

操作系统允许mysqld打开的文件数量。这个值可以设置得比较大，比如50000，最好在系统初始化安装时就设置了一个较大的值。可修改文件/etc/security/limits.conf来实现，命令如下。

```
vi /etc/security/limits.conf
* - nofile 50000
```

(2) max_connect_errors

此值应设置得比较大，如大于5000，以避免因为连接出错而超过出错阈值，导致MySQL阻止该主机连接。如被阻塞，则须手动执行flush-hosts进行复位。

(3) max_connections

允许并行的客户端连接数目。默认值100太小，一般会不够用。

生产环境中建议设置为2000~5000。注意，对于32位的MySQL由于有内存限制，连接数不能过大（建议小于800），否则可能会由于连接过多，造成MySQL实例崩溃。

(4) max_used_connections

MySQL Server启动后曾经到达的最大连接数。如果该值达到max_connections，那么某个时刻存在突然的高峰连接时，可能会有性能问题。

(5) threads_connected

当前打开的连接数量。这个值不能超过设置的max_connections*80%。需要注意及时调整max_connections的值。一旦连接数超过了max_connections，就会出现客户端连接不上的错误。

(6) aborted_connects

试图连接到MySQL服务器而失败的连接数。正常情况下，该值不会持续增加，出现连接失败的原因主要有如下几点。

- 客户端程序在退出之前未调用mysql_close()。
- 客户端的空闲时间超过了wait_timeout或interactive_timeout秒，未向服务器发出任何请求。
- 客户端在数据传输中途突然结束。

(7) Aborted_clients

由于客户端没有正确关闭连接导致客户端终止而中断的连接数。

出现下述情况时，服务器将增加“Aborted_clients”（放弃客户端）的状态变量。

- 客户端不具有连接至数据库的权限。
- 客户端采用了不正确的密码。

·连接信息包含不正确的信息。

·获取连接信息包的时间超过了connect_timeout秒。

我们可以使用如下的命令发现异常。

```
mysqladmin -uroot -p -S /path/to/tmp//3306/mysql.sock ext | grep Abort
```

也可以使用tcpdump来判断是什么原因导致了异常。

```
tcpdump -s 1500 -w tcp.out port 3306  
strings tcpdump.out
```

(8) thread_cache_size

服务器应缓存多少线程以便重新使用？当客户端断开连接时，如果线程少于thread_cache_size，则客户端的线程将被放入缓存。如果有新连接请求分配线程则可以从缓存中重新利用线程，只有当缓存空了时才会创建新线程。如果新连接很多，则可以增加该变量以提高性能。如果是大量并发的短连接，则可能会因为thread_cache_size不够而导致性能问题。生产环境中一般将其设置为100~200。

由于线程可以缓存，所以线程持有的内存不会被轻易释放。

(9) Threads_created

创建用来处理连接的线程数。应该监视Threads_created的增量，如果较多，则需要增加thread_cache_size的值。

以上对thread_cache_size的设置在高并发的时候会很有效。高并发时大量并发短连接对CPU的冲击不容忽视。

(10) threads_running

指同时运行的线程数目。这个值一般不会大于逻辑CPU的个数，如果经常有过多的线程同时运行，那么可能就意味着有性能问题。这个指标很重要，往往表明了一个系统的繁忙程度，它在系统爆发性能问题之前，会有一个上升的趋势，此时收集的性能信息，将有助于我们诊断复杂的性能问题。

(11) slow_launch_threads

如果这个值比较大，则意味着创建线程太慢了，可能是系统出现了性能问题，存在资源瓶颈，从而导致操作系统没有安排足够的CPU时间给新创建的线程。

(12) query_cache_size

为缓存查询结果分配的内存大小。一般设置为256MB。注意不要设置得太大。

可监控查询缓存命中率：Qcache_hits/(Qcache_hits+Com_select)。

更改这个值，会清空所有的缓存结果集，对于非常繁忙的系统，可能会很耗时，导致服务停顿，因为MySQL在删除所有的缓存查询时是逐个进行的。

(13) Qcache_lowmem_prunes

该变量记录了由于查询缓存出现内存不足，而需要从缓存中删除的查询数量，可通过监控Qcache_lowmem_prunes的增量，来衡量是否需要增大query_cache_size。

Qcache_lowmem_prunes状态变量提供的信息能够帮助你调整查询缓存的大小。它可计算为了缓存新的查询而从查询缓存区中移出到自由内存中的查询数目。查询缓存区使用最近最少使用（LRU）策略来确定哪些查询需要从缓存区中移出。

(14) InnoDB_buffer_pool_wait_free

一般情况下，是通过后台向InnoDB缓冲池中写入数据的。但是，如果需要读或创建页，并且没有干净的页可用，那么它还需要先等待页面清空。如果已经适当设置了缓冲池的大小，那么该值应该会很小。

(15) Slow_queries

查询时间超过long_query_time秒的查询个数。应该监控此变量的增量变化，一般1秒内不要超过5~10个，否则可能是有性能问题。

(16) Select_full_join

没有使用索引的连接数量。如果该值较大，则应该仔细检查一下表的索引。

(17) Created_tmp_tables

创建内存临时表的数量，如果Created_tmp_disk_tables比较大，则应该考虑增加tmp_table_size的大小。



注意 应该将`tmp_table_size`和`max_heap_table_size`简单调整到大小一样。32MB一般足够了。对这两个参数的控制通常基于内存引擎的临时表可以增长的阈值，若超过了这个阈值，就会转化成On-disk MyISAM表。

(18) `Created_tmp_disk_tables`

服务器执行语句时在硬盘上自动创建的临时表的数量。

(19) `Bytes_received`和`Bytes_sent`

可以用来监控MySQL的流量。

(20) `key_buffer_size`

MyISAM索引缓冲，实际用到多少就分配多少。不一定需要分配很大的空间，可参考实际观察到的值，不要大于实际值。如下命令可用于评估索引空间的大小。

```
SELECT SUM(INDEX_LENGTH) FROM INFORMATION_SCHEMA.TABLES WHERE ENGINE='MYISAM';
```

或者使用操作系统下的命令du进行统计。

```
$ du -sch `find /path/to/mysql/data/directory/ -name "*.MYI"'
```

如下公式将计算访问Key的命中率： $100 - ((\text{Key_blocks_unused} * \text{key_cache_block_size}) * 100 / \text{key_buffer_size})$ ，但是，该值没有什么实际意义，相对而言，`key_reads`更有实际意义，因此更值得关注，如下：

```
$ mysqladmin extended-status -r -i 10 | grep Key_reads
```

不要把`key_buffer_size`设置为0，至少也应设置为一个较小的值，比如32MB或64MB，因为MySQL的一些内部操作需要用到MyISAM引擎，如临时表。

(21) `Open_tables`

当前打开的表的数量。

(22) `Opened_tables`

已经打开的表的数量。

查看`Open_tables`及`Opened_tables`的增量时，如果`Opened_tables`的增量比较大，那么可能`table_open_cache`（或者`table_cache`）不够用了。如果`Open_tables`对比`table_cache_size`并不大，但`Opened_tables`还在持续增长，那么也可能是显式临时表被不断打开而导致的。

(23) `table_open_cache` (`table_cache` 5.1.3之前的参数名)

默认的设置太小了，生产环境中应该将其设置得足够大，数千到一万是比较合理的值。

检查`Opened_tables status`变量，如果该值比较大，而我们不经常运行`FLUSH TABLES`命令，那么应该增加`table_open_cache`的变量值。

(24) `table_definition_cache`

一般可以将其设置为足够高的值来缓存表定义，比如4096，这并不会耗费什么资源。默认的256太小了。

其他一些反应数据库访问请求、读写数据量的状态变量，这里将不再赘述。

11.3 数据库监控的实现

11.3.1 Nagios

使用Nagios对数据库进行监控的思路与之前讲述的心跳表大同小异，这里将简要介绍下其实现思路。

1) 监控主库的可用性。可以创建一个监控表监控主库的可用性，监控表有时间戳字段。Nagios每分钟更新监控表的数据，以测试主库可用性，如果连续多次失败，就判断为失败，并发送短信邮件报警。

2) 监控复制。定期检测，比如每隔5分钟，就读取从库的监控表最近的时间记录，判断滞后多少秒。

监控账号可以限制资源使用，示例如下。

```
GRANT SELECT,UPDATE ON db.name.* TO monitor user@'10.%' IDENTIFIED BY 'xxxxxxxxxxxxxx' WITH MAX_CONNECTIONS_PER_HOUR 360 MAX_USER_CONNECTIONS 3  
MAX_QUERIES_PER_HOUR 720 MAX_UPDATES_PER_HOUR 360 ;
```

11.3.2 swatch

可以使用swatch监控服务器日志和数据库日志，它可以实时监控日志，从而节省很多编写监控脚本的时间。

下面简单介绍下swatch的安装和使用方式。

swatch需要用到Perl 5.10，如果你的系统是Perl5.8，那么建议将其升级到5.10。

Perl升级到5.10的步骤如下。

```
perl -v  
cd  
mkdir pkgs  
cd pkgs  
wget http://www.cpan.org/src/perl-5.10.1.tar.gz  
tar zxvf perl-5.10.1.tar.gz  
cd perl-5.10.1  
.Configure -des  
make  
make install  
cd /usr/bin;mv perl perl.bak;ln -s /usr/local/bin/perl .
```

通过以下命令安装swatch所需要的模块。

```
cpan> install Date::Calc Date::Format Date::Manip File::Tail
```

swatch的安装方法如下。

下载swatch-3.2.3.tar.gz到本机。

```
tar zxvf swatch-3.2.3.tar.gz  
cd swatch-3.2.3  
perl Makefile.PL  
make  
make install  
cpan > install Proc::ProcessTable
```

这个步骤如果没有安装成功，则make test会失败，也可以手动执行编译安装，示例如下。

```
cd /root/.cpan/build/Proc-ProcessTable-0.45-OJ6Aeg  
perl Makefile.PL  
make  
make install
```

我们可以让swatch随系统启动，或者在root下添加守护，如下。

```
crontab -l  
## monitor log files using Swatch  
*/2 * * * * /root/crontab/sw.sh > /dev/null 2>&1  
cat sw.sh  
#!/bin/bash  
#source /root/.bash_profile  
#For monitoring log files ,looking for trouble.  
#20090311  
source $HOME/.bash_profile  
export PATH=/usr/local/bin:$PATH  
host_name= `hostname`  
exist_count=`ps -ef |grep "swatch" |grep -v grep |wc -l`  
echo "$exist_count"  
if [ "$exist_count" -eq 0 ]; then  
echo "starting swatch"  
#/usr/bin/swatch --config-file=/etc/swatch.conf --tail-file=/var/log/messages &  
swatch --config-file=/etc/swatch.conf \  
--tail-prog=/usr/bin/tail \  
--tail-args '--follow=name --lines=1' \  
--tail-file="/var/log/messages /usr/local/mysql/data/`hostname`.err" \  
--daemon  
echo "started swatch"  
fi
```

swatch并不知道如何处理异常日志，所有的规则都是在日常维护工作中不断积累下来的，以下提供的是笔者曾经使用过的一份配置文件。

swatch.conf

```
##### A simple example Start #####  
# watchfor /authentication failure|other message you want to be alerted/ #可使用正则表达式捕捉日志内的警告错误信息  
  
# threshold track_by="foo",type=limit,count=2,seconds=300 #若在  
300s之内捕捉到了信息，则执行动作  
,但最多只能执行  
2次  
,会忽略  
300s内的相同信息  
  
# threshold track_by="foo",type=threshold,count=2,seconds=300 #若在  
300s之内捕捉到  
2次信息，则执行动作  
,然后重新计时
```

```

# mail addresses=username1@ooea.com:abcd@ooea.com,subject="SSH:\ Invalid\ User\ ",when=1-6:8-17
#执行动作,发送邮件
# 可使用
when选项指定某个时间段才可执行动作.
when=day_of_week:hour_of_day.
# exec "command"
#执行动作,执行命令.

The command may contain variables which are substituted with fields from the matched line.
# perlcode [depth] arbitrary_Perl_code :可嵌入
perl代码.

#####
A simple example End #####
# This is Swatch configuration file. Usage: swatch -c=/etc/swatch.conf -t=/var/log/messages
# Added by garychen on 20070507
perlcode my $hostname='hostname';
watchfor /kernel BUG/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname Error "
###exec ""
watchfor /ERROR/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname Error "
###exec ""
watchfor /InnoDB: Warning/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname MySQL Error "
###exec ""
watchfor /ORA-
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname Oracle Error "
###exec ""
#watchfor /EXT3-fs error/
watchfor /error/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname system Error "
###exec ""
watchfor /Can't connect to localhost/
threshold type=limit,count=1,seconds=900
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname Memcached Error "
#exec ""
#watchfor /(.*PHP Warning.*)
# threshold type=threshold,count=10,seconds=900
#mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname php Error "
# exec "echo $1 >> /root/crontab/log/error_swatch.log"
watchfor /[alert]/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname Nginx Alert"
#exec ""
#watchfor /(.*\[error\].*)/
# threshold type=threshold,count=10,seconds=900
#mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname Nginx Error"
# exec "echo $1 >> /root/crontab/log/error_swatch.log"
watchfor /ip_conntrack: table full/
threshold type=limit,count=1,seconds=60
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname System Error"

#exec "echo $1 >> /root/crontab/log/error_swatch.log"
#exec "/root/crontab/modify_sysctl.sh"
watchfor /ALERT/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname System Error"
#
watchfor /worker process \d* exited on signal 9/
threshold type=limit,count=1,seconds=60
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname Nginx Error "
#exec ""
#
watchfor /messages suppressed/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname Nginx Error "
#exec ""
watchfor /mysql_error()/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname MySQL Error "
#exec ""
watchfor /Failed reading log event/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname MySQL Error "
#exec ""
watchfor /segfault at/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname TT Error "
#exec ""
watchfor /Out of memory/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname System Error "
#exec ""
watchfor /detected inconsistency/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname System Error "
#exec ""
watchfor /response failed/
threshold type=limit,count=1,seconds=300
mail addresses=username1@ooea.com:username2@ooea.com,subject="$hostname System Error "
#exec "

```

11.3.3 Cacti

Cacti等其他开源监控工具一般都提供了MySQL插件，可以通过添加插件方便地对MySQL进行监控和性能信息收集。Cacti的插件可以参考如下链接：

<http://www.percona.com/software/percona-monitoring-plugins>

下面对Cacti输出的一些图形做一些简单的说明。图11-1所示的是InnoDB事务计数图。

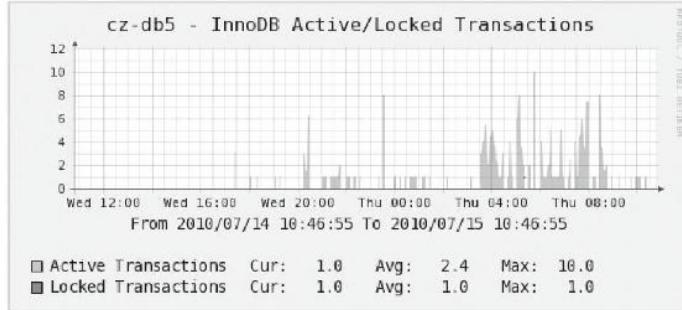


图11-1 InnoDB事务计数图

图11-1显示了事务的计数情况。一个活动的事务是指这个事务当前的状态是打开的，还没有关闭，也就是说，在BEGIN...COMMIT之间；一个正在运行的查询也是一个活动的事务（MySQL默认配置为事务自动提交，所以每个查询都被当作一个单独的事务）；一个锁定的事务是指处于LOCK WAIT状态的事务，通常是在等待一个行锁，但也可能是在等待表锁。

图11-2所示的是InnoDB缓冲池页的变动信息。

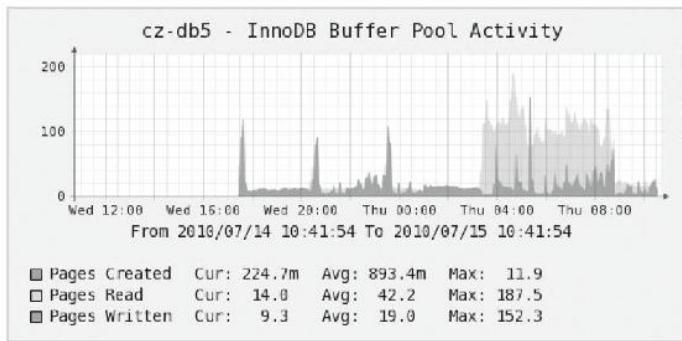


图11-2 InnoDB缓冲池页变动信息

图11-2给出了InnoDB缓冲池中页面的创建、读取和写入的频率，可以作为InnoDB吞吐率的一个指标。如果发现图中有突变，那么应该警惕。

图11-3所示的是InnoDB缓冲池中内存的使用情况。

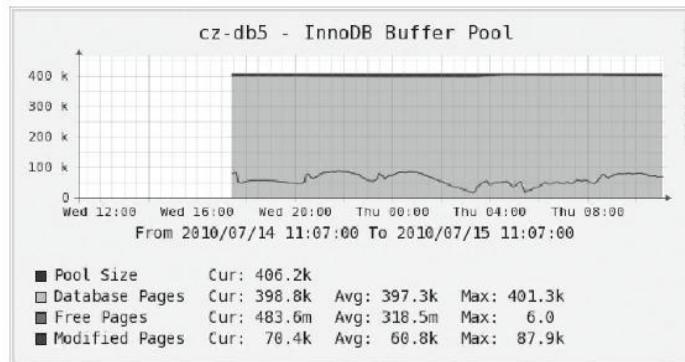


图11-3 InnoDB缓冲池中内存的使用情况图

图11-3显示了InnoDB缓冲池的一些基本信息，各项的含义如下。

·Pool Size: InnoDB Buffer Pool的大小。

·Database Pages: 已经使用的页。

·Free Pages: 自由空间。

·Modified Pages: 脏数据所占用的空间。

图11-4所示的是InnoDB Checkpoint Age信息情况。

图11-4所展示的InnoDB Checkpoint Age，等同于还没有应用检查点操作的数据字节数，如果这个时刻实例发生崩溃，那么恢复时就需要应用图中所示的这么多日志量。如果这个值接近日志文件的合计大小，那么可能你还需要增大日志文件。

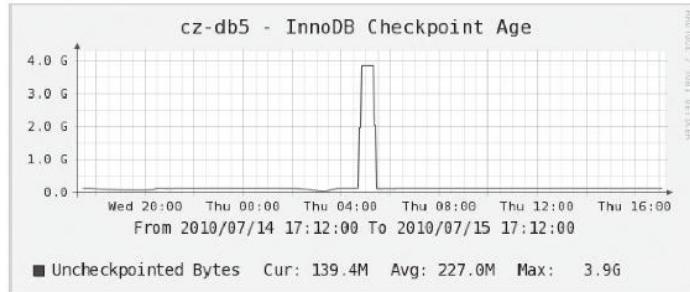


图11-4 InnoDB Checkpoint Age

图11-5是InnoDB I/O信息图。

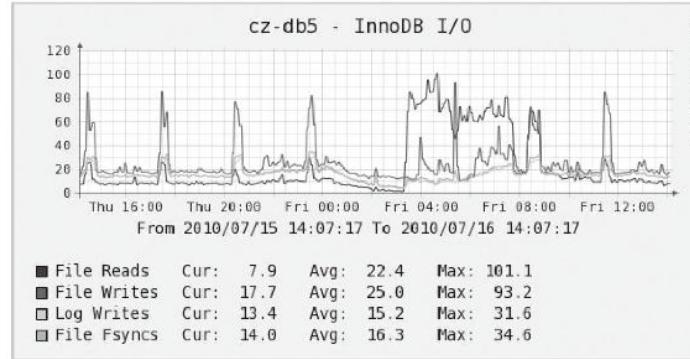


图11-5 InnoDB I/O信息图

图11-5展示了InnoDB I/O的统计情况，包括文件读写、日志写和Fsync()调用。Fsync是一项成本昂贵的操作，如果参数`innodb_flush_log_at_trx_commit`的值设置为1，那么在图11-5中可能会看到很高的File Fsyncs值。

图11-6是InnoDB I/O挂起信息图。

图11-6中应该没有挂起的I/O操作或挂起操作的值很小。如果在图中看到大量的Pending操作，那么我们可能需要更大的缓冲池，或者更快的存储。

图11-7是InnoDB查询修改记录的操作图。

图11-7中显示了InnoDB每秒执行SELECT、INSERT、UPDATE、DELETE操作的行数。从图中可以看到凌晨0点有一个高峰。

图11-8是InnoDB事务图。

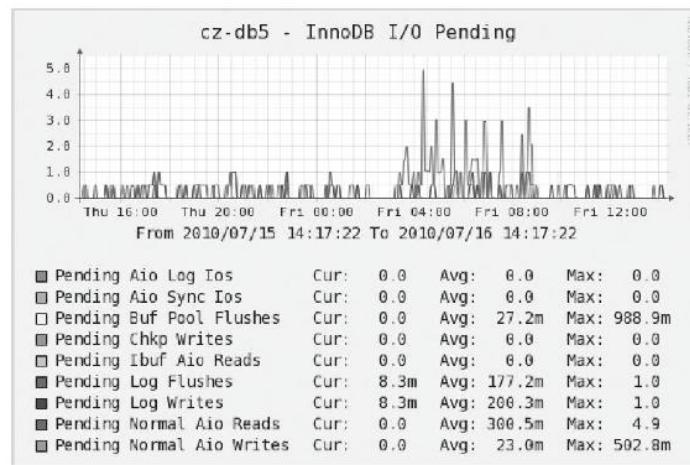


图11-6 InnoDB I/O挂起图

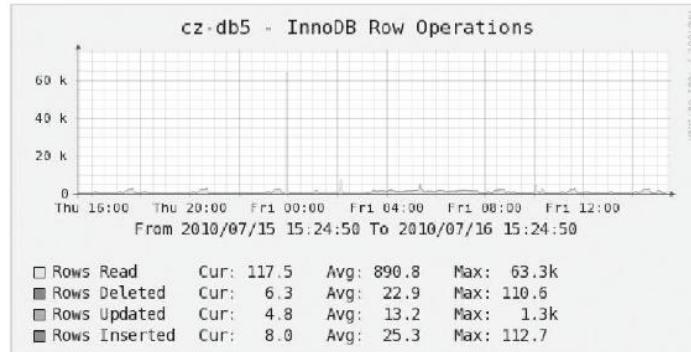


图11-7 InnoDB查询修改记录操作图

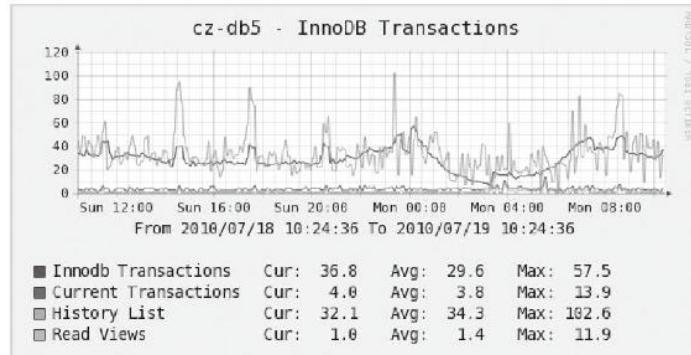


图11-8 InnoDB事务图

图11-8中的参数及其说明如下。

·InnoDB Transactions: 创建的事务。

·Current Transactions: 当前事务，不管处于何种状态，包括active、lock wait、not started等状态。

·History List: 未被清理的事务的列表长度，表征了最旧的事务，这些事务的存在可能是因为清理的速度跟不上事务的创建频率，也可能是因为有长时间运行的查询事务，为了维护一致性读而不能清理旧的行记录版本。

·Read Views: 多少事务有一致性快照。

图11-9是InnoDB连接图。

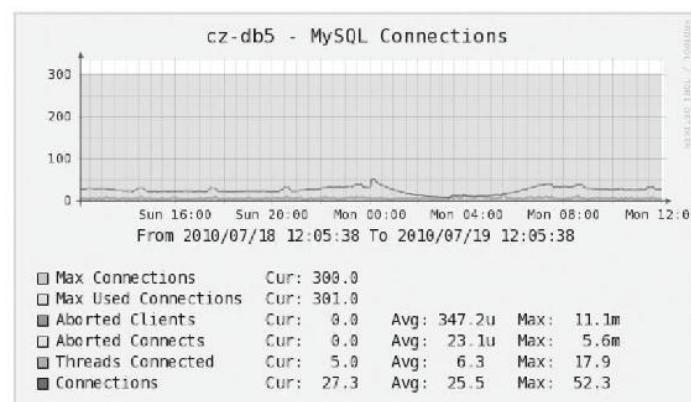


图11-9 InnoDB连接图

应关注图11-9中的Aborted Clients和Connections，Aborted Clients可能意味着连接超时退出或网络问题、账号验证错误、程序异常中断等情况的发生。

图11-10是MySQL句柄计数器信息图。

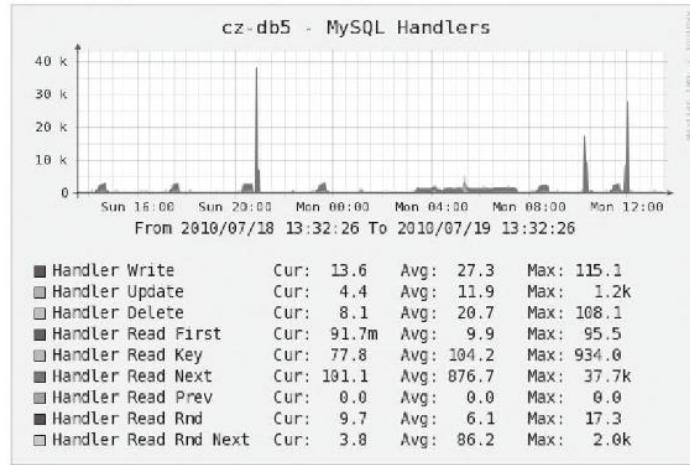


图11-10 MySQL句柄计数器图

各种句柄的计数，图11-10中的Handler Read Next有时会很大，表示可能有索引扫描。Handler Read Next指按照键顺序读取下一行的请求数。如果使用范围约束或执行索引扫描来查询索引列，那么该值会增加。

图11-11是一个网络传输流量统计图。

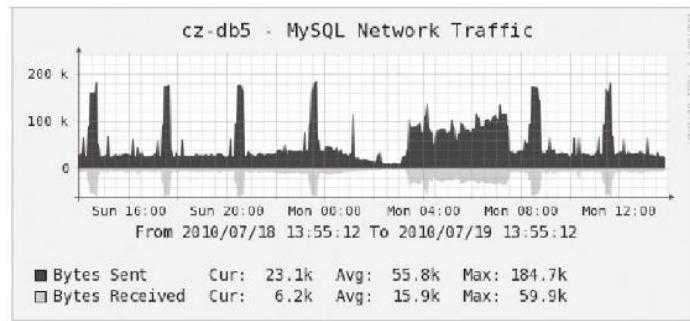


图11-11 网络传输流量统计图

图11-12是MySQL连接状态图。

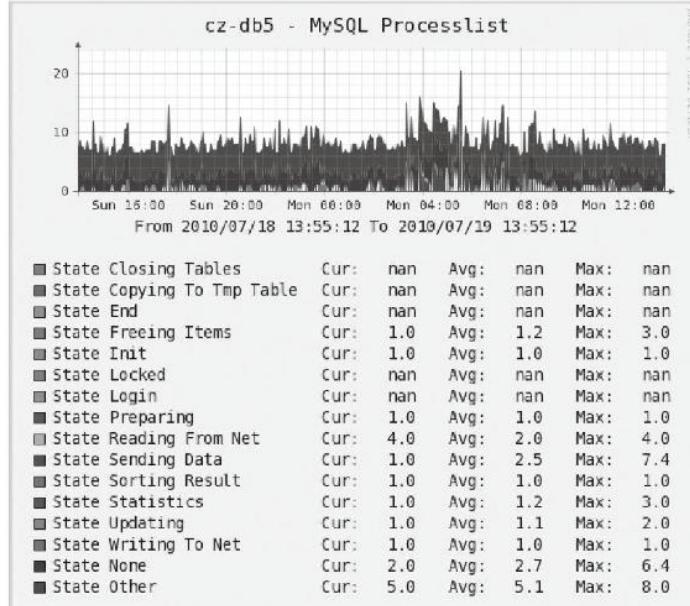


图11-12 MySQL连接状态图

图11-12所示的是MySQL连接各种状态的一个统计。在大部分情况下，你应该能看到很少的State Sending Data，对于图形突变，则需要谨慎探究是何种原因所导致的。

图11-13是不同类型的SELECT查询图。

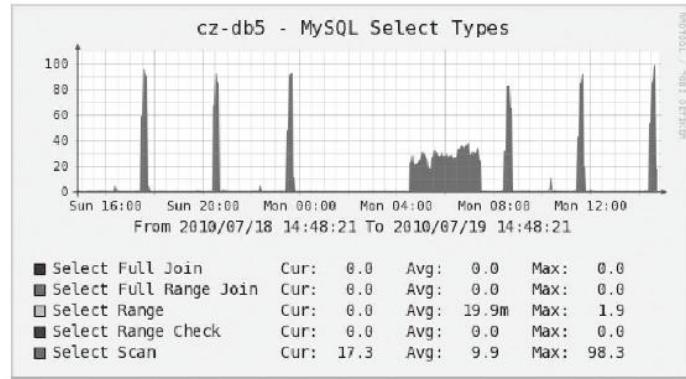


图11-13 SELECT查询图

图11-13显示的是不同类型的SELECT查询，一般情况下Select Full Join必须等于0，需要注意曲线的变化，如果有突变，则需要探明原因。

图11-14是MySQL表锁信息图。

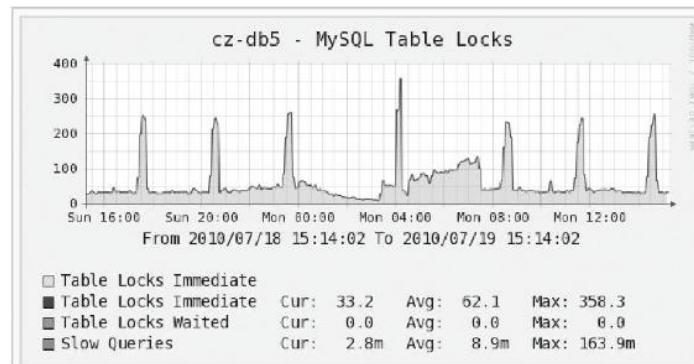


图11-14 MySQL表锁信息图

对于InnoDB来说，一般不用关注MySQL表锁信息图，如果有较高的Table Locks Waited，那么可能是由MyISAM表引起的。

图11-15是MySQL临时对象图。

对于图11-15，需要关注下Created Tmp Disk Tables，该值等于0为佳，如果比较高，比如大于5，则可能有性能问题。

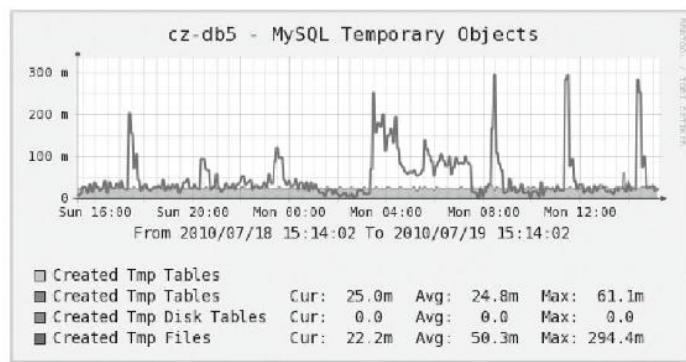


图11-15 MySQL临时对象图

11.3.4 如何打造一个强大的监控系统

官方的企业版监视器，基本涵盖了监控的各种要素，我们可以仿照它开发自己的监控系统。

我们的监控系统可能需要满足如下需求。

(1) 能实现实时查看和历史查看

我们需要满足实时查看性能数据及其可用性的需求，历史数据也要能够存储下来，保留一定的周期，以便在需要对其进行分析的时候，随时能够查询。

(2) 采用分布式的架构

仅仅在远程通过命令行对MySQL的状态进行监控可能还不够，可以看到开源工具的一些插件，选择了集中式的监控方式，即从一台监控机器上探测所有被监控

的主机，这种方式有一些弊端，事实上，主动和被动方式都采用会更灵活、更强大，也能监控到更多的信息。推荐的方式是部署一个分布式的Web应用程序，它由一个集中式的服务管理端和在每台被监控的MySQL服务器上安装的一个轻量级服务代理端组成。

(3) 可提供预警及建议功能

监控系统应能持续监控性能和可用性，在性能趋势偏离基准水平时发出报警，同时还能提供建议的配置和参数设定以改善性能。

(4) 丰富的图表

我们需要能够方便地查看所有的MySQL服务器，能够批量进行配置管理，可直观地查看一台服务器、自定义的组或所有服务器。一组丰富的实时图形和历史图形可帮助我们深入了解详细的服务器统计信息，可帮助我们全面深入地了解数据库性能、可用性、关键活动等信息。

(5) 可视化查询分析

可监视实时查询性能，查看执行统计信息，筛选和定位导致性能下降的SQL代码。MySQL在5.6版之后加强了Performance Schema的功能，所以我们可结合使用Performance Schema和MySQL Server 5.6直接从MySQL服务器收集数据，而无需额外的软件或配置。

(6) 发现并修复占用大量资源的查询

开发人员和DBA可通过相关的图形针对当时执行的查询比较执行参数，如服务器负载、线程统计信息或内存使用情况等。只需选中图形上的一个时间片就能找到最占用资源的查询，并找到可能导致更大性能问题的根源。

(7) InnoDB监视

监视影响MySQL性能的主要InnoDB指标。接收有关索引使用效率低下、锁定问题及InnoDB缓冲池使用情况等的警报，获取根据当前性能和趋势分析改进InnoDB配置的提示和技巧。

(8) 复制监控

可配合心跳表实现复制监控，在生产运维中，推荐采用人工输入主从复制拓扑架构信息的方式，并结合监控自动发现进行监控。因为如果只是自动发现，而我们对复制架构进行了误操作，那么我们自己就不能发现。我们需要了解所有MySQL主服务器和从服务器的性能、可用性和运行状况。特别是对于读写分离的架构，从库的可用性和复制延时，也显得尤其重要。

(9) 磁盘监视

趋势分析和预测可以帮助管理员预测未来的容量需求。可以根据用户定义的阈值（例如“如果磁盘空间12个月后将用尽请通知我”）向操作人员发出预警。

(10) 操作系统监视

直观地实时监视操作系统级别的性能指标，包括平均负载、CPU使用情况、内存使用情况、Swap使用情况、文件系统使用情况及磁盘I/O等。

按照需求分析，我们可以确定自己应该收集哪些信息，该如何设计库表结构，并验证表结构是否能够满足我们的业务需求。如果有大量的数据库服务器需要监控，那么还需要处理好实时收集数据和历史数据查询之间的资源争用问题，归档表、按时间分表、分区表都是可以考虑使用的技术。一旦我们实现了数据收集，就可以在这些标准化的数据之上进行预警、分析、统计等功能的开发，在后面更高级的阶段，还可以提供更丰富的咨询和建议功能，使监控平台变得更加智能。

11.4 数据库监控的可视化

现实工作中，我们不会经常去查看图形，特别是在有了很多图形的时候，更多的情况下，我们会接到报警，这个时候，才会去看图形，从图形上看到负载情况、资源使用情况有了变化，然后再去确定当时发生了什么？有没有做什么变更，从而快速定位问题的所在。

我们在监控数据、性能数据时往往有数据可视化的需求。可通过图形数据看到某种周期性的变动，比如每小时的波动，可能是有某些定时任务；每天的波动，可能和用户集中在某些时间段上网有关；每周的波动，可能是工作日访问请求大，非工作日访问请求少；季度的波动，可能是每个季度要生成一些报表。

监控展示数据所使用的图形一般是二维形式的，有折线图、散点图、热图、条形图及饼图等，三维图形在一些领域也有应用。以下将介绍几种常见的图形。

11.4.1 折线图

折线图（line chart）是用线段将各数据点连接起来的图形，它以折线的方式显示数据的变化趋势。折线图的特点是可以反映事物在一段时间内的趋势，它可以显示随时间（根据常用比例设置）而变化的连续数据，因此非常适合显示在相等时间间隔下数据变化的趋势。另外，在折线图中，数据是递增还是递减、增减的速度、增减的规律（周期性、螺旋性等）、峰值等特征都可以清晰地反映出来。所以，折线图常用来分析数据随时间的变化趋势，也可用来分析多组数据随时间变化的相互作用和相互影响。

折线图是生产环境监控系统中最常使用的图形，图11-16所示的就是一个折线图的案例。

从图11-16中，我们可以看到，在22:06分，查询达到峰值，而在凌晨时间段，查询量就很小。



图11-16 折线图示例

11.4.2 散点图

散点图又名散布图（scatter plot）。它是表示两个变量之间关系的图，又称相关图，是以一个变量为横坐标，另一变量为纵坐标，利用散点（坐标点）的分布形态反映变量统计关系的一种图形。

散点图用于分析两测定值之间的相关关系，它的优点是能通过直观醒目的图形方式反映变量间关系，以便决定用何种数学表达方式来模拟变量之间的关系。散点图不仅可传递变量间关系类型的信息，也能反映变量间关系的明确程度。通过作散点图对数据的相关性进行直观地观察，不但可以得到定性的结论，而且还可以剔除异常数据，从而提高用算法估算的准确性。

图11-17所示的就是一个散点图。

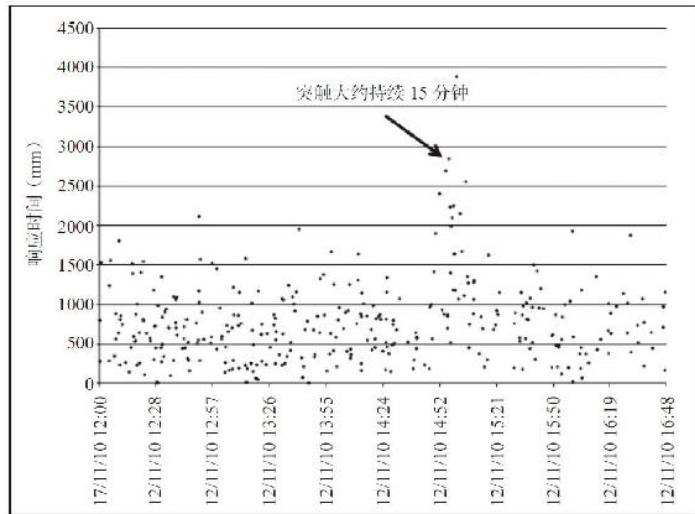


图11-17 Web访问日志散点图

图11-17是一个Web访问日志的散点图，可以看到在下午15点左右有一个响应时间变差的时间窗口。一般情况下，页面的访问响应可分为两类，一类是页面本身的响应就比较慢，它会一直表现得很慢，而另一类是页面在高峰时间段才会响应变慢，这往往意味着系统碰到了某些资源瓶颈。

散点图存在两个不足之外，一是如果点非常密集，那么点会重叠，相互之间很难区分；二是我们可能需要收集、存储和处理大量的数据。

对于大量数据，绘图的成本会较高，对比可以采用对日志进行取样的方式，仅针对取样点数据绘图，绘制的图仍然能够反映实际的响应时间分布，比如，如下的awk脚本，就可以采样1/3的数据。

```
cat access.log | awk '++n; if ((n % 3) == 0) { print $0 }' > access.log_sampled.txt
```

或者采用其他扩展性更好的绘图方式，比如热图。

11.4.3 热图

热图（heat map）的原理是把坐标点分组，每个分组的区域都称为bucket，它的颜色取决于在这个bucket里元素的数量，通过颜色变化的方式来表示坐标点的密集程度变化，解决了散点图坐标点过于密集所带来的问题，我们可用热图来分析Web服务器响应、磁盘访问延迟等指标。

图11-18所示的就是一个磁盘I/O响应时间的热图。

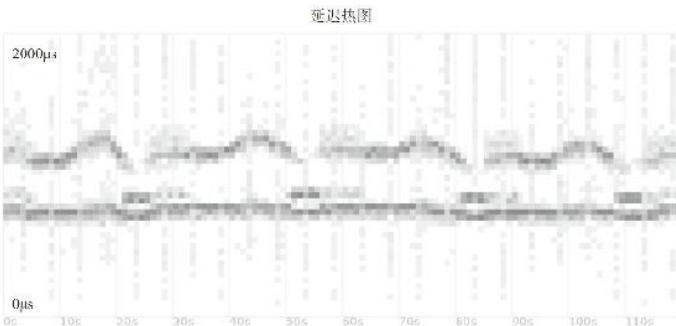


图11-18 磁盘I/O响应时间热图

可以看到大部分的响应时间都在2000μs以下，响应时间主要集中在两个区间，即深红色的两条颜色带。通过使用热图，我们可以描绘大量数据，并且使用渐变的色带来直观地展示数据的疏密程度或频率高低。现实中，热图应用在很多领域中，比如记录用户在Web页面内鼠标的点击位置，记录足球运动员的跑位情况等。

热图的不足之处是，它并不像折线图、条形图、饼图这样知名，很多人不知道有这种表示数据的方法，用户还需要了解其是如何展示数据的。

其他图形就不一一举例了。读者如果有兴趣，可阅读数据可视化的相关书籍。



小结 本章讲述了MySQL应该监控记录哪些信息及如何记录。我们需要具备一个意识，即应该持续监控一切事情，包括网络、系统、服务、用户行为等，基于翔实的数据，我们才能持续优化架构、辅助决策。强大的监控系统不仅仅是为DBA服务的，管理好自己的数据，友好地展现给研发、测试、运营、架构等各个团队，也为他们提供服务才是监控的意义所在。

第12章 MySQL复制

本章将为读者讲述MySQL的复制技术，首先，介绍最基础的主从复制，它是其他所有复制技术的基础，接着再为读者讲述各种复制架构的搭建，最后，列举了一些常见的复制问题及处理方式。复制技术是大部分MySQL高可用技术的基础，熟练掌握各种复制架构有助于制定适合自己公司的高可用方案，第13章将讲述MySQL的迁移、升级、备份和恢复，这些技能同样极大地依赖于对复制架构的理解。

12.1 基础知识

12.1.1 原理及注意事项

MySQL支持单向、异步复制，复制过程中一个服务器充当主服务器，而一个或多个其他服务器充当从服务器。有时我们也称从库为从服务器或从实例，意义上大致是类似的，不需要进行细致的区分。

(1) 复制的基本原理

在主库的二进制日志里记录了对数据库的变更，从库从主库那里获取日志，然后在从库中重放这部分日志，从而实现数据的同步。基本步骤类似如下。

- 1) 主服务器将更新写入二进制日志文件，并维护文件的一个索引以跟踪日志循环。
- 2) 从库复制主库的二进制日志事件到本地的中继日志（relay log）。
- 3) 从库重放中继日志。

将从服务器设置为复制主服务器的数据后，它将连接主服务器并等待更新过程。如果主服务器失败，或者从服务器与主服务器之间失去了连接，那么从服务器将保持定期尝试连接，直到它能够继续侦听更新为止。由--master-connect-retry选项控制重试间隔，默认时间为60s。

如果你想要设置链式复制服务器，那么从服务器本身也可以充当主服务器。

MySQL使用3个线程来执行复制功能，其中1个在主服务器上，另两个在从服务器上。当从服务器发出START SLAVE命令时，从服务器将创建一个I/O线程，以连接主服务器并让它发送记录在其二进制日志中的语句。主服务器可创建一个线程将二进制日志中的内容发送到从服务器中。该线程可以识别为主服务器上SHOW PROCESSLIST输出中的Binlog Dump线程。从服务器I/O线程读取主服务器Binlog Dump线程发送的内容并将该数据复制到从服务器数据目录中的本地文件中，即中继日志。第3个线程是SQL线程，由从服务器创建，用于读取中继日志并执行日志中所包含的更新。

由上可知，这样读取和执行语句将被分成两个独立的任务。每个从服务器都有自己的I/O和SQL线程。即使SQL线程执行得很慢，远远落后于主库，但I/O线程仍然可以从主库上获取所有二进制日志的内容，这样就可以允许主库清空二进制日志了，因为不再需要等待从库来读取二进制日志的内容。

(2) 复制的用途

复制有很多用途，比如跨IDC备份数据，使用读写分离架构扩展读，在从库上进行备份，使用从库测试数据库版本升级，高可用自动故障冗余切换等。生产中最广泛的用途无疑是进行数据备份，在备份过程中主服务器可以继续处理更新，并在主库不能提供服务的情况下接管服务。

(3) 复制的注意事项

·一般情况下，少量的从库，对于主库来说没有什么开销，但是如果部署了很多从库，就需要考虑从库对主库的影响了，网络带宽或I/O可能都会存在瓶颈。

·如果只是传送最新的二进制日志到从库，那么从库一般不会对主库有冲击，但如果由于某种原因，需要读取高并发主库上旧的日志，那就可能会带来严重的性能问题，因为主库要读取大量的旧日志，而这些日志没有被操作系统缓存，因此将导致主库I/O瓶颈，同时还有一个潜在的影响，会阻碍主库事务提交，因为MySQL的XA事务有其特殊性，在事务日志提交之前，需要确保二进制日志已写入。

·复制架构中的从库一般用于扩展读，对于扩展写没有什么用处，复制对于频繁读和少量写的系统好处最大。

回答下面的问题应该能够帮助你确定复制是否在多大程度上能够提高系统的性能。

1) 系统上的读写比例是什么？

2) 如果减少读取操作，一个服务器可以多处理多少写负载？

3) 网络带宽可满足多少从服务器的需求？

·由于目前MySQL5.1、5.5的复制是单线程的，所以复制可能会成为瓶颈，建议使用SSD来突破瓶颈。

·复制的架构和配置应尽量保持简单。

复制有一些限制和坑，但大部分都可以避免，很多会触发问题的高级特性普通用户根本用不着。所以保持自身的数据库配置简单是最好的规避出现复制问题的方法。比如，不要使用环状的复制架构，不要使用Blackhole引擎来实现复制，不要在配置文件内指定复制的过滤。建议生产环境保持简单，所有主从都是完全复制过去，同步所有的数据和权限。保持主从的完全一致，可以减少很多不必要的麻烦。

·建议将从库配置为只读，因为应用程序可能会配置错误，对从库进行写操作，将会导致数据的一致性，甚至丢失数据。

·互为主从的环境，一定要保证同一时刻只写一个数据库。

单向复制是健壮性最强的复制架构，但在实际中，可能会为了方便切换，往往是互为主从的环境。在这种情况下，一定要保证同一时刻只写一个数据库，以防止数据库同时写入相同的键值，导致主键冲突，复制失败。有些人使用双向复制，互为主从的两个库更新不同的表，认为这样可以加速复制，但实际上，双向复制并不能提高什么性能，服务器仍然要做同样的事情，只是锁的竞争更少些，因为源于另一个服务器的更新被序列化了，由于单线程复制，可能还会导致I/O瓶颈问题更突出。

MySQL复制目前不支持主服务器和从服务器之间的任何锁定协议来保证分布式（跨服务器）更新的原子性。这也意味着，在双向复制关系中，不应该同时写入主主配置的两个库，除非你确信任何顺序的更新都是安全的，或者除非你在客户端代码中知道怎样才能避免更新顺序错误。互为主从的复制模式，需要小心处理好自增键及主键的冲突，程序和表的设计应确保不会导致键冲突。由于存在很多约束和风险，所以，现实中的主主复制架构，我们一般采用的是Active-Passive模式而不是Active-Active模式。

·主从架构，如果从库太多，或者同时有很多从库要求传输日志，那么可能会导致主库负载上升。可以解决的方案是再配置一个从库，专门用来传递日志给其他从库。

·对于判断主从是否一致的问题，目前官方并没有一个成熟的解决方案，可以利用第三方的工具pt-table-checksum进行判断。

12.1.2 常用命令

我们可以使用SHOW BINARY LOGS查看当前主库的日志，在从库上执行SHOW SLAVE STATUS\G检查当前的复制状态，在主库上执行SHOW PROCESSLIST显示当前连接过来的从库线程，综合使用如上命令，我们可以大概判断当前的复制情况。

一般配置主从的大致步骤具体如下。

1) 如果当前已经有主从配置了，那么在从库上运行命令STOP SLAVE以停止复制。

2) 在从库上运行CHANGE MASTER命令设定连接主库的信息，配置主从。

3) 在从库上运行START SLAVE命令启动同步。

以下我们开始逐项介绍一些重要的命令。

在主服务器上，SHOW PROCESSLIST的输出看上去应该如下所示。

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
  Id: 2
  User: root
  Host: localhost:32931
    db: NULL
Command: Binlog Dump
  Time: 94
  State: Has sent all binlog to slave; waiting for binlog to
         be updated
  Info: NULL
```

其中，线程2是一个连接从服务器的复制线程。该信息表示所有主要的更新都已经被发送到从服务器上了，主服务器正在等待更多的更新出现。

在从服务器上，SHOW PROCESSLIST的输出看上去应该如下所示。

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
  Id: 10
  User: system user
  Host:
    db: NULL
Command: Connect
  Time: 11
  State: Waiting for master to send event
  Info: NULL
***** 2. row *****
  Id: 11
  User: system user
  Host:
    db: NULL
Command: Connect
  Time: 11
  State: Has read all relay log; waiting for the slave I/O
         thread to update it
  Info: NULL
```

该信息表示线程10是同主服务器通信的I/O线程，线程11是处理保存在中继日志中的更新的SQL线程。SHOW PROCESSLIST运行时，两个线程均是空闲的，都在等待其他更新。

请注意，Time列的值可以显示从服务器比主服务器滞后了多长时间。

1.SHOW MASTER STATUS、SHOW SLAVE STATUS命令解析

(1) SHOW MASTER STATUS

该命令用于提供主服务器二进制日志文件的状态信息，它需要SUPER或REPLICATION CLIENT权限，举例如下。

```
mysql> show master status;
+-----+-----+-----+
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+
| mysql-bin.000360 | 310 |          |          |
+-----+-----+-----+
```

以上命令显示了当前正在写入的二进制文件，以及当前的Position。

(2) SHOW SLAVE STATUS

该命令用于提供有关从库线程的关键参数的信息。如果你使用的是mysql客户端发布此语句，则可以使用一个\G语句终止符来获得更便于阅读的竖向输出版面。

SHOW SLAVE STATUS\G的输出类似如下。

```
mysql> show slave status \G
***** 1. row *****
Slave_IO_State: Waiting for master to send event
  Master_Host: 11.11.11.11
  Master_User: replic_user
  Master_Port: 3306
  Connect_Retry: 60
  Master_Log_File: mysql-bin.001672
  Read_Master_Log_Pos: 315022991
    Relay_Log_File: relay-bin.005035
    Relay_Log_Pos: 315023136
  Relay_Master_Log_File: mysql-bin.001672
    Slave_IO_Running: Yes
    Slave_SQL_Running: Yes
    Replicate_Do_DB:
    Replicate_Ignore_DB:
    Replicate_Do_Table:
    Replicate_Ignore_Table:
    Replicate_Wild_Do_Table:
    Replicate_Wild_Ignore_Table:
      Last_Error:
      Last_Error:
      Skip_Counter: 0
      Exec_Master_Log_Pos: 315022991
      Relay_Log_Space: 315023328
      Until_Condition: None
      Until_Log_File:
      Until_Log_Pos: 0
  Master_SSL_Allowed: No
  Master_SSL_CA_File:
  Master_SSL_CA_Path:
  Master_SSL_Cert:
  Master_SSL_Cipher:
  Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
  Last_IO_Errorno: 0
  Last_IO_Error:
```

```
Last_SQL_Errno: 0
Last_SQL_Error:
1 row in set (0.00 sec)
```

其中各参数及说明如下。

·**Master_Host**: 当前的主服务器主机。

·**Master_User**: 被用于连接主服务器的当前用户。

·**Master_Port**: 当前的主服务器接口。

·**Connect_Retry**: --master-connect-retry选项的当前值。

·**Master_Log_File**: I/O线程当前正在读取的主服务器二进制日志文件的名称。

·**Read_Master_Log_Pos**: 在当前的主服务器二进制日志中, I/O线程已经读取的位置。

·**Relay_Log_File**: SQL线程当前正在读取和执行的中继日志文件的名称。

·**Relay_Log_Pos**: 在当前的中继日志中, SQL线程已经读取和执行的位置。

·**Relay_Master_Log_File**: 由SQL线程执行的包含多个近期事件的主服务器二进制日志文件的名称。

·**Slave_IO_Running**: I/O线程是否被启动并成功地连接到主服务器上。

·**Slave_SQL_Running**: SQL线程是否被启动。

以上Slave_IO_Running和Slave_SQL_Running在正常情况下应该均为Yes。

·**Replicate_Do_DB**、**Replicate_Ignore_DB**:

使用--replicate-do-db和--replicate-ignore-db选项指定的数据库清单。

Replicate_Do_Table、**Replicate_Ignore_Table**、**Replicate_Wild_Do_Table**、**Replicate_Wild_Ignore_Table**:

使用--replicate-do-table、--replicate-ignore-table、--replicate-wild-do-table和--replicate-wild-ignore-table选项指定的表清单。

Last_Error、**Last_Error**: 多数最近被执行的查询返回的错误数量和错误消息。错误数量为0并且消息为空字符串, 则意味着“没有错误”。如果Last_Error值不是空值, 它也会在从库的错误日志中作为消息被显示。

·**Skip_Counter**: 最近被使用的用于SQL_SLAVE_SKIP_COUNTER的值。

·**Exec_Master_Log_Pos**: 来自主服务器的二进制日志的、由SQL线程执行的、上一个时间的位置 (Relay_Master_Log_File)。主服务器的二进制日志中的 (Relay_Master_Log_File, Exec_Master_Log_Pos) 对应于中继日志中的 (Relay_Log_File, Relay_Log_Pos)。

·**Relay_Log_Space**: 所有原有的中继日志结合起来的总大小。

·**Until_Condition**、**Until_Log_File**、**Until_Log_Pos**: 在START SLAVE语句的UNTIL子句中指定的值。

·**Seconds_Behind_Master**: 是从库“落后”多少的一个指示。一般是基于同一集群内网的主从集群, 此值应为0。本字段用于测量从库SQL线程和从库I/O线程之间的时间差距, 单位以秒计。

如果主服务器和从库之间的网络连接较快, 则从库的I/O线程会非常接近主服务器, 所以本字段能够十分近似地指示从库SQL线程比主服务器落后多少。如果网络较慢, 则这种指示不准确; 从库SQL线程经常能赶上读取速度较慢的从库I/O线程, 因此, Seconds_Behind_Master的值经常显示为0, 即使从库I/O线程落后于主服务器时也是如此。换句话说, 本列只对速度快的网络有用。

由于根据SHOW SLAVE STATUSG的输出估算具体的主从差异时间可能会不准, 异常情况下Seconds_Behind_Master的值为NULL, 或者显示不正常, 所以生产环境的实际监控一般是在主从中配置一个心跳表, 通过此心跳表来监控主从之间的时间差异。

2.CHANGE MASTER命令

这个命令在从库中执行, 可以配置所要连接的主库, 以及从哪里开始同步。常用的语法如下。

```
CHANGE MASTER TO
MASTER_HOST='11.11.11.11',
MASTER_PORT=port,
MASTER_USER='replic_user',
MASTER_PASSWORD='your_password',
MASTER_LOG_FILE='log file name',
MASTER_LOG_POS=position;
```

我们可以在正在运行中的数据库从库中动态修改连接主库的信息。例如修改复制用户的密码。

```
mysql> STOP SLAVE; -- if replication was running  
mysql> CHANGE MASTER TO MASTER_PASSWORD='new3cret';  
mysql> START SLAVE; -- if you want to restart replication
```

没有必要指定未发生改变的参数（主机、接口、用户等）。

·MASTER_HOST和MASTER_PORT指定了主库的IP和PORT。

·MASTER_LOG_FILE和MASTER_LOG_POS指定了主库的二进制日志的名称和位置。

·MASTER_USER和MASTER_PASSWORD指定了复制用户的账号和密码，将使用这个账号去连接主库，所以主库需要给予这个账号REPLICATION SLAVE的权限来复制数据。

·CHANGE MASTER会删除所有的中继日志文件并启动一个新的日志，除非您指定了RELAY_LOG_FILE或RELAY_LOG_POS。在此情况下，中继日志将被保持。

·CHANGE MASTER TO会去更新master.info和relay-log.info文件的内容。

3.START SLAVE和STOP SLAVE命令

我们常用的START SLAVE语句有3种。

(1) START SLAVE不带任何参数

不含选项的START SLAVE会同时启动两个从库线程。I/O线程从主服务器中读取查询，并把它们存储到中继日志中。SQL线程读取中继日志并执行查询。START SLAVE要求SUPER权限。

如果START SLAVE成功地启动了从库线程，则会返回，不会出现错误。但是，即使在此情况下，也有可能会出现这样的现象——服务器线程启动了，然后又停止了（例如，因为它们没有成功地连接到主服务器上，或者没有能够读取二进制日志，或者出现了其他问题）。START SLAVE对此不会发出警告。必须检查从库的错误日志，查看是否有由从库线程产生的错误消息，或者使用SHOW SLAVE STATUS检查它们运行是否正常。对于这种情况，我们在编写脚本的时候一定要留意。

(2) START SALVE启动单个服务器线程

```
START SLAVE IO_THREAD  
START SLAVE SQL_THREAD
```

(3) START SALVE指定到某个位置自动终止

可以添加一个UNTIL子句，指定从库应启动并运行，直到SQL线程达到主服务器二进制日志中的一个给定点为止。当SQL线程达到此点时，它会停止。如果在该语句中指定了SQL_THREAD选项，则它只会启动SQL线程。否则，它会同时启动两个从库线程。一般情况下，我们仅操控SQL线程。

```
START SLAVE [SQL_THREAD] UNTIL  
MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos
```

如下的一个例子，我们使用START SLAVE SQL_THREAD UNTIL命令，把从库指向新的主库，新主库需要打开log_slave_updates。

假设有A、B、C、D4个实例，A是主库，B、C、D是从库。现在需要调整架构为C、D是B的从库。我们可以按照如下的步骤进行调整。

1) 分别关闭B、C、D从库的SLAVE。

```
STOP SLAVE;
```

2) 查看主库A上的日志位置信息命令如下。

```
show master status;  
+-----+-----+-----+  
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB |  
+-----+-----+-----+  
| mysql-bin.000003 | 307827324 | | |
```

3) 在所有从库B、C、D上运行命令。

```
START SLAVE SQL_THREAD UNTIL MASTER_LOG_FILE = 'mysql-bin.000003', MASTER_LOG_POS = 307827324;
```

等同步到指定的位置时，自动断开SQL线程，此时B、C、D上面的数据应该是一致的。都同步到了A库的某个日志点。

在B库上运行SHOW MASTER STATUS记录下日志的位置信息log file name、log file position。

在C、D从库运行如下命令，指向新的主库B。

```
mysql > STOP SLAVE;
mysql > RESET SLAVE;
mysql >
CHANGE MASTER TO
MASTER_HOST='11.11.11.11',
MASTER_PORT=3306,
MASTER_USER='replic_user',
MASTER_PASSWORD='password',
MASTER_LOG_FILE='log file name',
MASTER_LOG_POS=log_file_positiion;
```

最后，在B库上运行START SLAVE命令，这样就可以调整A为主库，B为A的从库，C、D是B的从库。

主库可以运行如下命令显示从库。

```
SHOW SLAVE HOSTS;
mysql> SHOW SLAVE HOSTS;
```

MySQL 5.1版本需要在从库上配置好report_host、report_port，并需要重启从库，才能看到从库，这个过程比较繁琐，但新的5.5版本已经更友善了，可以直接看到连接到主库的从库信息，命令如下。

```
mysql> show slave hosts;
+-----+-----+-----+
| Server_id | Host | Port | Master_id |
+-----+-----+-----+
| 13584 | | 3307 | 13582 |
| 132110737 | | 3307 | 13582 |
+-----+-----+
```

12.1.3 参数设置

1.slave_exec_mode

复制冲突解决和错误检测可采用如下两种模式。

·STRICT默认。

·IDEMPOTENT忽略duplicate-key、no-key-found错误，一般在主主配置、环形复制等其他特殊情况下才使用，不推荐使用。

2.max_allowed_packet

默认的设置太小了，生产环境中建议配置大于16MB。如果太小了，可能会导致从库不能接收主库发过来的包，主从建议设置成一样的值。如果你有大的BLOB字段，可能还需要增加这个阈值。

3.请不要使用过滤选项

请不要使用复制过滤参数，除非你真的明确知道你在做什么。即使要使用，也建议只在从库上进行设置。

不要使用binlog-do-db、binlog-ignore-db这两个参数，如果在主库上设置了复制相关的过滤参数，它们可能会导致你不能进行时间点恢复，还可能导致你丢失数据。如果你只是因为I/O瓶颈，希望减少一些日志的写入，而临时禁用部分日志的写入，那么，建议你升级你的硬件。binlog-do-db、binlog-ignore-db这两个参数的含义具体如下。

(1) binlog-do-db=db_name

告诉主服务器，如果当前的数据库（即USE选定的数据库）是db_name，应将更新记录到二进制日志中。其他所有没有明确指定的数据库都将被忽略。如果使用了该选项，你应该确保只对当前的数据库进行更新。一个不能按照期望来执行的例子：如果用binlog-do-db=sales启动服务器，并且执行“USE prices; UPDATE sales.january SET amount=amount+1000; ”，那么该语句将不会被写入二进制日志。

(2) binlog-ignore-db=db_name

告诉主服务器，如果当前的数据库（即USE选定的数据库）是db_name，那么不应将更新保存到二进制日志中。一个不能按照期望来执行的例子：如果用binlog-ignore-db=sales启动服务器，并且执行“USE prices; UPDATE sales.january SET amount=amount+1000”，那么该语句将写入二进制日志。

建议不要使用replicate-do-db、replicate-ignore-db参数，这两个参数是在从库上进行设置的，这两个参数类似上面的两个参数，其实并不符合我们的预期。

*_do_db和*_ignore_db参数其实都仅仅只是针对当前的数据库，也就是说，如果我们USE到指定的库，然后执行了一条更新其他库的SQL，那么这些参数将都不起作用。

下面让我们详细解析下在从库上设置过滤的一些参数，我们来看下`--replicate-ignore-db=db_name`参数。

(1) `--replicate-ignore-db=db_name`

这个选项告诉从服务器不要复制默认数据库（由USE来选择）为`db_name`的语句。要想忽略多个数据库，则应多次使用该选项，且每个数据库使用一次。如果正在进行跨数据库更新并且不想复制这些更新，那就不要使用该选项。

如果使用`--replicate-ignore-db=sales`启动从服务器，并且在主服务器上执行下面的语句，那么UPDATE语句是不会复制的。

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

如果需要跨数据库更新，则应在从库上使用`--replicate-wild-ignore-table=db_name.%`。

下面我们再来看其他两个参数。

(2) `--replicate-ignore-table=db_name.tbl_name`

它将告诉从服务器线程不要复制更新指定表的任何语句（即使该语句可能更新其他的表Y。要想忽略多个表，则应多次使用该选项，且每个表使用一次。同`--replicate-ignore-db`对比，该选项可以跨数据库进行更新。

所以，如果我们需要在从库上忽略一些表的复制，或者使用`--replicate-ignore-table`参数，或者使用如下的`--replicate-wild-ignore-table=db_name.tbl_name`参数，它可以使用通配符进行匹配，功能也更强大。

(3) `--replicate-wild-ignore-table`

它告诉从服务器线程不要复制匹配给出的通配符模式的语句。要想忽略多个表，则应多次使用该选项，且每个表使用一次。该选项可以跨数据库进行更新。

例如：`--replicate-wild-ignore-table=foo% bar%`表示不复制数据库名以`foo`开始和表名以`bar`开始的表的更新。

如果表名模式为%，则可匹配任何表名，选项也适合数据库级语句（CREATE DATABASE、DROP DATABASE和ALTER DATABASE）。例如，使用`--replicate-wild-ignore-table=foo%.%`时，如果数据库名匹配模式`foo%`，则不复制数据库级语句。

有时我们会选择忽略复制`mysql`库来限制权限，不让主库的权限复制到从库，但这可能会带来很多后续问题，比如存储过程、events的权限问题等。



提示 强烈建议不要设置复制过滤选项。如果你一定要使用复制过滤，那么建议采用`replicate-wild-*`选项，它在绝大部分场合更适用。

如果我们需要临时禁用复制特性，那么我们还可以在会话级设置变量`SET sql_log_bin=0`，使当前的一些操作不被复制到从库。

4.`slave_compressed_protocol`

请慎重对待跨集群复制。跨集群配置的时候，可启用`slave_compressed_protocol=1`压缩传输数据，需要在主库进行压缩，在从库解压缩，由于压缩需要额外的CPU消耗，所以需要留意CPU资源是否充裕。

5.`read-only`

可以考虑为从库配置`read-only`选项，以保障数据安全，要注意SUPER权限的用户仍然可以写数据库。

6.`slave_net_timeout`

由于生产环境网络异常，可能会导致复制异常。即使`SHOW SLAVE STATUS`的输出正常，但此时可能已经停止复制了，`slave_net_timeout`的默认设置是1小时，因此很难避免因网络问题导致的复制异常中断，特别是跨IDC的复制，建议将其设置小于1分钟。

7.`--slave-skip-errors`

通常情况下，当出现错误时复制会停止，这个选项可以给你一个机会手动解决数据中的不一致性问题。当语句返回`slave-skip-errors`所列的错误时，该选项将会告诉从服务器SQL线程继续复制。

如果你不知道为什么会发生复制错误，那么请不要使用该选项。如果复制设置和客户程序中没有Bug，并且MySQL自身也没有Bug，那么应该是不会发生停止复制的错误的。滥用该选项会使从服务器与主服务器不能保持同步，将会导致数据不一致。

对于错误的代码，你应使用从服务器错误日志中错误消息提供的代码和`SHOW SLAVE STATUS`输出的错误代码。

如下设置将忽略1062和1053错误。

```
--slave-skip-errors=1062,1053
```

错误代码的具体释义请参考官方文档，这里不再赘述。

也可以（不建议）设置all值忽略所有的错误消息，例如：

```
--slave-skip-errors=all
```

8.skip-slave-start

`skip-slave-start`可以在命令行下或配置文件中使用，目的是在MySQL启动的时候不要启动Slave，这在某些故障情况下很有用，比如宕机后，有时我们希望先观察下情况，再启动Slave。或者有时我们希望能够进行手动调整，自己控制启动Slave的时刻，比如配置延时的从库。或者，有时我们使用`START SLAVE THREA UNTIL`命令，让从库成段地处理已复制的查询，使用`--skip-slave-start`选项来启动从库，可以防止当从库启动时，SQL线程开始运行。最好在一个选项文件中使用此选项，而不是在命令行中使用，这样，即使发生了意料之外的服务器重新启动，它也不会被忘记。

12.1.4 配置文件

默认情况下，中继日志使用`host_name-relay-bin.nnnnnn`形式的文件名，其中`host_name`是从服务器主机名，`nnnnnn`是序列号。用连续的序列号来创建连续的中继日志文件，从`000001`开始。从服务器跟踪索引文件中目前正在使用的中继日志。中继日志索引文件名默认为`host_name-relay-bin.index`。默认情况下，可在从服务器的数据目录中创建这些文件。可以用`--relay-log`和`--relay-log-index`服务器选项覆盖默认文件名。强烈建议指定默认文件名，即日志文件名不要有主机名前缀，文件名中不要带有主机名是为了方便迁移操作和故障处理。

中继日志与二进制日志的格式相同，并且可以用`mysqlbinlog`读取。SQL线程执行完中继日志中的所有事件并且不再需要中继日志之后，会立即自动删除它。没有直接删除中继日志的机制，因为SQL线程可以负责完成。

从服务器在数据目录中会另外创建两个小文件。这些状态文件的默认名为`master.info`和`relay-log.info`。它们包含了`SHOW SLAVE STATUS`语句的输出所显示的信息。

状态文件保存在硬盘上，因此从服务器关闭时不会丢失状态文件。下次启动从服务器时，读取这些文件以确定它已经从主服务器读取了多少二进制日志，以及处理自己的中继日志的程度。

由I/O线程更新`master.info`文件。`master.info`文件中的行和`SHOW SLAVE STATUS`显示的列的对应关系如表12-1所示。

表12-1 `master.info`文件中的行和`SHOW SLAVE STATUS`显示的列的对应关系

行	描述	行	描述
1	文件中的行号	8	Connect Retry
2	Master Log File	9	Master SSL Allowed
3	Read_Master_Log_Pos	10	Master_SSL_CA_File
4	Master_Host	11	Master_SSL_CA_Path
5	Master_User	12	Master_SSL_Cert
6	索引（不向 <code>SHOW SLAVE STATUS</code> 显示）	13	Master_SSL_Cipher
7	Master_Port	14	Master_SSL_Key

由SQL线程更新`relay-log.info`文件。`relay-log.info`文件中的行和`SHOW SLAVE STATUS`显示的列的对应关系如表12-2所示。

图12-2 `relay-log.info`文件中的行和`SHOW SLAVE STATUS`显示的列的对应关系

行	描述
1	Relay_Log_File
2	Relay_Log_Pos
3	Relay_Master_Log_File
4	Exec_Master_Log_Pos

当备份从服务器的数据时，你还应备份这两个小文件及中继日志文件。它们可用来在恢复从服务器的数据后继续进行复制。如果丢失了中继日志但仍然有`relay-log.info`文件，那么你可以通过检查该文件来确定SQL线程已经执行的主服务器中二进制日志的程度，例如如下命令。

```
cat relay-log.info
/usr/local/mysql/log//relay-bin.015161
401289501
mysql-bin.006800
401289356
```

如上信息表示，当前从库正好执行到主库日志文件mysql-bin.006800，执行的位置是401289356。

然后我们可以用Master_Log_File和Master_LOG_POS选项执行CHANGE MASTER命令来告诉从服务器需要重新从该点读取二进制日志。当然，要求二进制日志仍然在主服务器上，例如如下命令。

```
CHANGE MASTER TO  
MASTER_HOST='11.11.11.11',  
MASTER_PORT=3306,  
MASTER_USER='replic_user',  
MASTER_PASSWORD='your_password',  
MASTER_LOG_FILE='mysql-bin.006800',  
MASTER_LOG_POS=401289356;
```

MySQL 5.1的中继日志和relay-log.info、master.info文件并不是crash-safe的，也就是说，它们默认是不会实时刷新到磁盘的，那么在发生崩溃灾难的情况下，文件记录的信息可能是错误的，将会导致复制异常。如果使用的是MySQL 5.5，那么可以设置如下选项：

```
sync_master_info = 1  
sync_relay_log = 1  
sync_relay_log_info = 1
```

注意设置如上的参数将会带来很多开销。对于高并发写操作很频繁的业务，建议不要设置如上参数，否则将会严重影响性能。安全和效率往往不能兼得。

MySQL 5.5还有一个选项relay_log_space_limit，这个选项设置了所有中继日志可以使用的空间。意思是如果中继日志占用的空间超过了这个变量设置的阈值，那么I/O线程就会关闭，等待SQL线程应用日志释放空间。

12.1.5 复制模式

1.概述

MySQL可以使用如下3种复制模式。

- 1) 基于SQL语句的复制（statement-based replication）
- 2) 基于行的复制（row-based replication）
- 3) 混合模式复制（mixed-based replication）

对应地，我们可以设置3种类型的二进制日志格式，使用参数--binlog-format=type进行设置。

type的值可以是如下的值。

- 1) STATEMENT：基于语句的日志。
- 2) ROW：基于行记录的日志。
- 3) MIXED：混合日志模式，即默认是基于语句的日志，当需要的时候，将会使用基于行的日志。

MySQL在日志模式的选择上不同的版本默认值可能会不一样，建议在生产环境中使用MIXED，即混合模式的日志，一般情况下，它可以工作得很好。

我们可以在运行时动态修改日志格式，命令如下。

```
mysql> SET GLOBAL binlog_format = 'STATEMENT';  
mysql> SET SESSION binlog_format = 'STATEMENT';
```

在复制环境中，从库可以进行自动调整，以适应主库的row-base语句。

下面将详细介绍各种复制模式及其优缺点。

2.基于SQL语句的复制

基于SQL语句的复制（statement-based replication），也就是逻辑复制，MySQL 3.23开始支持。

基于语句的复制，复制将执行主库上所执行的语句，也就是说，在从库上执行的语句和在主库上执行的语句是一样的。

基于SQL语句的复制，其优点具体如下。

- 1) 相对于基于行记录的日志，它更简单，也更容易实现。
- 2) 数据库的二进制日志更小，因此，主从库之间传输的日志数据也更小。

3) 二进制日志的可读性更好，我们可以使用mysqlbinlog方便地读取二进制日志。

4) 更有利于排查问题，从库上执行的是和主库一样的语句。

基于SQL语句的复制，其缺点具体如下。

1) 有些操作将无法正确复制到从库，因为对于主库的操作，并不仅仅取决于SQL文本，还有一些不确定性的因素。不确定性的因素有如下之点。

·带LIMIT子句但没有使用ORDER BY的操作。

·修改数据的查询语句里用到了返回不确定性值的自定义函数和存储过程。

·一些函数在主从上执行的结果不一样，如UUID()、SYSDATE()、RAND()、VERSION().....

还有很多，这里就不一一列举了。

2) 从库需要锁住更多的记录，比如INSERT...SELECT...操作会需要锁定比基于行记录的复制多得多的记录，比如UPDATE一个表，如果没有索引，就会锁住整个表。

3) 复杂的、代价昂贵的语句需要在从库上再次执行，也就是运行整个语句，这样可能会比较慢，而基于行记录的复制，只需要修改指定的记录即可，不需要执行整条语句。

4) 对于非核心特性的功能支持力度有限，存储过程和触发器相关的复制Bug较多。

3. 基于行的复制

MySQL 5.1开始支持基于行的复制（row-based replication），它的适用范围更广泛，也可靠得多。基于行记录（row-based）的复制，其格式比较难以阅读，即使MySQL官方一直在改进其可读性。基于语句的复制很难处理各种高级特性，如视图、存储过程、触发器。如果你需要应用各种高级特性，那么推荐你使用基于行的复制模式。

基于行的复制优点具体如下。

1) 所有改变均被复制，对比基于语句的复制，这是一种更安全、更精确的复制。

2) 更少的锁定记录。

3) 对于存储器、触发器、自定义函数的特性也完善支持。

4) 二进制日志更有利于进行数据恢复，因为二进制日志里记录了数据的详细变更信息。

5) 更容易发现数据的不一致。比如主库中更改了1笔记录，而从库中不存在这笔记录，那么基于行记录的复制会报错而基于语句的复制则不会报错。

基于行的复制缺点具体如下。

1) 产生更多的二进制日志数据。

2) 二进制日志不易阅读，不方便使用mysqlbinlog解读日志。

3) 要求主从表结构一致，这样就限制了它的灵活性，因为生产环境有时需要临时修改从库的表结构，提升从库为主库。

4. 混合日志模式

笔者个人的建议是使用混合模式，即binlog_format=mixed，默认是使用基于语句的复制，但一旦MySQL检测到满足了一定的条件，那么它就会自动切换到基于行的复制。例如，在函数内使用了UUID()，更新了有自增列的表且调用了触发器或存储过程等情况，将会自动切换到基于行的复制。

12.1.6 复制兼容性

MySQL已经有了多个版本，有时我们需要在各个版本之间进行复制。如下是一些复制兼容性的注意事项。

·建议从库的版本高于主库，绝大部分情况下MySQL都支持此类复制。但也可能会碰到有些主库的语法，反而不支持更高版本的从库。

·从库建议使用主版本的最新版。

·主从之间不要跨越两个大版本号，那样可能会出问题，比如从MySQL4.1复制到MySQL5.1，为了降低风险，可以考虑在中间插入一个MySQL5.0版本的从库以避免一些意外问题的发生。

- 高版本到低版本的复制可能是可行的，但官方不保证支持。
- 升级生产环境之前，建议配置一个更高版本的从库运行一段时间，以验证复制功能。
- 即使主从复制数据正常，也不代表稳定性良好，在性能压力、网络异常的情况下，仍然可能会导致复制异常。生产环境大版本有差异的复制架构不要长期并存，应该尽量调整到一致，以免后续碰到其他问题。

12.2 配置主从复制

对于未上线的主机，即在主库没有任何写入的情况下，可以采用如下方式配置主从。

- 1) 在主从主机上部署好MySQL，并在主库上启用二进制日志，注意主从server-id必须不一样，server-id的设置可以使用IP的后8位加上端口（port）等其他标识信息，主库的配置文件类似如下。

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

- 2) 记录主库的日志文件名File和日志文件Position，命令如下。

```
mysql> show master status;
+-----+-----+-----+-----+
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000362 | 310 |           |               |
+-----+-----+-----+-----+
```

- 3) 在主库中创建复制账号，允许从库来访问，命令如下。

```
grant replication slave,replication client on *.* to replic_user identified by 'xxxxxxxxxxxx';
```

如果账户仅用于复制，那么replication slave的权限就足够了，但在本地查看从库（slave server）信息，还需要replication client权限。

- 4) 从库编辑配置文件，运行命令，配置主从。

编辑从库的配置文件。

下面将展示一个从库的配置文件示例：只有server-id必须设置，其他选项是可选的，具体命令如下。

```
log_bin = mysql-bin
server_id = 2
relay_log = /path_to_mysql_log/mysql-relay-bin
log_slave_updates = 1
read_only = 1
```

其中的参数及其说明如下。

·log_bin=mysql-bin: 建议主从配置一样的名字，不然在以后的配置中，处理问题会复杂很多。

·log_slave_updates=1: log_slave_updates决定了是否将从主库接收的更新写入从库自身的二进制日志里。将这个值设置为1，是方便以后以将这个从库提升为主库后，根据需要再配置一个从库，也方便数据恢复。

我们可以设想如下的场景。

如果主从复制的架构，主库提供服务，从库每天凌晨备份。

如果你的生产环境从库log_slave_updates是关闭的。

那么主库宕机后不能启动，你需要把数据库流量切换到从库，此时你需要在新的主库的基础上再制作一个从库。但是，请注意，你不能用从库的备份转储文件（dump文件）来做从库，因为凌晨备份的文件从凌晨到主库宕机这个时间段的日志并没有写入从库的日志，如果你使用这个转储文件，将会丢失很多数据，那么你需要在线重新导出一份数据来制作从库。如果你设置了log_slave_updates，那么从库的日志里就包含了所有时刻的数据更改，你就可以使用从库凌晨的备份文件在其他机器上直接制作从库了。

当然，设置这个变量也有弊端。如更大的I/O写入，不容易发现错误等。

设置了log_slave_updates可能不易发现错误，比如应用程序误写从库时，我们不能及时发现，因为我们可能会以为这是正常的更新。为了安全，我们需要在从库上设置read_only选项，设置了read_only=1之后，将只有SUPER权限的用户才可以修改数据。

在从库上执行如下语句，其中MASTER_LOG_FILE和MASTER_LOG_POS是第二个步骤记录的值。

```
mysql >
CHANGE MASTER TO
MASTER_HOST='11.11.11.11',
MASTER_PORT=3306,
MASTER_USER='replic_user',
MASTER_PASSWORD='xxxxxxxxxxxx',
MASTER_LOG_FILE='mysql-bin.000362',
MASTER_LOG_POS=310;
```



注意 千万不要使用在配置文件里指定master_host、master_port的方式，这些配置只在第一次启动MySQL时才生效。

5) 在从库上执行如下命令启动slave。

```
mysql > start slave
```

6) 在从库上确认复制正常。

```
mysql> SHOW SLAVE STATUS \G
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Seconds_Behind_Master 0
-----
```

前两项应该都是Yes。Seconds_Behind_Master应该不是NULL。

12.3 配置主主复制

配置为主主复制，需要解决的主要问题是自增键/主键冲突。

当将多个服务器配置为复制主服务器时，如果要使用自增列（AUTO_INCREMENT），那么应采取特殊的步骤以防止键值冲突，否则插入行时多个主服务器会试图使用相同的自增列值。

服务器变量auto_increment_increment和auto_increment_offset可以帮助协调多主服务器复制和自增列。

其中，auto_increment_increment用于控制自增列值增加的间隔。auto_increment_offset用于确定自增列值的起点。

假设有两台主机A、B，它们互为主从，那么配置可以如下。

A主机：

```
auto_increment_increment=3
auto_increment_offset=1
```

B主机：

```
auto_increment_increment=3
auto_increment_offset=2
```

我们还需要注意，除了自增字段不能互相冲突之外，所有表的键值也不能互相冲突，同一时刻的操作需要保证不会插入相同的键值。

还要留意复制的时序问题，一定要确保任一时刻只写一个库，主主复制更多的是为了故障冗余而不是为了能够多点写入。一般配置为Active-Standby，而不是Active-Active。

一般而言，配置为主主复制会导致维护更加复杂，可能还会带来隐患，需要更完善的监控措施和自动化手段。配置主主复制的步骤这里不再赘述，对每个库分别执行配置主从复制的步骤即可。

12.4 配置级联复制、环形复制

(1) 配置级联复制

假如需要配置成A→B→C→D→E这样的形式，箭头表示复制到，那么可按如下步骤进行。

1) 首先打开各实例的log_slave_update选项，首尾两个实例也可以不用打开。

2) 确保各主机的server-id不同。

3) 配置每一对主从，A→B，B→C，C→D，D→E。

注意节点越多，健壮性越差，建议不要超过4~5个节点。

(2) 环形复制

有一个现象需要留意：如果E又复制到A，就会成为环形复制，可以实现多点写入，此时也需要和“配置主主复制”一样关注键值冲突等问题。环形复制存在一个问题，如果某个节点被摘下，那么这个节点的写入事件将会在环内永远循环。因为只有最开始发起事件的节点才能过滤这类事件，所以摘下节点之前，应该确保已停止对其写入。

MySQL 5.6实现了GTID，这点大大提高了链式复制的健壮性。有兴趣的同学可以参考<http://dev.mysql.com/doc/refman/5.6/en/replication-gtids-concepts.html>。

12.5 跨IDC复制

跨IDC复制架构的部署与单机房部署链式复制（级联复制）的从库并没有区别，但由于网络的不稳定，可能会导致复制的不稳定，维护代价较高，而且可能需要外网IP才能进行复制，降低了安全性。但现实中，这种架构也有人使用，相对于使用应用程序实现的数据同步，数据库在某种程度上成本更低，也更容易确保数据的一致性。

下面将简述一些跨IDC进行复制的注意事项。

- 跨IDC的复制，建议还是采用普通的主从架构，而不要采用链式的复制架构，简单的主从架构更稳健。
- 尽量只在中心主库进行写入，其他机房只用于读，这样既可以简化架构，也可以避免多点写入带来的维护一致性的难题。如果是M-M的架构，也应该将一个机房作为备用（Standby），仅作容灾。
- 数据量较大的时候，网络可能会成为瓶颈，建议使用混合日志的复制模式。可在从库中设置`slave_compressed_protocol=1`压缩传输数据，此选项可进行动态设置。
- 由于跨IDC的主从复制，重新搭建代价比较大，在明确知道数据库出现何种错误时，可以忽略此错误，可使用“`slave-skip-errors=error_code1,error_code2...|all`”，但不要滥用，否则容易导致主从不一致而不自知。
- 由于跨IDC的复制，网络可能会不稳定，应用程序应该处理网络延时对用户体验的影响。

12.6 多主复制

关于多主复制，MySQL目前可以实现的思路和方法如下。

- 1) 使用一些开源的工具，如tungsten-replicator。
- 2) 自己写脚本对不同的主库进行轮询，获取日志，要跟踪每个主库的位置，此种方式比较复杂。
- 3) MySQL 5.7开始支持多主。可参考<https://www.percona.com/blog/2013/10/02/mysql-5-7-multi-source-replication/>。

12.7 延时复制

MySQL同步在快速的网络中是毫秒级的，如果有误操作，从库也会马上变更，对于一些频繁进行，而没有经过严格测试的升级，可能会带来风险。

可考虑配置一个延迟复制的副本，以改善故障情况下的可恢复性。

MySQL 5.6已经可以支持延迟复制，如果是5.1版本，可以用Percona公司出品的一个工具pt-slave-delay来实现延时复制。

下载地址为[wget percona.com/get/pt-slave-delay](http://www.percona.com/get/pt-slave-delay)

安装步骤此处省略。

语法格式为

```
pt-slave-delay [OPTION...] SLAVE-HOST [MASTER-HOST]
```

选项值一般可以用默认的，默认是延迟1小时。

如下是一个设置延时的例子。

```
pt-slave-delay u=xxxx,S=/tmp/mysql.sock,p=password --delay 1m --interval 15s --run-time 10m --log /path/to/delay.log -daemonize
```

以上命令表示后台运行这个工具10分钟（默认是永久运行的），从库保持一直滞后主库1分钟，间隔15秒每检查一次，那么理论上是延迟了1分钟15秒。

延时复制的原理为检查主库的日志到了哪里了（可以用SHOW SLAVE STATUS命令查看中继日志），对比已经应用的日志，就知道延迟的时间了。每隔1分钟检查一次（默认），不断启动、关闭replication SQL thread来保持主从一直延时固定的时间。

如果正在运行这个工具，那么按Ctrl+C退出后，它是友好地退出的，意思是它会启动复制SQL线程。

12.8 半同步复制

MySQL 5.5开始支持半同步复制（semi-sync replication），半同步复制提供了更好的灾难恢复性。

半同步的原理是，主库和它的从库都启用半同步特性，当一个从库连接主库时要标识自己是否支持半同步，如果主库启用了半同步，且拥有至少一个半同步从库，那么一个事务提交会阻塞直到确认至少一个半同步从库已经“接收到事务事件（event）”为止，否则会发生一个“超时”。

半同步从库在写入事件到中继日志（relay log）时，刷新到磁盘后才确认“接收到事务事件”。

如果发生一个“超时”，即没有任何一个半同步从库确认“接收到事务事件”，那么主库将自动切换到异步复制模式。

当至少一个半同步从库追赶上主库，主库又会自动切换到半同步模式。

这里的“半同步”，可以按如下这样理解。

对于传统的异步同步，主库写事务事件到二进制日志里，从库索取主库日志，这还不能确保事务事件被传送到从库。而对于全同步复制（fully synchronous replication），主库提交事务，必须等待从库也成功提交这个事务，才能完成这个事务，这样容易造成事务的延迟。所以，出现了半同步，半同步是介于异步和全同步之间的同步。

需要留意到是，半同步对于网络的要求很高，它仅适用于高速内网。虽然MySQL 5.5的半同步表现不佳，但是，据MySQL官方文档称，在新的5.7版本中，它已经得到了改善。

12.9 在线搭建从库

我们有多种方式可以在数据库提供服务的时候搭建从库，而不影响线上数据库或对其影响很小。在线搭建从库一般可分为两类，一种是在操作系统下做快照，另一种是利用自带的备份工具mysqldump制作备份。如下记录的是主从配置的一些常规步骤，一些基本的设置，比如参数的设置，这里将不再赘述，如果不加以说明，那么我们备份的库都是InnoDB引擎的表。

12.9.1 操作系统下对打包文件配置主从

1.已经有一主一从，增加一个从库

如果我们已经有了主从库，那么另外再搭建一个从库会比较简单，大概的步骤如下所示。

1) 关闭从库。

2) 打包相关文件到另外一台主机。

包括数据文件，如ibdata*、InnoDB事务日志文件ib_logfile*、master.info文件、relay-log.info文件和my.cnf配置文件。

3) 在新的数据库主机上配置相应的参数，注意server-id不要和其他数据库实例相同。

4) 一般来说，master.info的信息和relay-log.info的信息是一致的，你可以直接删除relay-log.info文件，重新启动，新的从库会按照master.info里的信息重新同步数据库。

5) 一些情况下即使正常关闭了数据库，也可能存在信息不一致的情况，relay-log.info里记录了当前应用到数据库主库的二进制日志的位置，这个值不同于master.info里记录的当前读取到的主库日志的位置，这种情况下，我们可以删除master.info文件，然后重新启动数据库实例，并按照relay-log.info里记录的信息，运行CHANGE MASTER命令重新同步主库的数据。

2.仅有主库，增加一个从库

如果我们只有一个主库，这个时候，我们希望制作从库，而主库正在提供服务，我们还希望对主库的影响最小，那么我们可以采用如下的方式制作从库，注意，这种方式仅适合MyISAM引擎的表。对InnoDB数据库不要使用这种方式制作从库。

1) 主库赋予从库访问权限。

```
mysql> GRANT REPLICATION SLAVE ON *.* TO 'replic'@'xxx' IDENTIFIED BY 'xxxxxxxxx';
```

2) 主库施加全局读锁，禁止更新和提交数据，并记录当前主库二进制日志的位置信息。

```
FLUSH TABLES WITH READ LOCK;
SHOW MASTER STATUS;
```

3) 另外打开一个会话，打包文件，一般情况下我们需要打包数据文件`ibdata*`和日志文件`ib_log*`，比如使用`tar`命令进行打包。

```
tar cvf data.tar *
```

4) 打包所有需要的文件后，在主库上进行解锁。

```
UNLOCK TABLES;
```

5) 将打包后的文件传递到远程主机，并删除多余的文件，InnoDB的数据文件`ibdata*`和日志文件`ib_log*`需要保留，配置文件`my.cnf`可能也需要保留，如下命令将远程传输文件。

```
scp data.tar mysql@11.11.11.11:/home/mysql/
```

6) 启动从库，使用`change`命令配置要从哪个主库进行同步，并启动`slave`。

```
CHANGE MASTER TO
MASTER_HOST='11.11.11.11',
MASTER_PORT=3306,
MASTER_USER='replic_user',
MASTER_PASSWORD='yourpassword',
MASTER_LOG_FILE='mysql-bin.000084',
MASTER_LOG_POS=521259880;
show slave status \G
start slave;

show slave status \G
```

以上的`MASTER_LOG_FILE`和`MASTER_LOG_POS`就是第2)个步骤记录的主库二进制日志的位置信息。

注意`FLUSH TABLES WITH READ LOCK`这条语句必须等待其他查询语句的完成，所以可能会耗时很长才执行完这条语句，甚至超时退出。

以上制作从库的方式，对于都是MyISAM引擎表的数据库比较适用，但InnoDB的后台进程即使是施加了全局锁，在`tar`打包文件的过程中，也仍然会去写InnoDB的数据文件，可能最终的数据文件并不能被用作从库，或者不能启动，所以，请不要对InnoDB数据库使用这种方式制作从库。

如果不使用`tar`命令，而是使用操作系统下的快照技术，那么对于InnoDB引擎的数据库采用如上的方式也可以制作从库，我没有去验证过，但理论上是可行的。因为我们在施加全局锁的时候，可以获取数据库文件的一个快照，但要注意，所有的文件都应该放在同一个分区里，这种方式下获得的快照所有文件都将是一致的。

12.9.2 利用mysqldump制作从库

1. 在主库上执行mysqldump制作从库

如果我们希望在主库上导出所有数据，制作从库，那么我们可以在主库上运行`mysqldump`命令先制作一个备份文件。`mysqldump`的命令类似如下。

```
mysqldump --flush-logs
--master-data=2 --single-transaction --hex-blob -R -f
--all-databases > /path/to/dir/databases.sql
```

我们通常称`mysqldump`导出的数据文件为转储文件或`dump`文件，`master-data=2`会生成被注释掉的`CHANGE MASTER TO`语句，存储在转储文件里，我们可以利用这个信息来创建从库。`master-data`的默认值是1，会生成自动执行的语句，由于我们一般不希望自动执行，所以我们将该值设置为2。

`single-transaction`参数表示制作一个一致性的备份集，对于InnoDB，制作一致性备份集的时候不会锁表，仍然可以读写数据，这点对于在线备份很重要。对于MyISAM表，仍然会需要锁表。`mysqldump --flush-logs --master-data=2 --single-transaction --hex-blob -R -f --all-databases`这个命令，它只是在一开始的瞬间会请求锁表，所以它对系统的影响很小。

对于`--master-data=2`，如果不加`single-transaction`参数，那么它在自动启用`--lock-all-tables`备份的过程中会锁表。

实际创建数据库从库的步骤具体如下。

1) 部署好从库实例，此时数据为空。

2) 在主库上执行`mysqldump`命令，将数据导出为SQL文件。

3) 在从库上导入此SQL文件。

4) 配置主从。

根据转储文件（SQL文件）的CHANGE MASTER语句提供的信息，可以生成相应的CHANGE MASTER命令，并在从库中执行。

我们也可以使用另外一种方法，在主库上运行命令“FLUSH TABLES WITH READ LOCK;”然后使用mysqldump导出数据，利用此转储文件来制作从库，这种方式不仅对MyISAM有效，对于InnoDB也有效，具体步骤如下。

1) 在主库上运行命令锁表、备份。

```
FLUSH TABLES WITH READ LOCK;
SHOW MASTER STATUS;
```

记录SHOW MASTER STATUS命令的输出结果。

此时，不要关闭如上的会话连接。

另外再开一个会话，运行mysqldump命令备份。

```
mysqldump -uroot -p --all-databases > /a/path/mysqldump.sql
```

mysqldump命令执行完毕之后，在主库上原来的会话连接里，运行如下的命令解锁。

```
UNLOCK TABLES;
```

2) 向从库导入dump文件。

如果之前启动了SLAVE，则关闭SLAVE。

```
STOP SLAVE;
```

运行如下命令导入数据。

```
mysql -uroot -p < mysqldump.sql
```

运行如下命令，修改要连接的主库信息。

```
RESET SLAVE;
CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000001',
MASTER_LOG_POS=98;
START SLAVE;
```

以上MASTER_LOG_FILE和MASTER_LOG_POS的信息就是上述第1)个步骤中运行SHOW MASTER STATUS记录的值。

3) 运行命令“SHOW SLAVE STATUS;”检查复制状态，以下两项的值应该都是YES。

```
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
```

2. 在从库上执行mysqldump制作从库

如果不能关闭从库，那么我们一般采取关闭Slave的SQL线程，然后导出数据的方式制作从库。

原理：关闭从库的复制SQL线程后，从库将不再被更新，这个时候，可以认为我们获得了一个一致性的快照。关于这个快照和原来主库的主从关系，我们可以运行SHOW SLAVE STATUS\G命令来查看。具体步骤如下。

1) 关闭Slave的SQL线程，并获取SHOW SLAVE STATUS的信息。

```
mysql> STOP SLAVE SQL THREAD;
mysql> SHOW SLAVE STATUS;
```

2) 对于SHOW SLAVE STATUS命令的输出，我们关注的是如下两项。

```
Relay_Master_Log_File
Exec_Master_Log_Pos
```

新的从库应该从主库的如上位置开始重新同步。

3) 导出数据库，命令如下。

```
shell> mysqldump -
```

```
master-data=2  
all-databases > dumpfile
```

4) 重新启动Slave，命令如下。

```
mysql> START SLAVE;
```

5) 向新的slave机器导入数据，命令如下。

```
shell> mysql < dumpfile
```

6) 对新的slave实例，运行如下命令恢复同步，`file_name`、`file_pos`就是我们第2) 个步骤中记录的`Relay_Master_Log_File`和`Exec_Master_Log_Pos`的值。

```
mysql> CHANGE MASTER TO  
MASTER_LOG_FILE = 'file_name', MASTER_LOG_POS = file_pos;
```

12.10 配置日志服务器

如果配置主从，从库需要读取的是比较旧的日志，而这些日志没有被主服务器操作系统缓存，必须从磁盘中进行读取，那么可能会导致大量的磁盘读写，从而严重影响数据库繁忙的生产系统，可以考虑的办法是不要直接从主库中进行日志传递，而是专门搭建一个日志服务器。

配置完日志服务器之后，把从库索取的日志文件放到日志服务器中，然后再从这个日志服务器进行同步。

关于日志服务器的配置，可参考<http://mysqlsandbox.net>，我们也可以使用日志服务器来进行时间点恢复，利用复制来进行时间点恢复而不是用`mysqlbinlog`来进行，注意原因如下。

- 复制是被久经考验的，而`mysqlbinlog`则不是那么可靠。

- 复制的速度更快。

- 复制可以更方便地查看进度。

- 复制有更多控制，更容易处理复制错误，也可以过滤事件。

配置日志服务器进行时间点恢复的步骤如下。

1) 部署安装。

下载MySQL二进制官方安装包。

```
wget mysql-5.1.58-linux-x86_64-glibc23.tar.gz
```

下载sandbox，创建安装用户。

```
wget https://launchpad.net/mysql-sandbox/mysql-sandbox-3/mysql-sandbox-3/+download/mysql-Sandbox-3.0.28.tar.gz  
useradd sandbox  
usermod -G mysql sandbox
```

解压到目录`/home/sandbox/pkgs/mysql-Sandbox-3.0.28/`，进行安装。注意不要将解压的二进制包的目录删除了。

```
# as normal user  
export PATH=$HOME/usr/local/mysql/bin:$PATH  
##export PERL5LIB=$HOME/usr/local/lib/perl5/site_perl/5.8.8  
perl Makefile.PL PREFIX=$HOME/local/sandbox  
make  
make test  
make install
```

将`/home/sandbox/local/sandbox/bin`路径添加到PATH变量。

执行如下命令安装MySQL实例。

```
make_sandbox /home/sandbox/pkgs/mysql-5.1.58-linux-x86_64-glibc23.tar.gz
```

安装成功后，`/home/sandbox/sandboxes/msb_5_1_58`下同时生成了很多便于管理的脚本，如`start`、`stop`、`use`等。

配置字符集，并添加日志临时目录（默认生成的实例是没有日志的）。

```
./stop
```

修改配置文件。

```
[client]
default-character-set = utf8
[mysqld]
character-set-server = utf8
default-storage-engine=innodb
./start
./use
> status
```

2) 以下命令将查找最近的二进制文件，并且将日志传递到sandbox主机，准备测试。

```
find ./ -type f -name "mysql-bin.*" -newer mysql-bin.025050 |xargs ls -lrt > /tmp/all_files.txt
scp -P 9922 -p `cat /tmp/all_files.txt` sandbox@11.11.11.11:/home/sandbox/mysqllog_tmp
```

3) 配置日志服务器。

通过以上步骤，日志目录放在/home/mysql/mysqllog_tmp处，我们可运行如下命令生成日志索引文件mysql-bin.index。

```
cd /home/sandbox/mysqllog_tmp
ls -l /home/sandbox/mysqllog_tmp/mysql-bin.[0-9]* > mysql-bin.index
./stop
```

修改配置文件my.sandbox.cnf，添加如下配置项。

```
log_bin = /home/sandbox/mysqllog_tmp/mysql-bin
log_bin_index = /home/sandbox/mysqllog_tmp/mysql-bin.index
./start
```

启动成功。

```
./use
> show binary logs
```

4) 配置复制用户。

```
./use --user=root
> GRANT REPLICATION SLAVE ON *.* TO 'rsandbox'@'10.%' IDENTIFIED BY 'rsandbox';
```

5) 配置主从同步，进行时间点恢复，恢复到指定时间。

基本步骤如下。

向从库导入一份历史备份，配置主从同步，然后应用日志服务器的日志，我们可以设置同步到某个时间点。可使用如下命令，同步到某个指定的位置。

```
STARTSLAVE [SQL_THREAD] UNTILMASTER_LOG_FILE='log_name',MASTER_LOG_POS=log_pos
```

12.11 常见的复制问题及处理方法

以下将叙述一些常见的复制故障处理。有些复制故障是因为从库被误操作而导致的，此时可能要修复数据或重做从库，此类故障的处理，这里就不做叙述了。

12.11.1 跳过复制错误

在明确知道数据库出现了何种错误时，可以忽略此错误，但不要滥用，跳过错误的命令如下。

```
STOP SLAVE;
SET GLOBAL sql_slave_skip_counter = 1;
STARTSLAVE;
```

12.11.2 临时表和复制

必须保证干净地关闭从库，否则复制可能会出错。因为在复制的时候，从库的临时表也在进行同步，如果关闭了从库，再重启的时候，就没有临时表了，那么那些对临时表的更新就不能被复制过来，从而就会复制出错。所以需要干净地关闭临时表，在没有临时表的时候，干净地关闭数据库。

如下步骤可避免复制出错。

1) 运行“STOP SLAVE SQL_THREAD;”命令。

2) 运行SHOW STATUS检查Slave_open_temp_tables的值，具体命令如下。

```
SHOW STATUS like '%slave%';
```

3) 如果Slave_open_temp_tables不等于0, 那么运行START SLAVE SQL_THREAD, 然后重复以上步骤。

4) 如果Slave_open_temp_tables=0, 那么, 那么可以关闭数据库的实例了。

或者可以用另外一种方式, 将临时表的前缀都配置为某个名称(如norep), 然后用选项--replicate-wild-ignore-table=norep%将这些表配置为不复制。

临时表的复制问题主要是出在基于语句的复制模式, 如果使用row-based复制, 那么临时表是不会被复制的, 如果你希望一劳永逸, 可以考虑使用基于行的复制模式。

12.11.3 内存表和复制

如果使用的是内存引擎的表, 那么从库重启也可能会导致复制中断。

1) 如果主库重启, 那么内存表将会是空的, 会写入一个“DELETE”语句到二进制日志, 通知从库清空数据, 这种情况下一般不会有复制问题。

2) 如果从库重启, 那么内存表是空的, 这会导致和主库的数据不一致, 主库复制过来的操作将无法正常运行, 复制将会失败。基于行的复制可能会出现错误“Can't find record in'memory_table'”。

此种情况下的复制中断没有太好的解决方案。如果可能, 我们可以用InnoDB来代替内存表。如果你实在需要使用内存表的话, 可以考虑设置IDEMPOTENT(这个选项对所有其他表都生效, 因此需要谨慎使用), 或者忽略报错的错误号, 比如忽略1032错误, 或者还可以在从库中忽略要复制的内存表。

另外, 基于语句的复制, 比如INSERT INTO...SELECT FROM memory_table可能向主从库中插入不一样的数据。

如果设置了内存表的大小, SET global max_heap_table_size value, 那么这个变更不会被复制到从库的, 你需要确保主从都做了变更。

虽然主从的内存表的数据可能会不一致, 但是如果应用程序逻辑可以确保内存表只是用作缓存, 那么一般是不会有太大的问题。

12.11.4 主库宕机重新启动成功, 但复制关系中断

主库、从库宕机都可能导致复制关系中断。

一般情况下, 生产环境中出现的复制故障主要是主库宕机后, 从库找不到同步的主库日志位置信息, 需要手动处理。

主库二进制日志并不是实时刷新的, 主库宕机后, 部分日志丢失了, 但是更多的日志已经被传送到了从库, 结果从库的数据比主库的还要新, 这种情况下往往会导致主键冲突。我们需要进行临时设置让从库忽略主键冲突的错误。

待主从之间的复制稳定之后, 建议立即取消忽略的错误号, 仍然执行严格的复制检查。

12.11.5 主库宕机重启不成功

如果主库宕机重启不成功, 则需要选择其中一台从库做主库, 具体步骤如下。

1) 通过master_log_file、read_master_log_pos可以判断哪个从库是最新的。

2) 确认该从库已经应用了所有日志, 提升该从库为主库。

3) 其他从库自行检查自己最新的日志, 判断是哪个时间点中断了, 是哪条SQL, 然后通过这条SQL去“新主库”查找具体的位置点(position), 并从这个点开始同步。

由于检索相关SQL比较耗时, 对于高并发的业务, 可能会难以定位到具体的位置点, 如果可以接受部分数据误差, 那么我们也可以直接选择故障时刻的大致日志位置点或凭经验决定从哪里开始同步。如果确实难以定位, 但对于数据的一致性要求又很高, 那么我们可以对新的主库重新制作从库, 以保证数据的准确性。

12.11.6 多个从库的server-id相同

如果主库的一些从库存server-id相同的情况, 那么从库的MySQL错误日志里将会有大量的重连和断开的错误, 但它不会明确告知我们server-id有重复。

配置不同的server-id即可解决此问题。

12.11.7 锁定导致的复制延时

如果是基于语句的复制，那么从库上的操作中锁定可能会比较多，从而影响复制的速度，比如INSERT...SELECT这类语句会锁住所有数据。由于从库上的操作是逐个顺序执行的，因此长时间的查询，可能会导致大的延时。解决方法具体如下。

- 1) 分割查询，尽早释放资源。
- 2) 可以考虑采用SELECT INTO OUTFILE，然后LOAD DATA INFILE的方式。

12.11.8 对MyISAM引擎的表恢复数据

如果某个MyISAM表的数据有问题，需要恢复到某个时间点的数据，那么我们可以采用如下的便捷方法进行恢复。

- 1) 主库LOCK TABLE table_name READ。
- 2) 主库FLUSH TABLES。
- 3) 复制备份的MyISAM数据文件，覆盖掉主库和从库中的这个表。
- 4) 主库解锁表UNLOCK TABLES。

以上方法在理论上是可行的，主要是为了确保在将备份文件复制到主库上的时候，主从复制也是正常的。有时我们在误操作表之后，希望能够恢复数据，这时就可以采用这样的办法了。

12.11.9 如何彻底清除Slave设置

MySQL 5.1并不能干净清除复制信息，比如，我们在本机执行如下命令。

```
STOP SLAVE;
RESET SLAVE;
```

以上命令将清除master.info和relay-log.info。

但我们仍然可以在如下命令中看到一些残留信息。

```
SHOW SLAVE STATUS \G;
```

残留信息输出如下。

```
mysql> SHOW SLAVE STATUS \G;
***** l. row *****
Slave_IO_State:
Master_Host: appxxx
Master_User: test
Master_Port: 3306
Connect_Retry: 60.....
```

理论上清除废旧的文件，然后重启MySQL即可，那么有没有更好的不需要重启的办法呢？可以试试下面这个办法。

```
CHANGE MASTER TO MASTER_HOST=""
```

这个时候虽然“SHOW SLAVE STATUS\G;”没有任何Slave相关信息的输出了，但是重新生成了master.info和relay-log.info文件，这样很不友好。

然后我们再运行“RESET SLAVE”；命令，这次就可以清除所有信息了。此时可手动清除残留的*relay*文件。

12.11.10 网络异常导致的复制延时

MySQL在网络异常的时候，也可能会延时很久，然而我们并不知道，虽然SHOW SLAVE STATUS\G输出里的Seconds_Behind_Master显示为0，但可能已经延时很久了。建议把slave_net_timeout参数设置得小一些，比如小于1分钟。



小结 本章介绍了MySQL的复制技术，所有其他复制架构的基础都是主从复制，我们应该熟练掌握主从复制，熟悉复制相关的各种文件。复制故障是生产环境中比较常见的问题，特别是有大批量机器的时候，为了减少复制问题的发生，我们应保证复制架构的简单，尽量少用MySQL的各种高级特性，比如内存表、临时表、视图、存储过程、触发器等。

本章将为读者讲述数据库的各种维护任务：迁移、升级、备份和恢复。因为每个人熟悉的工具不同，其对应的迁移、升级、备份和恢复的方式也都略有不同，本书将尽量对笔者认为最具普遍性的一些操作进行讲述。另外还整理出了一些注意事项，DBA需要有缜密的思维，要考虑到可能出现的各种情况，并能够冷静地处理异常情况。

13.1 升级

MySQL的升级主要有两类，一种是对数据库表结构或数据的变更，另一种是数据库版本的升级。

13.1.1 升级表结构或变更数据

可以直接在命令行下键入SQL语句进行升级，或者执行SQL文本文件，为了避免乱码和各种字符集的转换，建议文本文件的字符集与MySQL服务器字符集，以及数据库连接的设置都保持一致，生产环境中建议都统一为utf8。

生产环境中，一般是通过SQL脚本进行升级，以下将详述这种升级方式，大致步骤如下。

1) 首先确认要升级的数据库信息：数据库IP、端口、数据库名，是否有多个分库需要升级等。

2) 检查升级的脚本。

如果脚本中存在非英文字符，则需要确保是utf8无BOM格式的文件，检查语法是否正确，是否有异常符号，比如全角符号、Windows换行符等。

对于重大的操作，比如删除数据库，如果有疑问，请事先和研发、测试人员进行确认。

3) 评估升级对生产的影响，以及升级所耗费的时间。如果可能会影响到生产，则应该和研发、测试、产品、运营等工作人员沟通协调升级的方式，是否停服，是否需要另选时间，比如在凌晨负荷低峰时期执行。在操作过程中，应该尽量做到不影响生产负荷，一些操作可能导致服务可用性下降，比如在大表上修改表结构。一些操作大量数据的语句，比如INSERT INTO SELECT、CREATE TABLE AS SELECT语句，需要锁表，可能也会导致服务可用性的下降。对于大的更新及删除语句可考虑分拆成多条语句执行。尽量平均分布负荷，以减少对生产负荷的冲击。

4) 原则上，升级操作，应该是被测试验证过的。如果是复杂的升级，往往还需要进行模拟演练。

5) 升级前，应该备份数据。备份的原则是能够尽快回滚，如果要升级的表比较多，可以考虑进行一次全备。注意存储过程和触发器的备份方式不同于普通数据。

6) 执行操作前，应该检查是否连接到了正确的数据库，检查执行环境，比如操作系统、mysql客户端，可以在mysql命令行提示符下运行STATUS进行验证。例如，以下命令将验证客户端、连接、数据库等信息，我们推荐使用utf8字符集。

```
mysql> STATUS
-----
mysql Ver 14.14 Distrib 5.1.70, for unknown-linux-gnu (x86_64) using readline 5.1
Connection id:          1109769683
Current database:       db name
Current user:          root@localhost
SSL:                  Not in use
Current pager:         stdout
Using outfile:          ''
Using delimiter:        ;
Server version:        5.1.70-log MySQL Community Server (GPL)
Protocol version:      10
Connection:             Localhost via UNIX socket
Server characterset:   utf8
Db     characterset:   utf8
Client characterset:   utf8
Conn. characterset:    utf8
UNIX socket:           /tmp/3306/mysql.sock
Uptime:                339 days 21 hours 34 min 26 sec
```

以上连接、客户端、数据库、数据库服务器的字符集都设置为utf8了。检查是否连接到了正确的数据库及主从库。Current database及UNIX socket都标识了当前正在更新的库。

7) 升级前，可能需要停止写入，或者停止一些守护，应该和应用服务器的维护人员一起确认是否已经停止了相关的写入或守护。

8) 可以考虑把升级记录到日志里。连接时加-v参数，执行SQL语句时使用tee filename把操作语句的日志输出到文件，执行完毕后进行操作检查。这也是一个好习惯，方便以后我们进行回溯和检查升级的细节。

9) 对于表结构的调整，如果分配更多的资源能缩短执行的时间，那么可在会话级调大资源分配。

以上是升级的大致步骤，现实中，升级的准确性，不仅仅取决于经验、技能，也取决于流程、研发与测试环节的完善，这点不容忽视。

对大表的升级

MySQL更改表结构时，如果是大表，则可能会导致性能问题，因为MySQL 5.1、MySQL 5.5更改表结构的绝大部分操作就是生成一个新结构的临时表，然后把数

据插入到新的表，逐条插入记录，同时修改索引，在将所有数据都复制到新的表之后，删除旧表，然后将新表重命名为旧表的名字，实现新旧表的切换。这种操作成本高，还会导致不能写入数据，对于许多生产系统来说，这是不可接受的，因为它会严重影响服务的可用性。

由于修改大表的表结构时，需要复制一份数据，所以要留意空间是否足够。

MySQL 5.5提供了更多的在线修改表结构的功能，InnoDB也支持通过排序的方式来创建索引，而之前的MySQL 5.1是以逐行插入数据的方式创建索引的。但是，相对于一些NoSQL产品，MySQL在线DDL的能力还是比较弱的，需要尽量小心地避免修改大表结构。在MySQL 5.6版本中，官方更进了一步，增强了很多ALTER TABLE操作，并尽量避免了复制整张表的数据。在修改表结构的同时，允许SELECT、INSERT、UPDATE、DELETE语句继续执行，这种特性也称为在线DDL（online DDL）。

如果因为更改表结构而导致的写阻塞，那么有什么办法可以减轻或避免呢？Percona工具“online schema change”可以做到在线修改表结构而不阻塞服务。M-M架构也可以进行变通解决。或者你可以把数据库升级到MySQL5.6，从而利用其在线修改表结构的特性。

变更数据或导入数据的时候，需要留意该操作对生产的影响，应该控制记录更新的频率，平均分布生产负荷。对于不同的硬件，具体控制的写入频率也应不同，对于普通的SAS硬盘，建议控制在每秒小于200条记录；对于SSD，由于IOPS大大提高，因此每秒控制在500~1000条记录也是可以的。实际更新数据的频率还取决于生产系统的繁忙程度，以及对缓存的影响，更新数据有一个原则，那就是如果不赶时间，那么慢一些会更安全。

变更数据可能会导致从库延时较长，如果从库也提供生产服务，那么需要留意延时的影响。平均分布负荷，将修改或导入数据的操作切割为更小的单元，可以缓解延时。

对于导入大量数据的操作，我们可能难以判断导入的进度，可以大致估算下磁盘的空间，估计目前的进度和剩余的时间，例如如下命令。

```
[mysql@db1000] $ du -sh ; sleep 3600;du -sh
```

以上命令可以查看空间增长，了解大概导入了多少数据。

有时，我们希望利用主从复制架构实现平滑更改大表的表结构，即，我们先在从库上更改完大表结构，把数据库流量切换到从库，然后再更改主库的表结构，再把数据库流量切换回来，注意，在操作从库和主库之前我们需要在会话(session)中运行Set sql_bin_log=0。具有SUPER权限的客户端可以通过SET sql_log_bin=0的语句禁止将自己的语句记入二进制记录。这样就不会在修改大表结构时对它的从库造成影响。

13.1.2 MySQL版本升级

MySQL的版本升级，需要考虑许多因素，如果是大版本的升级，建议首先仔细研读官方的升级文档，因为可能会有一些细节并不包含在常用的步骤之内。如下是升级的注意事项。

·升级前应仔细阅读官方的升级文档。

·升级后，应再重启以确认是否正常。

·mysqldump备份的转储文件可用于升级。由mysqldump导出来的转储文件，相对于物理文件形式的备份来说，兼容性更高。

使用转储文件升级的步骤大致是：先用mysqldump把数据导出来，备份权限，然后升级MySQL代码，并运行mysqld_upgrade脚本进行升级，然后导入转储文件，恢复权限信息。如果是从库，那么需要在升级前停止复制，升级后恢复同步。一般情况下不需要导入MySQL系统库，因为新版本的MySQL系统库可能会有结构上的变更，如果一定要导入MySQL数据库，则要记得使用FLUSH PRIVILEGES生效权限，并使用mysql_upgrade进行修复。

·主从架构升级的话，应先升级从库，从库高于主库一般是可行的，反之则可能出问题。

·不要跨越大版本进行升级，比如，4.1升级到5.0是正确的策略，其他跳过版本的升级可能会有问题，如果升级4.0到5.1，那么合适的策略是按照4.0→4.1→5.0→5.1这样的顺序进行升级。

如下是升级的具体步骤举例，可供读者参考。

1) 如是主从架构，则先升级从库。

2) 关闭MySQL实例，并确认已经停止了服务。

3) 备份并删除MySQL旧版本。注意，不要误删除了数据。

4) 安装新版本MySQL，并启动新版本，确认数据能正常加载。为了使数据更安全，可以配置启动时不开启复制，对于小版本的升级，不停止复制一般也不会有问题。

5) 运行mysqld_upgrade脚本升级。如果是主库，可禁用二进制日志写入(skip-write-binlog)。

```
bin/mysql_upgrade -uroot -ppassword -S /path/to/mysql.sock --skip-write-binlog
```

6) 确认升级成功，重启MySQL，确认MySQL启动正常。

13.2 新业务部署上线

新业务上线会涉及相关的其他团队，这里仅做简单的描述，作为DBA，必须做到如下的一些步骤。

首先，我们需要检查数据库服务器，并配置好主从环境。

其次，我们要确认监控、备份、收集信息的策略是否已经实现。

然后，执行SQL脚本部署新数据库和初始化数据，如有遗留的测试数据，请务必先清理掉。

最后，配合其他团队调试并上线，上线后，还需要观察一段时间。

13.3 迁移

13.3.1 迁移步骤

一般迁移数据库时有如下4个步骤。

(1) 迁移数据库策略的制定

确定迁移前后的架构和物理部署的变化、迁移的方法、迁移的时间、使用的迁移工具、迁移的风险、迁移的回滚策略等。

如果迁移数据是完全通过应用程序来实现数据迁移的，那么DBA往往不需要进行任何操作，需要做的事情主要是监控迁移过程中数据库负载的变化。

比较常见的迁移数据库的方式是，DBA在新的数据库主机上部署新的基础环境，例如，我们有A、B一对数据库主从，因为A、B主机的磁盘空间不够，我们需要将数据库迁移到C、D主机，然后在C、D主机上部署一对数据库主从。接着通知应用服务器的运维工程师修改应用程序的数据库配置文件指向新的数据库主机C，最后，运维工程师重启应用程序，实现对数据库的切换。

一些公司，实现了中间件，数据库流量都通过Proxy，然后到达后端的MySQL，此类数据库流量的切换一般都比较简单，只需要在平台上修改数据库流量的指向，指到其他的数据库主机即可。

我们这里讨论的主要是另一种情况，即通过修改应用服务器配置文件，或者通过修改内网DNS的方式将数据库流量切换到新的数据库主机。

(2) 确定迁移数据库的具体步骤

细化迁移的步骤，预估各个步骤所需要的时间。我们不仅要考虑实际操作的时间，也需要考虑确认的时间，预留处理异常的时间。基础环境的准备可以预先完成，并且添加必要的监控。

(3) 迁移数据库，并检查

实际操作中，应该严格按照拟定步骤的顺序来执行，尽量做到做完一项，确认一项。

(4) 迁移后的处理

迁移后，要注意清理废旧的环境，对新环境进行完善监控，必须要留出足够的人力和时间用于观察。

对于切换、迁移数据库，有如下一些注意事项。

1) 切换前检查新旧服务器的软硬件配置和网络配置。

常见的检查项目有RAID卡、磁盘阵列、CPU、MySQL版本、变量和参数文件的差异、网络、防火墙配置等，可以用Percona工具pt-config-diff检测来对比迁移前后新旧主机参数的差异，它不仅可以检测参数文件的差异，还可以检测实际变量和参数文件之间的差异。

我们需要关注的参数主要是连接数、字符集、内存参数、步长和偏移量、事务隔离级别、日志保留策略、是否只读、server_id等。保持主从软硬件的基础环境和配置的一致性，可以让你后期的维护更简单。需要留意的是，应用程序的逻辑不应该依赖于参数文件，比如应用程序的逻辑不能只依赖于偏移量、步长等参数。主从库的参数配置最好一致，这样部署一个新的从库时，就只需要修改下server-id即可。

2) 主从库的权限设置应该一致。由于数据库的服务器一般处于内网，没有外网IP，相对来说比较安全，那么建议不要对权限设置得过细，否则迁移环境时，还需要考虑修改权限，而DBA很可能会忽视了这点。

3) 如果是新搭建的主从环境，需要确认主从库同步是否正常，可以使用SHOW SLAVE STATUS命令进行确认。

- 4) 如果是主从复制架构，则需要把数据库流量切换到从库，需要在切换前记录从库的SHOW MASTER STATUS信息。
- 5) 应用程序切换数据库流量时，需要防范数据库同时写入主从库的可能性。如果我们有多台应用服务器，可以临时关闭大部分应用服务器，仅留下一台应用服务器，这样重启时，就不会发生同时写入主库和从库的情况。为了安全，我们可能不得不把主库设置成只读。需要注意的是，只读的设置对于Super权限的连接用户是无效的。如果同时写入主库、从库，可能会有主键冲突，从而影响复制，如果发生了主键冲突，可以考虑忽略部分数据或重做从库。
- 6) 对于数据库流量的切换，需要运行命令确认切换是否成功，SHOW PROCESSLIST、SHOW MASTER STATUS、SHOW SLAVE STATUS等命令可以协助你确认信息。我们还可以用tcpdump或iptables命令判断是否还有流量访问旧的数据库端口。
- 7) 旧的数据库如果已经不再使用，则必须及时清理和删除数据。如果已经不需要同步关系，则请务必及时清理掉同步关系，以免误操作数据，同步了生产库。如果将从库提升为主库，则需要停止从库部署的备份脚本。
- 8) 尽量在迁移前就部署好监控，迁移后，需要检查各项的状态信息、性能收集、备份、监控是否完备。

13.3.2 切换数据库时长短连接的影响

由于长连接往往缓存了数据库IP，因此需要重启应用服务器才能实现对数据库访问的变更。以Nginx+PHP+fpm为例，当我们使用fpm来管理PHP进程时，MySQL长连接缓存了访问数据库的IP；当我们更改数据库路由时，比如，更改了数据库配置文件，或者更改了内网DNS，我们需要重启php-fpm管理器才能生效。

如果是Nginx+fpm+PHP的部署，那么PHP进程数×主机数=实际的长连接数。如果查询缓慢则会导致所有的PHP进程都无法处理其他的请求，因为PHP进程需要等待数据库的响应。

短连接可能也会缓存对数据库的访问IP，比如，PHP应用，使用短连接访问数据库，配置文件内存储的数据库实例的DNS域名，如果更改了内网DNS，那么无须重新启动服务即可生效，但如果数据库配置文件中存储的是主机名，那么更改/etc/hosts可能会无效，因为主机名被缓存了，我们仍然需要重新启动应用服务器。

Java服务器默认也缓存了DNS结果，你可以将其设置为禁止缓存，但很少有人这么做，所以更改了内网DNS，或者更改了配置文件内的数据库IP，也需要重启Java服务器。

现实中，不同的应用服务器可能会有不同的行为，所以我们需要了解，在更改了数据库配置文件之后，我们的应用服务器是否需要重新启动才能生效。

13.4 生产环境常用的备份策略

为什么我们需要备份呢？

备份的主要目的是为了灾难恢复，备份还可以用于测试应用、回滚数据修改、查询历史数据、审计等。

之前我们在其他章节已经讨论了备份的一些方式和工具的使用方法，本节将从生产运维的角度介绍备份策略的制订。由于生产环境一般是主从架构，因此本书将基于这种架构，阐述备份的一些策略和方式。

如果是拥有大规模数据库集群的公司，你可能需要专门规划执行数据库备份的机器和海量分布式文件系统，以存储备份，你可能还需要有专门的检测系统、调度系统、恢复测试系统、预警系统和保留策略，以应对大量数据库的备份。

13.4.1 备份策略

我们需要制订备份策略并文档化。我们需要考虑许多因素，比如，数据的重要程度、是否需要时间点恢复、数据量的大小、是否有法律规定要保留多久、需要多久恢复、用户可以接受多久恢复、用户是否可以接受部分功能不可用及其他一些因素。一般常见的方法是每天选择负荷小的时间段进行备份，然后将备份保留一段时间。

我们推荐在从库上进行备份，对于负荷比较小的业务，你也可以选择在主库上进行备份，前提是不影响生产负荷。

对于数据量大的备份，每次制作一个全量从库的代价可能会很大，定期全量备份，然后再进行增量备份，是值得考虑的方法。

备份服务器应该视为与生产服务器一样重要，甚至更重要。

备份文件应该和数据库主机物理分开，可以选择FTP上传或其他网络传输方式把备份文件保留在独立的备份服务器上。

我们也可以使用NFS挂载文件系统的方式进行远程备份，如下是挂载远程文件系统的一个例子。

```
cat /etc/rc.local
mount -t nfs -o rw,bg,hard,nointr,rsize=4194304,wsize=4194304,tcp,vers=3,timeo=600,noac 11.11.11.11:/home/nfs_dir /home/mysql/nfs_from_db1000
```

写入/etc/rc.local的原因是希望在系统启动时挂载。rsize、wsize参数的默认值都比较小，可以适当加大。注意要使用sync的方式，不要使用async的方式进行挂载

需要留意到是，使用NFS的方式可能不够稳定，而且客户端是一个串行的机制，无法提高吞吐，使用NFS的方式，还有潜在的安全风险，但由于NFS简单方便，所以还是有许多人在使用它。

我们应该制定完善的数据保留策略，有日备份、周备份、月备份、季备份，原则上，最近的备份应该每天都保留，随着时间跨度的增长，我们可以保留更少的备份。

一般我们使用热备份，而不使用冷备份。“Hot”的本意是不用关闭MySQL服务，不影响服务。由于InnoDB支持在线热备份，所以建议都使用InnoDB引擎。而MyISAM引擎进行备份的时候必须锁表，这点很可能会影响到服务的可用性。

13.4.2 备份建议

如下是一些备份的建议。

·对于很大的数据库，物理备份更适合。

对于小数据库，用逻辑备份mysqldump就可以了，这样也简单；对于非常大的数据库，逻辑备份恢复得太慢，可能会出错，建议选择物理备份。由于大数据库每天全备成本太高，使用增量备份或保留副本也许是更好的选择。mysqldump进行远程备份的时候，需要确保网络的稳定性，如果备份的时间比较长，而网络又不太稳定，则可能会由于网络波动而导致备份失败，增加备份的重试机制或先备份到本地，再上传或同步到其他服务器会更好。

·我们应该按重要程度保留多个备份。

·我们应定期提取备份验证，衡量恢复所需要的资源，以及恢复的速度和效果。

·主库应该启用二进制日志。

主库应该启用二进制日志，以便搭建从库，做时间点恢复。expire_logs_days参数至少要跨越2~3个备份；

·在从库中启用log_slave_updates，如果没有启用，则应考虑是否备份主库上的二进制日志。

·监视你的备份和备份过程。

·应该考虑备份文件的安全。

13.5 常用备份方式和恢复方法

MySQL的备份方式可以分为物理备份和逻辑备份两种，物理备份主要是通过备份数据文件，以快照的方式进行备份，如果我们使用了LVM或一些存储系统，那么快照将是最方便的方式。备份数据文件或快照往往可以更快地备份和恢复。

本节主要讲述在操作系统下备份数据文件或逻辑备份，对于快照方式不进行叙述，大家可以参考LVM或存储系统做快照的一些技术。

常用的备份工具有mysqldump和Percona XtraBackup。mysqldump是官方自带的一种逻辑备份工具，它的兼容性很好，与版本无关，恢复起来更方便，没有理由不使用它。一般小公司，生产环境大都是用mysqldump进行备份的。Percona XtraBackup是Percona公司提供的工具，对于备份大数据很有效。

13.5.1 使用dd备份和恢复数据

使用dd备份数据是传统的办法，虽然现在已经用得很少。

如下是一些备份的例子。

1) 将本地的/dev/hdx整盘备份到/dev/hdy。

```
dd if=/dev/hdx of=/dev/hdy
```

2) 备份/dev/hdx全盘数据，并利用gzip工具进行压缩，保存到指定路径。

```
dd if=/dev/hdx | gzip >/path/to/image.gz
```

3) 将备份文件恢复到指定盘。

```
dd if=/path/to/image of=/dev/hdx
```

4) 将压缩的备份文件恢复到指定盘。

```
gzip -dc /path/to/image.gz | dd of=/dev/hdx
```

5) 将光盘数据复制到root文件夹下，并保存为cd.iso文件。

```
dd if=/dev/cdrom of=/root/cd.iso
```

13.5.2 使用mysqldump备份和恢复数据

mysqldump这个命令在之前的章节里已经有过许多讲述，对于普通数据的备份，这里将不再赘述。我们一般使用如下命令备份整库。

```
mysqldump --databases db_name > db_name.sql
```

使用如下命令恢复数据。

```
mysql db_name < db_name
```

或者可以使用source命令来执行。

```
mysql > source /path/to/db_name.sql
```

生产环境中一般为主从配置，那么我们一般会在从库上配置备份脚本，进行定时备份。如下是一个使用mysqldump备份的示例。

```
mysqldump --flush-logs --master-data=2 --hex-blob -R -E -f --all-databases 2>> /path/to/log | gzip > sql.name.gz
```

对于InnoDB备份，我们可以加上--single-transaction，实现无阻塞备份。生产环境一般在从库中进行备份，那么我们还可以把复制SQL线程临时关闭以进行备份。

有时我们需要指定字符集参数进行备份，如在MySQL 5.5及以上的版本中，如果你使用了utf8mb4字符集，那么需要添加参数--default-character-set=utf8mb4。

在MySQL 5.6中，由于安全性的需要，你需要把备份账号的密码保存在配置文件中或MYSQL_PWD环境变量内，比如，你可以在脚本里执行export MYSQL_PWD=your_password。

我们需要留意其他一些数据的备份，比如视图、存储过程、触发器和事件。

由于视图有依赖，如果基础表不存在或没有权限，那么视图的导出将会失败，而且会导致mysqldump命令的退出，为了避免无效视图导致备份库失败，我们可以在导出的时候，添加一个参数-f强制导出数据而不是中途退出。

对于mysqldump的备份，需要检查其输出，检查是否有错误或警告，正常备份结束后，应该有“Dump completed on”字样，我们可以使用--result-file参数保存mysqldump结果。

对于报警或错误的信息，我们需要及时处理，比如视图中引用了不存在的表。在我们导入视图之前，需要处理好基表存在的问题。

存储过程、视图、触发器的导出信息里可能带有DEFINER信息，在导入的时候，因为目的库中并不存在相应的用户信息或缺少权限，因此我们需要把DEFINER信息批量替换成合适的用户。如果不指定DEFINER信息，那么系统会自动使用默认的用户。

如下是一个导出存储过程的例子。

```
mysqldump -uroot -S /path/to/tmp//3306/mysql.sock -p -td -R --triggers=false db_name > db_name_procedure.sql
```

如下是一个导出触发器的例子。

```
mysqldump -uroot -S /path/to/tmp//3306/mysql.sock -p -td db_name > db_name_trigger.sql
```

如下例子将仅导出数据，而不包括存储和触发器。

```
mysqldump -uroot -S /path/to/tmp//3306/mysql.sock -p --triggers=false db_name > db_name_data.sql
```

导入视图、导入触发器和导入数据类似。但导入触发器时，需要先删除数据库中原有的触发器。由于导出触发器的转储文件里没有DROP TRIGGER语句，因此我们需要手动生成DROP TRIGGER的语句，命令如下。

```
SELECT TRIGGER_SCHEMA,TRIGGER_NAME,DEFINER FROM information_schema.triggers;  
select concat('drop trigger ',TRIGGER_NAME,';') into outfile '/tmp/drop_trigger.sql' from information_schema.triggers where TRIGGER_SCHEMA='db_name';
```

在MySQL 5.1版本中，如果运行FLUSH LOGS命令，会把我们的MySQL错误日志清除掉，并备份到一个文件，但是如果多次FLUSH LOGS，就难以追踪错误信息了，因为有用的信息都被过滤掉了。

如果是mysqldump带--all-database选项，那么每备份一个数据库，就会切换一次日志（FLUSH LOGS）。

如果是带--lock-all-tables选项，或者是--master-data选项，那么仅需要切换一次日志，而且在这个时刻，所有表都会被锁住。

另外，对于二进制日志的导出，需要添加一个hex-blob选项。

13.5.3 使用Percona XtraBackup备份和恢复数据

1.概述

Percona XtraBackup是一个免费的MySQL热备份软件，支持在线热备份InnoDB和XtraDB，也可以支持MyISAM表的备份，不过MyISAM表的备份需要在锁定表的情况下进行。本书对于Percona XtraBackup的叙述是基于2.2版本的。读者实际使用此工具时，请参考相关版本的官方帮助文档。

Percona XtraBackup有3个主要的工具：xtrabackup、innobackupex、xbstream。它们的特点分别如下。

1) xtrabackup：是一个编译了的C二进制文件，只能备份InnoDB/XtraDB数据。

2) innobackupex：是一个封装了xtrabackup的Perl脚本，除了可以备份InnoDB/XtraDB之外，还可以备份MyISAM。

3) xbstream：是一个新的组件，能够允许将文件转成xbstream格式或从xbstream转到文件格式。

你可以单独使用xtrabackup工具，但是我们推荐用innobackupex来进行备份，因为innobackupex本身就已经包含了xtrabackup的所有功能。

xtrabackup是基于InnoDB的灾难恢复功能进行设计的，备份工具复制InnoDB的数据文件（datafile），但是，由于不锁表，这样复制出来的数据就会不一致。

InnoDB维护了一个重做日志，包含InnoDB数据的所有改动情况。在xtrabackup备份InnoDB的数据同时，xtrabackup还有另外一个线程监视着重做日志，一旦日志发生变化，就把发生了变化的日志块复制走。这样就可以利用此重做日志做灾难恢复了。

以上是备份过程，如果我们需要恢复数据，则在准备（prepare）阶段，xtrabackup就需要使用之前复制的重做日志对备份出来的InnoDB数据文件进行灾难恢复，此阶段完成之后，数据库就可以进行重建还原了。

Percona XtraBackup对MyISAM的复制，是按这样的一个顺序进行的：先锁定表，然后复制，再解锁表。

2.Percona XtraBackup的安装和部署

在<http://www.percona.com/downloads/XtraBackup/LATEST/>上可以下载最新的Percona XtraBackup二进制包。

直接解压可以看到有两个目录，其中，bin目录就是存放之前说过的备份工具，share目录存放着Percona XtraBackup的测试脚本。

这里解释bin目录中各个文件的意义。

除了之前说过的3个工具innobackupex、xtrabackup、xbstream之外，我们还可以看到几个之前没有提到过的文件，它们分别是xtrabackup_51、xtrabackup_55、xtrabackup_56。

这是Percona XtraBackup为了保证对InnoDB发行版的有效兼容而采取的一种人性化的做法。下面来看看这些命令的作用范围，如表13-1所示。

表13-1 xtrabackup的兼容性

Server (适用的 MySQL 版本)	xtrabackup binary (可执行文件名称)	Server (适用的 MySQL 版本)	xtrabackup binary (可执行文件名称)
MySQL 5.1.*	xtrabackup_51	MariaDB 5.5.*	xtrabackup_55
MySQL 5.1.* with InnoDB plugin	xtrabackup	MariaDB 10.0.*	xtrabackup_56
MySQL 5.5.*	xtrabackup_55	Percona Server 5.0	xtrabackup_51
MySQL 5.6.*	xtrabackup_56	Percona Server 5.1	xtrabackup
MariaDB 5.1.*	xtrabackup	Percona Server 5.5	xtrabackup_55
MariaDB 5.2.*	xtrabackup	Percona Server 5.6	xtrabackup_56
MariaDB 5.3.*	xtrabackup		

innobackupex的备份过程中，会生成一些文件，如下是一些生成的文件。

·xtrabackup_checkpoints：此文件包含了LSN号与备份类型。此文件被用于增量备份恢复。

·xtrabackup_binlog_info：此文件记录了备份时刻二进制日志的位置，即SHOW MASTER STATUS的结果。

·backup-my.cnf：此文件仅仅记录了备份所需的my.cnf中的选项，它并不是my.cnf的备份。恢复的时候，需要确认目的地的配置文件也要与这个文件记录的一致。

·xtrabackup_binlog_pos_innodb：此文件记录了备份时刻InnoDB的二进制日志的位置。

·xtrabackup_slave_info：当使用--slave-info选项在从库进行备份时，将会记录CHANGE MASTER语句，以便日后用于搭建新的从库。也就是说它记录了我们正在备

份的从库的SHOW SLAVE STATUS\G里的Relay_Master_Log_File和Exec_Master_Log_Pos的值。以便于我们在这个从库的基础上，为主库搭建其他从库。

其他文件略。

3.如何全备

以下是一个全备的例子，命令中，\$day_backup_dir是我们保存备份的目录。\$bak_file是我们的最终备份文件。由于数据量很大，因此我们最好使用管道输出到pigz压缩命令，进行压缩。pigz是一个开源的压缩工具，可以并行压缩文件，如果你的Linux系统没有安装，那么请下载并安装它。

```
innobackupex-1.5.1 --defaults-file=${def_myconf} --slave-info --user=$user --password=$password --tmpdir=${tmp_dir} --stream=tar $day_backup_dir
2>>$xtrabackup_log |pigz -p 8 > $bak_file 2>>$xtrabackup_log
```

一些参数介绍如下。

--defaults-file=[my.cnf]: 默认的配置文件，注意放在所有参数列表中的第一个。

--slave-info: 在备份一个复制架构中的从库时，这个选项非常有用。它记录了备份时刻正在应用的主库的二进制日志名和位置，这些信息被记录到了xtrabackup_slave_info文件中，这个有点像mysqldump中的CHANGE MASTER标志。当你需要为主库搭建新的从库时，通过对这个从库的备份加上xtrabackup_slave_info文件中的二进制位置来恢复同步。

--tmpdir: 设置这个参数是为了避免在主机上对多个实例进行备份的冲突，因为可能都要向同一个临时目录写入同样的文件。备份过程中可能将大量数据写入到tmpdir的默认值/tmp中，所以需要将这个值设置到非/tmp目录中，以免/tmp目录占满影响备份及系统的其他正常服务。

--stream tar: 使用这个流备份选项，我们可以使用tar进行打包。

实际应用中，还有其他一些选项需要留意。

--databases: 指定备份的数据库，若没有指定则默认备份所有数据库。

--export: 导出个别表，以便于导入到其他服务器上。

具体的innobackupex选项说明可以参照官方文档：

http://www.percona.com/doc/percona-xtrabackup/innobackupex/innobackupex_option_reference.html。

4.如何增量备份

增量备份的原理如下。

1) 首先完成一个完全备份，并记录此时的检查点LSN，你需要一个全备才能恢复一个增量的改变，若没有一个全备作为一个基准，那么你的增量备份就是没有意义的。

2) 然后进行增量备份时，比较表空间中每个页的LSN是否大于上次备份的LSN，若是则备份该页并记录当前检查点的LSN。

相关的选项如下。

--incremental: 建立增量备份。

--incremental-basedir= DIRECTORY: 全备目录，主要用于作为增量备份的基准。

--incremental-dir= DIRECTORY: 增量备份目录。

--incremental-lsn: 用于增量备份的日志序列号。

增量备份的步骤具体如下。

步骤1：全备。

```
innobackupex /data/backups
```

这样的全备，会在全备目录下生成一个带有时间标记的目录，即BASEDIR，我们假定BASEDIR是/data/backups/2013-03-31_23-01-18，该目录即为备份的目录，

你也可以通过innobackupex-no-timestamp覆盖这种行为，备份文件将放在给定的目录下。我们可以看到在BASEDIR目录下有一个文件xtrabackup-checkpoints，里面记录了备份的类型和起始点的位置，如下所示。

```
backup_type = full-backuped
from_lsn = 0
```

```
to_lsn = 1291135
```

步骤2：第一次增备。

```
innobackupex --incremental /data/backups --incremental-basedir=BASEDIR
```

这样就有一个增量备份，存放在/data/backups目录下一个带时间戳的目录中，假定是/data/backups/2013-04-01_23-01-18，称之为INCREMENTAL-DIR-1，如果此时还想要再进行一次增量备份的话，那么类似地，也需要一个基准，现在的基准就变成了刚刚完成的增量备份INCREMENTAL-DIR-1了，我们检查INCREMENTAL-DIR-1目录下的xtrabackup-checkpoints文件，可以看到如下信息。

```
backup_type = incremental
from_lsn = 1291135
to_lsn = 1352113
```

步骤3：第二次增备。

```
innobackupex --incremental /data/backups --incremental-basedir=INCREMENTAL-DIR-1
```

这次增备是在/data/backups下创建了目录/data/backups/2013-04-02_23-01-18用于保存增量备份，称之为INCREMENTAL-DIR-2，检查INCREMENTAL-DIR-2目录下的xtrabackup-checkpoints文件，内容如下。

```
backup_type = incremental
from_lsn = 1352113
to_lsn = 1358967
```

你会发现在每一个备份中，不管是全备，还是增备，它们的目录中都有这样一个文件：xtrabackup-checkpoints。我们也可以通过这些位置点来备份。例如如下语句。

```
innobackupex --incremental /data/backups
--incremental-lsn=1291135
innobackupex --incremental /data/backups
--incremental-lsn=1358967
```

5.如何恢复全备

全备后的文件，不能直接用于恢复数据，因为还存在数据不一致的情况，需要应用事务日志，来确保数据文件的一致性。这也是准备阶段的一个目的。一旦这些操作完成了，数据就可以被用作恢复还原了。

相关选项及其说明如下。

--apply-log: 实际恢复数据时，我们需要先对备份的数据应用事务日志，即在恢复的第一阶段应用日志，如果你不指定--defaults-file参数，那么这里将默认使用backup-my.cnf文件里参数应用日志。

--copy-back: 将之前所做的备份复制到原来的数据目录中。

--redo-only: 在进行增量备份恢复时将会用到。

1) 准备阶段。

例如，运行如下的命令。

```
innobackupex --apply-log /path/to/BACKUP-DIR
```

这里还可以加入--use-memory选项来确保内存，因为这个准备阶段的进程会消耗很多内存。

```
innobackupex --apply-log --use-memory=4G /path/to/BACKUP-DIR
```

2) 恢复还原数据。

在准备阶段完成之后，通过--copy-back选项来完成把备份恢复到服务器的datadir目录下的操作。注意先要备份然后删除数据目录(datadir)下原有的文件，可以保留配置文件my.cnf。

```
innobackupex --defaults-file=/path/to/datadir/my.cnf --copy-back /path/to/BACKUP-DIR
```

3) 在启动mysql服务器之前，要先确认文件的参数文件、文件属主等信息是正确的。可能还需要将文件的所有者信息更改一下。

```
chown -R mysql:mysql /var/lib/mysql
```

4) 启动MySQL，确认服务正常。

实际生产环境中，我们一般会压缩备份文件，所以，在恢复重建数据库之前，我们需要先解压文件。



注意 解压tar文件时要加参数-i，如，tar-xf2013-02-19_12-20-49.tar.gz

6.如何恢复增量备份

增量备份的恢复类似于全量备份的恢复，也有两个阶段，准备阶段和恢复数据阶段。

(1) 准备阶段

与全量备份的准备阶段有所不同，这个阶段需要注意的问题更多。

- 对于每一个增量备份，只有已经提交了的事务才能被重做。这个过程是将全备的内容与增量备份的内容合并到一起。
- 那些没有被提交的事务必须被回滚掉，以得到一份可以用来恢复的数据。

具体步骤如下。

1) 对基本备份进行准备。

innobackupex--apply-log--redo-only BASE-DIR (BASE-DIR即之前全备的那个目录)，运行完毕后，你会看到类似如下的输出。

```
120103 22:00:12 InnoDB: Shutdown completed; log sequence number 1291135  
120103 22:00:12 innobackupex: completed OK!
```

2) 合并第一次的增量备份。

```
innobackupex --apply-log --redo-only BASE-DIR --incremental-dir=INCREMENTAL-DIR-1
```

3) 合并第二次的增量备份。

```
innobackupex --apply-log BASE-DIR --incremental-dir=INCREMENTAL-DIR-2
```

如果有“completed OK!”字样，则表示应用准备成功。

注意--redo-only选项，对最后一个增量备份不要使用--redo-only选项。

4) 合并完所有的增量备份之后，我们运行如下命令来准备好整个数据库文件。

```
innobackupex --apply-log BASE-DIR
```

现在我们的备份文件可以用来进行恢复还原了。

(2) 数据恢复（restore）阶段

在完成了增量备份的准备阶段后，现在的基准目录（base+incremental=full）就像是做了一个全备的目录，可以直接进行重建。

```
innobackupex --copy-back BASE-DIR
```

7.时间点恢复

通过innobackupex和MySQL服务的二进制日志文件可以进行基于时间点的恢复，将数据库恢复到历史的某个状态。二进制日志中保存着对数据库的操作细节，你可以用一个历史备份再加上二进制日志来将数据库恢复到某个时刻。

时间点恢复的过程大致如下。

我们先通过innobackupex做一次全备。

```
innobackupex /path/to/backup --no-timestamp
```

接下来，就准备恢复日志应用。

```
innobackupex --apply-log /path/to/backup
```

假设某个时刻已经过去了，想恢复数据库到该时刻，那么应该首先知道当前的二进制日志情况。

```
mysql> SHOW BINARY LOGS;
mysql> SHOW MASTER STATUS;
```

第一个查询将会告诉你包含了哪些二进制日志文件，第二个查询将会告诉你哪个二进制日志文件当前正在使用，以及当前的日志位置点。

之后，你可以通过之前所做的备份中的xtrabackup_binlog_info文件，找到备份到的日志编号及日志位置。

```
cat /path/to/backup/xtrabackup_binlog_info
```

通过之前所做的备份对数据库进行恢复。

```
innobackupex --copy-back /path/to/backup
```

此时，数据已经达到了某个时刻，可以应用之前得到的信息，再使用mysqlbinlog工具进行基于时间点的恢复。

如下是基于时间点恢复的一个案例。我们需要在一台测试机器上将生产环境的数据库恢复到某个时间点。

1) 为生产环境数据库建立一个全备，然后把所有文件传输到测试机器上。

```
innobackupex /path/to/backup --no-timestamp
```

2) 在测试机器上准备恢复，应用日志。

```
innobackupex --apply-log /home/backup/full/
```

3) 在生产环境中查询当前数据库的日志位置。

```
mysql> SHOW BINARY LOGS;
+-----+-----+
| Log_name | File_size |
+-----+-----+
| mysql-bin.000001 | 126   |
| mysql-bin.000002 | 1306  |
| mysql-bin.000003 | 126   |
| mysql-bin.000004 | 497   |
+-----+-----+
mysql> SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000004 | 497 | | |
+-----+-----+-----+-----+
```

4) 到备份目录中去查看我们备份时刻的日志位置。

```
$ cat /home/backup/full/xtrabackup_binlog_info
mysql-bin.000003 57
```

5) 在测试机器上进行恢复。

```
innobackupex --copy-back /home/backup/full/
```

此时，数据库已经恢复到mysql-bin.00000357这个点的数据了。

6) 传输最新的日志文件到测试机器上，检查需要应用的日志记录。

```
$ mysqlbinlog /home/backup/full/mysql-bin.000003 /home/backup/full/mysql-bin.000004 \
--start-position=57 > mybinlog.sql
```

此时，会生成一个mybinlog.sql文件，这个文件就是mysql-bin.00000357之后的所有操作的日志。我们可以打开来查看下，以确认是不是我们需要的日志。

7) 如果我们希望恢复到11-12-25 01:00:00的数据，那么可以执行如下的语句。

```
$ mysqlbinlog /home/backup/full/mysql-bin.000003 /home/backup/full/mysql-bin.000004 \
--start-position=57 --stop-datetime="11-12-25 01:00:00"
| mysql -u root -p
```

8.其他注意事项

使用xtrabackup进行备份的一些注意事项如下。

·可以使用--parallel参数并行复制文件，但如果你只有一个数据文件（ibdata文件），那将不会有太大的好处，使用独立表空间有利于改善I/O，但是也要注意如果文件碎片很多，可能会导致严重的I/O瓶颈。

·对于I/O资源比较紧缺的服务，或者为了减少对生产的影响，可以采用--throttle选项限制I/O。

·对于从库的备份，建议加-slave-info选项。xtrabackup_slave_info文件记录了主库（Master）的位置（Position）信息，可以用此信息来搭建从库。

·可以加--safe-slave-backup这个选项，这个选项将使得从库在备份的时候会关闭SQL线程。

·对于流备份的tar包，我们可以考虑压缩，或者传输到远程服务器上，命令如下。

```
innobackupex --stream=tar ./ > backup.tar  
innobackupex --stream=tar ./ | gzip - > backup.tar.gz  
innobackupex --stream=tar ./ | ssh user@desthost "cat - > /data/backups/backup.tar"
```

·解包的操作必须加-i参数，否则会出错，如tar-ixvf backup.tar。

·对于恢复操作，一定要确认是否成功，下面的命令

```
$ innobackupex --apply-log /path/to/BACKUP-DIR
```

如果执行成功，则应该输出有类似于“11122501:01:57 innobackupex:completed OK!”的字样。

·建议备份和恢复所使用的xtrabackup版本是一致的。

13.5.4 使用mysqlbinlog进行时间点恢复

mysqlbinlog是一个从二进制日志读取语句的工具。

当我们使用mysqldump对主库进行备份时，我们需要添加参数--master-data=2--single-transaction，生成的备份转储文件会存储备份时刻的二进制日志位置信息，我们可以从这个点开始，进行时间点恢复。

当我们在从库上进行备份时，可以关闭从库的SQL线程，然后进行备份，并记录SHOW SLAVE STATUS\G的输出到备份文件，SHOW SLAVE STATUS\G的输出中记录了当前应用到了主库的那个位置点的信息。

```
Relay_Master_Log_File: mysql-bin.000013  
Exec_Master_Log_Pos: 18761441
```

我们可以从这个点出发，进行时间点恢复。

如下的例子说明了如何利用mysqlbinlog进行时间点恢复。

假设要恢复到2010年8月12日下午15:00。

首先，我们将备份文件导入一个临时测试库。

然后，我们查询备份文件，确认备份时刻的主库二进制文件的位置。我们把包含这个位置点及之后的所有日志文件都传输到临时测试库主机上，假设我们只需要应用一个日志文件。

接下来，我们就可以应用日志了，例如如下语句中：

```
mysqlbinlog --start-position=18761441 --stop-datetime=  
2010-8-12 15:00:00  
mysql-bin.000013 | mysql -uroot -p
```

18761441这个位置，是我们从备份文件里查询到的。

1) 为了保险，你也可以mysqlbinlog输出到文件，然后检查此文件，再用mysql执行。

```
mysqlbinlog -  
start-position=18761441 -  
stop-datetime=  
2010-8-12 15:00:00  
mysql-bin.000013 > /tmp/bin.txt  
mysql -uroot -p < /tmp/bin.txt
```

2) 如果有多个mysql二进制日志，最安全的方式是所有的二进制日志都顺序应用一个连接，如果将多个连接同时导入数据，可能会导致错误。

如果应用多个二进制日志文件，我们可以使用如下的方式来匹配文件。

```
mysqlbinlog binlog.[0-9]* | mysql
```

或者按顺序列出所有需要执行的日志文件。

```
mysqlbinlog binlog.000001 binlog.000002 binlog.000003 | mysql
```



小结 本章介绍了DBA的一些日常维护工作：迁移、升级、备份、恢复。升级不会是一个轻松的工作，特别是在频繁发布升级的公司。DBA除了要熟练地掌握基本功之外，还需要规范各种升级操作，减少过多的升级和误操作的可能性。数据是公司的生命，DBA应该仔细检查备份策略并保护好数据的安全。DBA应该熟悉各种备份和恢复方式，因为这决定了你能够快速恢复服务的能力。

第14章 运维技巧和常见问题处理

DBA的成长，离不开对各种问题的处理。本章将为读者介绍一些运维技巧和常见问题的处理方法。我们需要意识到，别人的经验代替不了自己的经验，所以，多实践、多处理问题，最终会帮你成为一名训练有素的DBA。

14.1 MySQL运维技巧

14.1.1 使用lsof命令恢复文件

如果你在Linux下不小心删除了一个文件，现在想要恢复这个文件，那么lsof命令就能派上用场了。

首先补充下关于lsof命令的基础知识。

lsof是Linux自带的工具，其他Unix系统可能需要自己进行编译安装，它可以显示打开的文件和网络连接，以了解关于系统的更多信息，了解应用程序打开了哪些文件或哪个应用程序打开了特定的文件，可以使我们做出更好的决策。对于数据库维护，lsof的一个重要作用就是可以帮助我们恢复被删除的文件。

/proc是一个目录，其中包含了反映内核和进程树的各种文件。与lsof相关的信息大多数都存储在以PID（进程ID）命名的目录中，所以/proc/1234中包含的是PID为1234的进程的信息。

比如我们查看某个MySQL进程的信息，它打开了一个文件func.MYD。

```
lsof -p 28400 |egrep "COMMAND|func.MYD"
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
mysqld 28400 mysql 291u REG 8,17          0       65550 /data/to/path/mysql/func.MYD
```

我们可以使用“/proc/\$pid/fd/\$fd”的形式访问文件。

```
ll /proc/28400/fd/291
lrwx----- 1 root root 64 Aug 13 15:25 /proc/28400/fd/291 -> /data/to/path/mysql/func.MYD
```

其中，\$pid是打开文件的进程id。\$fd是文件描述符，应用程序通过文件描述符识别该文件。我们可以使用命令lsof -p "\$pid"确认下信息。

当进程打开某个文件时，只要该进程一直保持打开该文件，即使将文件删除掉，它也依然存在于磁盘中。这就意味着，进程并不知道文件已经被删除了，它仍然可以对打开该文件时提供给它的文件描述符进行读取和写入。除了该进程之外，这个文件是不可见的，因为已经删除了其相应的目录条目。

如果可以通过文件描述符查看相应的数据，那么就可以使用I/O重定向将其复制到文件中，如cat/proc/28400/fd/291>/data/to/path/mysql/func.MYD。对于许多应用程序，尤其是日志文件和数据库，这种恢复删除文件的方法非常有用。

如下的案例演示了如何恢复误删除的Apache服务器的access_log。

1) 首先运行如下命令。

```
lsof |grep access_log
```

输出结果类似如下。

```
httpd 26120 apache 42w REG 253,0 5852 12222531 /apachelogs/access_log (deleted)
```

我们可以看到，最后有“(deleted)”字样，好消息是进程(26120)仍然持有这个文件句柄，如果没有进程打开这个文件，那么我们将永远失去这个文件了。

2) 然后到/proc文件系统下去查找信息。

```
more /proc/26120/fd/42
```

以上26120是进程id，42是设备描述符（fd）。

3) 最后，我们可以把这个文件重定向到原来的位置，例如如下语句。

```
cat /proc/26120/fd/42 > /apachelogs/access_log
```

如果我们不知道进程id，那么我们可以使用如下命令来查找被删除文件的信息。

```
lsof -nP | grep '(deleted)'
```

或者使用如下命令。

```
find /proc/*/fd -ls | grep '(deleted)'
```

仍然以上面的access_log为例，如果你删除了access_log，但你都会发现空间并没有被释放，因为进程仍然持有这个文件。这时你可以选择重启Apache服务生效，也可以使用lsof命令找到这个文件，并截断这个文件。

正常的截断access_log的命令如下所示。

```
> /path/to/the/file.log
```

现在这个文件被我们在操作系统下删除了，那么我们可以用如下的方式来截断它。

```
> "/proc/$pid/fd/$fd"
```

14.1.2 如何删除大文件

删除一些大文件时，不可避免地会对当前的I/O造成冲击，会对数据库造成许多慢查询。特别是上百GB大的文件，影响可能会长达几十秒。在ext3系统上，表现尤差。可以采用如下的一些方式来缓解对系统的影响。

·选择闲时，如凌晨，执行删除操作，可使用crontab调度或使用at命令。

·使用其他文件系统，文件系统ext4、xfs对大文件操作的性能远好于ext3。

·使用truncate工具，分段删除文件。具体步骤如下。

1) 下载并安装truncate。

```
wget http://ftp.gnu.org/gnu/coreutils/coreutils-8.9.tar.gz
tar -zxvf coreutils-8.9.tar.gz
cd coreutils-8.9
./configure
make
cp src/truncate /usr/bin/
```

2) 如下是一个删除大文件的脚本。

```
cat rm_large_file.sh
#!/bin/bash
## 调用
truncate命令删除文件
,仅针对大文件
(大于几个
GB的文件
)
## 调用方式
./rm_large_file file_name
if [ "$#" != "1" ] ; then
    echo "please input file name"
    exit 99
fi
filename=$1
filesize=`ls -lh $filename | cut -d\  -f5| cut -dG -f1`#
# 文件大于
1GB时
,且必须是数字
if [[ ${filesize} == *[!0-9]* ]] ; then
    echo "warning:非数字
,可能没有
1GB"
    exit 99
fi
if [ ${filesize} -le 1 ];then
    echo "too small size"
    exit 88
fi
if [ ${filesize} -ge 500 ];then
    echo "too large size,please modify the shell scripts"
    exit 88
fi
sleepetime=3
echo "truncate file $filename ... ,sleep $sleepetime seconds per truncates"
```

```
date_start=$(date +%s)
for i in `seq $filesize -1 1` \
do
    sleep $sleep_time
    echo "truncate to ${i}G"
    truncate -s ${i}G $filename
done
rm $filename
date_end=$(date +%s)
echo "date "+%Y-%m-%d %H:%M:%S" . rm file $filename completed. ($((date_end-date_start)) sec)"
```

14.1.3 获取吞吐信息

如下命令，将实时显示MySQL的吞吐信息。这条命令是从网上摘录的。

```
mysqladmin -uroot -p -i 2 extended-status | awk -F " |" 'BEGIN { count=0; } { if($2 ~ "/Variable_name/ && ++count%15 == 1){print "-----|-----|-----|---" MySQL Command Status --|---- InnoDB row operation ----|-- Buffer Pool Read ---"; print "---Time---|---QPS---|select insert update delete| read inserted updated deleted| logical physical";} else if ($2 ~ "/Queries/){queries=$3;} else if ($2 ~ "/Com_select /){com select=$3;} else if ($2 ~ "/Com_insert /){com insert=$3;} else if ($2 ~ "/Com_update /){com update=$3;} else if ($2 ~ "/Com_delete /){com delete=$3;} else if ($2 ~ "/InnoDB_rows_read/){innodb_rows_read=$3;} else if ($2 ~ "/InnoDB_rows_deleted/){innodb_rows_deleted=$3;} else if ($2 ~ "/InnoDB_rows_inserted/){innodb_rows_inserted=$3;} else if ($2 ~ "/InnoDB_rows_updated/){innodb_rows_updated=$3;} else if ($2 ~ "/InnoDB_buffer_pool_read_requests/){innodb_lor=$3;} else if ($2 ~ "/InnoDB_buffer_pool_reads/){innodb_phr=$3;} else if ($2 ~ "/Uptime / && count >= 2){ printf("%s|%9d",strftime("%H:%M:%S"),queries);printf("%6d %6d %6d",com_select,com_insert,com_update,com_delete);printf("%8d %7d %7d",innodb_rows_read,innodb_rows_inserted,innodb_rows_updated,innodb_rows_deleted);printf("\r%10d %11d\n",innodb_lor,innodb_phr);} }'
```

14.1.4 传输大文件

迁移或恢复备份的过程有时需要传输大文件，传输大文件时需要注意如下两点。

- 1) 用scp进行传输的时候，如果可能造成主库所在机器的I/O紧张，那么可能需要考虑限速(-l参数)，以免影响数据库主机上的其他实例。
- 2) 可考虑使用管道，以减少I/O操作，节约时间。如下命令将利用管道把文件压缩输出到远程服务器上。

```
gzip -c /backup/mydb/mytable.MYD | ssh root@server2 "gunzip -c - > /var/lib/mysql/mydb/mytable.MYD"
```

如下命令将利用管道把mysqldump备份的数据输出到远程服务器上。

```
mysqldump -uroot db_name | gzip -c | ssh mysql@11.11.11.11 "gunzip -c - > /home/mysql/db_name.sql"
```

zcat命令也比较方便实用，可以不用解压缩大文件，直接应用，例如如下命令。

```
zcat xxx.gz | mysql -uroot -p
```

如下命令将合并远程传输和压缩操作，以节省时间。

```
ssh mysql@11.11.11.11 "cd /home/mysql/data ;tar -zcvf - data" | cat > data.tar.gz
```

14.1.5 记录连接用户

我们可以通过设置init_connect参数来记录连接到数据库的用户。如下命令中的accesslog表将存储连接的用户。

```
mysql> show variables like 'init_connect';
| init_connect | insert into db_name.accesslog(thread_id,log_time,localname,matchname) values(connection_id(),now(),user(),current_user());
```

注意还需要分配这个程序账号对表accesslog的INSERT权限。

```
GRANT INSERT ON `db_name`.'accesslog` TO user_name@'10.%' ;
```

14.1.6 如何判断表的碎片

更连续、更紧凑的数据块可以让性能变得更好。碎片化的表会导致一些操作比较慢，如索引范围查找，尤其是对于覆盖索引类的查询。

数据变得碎片化，可能是出于如下原因。

- 行记录自身碎片化，一笔记录被存放在多个地方。

- 逻辑顺序的块或行记录并未顺序存储于磁盘中。

- 自由空间碎片化。

MySQL目前并没有足够的信息来帮助我们判断一个表是否有很多碎片，但是我们可以通过其他一些方式来判断。

一般情况下，当我们对一个大表进行全表扫描的时候，SHOW INNODB STATUS\G如果显示平均I/O SIZE比较小，比如20KB，那么这个表的碎片可能就比较多，例如如下语句。

```
FILE I/O
... reads/s, 20534 avg bytes/read, ... writes/s, ... fsyncs/s
```

对大表进行全表扫描，可以使用如下语句进行模拟。

```
SELECT count(*) FROM tbl WHERE non_idx_col=0
```

在操作系统下，我们可以使用cat命令判断碎片是否比较严重，如cat/dev/sdb1>/dev/null。下面我们通过一个例子来进行说明。

正常的情况下，6块15K转速的SAS盘所组成的RAID1+0，I/O吞吐率可以达到300MB每秒，如果我们使用如下命令检查到每秒只有几十MB，那么表的碎片可能比较严重了。

```
cat table.ibd > /dev/null
```

当使用独立表空间时，table.ibd是表的数据文件。

一般情况下，我们极少碰到表的碎片所导致的性能问题，但在突然的大规模的数据变更下，碎片可能会比较严重。一般有如下三种办法整理碎片。

- OPTIMIZE TABLE命令优化。

- ALTER TABLE TABLE_NAME ENGINE=ENGINE。

- 重新导出导入数据。

推荐使用OPTIMIZE TABLE命令进行优化。需要留意的是执行OPTIMIZE TABLE命令时会锁表，你将不能继续写入数据。

我们也可以借助一些开源的工具来判断数据文件的碎片，对于独立表空间，我们还可以通过查询information_schema.tables的DATA_FREE列来衡量碎片化的程度。

```
SELECT
    ENGINE,
    TABLE_NAME,
    Round(DATA_LENGTH / 1024 / 1024) as data_length,
    round(INDEX_LENGTH / 1024 / 1024) as index_length,
    round(DATA_FREE / 1024 / 1024) as data_free,
    DATA_FREE / (DATA_LENGTH + INDEX_LENGTH) as ratio_of_fragmentation
FROM
    information_schema.tables
WHERE
    DATA_FREE > 0;
```

碎片化比较严重，不一定就是有性能问题，即使以上碎片化的比率达到20%甚至30%，你应该在确认性能问题的原因就是表的碎片化后才能采取行动。

14.1.7 快速关闭MySQL

如果InnoDB缓冲区（innodb_buffer_size参数）很大，缓冲区内的脏数据太多，那么关闭的时候必须把脏数据刷新到磁盘，这个过程可能会很漫长，从而导致关闭服务的时间过长。

我们可以临时设置innodb_max_dirty_pages_pct=0，然后等脏数据大部分都刷新到磁盘后（查看SHOW INNODB STATUS输出中的Modified db pages，这个值应该比较小），再手动关闭数据库。

可以采用如下的办法。

1) 运行命令“SET GLOBAL innodb_max_dirty_pages_pct=0;”。

2) 运行命令mysqladmin ext-i10|grep dirty检查状态变量Innodb_buffer_pool_pages_dirty，等到它接近0的时候关闭它，如果是生产繁忙的系统，这个值可能会一直偏大。

3) 待Innodb_buffer_pool_pages_dirty的值很小时，就可以用mysqladmin关闭MySQL了。

对于某些需要快速关闭和重启MySQL的情况，这种方法是适合的，因为我们可以预先运行第一个步骤的命令。

另一种办法是设置innodb_fast_shutdown=2（默认为1，可以动态修改该值），不过不到万不得已时不要这么做，因为虽然这样可以快速关闭MySQL，但启动的时候要执行更多的恢复操作。



注意 对于InnoDB的数据库，FLUSH TABLES是没有用的，FLUSH TABLES是针对MyISAM这类引擎的。

14.1.8 如何预热数据

预热数据是为了能把热点数据加载到内存中。可以考虑的一个方法是，执行一次全表扫描（full table scan），如下是一个全表扫描的例子，在一个4块15K转速的SAS盘所组成的RAID1+0的数据库主机上执行如下查询，查询以一个非索引的列为条件，执行COUNT操作。

```
SELECT COUNT(*) FROM tbl WHERE non_idx_col=0;
```

通过*iostat*命令可以看到I/O比较高，顺序读取磁盘吞吐有每秒百MB以上，如果比较低，只有每秒几MB，那么这个表的碎片化可能严重或硬件有问题。

我们可以使用SHOW INNODB STATUS命令检查下预热的效果。检查FILE I/O节，可以看到每秒有几百次的I/O，每次I/O在百KB左右。这与操作系统命令*iostat*的输出类似（见avgqrq-sz项）。检查ROW OPERATIONS节，可以看到每秒有数十万条记录的读取量。

我们也可以把预热数据要执行的SQL通过*init_file*参数来执行，这样就可以在系统启动的时候执行了。

14.1.9 临时禁止数据库访问

我们可以使用防火墙工具*iptables*临时禁止网络访问。例如如下语句。

```
iptables -A INPUT -p tcp --dport 3306 -j DROP
```

或者配置参数*skip-networking*临时禁止网络访问。

14.1.10 获取MySQL连接、用户

以下查询可用于获取长连接的用户连接。

```
SELECT LEFT(host, IF(LOCATE(':', host), LOCATE(':', host), LENGTH(host) + 1) - 1
) AS
host_short, GROUP_CONCAT(DISTINCT USER) AS users,COUNT(*)
FROM information_schema.processlist
GROUP BY host_short
ORDER BY COUNT(*),host_short;
```

14.1.11 更改数据库名

MySQL并没有直接修改数据库名的管理命令，如果需要修改数据库的库名，有如下两种方法。

· 使用*mysqldump*导出该数据库下的所有表，然后创建新的数据库，然后使用*mysql*命令再把表导入新的数据库，最后删除旧的数据库。

· 重命名表，具体步骤如下。

1) 新创建数据库newdb。

```
mysql> CREATE DATABASE newdb;
```

2) 生成重命名表的语句。

```
mysql -N -e "SELECT CONCAT('rename table olddb.',table_name,' to newdb.',table_name,';') FROM information_schema.TABLES WHERE TABLE_SCHEMA='olddb';" >
rename_mysql_name.sql
```

3) 执行*rename_mysql_name.sql*。

```
mysql -uroot -p < rename_mysql_name.sql
```



注意 重命名表的操作会导致连接中断，所以你的应用程序需要有重连的机制。

14.1.12 批量KILL连接

有时生产环境突然出现性能恶化，登录MySQL，运行SHOW PROCESSLIST命令，发现有大量查询正在执行，这时你打算手动KILL掉应用程序中过来的运行时间超过200s的所有的数据库连接。

```
mysql> SELECT CONCAT('KILL ',id,';') FROM information_schema.processlist
```

```
WHERE user<>'root' AND Command='Query' AND db='db_name' AND time > 200 INTO OUTFILE '/tmp/a.txt';
mysql> SOURCE /tmp/a.txt;
```

你可以添加更多的筛选条件。

如下是一个KILL掉被阻塞的连接的例子，这是一个临时的解决方案，彻底解决问题需要尽快找到导致阻塞的原因。

```
for id in `mysqladmin processlist|grep -i locked|awk '{print $1}'`  
do  
    mysqladmin kill ${id}  
done
```

14.1.13 记录运行时间长的查询

如下命令将记录运行时间超过120s的查询。

```
mysql -uroot -p -e "show full processlist" |grep "Query" |grep "select" |egrep -v "root|Sleep|Locked|INSERT|DELETE|UPDATE" | gawk '{if(strtonum($6)>120){print $0;}}' | grep db_name > /tmp/long_running_process.lst
```

14.1.14 删表分表

如果分表是类似于table_name_20100923这样的格式，现在我们需要删除3个月之前的数据，那么我们可以使用如下语句生成批量DROP TABLE的语句。

```
SELECT  
    CONCAT('DROP TABLE ', table_name, ';')  
INTO OUTFILE '/tmp/file' from  
information_schema.tables  
WHERE  
    table schema = 'db name'  
    AND table_name like 'table_name_201%'  
    AND table_name < CONCAT('table_name_',date_format(date_sub(now (), interval 90 day),'%Y%m%d'))
```

14.2 常见问题

14.2.1 忘记root密码

如果忘记了root密码，可以按如下步骤进行处理。

- 1) 先关闭MySQL服务，你可以使用自启动服务脚本关闭MySQL，或者直接在操作系统下kill掉服务。
- 2) 然后修改配置文件，添加--skip-grant-tables参数，然后重新启动MySQL服务，此时我们可以无密码登录，然后修改权限表，命令如下。

```
UPDATE mysql.user SET password=PASSWORD('new password') WHERE user='root';
```

- 3) 修改配置文件，去掉启动参数--skip-grant-tables，重新启动MySQL。这时你就可以使用新密码了。

14.2.2 InnoDB同时打开事务最大不能超1023个

对于MySQL 5.1，如果并发事务超过1023个，InnoDB将报错，报错语句为“**InnoDB: Warning** cannot find a free slot for an undo log”。程序也会报错，报错语句为SQL state[HY000];error code[1637];Too many active concurrent transactions;。

解决方式如下。

- 使用MySQL5.5或之后版本。
- 使用Percona分支版本也可以解决。

14.2.3 连接不上MySQL

如果连接不上MySQL，将输出类似如下的错误信息。

```
shell> mysql  
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)  
shell> mysql  
ERROR 2002: Can't connect to local MySQL server through socket  
'/tmp/mysql.sock' (111)
```

可能的原因如下。

- 数据库服务器没有启动。
- 连接了错误的端口或套接字（socket）文件。
- 服务器或客户端程序不具有访问包含套接字文件的目录或套接字文件本身的权限。

还可以使用`shell>netstat -ln|grep mysql`来确定下socket文件的位置。

如果报出如下错误：

```
Access denied for user 'user'@'ip_address' (using password: YES)
```

那么原因一般是密码错误。

Access denied错误消息将会告诉你，你正在使用哪个用户句尝试登录，你正在试图连接到哪个主机，是否使用了密码。通常，你应该在**user**表中有一行记录能够正确地匹配错误消息中给出的主机名和用户名。例如，如果遇到包含了**using password: NO**的错误信息，则说明你登录时没有密码。

如果报出如下错误：

```
Host ... is not allowed to connect to this MySQL server
```

那么这是因为在**user**表中没有匹配你运行命令的主机的行，可能是IP被限制了。

14.2.4 主机的host_name被屏蔽

如果遇到下述错误，则表示**mysqld**已收到了来自主机“`host_name`”的连接请求，但该主机被屏蔽了。

```
Host 'host_name' is blocked because of many connection errors.  
Unblock with 'mysqladmin flush-hosts'
```

可运行命令“`mysqladmin flush-hosts`”解除屏蔽。

`max_connect_errors`变量设置了最多允许多少次连接中断，如果超过了这个阈值，MySQL就会屏蔽主机的后续请求。直到你执行了`mysqladmin flush-hosts`命令，或者发出了`FLUSH HOSTS`命令为止。

14.2.5 连接数过多

当你试图连接到**mysqld**服务器时遇到“`Too many connections`”错误，这表示所有可用的连接均已被其他客户端使用。允许的连接数由`max_connections`系统变量来控制。默认值为100。如果需要支持更多的连接，则需要设置变量`max_connections`，命令如下。

```
mysql> SET GLOBAL max_connections = 3000
```

mysqld实际上允许`max_connections`+1个客户端进行连接。额外的连接保留给具有SUPER权限的账户。通过为系统管理员而不是普通用户授予SUPER权限（普通用户不应具有该权限），系统管理员能够连接到服务器，并使用`SHOW PROCESSLIST`来诊断问题，即使已连接的无特权客户端数量已达到最大值也同样。处理问题的步骤大致如下。

1) 查看当前的连接数`Threads_connected`，曾经最大的连接数`Max_used_connections`。

```
mysql> SHOW GLOBAL STATUS LIKE '%conn%';
```

2) 检查下当前线程的详细信息，线程是否大量累计，被阻塞。

```
mysql> SHOW PROCESSLIST ;
```

以上步骤，一般可以判断原因，必要的话，可以运行`KILL`命令，临时`KILL`线程。

3) 如果需要临时增加连接数阈值，可运行如下命令。

```
mysql> SET GLOBAL max_connections=new_value;
```

4) 如果需要永久变更，则要记得同步更改配置文件`my.cnf`。

14.2.6 处理磁盘满

出现磁盘空间满的情况时，MySQL将会每分钟检查一次，查看是否有足够的空间写入当前行。如果有足够的空间，则将继续，就像什么也未发生一样。每10分钟会将1个条目写入日志文件，提醒磁盘满状况。在磁盘满的情况下，可以临时把一些大文件挪走，比如二进制日志文件，如果不需要马上切换日志，一般是不会有这些问题的。

14.2.7 表损坏

表损坏主要出现在MyISAM引擎的表发生损坏的情况下，如果MySQL主机突然崩溃，或者强制关机而没有正常关闭MySQL服务都可能导致MyISAM表损坏。当在表中查询数据时候，你会碰到报错。

一个被损坏了的表的典型症状如下。

- 1) 当在从表中选择数据时，你会得到如下错误。

```
Incorrect key file for table: '...'. Try to repair it
```



注意 磁盘空间，如临时表所在的操作系统分区满了的情况下，也会有这种报错。

- 2) 查询不能在表中找到行，或者返回不完全的数据。
- 3) 提示错误信息：Error:Table 'p' is marked as crashed and should be repaired.
- 4) 打开表失败：Can't open file。

MyISAM表可以采用以下步骤进行修复。

- 1) 使用REPAIR TABLE命令或myisamchk工具来修复。
- 2) 如果上面的方法修复无效，则使用备份来恢复表。

建议在配置文件里添加自动修复表的参数，即myisam-recover=default，这样系统会在启动的时候自动帮你修复表。如果表被标记为“not closed properly”或“crashed”，那么MySQL会检查该表，并写入信息“Warning:Checking table...”到错误日志，如果表需要修复，则执行修复，并写入“Warning:Repairing table”信息到错误日志。如果你的MySQL实例没有崩溃过，但是出现了大量的这类信息，那么可能是哪里出问题了，你需要做进一步的诊断。

如果是大表，修复表会很耗时，还可能会影响到服务。

14.2.8 查看锁的等待

新的MySQL版本5.5增加了一些视图，用于查看锁的等待情况，例如：

```
SELECT r trx_id AS waiting_trx_id, r trx MySQL_thread_id AS waiting_thread, TIMESTAMPDIFF(SECOND, r trx wait_started, CURRENT_TIMESTAMP) AS wait_time, r trx query AS waiting_query,
l loc_table AS waiting_table_lock,
b trx_id AS blocking_trx_id, b trx MySQL_thread_id AS blocking_thread,
SUBSTRING(p.host, 1, INSTR(p.host, ':') - 1) AS blocking_host,
SUBSTRING(p.host, INSTR(p.host, ':') + 1) AS blocking_port,
IF(p.command = "Sleep", p.time, 0) AS idle_in_trx,
b trx query AS blocking_query
FROM INFORMATION_SCHEMA.INNODB_LOCK_WAITS AS w
INNER JOIN INFORMATION_SCHEMA.INNODB_TRX AS b ON b trx_id = w.blocking_trx_id
INNER JOIN INFORMATION_SCHEMA.INNODB_TRX AS r ON r trx_id = w.requesting_trx_id
INNER JOIN INFORMATION_SCHEMA.INNODB_LOCKS AS l ON w.requested_lock_id = l.lock_id
LEFT JOIN INFORMATION_SCHEMA.PROCESSLIST AS p ON p.id = b trx MySQL_thread_id
ORDER BY wait_time DESC
```

或，

```
SELECT CONCAT('thread ', b trx MySQL_thread_id, ' from ', p.host) AS who_blocks,
IF(p.command = "Sleep", p.time, 0) AS idle_in_trx,
MAX(TIMESTAMPDIFF(SECOND, r trx wait_started, NOW())) AS max_wait_time,
COUNT(*) AS num_waiters
FROM INFORMATION_SCHEMA.INNODB_LOCK_WAITS AS w
INNER JOIN INFORMATION_SCHEMA.INNODB_TRX AS b ON b trx_id = w.blocking_trx_id
INNER JOIN INFORMATION_SCHEMA.INNODB_TRX AS r ON r trx_id = w.requesting_trx_id
LEFT JOIN INFORMATION_SCHEMA.PROCESSLIST AS p ON p.id = b trx MySQL_thread_id
GROUP BY who_blocks ORDER BY num_waiters DESC\G
```

也可以使用mysqladmin debug命令查看锁的等待情况，mysqladmin debug命令将会把锁的信息打印到MySQL Server的错误日志（error log）中。

14.2.9 mysqldump备份报错

将mysqldump备份到远程管道，或者慢速设备（如NFS）中时，可能会出现如下的报错信息。

"Got timeout writing communication packets".

或者报错信息如下。

110421 2:07:01 [Warning] Aborted connection 237201 to db: 'db_01' user: 'root' host: 'localhost' (Got timeout writing communication packets)

你可能需要增加net_write_timeout参数才可以确保不会出错。

14.2.10 Table'tbl_name' doesn't exist

由于MySQL是使用目录和文件来保存数据库和表的，因此如果它们位于区分文件名大小写的文件系统上时，数据库和表名也区分文件名大小写。

如果提示如下错误。

Table 'tbl_name' doesn't exist
Can't find file: 'tbl_name' (errno: 2)

则有可能确实不存在这个表，但也可能表是存在的，但你没有正确引用它，或者没有权限。

14.2.11 root账号权限异常

如果root账号异常，比如，误删除了root账号，那么你可以采取下面的方式来处理，建议不到万不得已时，不要使用下面的方法。更保险的办法还是关闭实例，然后直接复制其他实例的mysql库，重启后进行适当的修改即可。

恢复root账号的具体步骤如下。

1) 关闭MySQL。

2) 不加载权限表启动MySQL，即运行mysqld_safe--skip-grant-tables&。

3) 运行mysql -u root -p回车, 执行如下的查询。

4) 然后，重启mysqld即可。

有时我们可能会发现root不能给其他用户赋予权限。

```
mysql> GRANT EVENT ON *.* to event user@localhost;
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
SELECT host,user,Grant_priv,Super_priv FROM mysql.user WHERE user='root'; #可发现
root@localhost的
Grant_priv值为
N
```

如下命令将给root账号恢复GRANT权限。

```
UPDATE mysql.user SET Grant_priv='Y', Super_priv='Y' WHERE user='root' and host='localhost';
FLUSH PRIVILEGES;
mysql > EXIT
mysql >
mysql > GRANT EVENT ON *.* to event user@localhost;
```

14.2.12 SHOW PROCESSLIST输出中有大量unauthenticated user连接

SHOW PROCESSLIST输出中有大量unauthenticated user连接时，如果连接很频繁，则客户端可能会报错“Can't connect to MySQL server on”。

一般出现这种异常的原因是，数据库开了域名反向解析从而导致客户端连接超时，解决方案如下：

1) 把服务的DNS反向解析功能关掉

2) 构建自己的DNS解析或更改hosts文件，使其能够快速解析域名

14.2.13 统计information_schema里面的元数据信息缓慢

有时统计information_schema里面的元数据信息缓慢，这种情况一般发生在统计许多表、大表或分区表的时候，对information_schema执行的一些查询，如SHOW TABLE STATUS、SHOW INDEX等操作，会导致MySQL Server计算统计信息。由于查询information_schema里的信息缓慢，甚至还可能会导致服务器出现性能问题，许多人改用操作系统命令行工具进行统计，比如使用find、du等命令统计空间占用情况等。

由于可能会影响到服务器性能，因此对information_schema的查询要慎重使用，对于拥有大量数据的MySQL Server可能会导致严重的性能问题。一些监控工具、监控脚本就存在这样的严重问题。

解决办法是设置变量SET GLOBAL innodb_stats_on_metadata=0以避免产生性能问题。innodb_stats_on_metadata=0表示在查询information_schema时，不自动更新统计数据。

InnoDB的统计信息并不是持久化到硬盘里的，而是动态收集的，存储在内存中的。MySQL 5.6、Percona Server可以设置参数，对统计数据进行持久化。

14.2.14 Aborted_connects、Aborted_clients异常升高

有时我们会观察到状态变量Aborted_connects、Aborted_clients在不断增长。

```
mysqladmin ext | grep Abort  
mysqladmin ext | grep Abort | grep -v 0
```

用“--log-warnings=2”选项启动mysqld，可获得关于连接的更多信息。这样，就能将某些断开连接错误记录到hostname.err文件中，例如如下语句。

```
010301 14:38:23  Aborted connection 854 to db: 'users' user: 'josh'
```

你也可以动态设置这个参数。

如果客户端成功连接到服务器但是因异常断开了连接，那么MySQL Server的状态变量Aborted_clients将增加，并将“Aborted connections”（放弃连接）消息记录到错误日志中，可能的原因有如下几点。

- 客户端程序在退出之前未调用mysql_close()。
- 客户端的空闲时间超过wait_timeout或interactive_timeout秒，未向服务器发出任何请求。
- 客户端在数据传输中途突然结束。

如果客户端甚至不能连接到MySQL Server，那么MySQL Server将会增长Aborted_connects变量，不成功的连接尝试可能是因为如下的原因。

- 客户端没有权限连接数据库。
- 客户端密码错误。
- 连接信息包不含正确的信息。
- 获取连接信息包的时间超过connect_timeout秒。

我们可以使用tcpdump来获取可能出错的原因，比如可以用如下方式检测到密码错误。

```
tcpdump -s 1500 -w tcp.out port 3306  
strings tcpdump.out
```

14.2.15 MySQL server has gone away错误

有时会出现“MySQL server has gone away”的报错，一般同时还会有“Lost connection to server during query”的报错。发生MySQL server has gone away的最常见原因是连接闲置超时，被服务器中断连接，默认情况下，服务器关闭空闲时间超过8小时的连接，我们可以设置变量wait_timeout，改变8小时的默认值，一般同时还需要修改interactive_timeout。

mysql命令行默认是重连的，但有一些应用程序，也许并没有重连的机制，这往往会导致执行失败。

导致MySQL server has gone away错误的一些其他原因如下所示。

- 使用KILL语句或mysqladmin kill命令杀死了正在运行的线程。
- 在关闭了与服务器的连接后试图运行查询。这表明应更正应用程序中的逻辑错误。

- 在客户端的一侧遇到TCP/IP连接超时错误。
 - 在服务器端遇到超时错误，而且禁止了客户端中的自动再连接功能。
- 如果向服务器发出了不正确或过大的查询，也会遇到这类问题。如果mysqld收到过大的或无序的信息包，它会认为客户端出错，并关闭连接。如果需要执行较大的查询（例如，正在处理大的BLOB列），则可通过设置服务器的max_allowed_packet变量，增加查询限制值，该变量的默认值为1MB。

14.2.16 信息包过大错误

通信信息包是发送至MySQL服务器的单个SQL语句，或者发送至客户端的单一行。在MySQL 5.1服务器和客户端之间最大能发送的信息包为1GB。

当MySQL客户端或mysqld服务器收到大于max_allowed_packet字节的信息包时，将发出“(ER_NET_PACKET_TOO_LARGE)”，错误，并关闭连接，错误如下。

```
Error: 1153 SQLSTATE: 08S01 (ER_NET_PACKET_TOO_LARGE)
Message: Got a packet bigger than
max_allowed_packet' bytes
```

有一些客户端还会在包过大时，提示“Lost connection to MySQL server during query”的错误。

客户端和服务器均有自己的max_allowed_packet变量，因此，如你打算处理大的信息包，则必须增加客户端和服务器上的该变量。

如果你正在使用mysql客户端程序，这时要想将max_allowed_packet变量设置为较大的值32M，可用下述方式进行修改。

```
mysql> set global max_allowed_packet=32*1024*1024;
```

配置文件可修改如下。

```
[mysqld]
max_allowed_packet=32M
```

增加该变量的值很安全，这是因为仅当需要时才会分配额外的内存。例如，仅当你发出长查询或mysqld必须返回大的结果行时mysqld才会分配更多的内存。该变量之所以取较小的默认值也是一种预防措施，以捕获客户端和服务器之间的错误信息包，并确保不会因为偶然使用大的信息包而导致内存溢出。

14.2.17 内存溢出

32位机器中有内存寻址的限制，注意不要突破2GB（一般32位系统有2.5~2.7GB的限制）的限制，否则很容易导致MySQL崩溃，如下是一段崩溃时候的报错信息。

```
"100201 11:49:29 [ERROR] /usr/local/mysql/bin/mysqld: Out of memory (Needed 2095208 bytes)"
100108 10:42:12 [ERROR] /usr/local/mysql/bin/mysqld: Out of memory (Needed 2095392 bytes)
100108 18:30:10 [ERROR] /usr/local/mysql/bin/mysqld: Out of memory (Needed 156 bytes)
100108 18:30:10 -
mysqld got signal 11 ;
```

如果使用mysql客户端程序发出了查询，并收到下述错误之一，则表示mysql没有足够的内存来保存全部查询结果。

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

14.2.18 MySQL单张表为多大才合适，为什么大表会慢

笔者建议单个表的数据在千万条以下，主要是因为MySQL 5.0和MySQL 5.1在线DDL的能力太弱，MySQL 5.6和5.7对于在线表结构的变更做了许多优化，已经极大地缓解了修改表结构对于生产系统的影响。性能往往不是制约数据量的主要因素，如果你修改表结构的代价比较高，而你的磁盘性能并不高，那么你就应该未雨绸缪，把数据表限制得小一些。如果你能够确保修改表结构并没有太大的影响，那么几亿以上条数据的表也是可以接受的。

生产环境中，研发人员往往担心表太大了性能会下降，但是性能下降，往往是受多个因素的影响，如果优化得好，资源配置适当，表的设计和访问充分利用了MySQL的簇表结构，比如，访问是基于主键，或者基于主键的范围查找，那么亿条数据级别的表也是可以很快的。

InnoDB缓冲很重要，如果我们的热点数据能够被缓存，当我们对大表的访问能够命中缓冲时，那么性能显然也会很好。

对于一些大表的访问，如果随机读过多，那么也可能会导致严重的性能问题，有时顺序读可能还会更快些。顺序读，意味着我们选择的是全表扫描或基于主键的范围查找。

项目初期，研发人员对于数据访问的模式可能还不太了解，设计了比较符合范式的表，那么查询数据时往往需要连接多张表，在数据量不大的情况下，这点可

能不能成为问题，但是一旦表的数据量增加了，连接的代价就会越来越大，因为利用索引连接表，往往意味着大量的随机读。所以在OLAP这种存在很多大表的应用中，应尽量避免出现连接。

清理大表的数据时，归档数据也是一种可以考虑的方式，我们可以把历史表分离到更差的机器或磁盘中。

对于OLTP应用，如果必须对大量数据进行操作，那么分批地小批量获取数据将会更佳，因为MySQL不擅长同时处理大量短小的事务和一个巨大的事务。

14.2.19 MySQL最大能支持多大的并发查询

对于普通的数据库主机（硬盘采用SSD），一般简单的查询（读写混合）可以达到3000~5000 QPS（高并发的小结果集）。如果是纯基于主键的查询，则QPS可以更高。但如果是复杂的查询，可能就只会有几百的QPS。复杂的查询是指诸如分组、排序、批量操作数据、统计之类的消耗资源的查询，或者会涉及复杂的计算。你可以尝试分解复杂的查询，把一些排序、计算之类的操作放到应用程序中去实现。

14.2.20 创建索引出错

创建索引可能会报错“ERROR 1170(42000)”。

如果是对BLOB或TEXT字段建立索引，则需要设置键的长度，否则会出错。

```
mysql> create index idx_col on table_name(col);
ERROR 1170 (42000): BLOB/TEXT column 'col' used in key specification without a key length
```

正确的做法是，设置键的长度

```
mysql> create index idx_col on table_name(col(255));
```

14.3 故障和性能问题处理

14.3.1 通过减少文件排序和临时表提高性能

通过检查SHOW PROCESSLIST输出或慢查询日志，我们可以筛选出开销大的SQL，通过EXPLAIN检查它们的执行计划，我们会发现它们可能扫描了过多的记录数，并且Extra列的输出类似于“Using where;Using temporary;Using filesort”。

Using filesort意味着你不能利用索引进行排序，Using temporary意味着查询使用了临时表来存储数据，如果临时表超过了限制，那么还可能需要转变成磁盘临时表，在高并发的情况下，可能还会导致严重的性能问题，如果通过SHOW PROCESSLIST看到有输出“Copying to tmp table on disk, Copying to tmp table on disk”的信息，则应该避免，即使当时没有出现性能问题，以后也可能会导致性能问题。

如果我们把Using temporary和filesort优化掉了，往往也就解决了性能问题。对于临时表的优化，请参考6.2.10节，对于filesort的优化，解决的思路就是尽量利用索引进行排序，如果实在不能利用索引排序，可以通过限制排序集合的数据来缓解性能问题。我们在6.2.8节对filesort的优化做了详细介绍。

14.3.2 通过慢查询快速定位导致性能问题的SQL

我们在4.3节详细介绍了慢查询日志。普通情况下，我们通过检查慢查询日志里扫描的记录数，返回的结果集及响应的时间可以大致判断出不良SQL。

一般在没有严重性能问题的时候进行检查会更好，因为这个时候，还没有出现SQL互相等待，资源严重竞争的问题，一旦出现互相等待，慢查询的大量输出往往会扭曲分析结果。对于慢查询日志里出现了太多条目这种情况，我们可以通过一些方法筛选出更值得怀疑的SQL，扫描记录数非常多的慢查询记录，仍然是重点怀疑的对象，如果并发还比较多的话，则更值得怀疑。检查出现初期性能问题的慢查询，干扰会更少，另外，往往在一个严重消耗资源的不良SQL执行完后，会马上出现大量的慢查询，因为之前的这些本来应该运行得很快的SQL被阻塞了。

如何定位到具体的不良SQL，需要经验和技巧，通过不断地累计经验，你会越来越熟悉通过慢查询快速定位问题，如果有自动化的收集信息的工具会更好，你可以定期扫描慢查询日志，记录这些慢查询日志，通过人工或自动的方式分析和报警。

14.3.3 定位导致了性能问题的客户端/应用服务器

许多时候，我们碰到的性能问题都是来自于一些特定的应用服务器，这个时候，要求我们能够快速定位到连接到MySQL的可疑应用服务器，SHOW PROCESSLIST输出中的IP信息会帮助我们找到可疑的应用服务器，甚至可疑的应用服务，比如，我们在host列看到的信息不仅包括IP信息也包括端口信息：192.168.1.70:45384，通过在IP信息所指的应用服务器中运行lsof或netstat命令，我们可以找到对应的操作系统进程ID，如下所示。

```
netstat -ntp | grep :45384
tcp        0      0 192.168.1.70:45384  192.168.1.82:3306  ESTABLISHED 28540/php-cgi
```

以后就可以对进程id为28540的php-cgi进程做更多的诊断。

如果发现我们的数据库连接里有大量的sleep状态的连接，那么可以使用如上的方式找到对应应用服务器上的服务。



小结 本章介绍了运维等一些技巧及常见问题的处理。在日常运维过程中，建议大家平时积累自己的知识库，记录下故障处理过程中的现象、影响、处理措施、原因分析、后续改善等信息。一些小小的应用技巧在关键的时候很可能会帮到你的大忙。随着你经验的不断丰富，将逐渐能够避免大部分可能出现的问题。成熟的运维体系和训练有素的工程师团队面对的MySQL问题将不会是本书所列举的一些常见小问题，如果你所在的公司业务规模扩充得很快，那么你遇到的将更多的是性能、可扩展性方面的问题。笔者将在最后一篇，性能优化和架构篇，着重讲述这些内容。

第15章 运维管理

随着各种技术的快速发展，现今的DBA可以比以前的DBA维护多得多的数据库实例。DBA已经越来越像一个资源的管理者，而不是简单的操作步骤执行人。本章将为读者介绍规模化运维之道。首先，我们讲述规模化的相关知识，然后再简要介绍下服务器的采购，最后，笔者将分享一些运维管理规则，希望能起到抛砖引玉的作用。

15.1 规模化运维

对于机器比较少的公司，我们可能不需要太过关注一些规模化运维的原则，这个时候更值得优化的是人员成本。而在拥有了大量机器之后，我们必须考虑如何高效地运维大规模的数据库主机，这里面有一些要点需要把握，比如资源利用、资源隔离、虚拟化、标准化、自动化等，依据你的生产环境的实际情况，会有不同的侧重点。本文主要是介绍一些思路，实际实现的方法大同小异，按照合适的方法论，你也完全可以构建自己的高效化运维平台。

15.1.1 基础环境

运维有一定规模的数据库机器，需要做到软硬件基础环境的简单化和标准化。拥有稳定的底层，才能确保数据库正常的运行。我们的基础环境要满足一些要点，可以归纳为简单化、标准化、自动化、文档化。这一系列要点有一个根本的目的，那就是尽可能高效地运维数据库机器。

我们需要首先从底层基础设施的标准化开始入手，这是基础，只有标准化了，我们才好做运维平台，开发运维工具。有了标准化的数据，我们才能方便地构建性能模型和容量模型，才能在这个基础之上延伸更多的应用、使平台变得越来越智能，可以说，标准化是智能化的基础。

基础环境配置的标准化和统一，将给后续的运维带来便利，所以务必要在一开始就有步骤地进行实施，保证了基础环境的标准化，才能在后续实现大规模的自动化和信息收集。

基础环境中的一些注意事项如下。

(1) 操作系统的版本要统一，不要追求操作系统的先进性

MySQL推荐运行于Linux下。据统计，90%的MySQL用户使用Linux做生产环境，80%的MySQL用户使用Linux做开发环境，大部分大网站也都使用Linux。各大厂商都大力支持Linux，Linux同时也拥有成熟的开源社区，形成了良好的生态。Linux系统的稳定性很好，各种主流数据库软件，也都在Linux系统上获得了长足的发展，尤其是MySQL。Linux对比其他操作系统，能更快地反映出最新软硬件的发展，对于最新的硬件有更好的支持。所以，如果没有什么特殊的更好的理由，建议大家也在Linux下使用MySQL。

操作系统技术发展到现在，管理越来越简单，特性越来越丰富，但是核心的东西相对变得较少，对于操作系统，笔者觉得应该保守些，我们的数据库服务器是面向企业的，面向海量用户的，上面运行的都是服务级别的应用，稳定性才是最值得看重的。

操作系统上自带的各种应用，如gcc、MySQL，都远远落后于最新的版本，这是一种自然的事情，因为它追求的是稳定性。你可以安装那些新的应用，但这些新的版本可能并未经历相应操作系统版本的大量实际验证，可能还需要不断地修复一些Bug，或者稳定性还有待增强。当然，服务器软件的版本比当前的稳定流行版本超前或滞后太多也可能会有隐患。

架构简单的一个重要前提是标准，应用程序/网络服务器软件使用相同的基础平台，而不是各种版本的操作系统都上。操作系统的不统一将在未来使运维就得很复杂，因为哪怕是一点小小的不同，都可能造成系统管理员的困惑，并不是每个软件的开发者都熟悉各种操作系统，熟悉各种不同版本的操作，并能够让自己的软件版本完美地运行于各种版本的操作系统之上。为了兼容各种操作系统，你可能要进行许多特殊的处理，从而难以做到更完善的自动化，导致可维护性下降。可维护性是一个重要的指标，降低了维护的难度，才能充分考虑扩展性和自动化，才能借助各种开源工具高效地管理服务器。

我们所使用的操作系统应该是应用得最广泛的，配置也应该是基于主流的基础设置，这样可以大大降低学习和维护的难度，学习新系统的也是需要成本的，而且，对于数据库服务器来说，操作系统的各种新特性对于整体系统的性能优化影响并不大。如果更改一个参数对于性能的改善并不是非常大，那么建议尽可能不要去动它。

(2) 应该使用64位的操作系统

64位的操作系统对比32位的操作系统有许多的好处，一般情况下，它的兼容性更好、性能更好、资源利用率更好，所以，建议在生产环境中，不是出于特殊的原因，都应该使用64位的。

(3) 自动化你的部署

许多中小公司，在自动化运维没有发展之前，信息的组织依赖于许多表格，部署过程都是按文档的顺序逐步来进行的，部署完一个服务后，还需要一个长长的检查列表来核对部署是否正确，由于检查列表往往是基于工程师经验的不断累积，是基于前任的经验，由于技术或产品的不断更新，新的问题不断出现，所以这份检查列表（checklist）需要保持持续更新，这些方式对于小规模的机器可能还比较适合，但对大规模的服务部署上下线，人工检查显然是不适合的，大规模的服务部署将需要自动化的检查手段。

规模化的运维，可以通过一些自动化手段，让部署、上下线操作变得更容易，基本上不需要你介入。你能够通过自动检测、自动处理的方式上下线数据库资源。

我们可以定制操作系统、编写脚本，自动化部署各种操作，也有一些开源软件的方案，比如使用puppet进行配置管理。一些公司，还专门设计了应用运维平台、数据库运维平台，在一个统一的平台上进行数据库生命周期内的各种工作。总之，你在部署维护上所耗费的时间越少，你就越有时间针对性地进行系统架构的改造和前瞻性的规划。

(4) 了解你的生产负荷，搭建监控平台，收集一切信息

我们应该熟悉服务是I/O密集型的、内存消耗型的，还是CPU密集型的，对于大规模部署的机器，越了解你的生产负荷，你就越知道它适合部署在什么样的机器上，应该如何充分利用资源，由于历史问题或时间精力的关系，许多公司在最开始发展的时候，往往没有一个标准，针对特定的负荷选择硬件，更多地是凭着个人的经验去采购，在到达一定规模后，逐渐构建了自己的生产负荷的模型，这个时候，采购硬件将更多地依据线上的生产负荷数据。

熟悉你的负荷，你才能提前升级硬件或扩容，对于数据库类的应用，要重点关注内存的资源瓶颈，硬盘、CPU、网络等资源瓶颈往往只会使程序变得缓慢，这点也许能忍受，但一旦出现内存瓶颈，就好比高速行驶的汽车撞上了一堵墙，你可能会碰到无法预料的严重后果，所以，请务必关注内存瓶颈，某些情况下，I/O瓶颈比较突出，可能就是因为内存分配得不够所导致的。

我们应持续不间断地收集信息，在现有数据的基础上，分析趋势，构建模型。有了数据，也方便我们进行性能调优，调整架构设计，从而验证程序变更的效果。

(5) 不要在数据库机器上部署其他服务

复杂的环境将导致整体系统的不稳定性，导致复杂的诊断。

以上5点主要说明了运维数据库机器的一些关注点，这也是早期中小型公司可能犯的错误，特别是最后一点，为了利用资源，在数据库机器上部署其他服务，往往会导致出现更多的问题。

有了好的基础，我们才能适应未来的真正的大规模的数据库主机运维。当你的公司规模变得更大的时候，你的数据库运维成本不会增加太多。

15.1.2 虚拟化

在计算机技术中，虚拟化是一种资源管理技术，是将计算机的各种实体资源，如服务器、网络、内存及存储等，予以抽象、转换然后呈现出来，打破实体结构间的不可切割的障碍，使用户可以用比原本的配置更好的方式来应用这些资源。一般所指的虚拟化资源包括计算能力和数据存储。

虽然虚拟化在一些场景和一些应用中取得了成功，我们也总是说虚拟化节约了成本，但我们有必要思考一下，真的是虚拟化节约了成本呢，还是有其他的因素帮助节约了成本？影响成本的因素有哪些？虚拟化是如何影响成本的？

计算服务器虚拟化的成本时需要考虑4个因素：硬件成本、能源成本、软件成本和人力成本。你需要综合评估虚拟化改造对成本的影响。

市场上，有一种观点，认为虚拟化技术可以大大节省成本，其实，服务器虚拟化技术是否能够带来成本节约及节约多少都取决于自身的架构。如果一台物理机上运行了多个虚拟机，但它的资源利用率并不高，那么其实每个虚拟机的成本也不低，本质上，如果你的程序能够充分利用软硬件资源，那么服务器虚拟化在削减硬件成本方面的成效就不那么明显了。你要确保，在主机上部署多个虚拟机时，增加虚拟机密度所消耗的成本不会超过所得到的收益。

那么，为什么还是有那么多公司宣称虚拟化节省了大量成本呢？这是因为他们的机器规模已经很大了，但利用率的问题一直没有得到解决，软件程序架构一直无法充分利用资源，主要是CPU资源，对于这些应用来说，通过部署大量虚拟机，确实很容易调节硬件的利用率。然而，这并不是说虚拟机大大节省了成本，更准确的说法是，他们之前没有真正地关注服务器利用率的问题。如果能够通过配置软硬件资源达到充分利用硬件资源，那么也许可以达到更好的性价比，毕竟虚拟机在程序和底层硬件中间增加了一个层次，多了一层转换的开销。

目前主要是应用服务器的虚拟化，而数据库的虚拟化还少有人做，原因在于数据库的高I/O负荷难以被隔离，且多个虚拟机对底层存储设备的操作效率不高。另一个需要考虑的因素是数据库的安全性比较高，如果一台普通的物理机宕机，可能会导致上面多个MySQL的数据丢失和损坏，这点是DBA不能接受的。数据库可以

虚拟化的一个场合是你存在大量的小数据库，数据库的QPS很小，没有什么读写，业务场景单一，这种情况下，使用虚拟机可以简化管理成本。

15.1.3 关于去IOE

去IOE是一个比较流行的说法，即去掉IBM、Oracle、EMC这些软硬件设备，以其他的解决方案来代替。IBM的服务器+Oracle数据库+EMC存储是非常流行的组合，大量的企业都在使用这样的架构。但是随着IT领域的不断发展，PC服务器、固态硬盘、开源数据库的推广，人们有了更多的选择，这个时候一些公司开始尝试，替换掉自己企业内部的IOE的某一部分，甚至全部替换掉。这方面比较典型的案例是阿里巴巴的“去IOE运动”。

支持或赞同去IOE的人都不在少数，我认为这个说法有些简单，“口号”可能掩盖了许多问题，传统领域和互联网领域的工作人员、软硬件的协同工作方式存在很大的不同。对于互联网行业，往往一开始就是LAMP架构，使用相对廉价的PC来构建服务，所以这个去IOE的运动更多地是针对传统行业而言，一些本身就属于互联网行业的公司，如果其使用了IOE，那么，可能会逐渐无法满足其不断增长的规模，IBM、Oracle、微软这些大厂商，自身并没有运营大规模系统的经验，提供的解决方案，只适合中小型公司，所以京东、阿里巴巴等公司才会不断地把数据迁移到大规模的MySQL集群上，如果非IOE的方案能够提供更低的成本，更好的性能，那么为什么不去尝试呢？

对于绝大部分传统行业，仍然要回归到商业的本质，你要满是什么要求，达到什么目的，不能为去IOE而去IOE。如果你的系统已经很稳定了，你对自己的生产配置有信心，你的公司也需要稳健，也许你应该再思考下，去IOE所带来的好处和坏处，你应该综合考虑成本，而不是简单地拒绝商业公司的软硬件产品。

使用IOE的好处是，当你的系统中某一环节出现问题时，你能迅速地向其他出现过类似问题的用户请教。同时这三家厂商已经磨合得非常好，在向他们寻求帮助的时候也更简单一些。这样能够把出现错误的几率降到最低，同时为你节省大量的时间。虽然开源解决方案似乎软件成本低，但是同时人员的成本也需要考虑，你需要投入更多的人员培训成本，甚至雇佣一些专业的软件设计人员来解决问题，同时，传统行业公司的业务可能是非常复杂的，开源数据库往往难以达到商业数据库所支持的强大的功能和丰富的特性。

15.1.4 资源利用和隔离

硬件的发展很快，目前单机的性能数据也在不断提升，固态硬盘已经在互联网公司获得大规模的使用，可以说，价格已经不成问题，许多公司都配备了固态硬盘或FLASH卡，相对于传统的机械硬盘，固态硬盘有一个数量级的性能提升，而FLASH卡有2~3个数量级的提升。长期以来困扰DBA的I/O瓶颈问题得到了极大的缓解。内存现在也很便宜，许多数据库主机的内存标准配置已经达到了128GB。Intel的CPU性能也提升得很快，打开超线程后，可以拥有24个、48个甚至96个超线程。

为了充分利用多处理器/多核系统，程序需要有并行运行的能力，也就是说，可以同时在多颗CPU核上运行。早期的官方MySQL版本由于使用了旧的InnoDB引擎，导致扩展性有限，难以充分利用CPU资源，Oracle收购MySQL之后，新的版本MySQL 5.5、5.6和5.7都使用了新的InnoDB引擎，扩展性大大提高。但由于存在一些限制，MySQL实例还是难以充分利用我们的硬件CPU资源。

由于在单机上仅仅部署一个MySQL已经无法充分利用机器了，所以我们往往在一台单机上部署多个MySQL实例以充分利用资源。这样就可能出现各个实例资源争用的情况，因此我们有必要对主机上的MySQL实例做一定程度的隔离。

目前在业内推荐使用的资源隔离的方案是CGroup，它是Linux内核提供的一种资源隔离技术，可以对CPU、内存、I/O等资源进行隔离。CPU和内存相对来说比较好隔离，磁盘I/O则不太好隔离，可以考虑在更上层做限制。如果需要做数据库的云平台，CGroup技术是很实用的，它比基于虚拟机的资源隔离更高效，由于CGroup对内核有要求，而且也比较复杂，所以许多公司并没有使用这项技术，但它已经在一些商用的云平台中得到了使用。

一些DBA使用的是更简单的绑定CPU的策略，通过numactl或task等命令把MySQL实例绑定到某颗CPU上，绑定CPU不仅在一定程度上隔离了CPU资源，通常也能获得比较大的性能提升。建议单机部署多个实例，除了资源利用，还有一个原因，MySQL对于多核CPU的利用率一直不佳。对于官方版本MySQL 5.1，我在生产环境中很少看到能跑满6个核的，随着核数的增加，MySQL的吞吐并不能线性扩展，虽然MySQL/InnoDB一直在改进这个问题，但在可以预计的相当长的一段时间内，MySQL将无法充分利用到目前的8核、12核的CPU，所以我们需要提升MySQL对于CPU的利用率。绑定CPU就是一种比较有效的手段，比如，我们可以使用如下命令绑定mysqld到特定的CPU节点：

```
numactl --cpunodebind=0 --localalloc
```

绑定CPU，要注意冲突，如果你绑定了一颗本来就很繁忙的CPU，那么即使有空闲的CPU，你也利用不上它。

关于NUMA及numactl的详细介绍，请参考18.3.2节。

其他资源也可以进行适当的隔离，比如通过多个IP的方式，把MySQL绑定到不同的网卡上。

以上针对的主要是多实例的资源隔离，我们也可以在数据库上做一些资源限制，MySQL支持对用户的简单的资源限制，比如允许一定时间内运行命令的次数、进行连接的次数，但MySQL的资源管理相对于传统的商业数据库，比如Oracle，还是很粗陋的，没有充分反映连接用户所消耗的资源，比如用户查询扫描的记录数就不知道，所以，仅仅适用于一些特定的场景，比如监控系统所使用的用户，就可以限制一下它的资源使用，以避免监控用户异常影响到生产负载。如果需要达到类似Oracle限制资源的功能，有如下两种策略，一是改造MySQL，让MySQL收集用户的资源使用信息，在用户达到阈值时，自动限制用户对资源的使用，比如降低用

户访问数据库的速度。二是用户程序通过Proxy（中间件、代理）访问数据库，中间件收集资源的使用信息，对照用户的资源限制（连接数、QPS、流量等），对用户访问进行资源隔离，比如中间件可以自动对访问量很大的业务进行限流。

15.1.5 关于备机、备份

对于应用服务器，在大量服务器下，更多的是考虑弹性扩展的能力，可以动态地添加计算资源，这比预留一些备用节点更适合。而对于数据库机器，一般选择主从架构，留一个空闲的备机作备用。

在大批量机器下，许多人会怀疑保留一个完全空闲的备机的合理性。我不确定以后随着技术的发展，是否会有个很好的方案，可以用少得多的机器支撑业务。但目前来说，对于绝大部分企业，使用主从架构，保留一个空闲的从库，是最简单、最稳健的方式。

我比较怀疑国内的公司是不是都严格遵守了“N+1”的策略。一主一从的架构，如果严格执行，可能有许多备用服务器。不过现在的数据库服务器都比较强劲，多实例下，已经节省了许多备用资源。

大量的节点，用于备份中心的投资自然就会很高，但一般来说，对数据进行备份的成本远远小于丢失数据带来的损失。如果你考虑到这一点，那么你将没有理由削减备份的投资。

业内的数据库服务器一般在从库进行备份，但是随着数据越来越大，也需要留意大数据或大量节点下的一个趋势，数据使用副本，不需要定期备份也是可能的。

15.2 服务器采购

服务器采购需要在性能和成本之间做一个平衡，建议读者跟踪使用主流的配置，主流的硬件由于是大批量生产，因此更容易降低成本，比如，我们倾向于使用普通的服务器，双路CPU就足够了，没有必要考虑昂贵得多的4路CPU；再比如，内存条的选购，许多公司选购8GB，而如果是选购16GB一根的内存条，就会贵得多了。也许以后16GB会成为主流，那么8GB反而就不划算了。SSD的大规模使用同样是一个成本不断降低的例子。你需要不断跟踪硬件的发展以挑选最划算的配置。

当我们采购硬件或部署新的系统时，我们可能被要求选择更经济的方式，以合理的成本实现目标的性能要求。影响性能的因素有许多，比如CPU的个数、磁盘的个数、磁盘RAID级别、内存容量、Flash设备的使用方式及文件系统的设置等。

为了实现以最小的成本实现性能的需求，我们可能需要做许多测试和验证，因为我们需要组合许多不同的软硬件的搭配。更具实践性的方式是，依据经验，选择测试某些组合下的性能，最终确定何种配置能够满足你的需求。

如果我们知道最大可能的硬件配置，那么可以按如下步骤选择配置。

- 1) 测试所有组件都是最佳配置时候的性能。
- 2) 逐个改变各个部件的配置，然后测试性能。
- 3) 通过以上步骤，我们可以得出大致的结论，当我们使用更低的成本，减少某个部件的配置时，比如减少内存，我们的性能会损失多少。
- 4) 然后，从最大配置开始，我们逐步调整各种部件的配置，最终得到一个组合，能尽可能以最小的成本实现性能的需求。
- 5) 再次测试，验证这个配置是否满足需要。

每个公司所选择的标配服务器都不尽相同，因为需要契合自己的业务，考虑的角度就会不一样。而且随着市场的变化，主流配置也许很快就过时了，在此就不列举服务器具体配置的例子了。读者可以Google主流互联网公司的配置或和业内的同行进行交流。没有哪一个技术人员可以说自己的配置是最优的，相比较选择最优的配置，如何充分利用现有资源、让硬件资源充分为业务服务才更具有实际意义。

15.3 运维规则

为什么我在基础知识里，增加了一项运维规则的介绍呢？对于运维，除了平台、工具、知识、经验，意识也是非常重要的，有正确的认知、意识，就可以让运维数据库得心应手，又稳又好地运行大规模的数据库集群离不开一些行之有效的规则，可以说，意识在某种程度上决定了我们的运维质量。

以下将重点介绍数据库运维的36条规则。这些规则可能互相之间有冲突，不同的人，可能侧重点也不同，但总体目标是一致的，都是为了服务的质量。读者也可以跳过本节，待有一定的经验后再阅读本节收获会更大。

15.3.1 确保基础网络稳定可靠

因为网络在应用层软件和数据库软件的下一层，因此网络的不可靠，将直接影响到数据库服务器和应用服务器的稳定和性能，网络的复杂性，也可能导致应用软件变得复杂，对此应该有清晰的认识，许多软件架构师或运维人员往往低估了网络对于系统的影响。现实中，许多软件都是基于网络良好的情况下设计的，当碰到复杂的网络问题时，可用性将大大降低。

15.3.2 应构建性能模型，进行容量规划

一些较大的公司，可能有比较完善的性能模型，以尽可能地进行容量规划。而小公司，可能更多地信任监控机制，并没有进行容量规划。随着公司地不断发展，容量模型是需要逐步建立的，至于效果如何，也需要有清晰的认识，现实世界的生产可能比模型复杂多了。

传统行业的容量规划，往往比较固定，可以预知，因此按生产任务来安排即可。而互联网行业有许多变数，业务的增长可能是爆炸式的，新增的业务，有时会资源紧张，有时资源又十分空闲。如果不能从更高的规划角度去管理资源，可能会导致手足无措。作为小的技术团队，可能不太了解高层的实际想法，但也应该尽可能地贯彻传达高层的一些想法、方向，避免导致资源浪费。

容量规划，应该提早发现是否需要扩容，要更主动。需要留有一定的余量，这样才能心中有数、遇事不慌。如果流量突然增长，可能会导致业务受到影响，甚至下线，我们可以理解这也是某种程度的单点，需要尽力避免。

应该把容量规划作为一个常规的工作定期检查。如果有合适的预测模型会更好，但更多情况下可能仍然是基于自己的经验分析，对业务了解得越深，对性能的规划，就会更准确、更有前瞻性。

15.3.3 优先扩容，再考虑优化

尽量不要在容量和性能的高度压力下考虑优化，先扩容，把危险症状降低下来，然后再考虑优化，往往是更靠谱的，除非你有把握，能够在短时间内通过调整让性能瓶颈消失。

15.3.4 保持简单

生产中的异常往往是由复杂性导致的。我们要区分哪些复杂是必然的，哪些是由于“想当然的”或“错误的理解”导致的。比如，跨IDC的网络复杂性就是必然的，需要更复杂的处理策略。而过多的程序数据流层次，就可能是不需要的，层次多了，再加上没有合适的协议约定，往往会导致连锁反应，使诊断困难、开发复杂化。

我们不要因为解决问题，而在你的架构中引入“新的问题”。对于核心架构或算法的调整，往往会导致异常，“回归测试”可以发现一些问题，但更多地依赖于研发人员对于风险的认识，应尽可能地解耦，否则调整的代价太大，引入的问题也会更多。

15.3.5 监控一切

监控一切，记录一切数据。当我们有了数据，才能验证自己的想法，才能辅助我们进行决策。监控的不仅仅是性能数据，也包括了产品、运营、研发各个部门所关心的数据。多记录一些数据，总不会有坏处，有时即使某些数据看起来似乎没有什么用，但在不久的将来，可能就会派上用场了。

15.3.6 处理监控报警

应该注意监控报警是能够采取措施的，或者说，能够找到合适的人来处理的。我们在部署监控平台时，容易犯的一个错误是，报警太多，而有很多报警，却是不需要处理的，每个人每天关注事物的时间总是有限的，所以要注意报警规则的有效性。一些不能处理，或者不需要及时处理的报警，往往属于趋势统计分析的范畴。我们完全可以选择在其他时间段进行处理。如果一个运维工程师频繁收到报警短信，可能就把真正值得关注的信息给忽略了，或者由于太多报警短信，导致他不再查看短信报警，以免严重影响自己的生活和工作。

15.3.7 不要重复“造轮子”

不要重复“造轮子”，也不要什么都从外部获取，如工具、代码、框架等。需要考虑的是在合适的时间以合适成本切入，投资回报率也是需要考虑的。

一般来说，每个公司都存在重复“造轮子”的现象，而且许多人都热衷于此，可能需要用这样的项目来证明自己。但是，他们并没有考虑到一个重要的指标：投入/产出比。如果能够充分利用社区的成果，利用公司已有的成熟框架，那么可以大大加快自己的项目进度，因此，为什么非要自己做一个呢？也许，有些人考虑的是，重复造轮子，可以真正锻炼到团队，毕竟从头开始做一个东西，所累积的经验值可能比一般的项目多得多，往往有助于个人的成长和公司后续的项目。

对于开源产品应该尽量选择国外的产品，笔者这么说有些无奈，虽然国内有许多公司都在拥抱开源，但更多的是个人行为，普遍来说，国内的开源产品，往往缺乏维护，缺乏更高层次的性能、架构和扩展意识，在和国外的开源产品的竞争中，一般都会败下阵来，随着核心人员的流失，或者成员自己的KPI都难以保证，往往不能继续发展下去。所以在使用其他公司的开源产品的时候，特别是缺乏社区参与的产品时，一定要谨慎，最好能够确保自己有足够的能力进行修改，有专门的源码研究人员，否则一旦发生了生产事故，Debug本身也需要时间，更何况是不熟悉代码之下的Debug。

15.3.8 允许出错

允许出错的运维文化，传统的绩效考核（KPI）可能会对此形成不必要的桎梏。人往往从错误中才能得到成长，所以犯一些错误都是可以理解的，关键是我们要建立一套机制，让错误能够尽可能快速地被修复，限制错误影响的范围，并且我们需要能够总结归纳错误，从错误中得到成长，这不仅是个人的成长，也是组织成长的方式。

国内的现状，确实有些片面地放大了故障现象。即使是Google、Facebook、Twitter、Amazon这样的公司，也会偶尔出故障，影响面不一定比国内的公司小。这个世界上，只要存在着硬件载体，就必然伴随着各种各样的故障。有时为了追求高可用性，设计复杂的架构，或者准备过多的冗余设施，往往会导致解决方案的成本剧增，而解决方案的复杂性，可能会增加维护成本和后期改造的难度。国内的众多公司，真正需要99.99%的高可用的到底有多少呢？有多少不能承受的单点故障呢？许多时候，产品才是王道，短期的失效，可能并不会影响到用户的流失。我们应该对可用性进行管理，区别各种服务的等级，对可用性要求高的服务进行专门优化。

有时出现性能问题，往往是一件好事，因为这往往伴随着流量的巨大增长。而在一定的时间内，问题总是可以解决的，我还从来没有碰到过用时间解决不了的技术问题，最重要的是经过问题的解决和总结，经验和技术都能够上一个台阶。

当然，我不是鼓励冒险主义，有计划的冒险才是可取的。在不同的时间段，解决不同的技术问题，往往是对现实的反映。超前或滞后太多，也不可取。

生产环境应该允许犯错误，而且应该是建立在可控的前提下。备份、备份、再备份，保证可回滚，是一个好习惯。

重复性的失误，往往可以找到客观规律，然后用流程、规范和工具避免错误。

失误，往往还出现在周末，出现在非正常升级时间，在打破常规的情况下，在人的体力、智力处于低谷期的时候，将增加故障的概率，毕竟人不是机器，由于生理问题可能会导致出现错误。

所以，很多问题或故障的发生，表面上看是技术、经验问题，但更本质的还是属于人员组织的问题。团队管理者需要知道组员能否适应需要，关注其成长，给予适当压力，但也别过度了，并且还要提供适当的支持。有句话说得好“让合适的人做合适的事”。

15.3.9 设置备用角色

备用角色的作用不容置疑。有备用角色，才可以让我们的工作不被打断，当主要角色请假，或者因为过度劳累，备用角色可以马上启用。这样可以让我们的工作不会陷入被动。

15.3.10 仔细阅读产品文档

在进行任何操作之前，都建议详细阅读文档。产品的说明手册，比如RAID卡的说明文档，就需要仔细阅读，以便选择合适的参数配置。

通常来说，默认的配置并不适合于生产环境，关于数据库的升级，网上可能有各种操作说明文档，但仍会遗漏许多细节，而且在特定的生产环境中什么都有可能发生，因此，要详细阅读相关版本的升级帮助，甚至准备在必要的情况下进行降级的策略。

15.3.11 画数据流图和物理部署图

由于公司有分工，某些人往往只负责部分系统，缺乏对整体系统的把握。有可能的话，应用系统运维工程师应该画出自己的物理部署图，从而了解自己的系统，对于数据流图，软件研发人员也应该将其画出，以便相关人员参考和诊断问题。如果有可能的话，也可以画出整体网络的拓扑图让运维人员进行了解。有了相关的网络、物理部署和数据流图，我们才能更准确地定位问题的所在。

15.3.12 要有版本控制

我们做的所有事情和变更，都应该尽可能地纳入版本管理。文档、应用程序配置、监控配置这些都比较容易实现版本控制，版本管理系统容易管理文档和代码，但其他类型的配置就不容易实现版本控制，比如交换机、路由器、防火墙、操作系统的配置等，我们应该尽可能地实现它们的版本控制。

15.3.13 解决问题要用合适的工具

有些工具比较通用，但对于特定的问题可能就会不适合，有些工具只针对特殊的场景，那么我们就要看，对我们是否真的有用。一般来说，通用的工具只适合初期，到了规模庞大的时候，往往需要针对特定的需求选择特定的工具。对于复杂的问题往往不能轻易确定应该如何选择工具，这个时候，你需要将这个问题分解为一系列的小问题，这样才能方便你选择合适的工具。有些工具可能需要你自己开发，有些使用既有的工具即可，对此应该有一个衡量和评估的过程。

15.3.14 系统工程师要具备定位瓶颈的能力

我们需要监控一切，这样才能预先发现系统的瓶颈。对于一些资源的争用，通过监控系统就能够直观地反映出来。而对于一些隐藏比较深的资源瓶颈和系统瓶

颈，往往需要我们利用各种工具，靠经验去分析和判断。我们需要有意识地尽可能地通过监控系统去发现问题，让监控系统变得越来越智能，越来越少地依赖于人的经验。

运维工程师要分清楚是哪些资源出现了瓶颈，不要混淆了现象和原因。没有足够经验的工程师经常会犯这个错误。

高级工程师和初级工程师有一个很大的区别，高级工程师知道如何去定位瓶颈所在。他们不仅知道如何使用工具，还知道何时、何地、为什么去使用工具，这样，他才有可能在问题爆发之前，就定位到瓶颈所在。那么作为运维工程师，就有必要去训练这种技能。自己测试和验证，然后通过Wiki分享或组内分享都是可以考虑的方式。

定位瓶颈，还需要了解较多的其他领域的知识，因为数据可能要经过很多环节，如本地电脑、浏览器、DNS服务、负载均衡设备、应用服务器等。在自己熟悉的工具和领域之外，了解其他领域大概有一些什么方法和工具是很有帮助的。

15.3.15 确保无线网络的稳定

随着人们工作、生活的变迁，越来越多的人趋向于移动办公，在公司内部，很多人也是用笔记本接入无线网络的，所以需要保证办公无线网络的稳定、方便和安全。

15.3.16 确保访问生产网络时有备用的访问方式

现在许多人是在家里办公或处理故障的，那么公司需要保证在非办公区也能够访问生产网络，员工在外出或旅游的时候，应该带上电脑、上网卡等设备，保证在需要的时间内能够及时响应。

许多公司是使用VPN设备来远程访问生产网络的，VPN设备应该也部署在生产机房中，而不是放在办公网络里。VPN设备不应该是唯一的访问方式，我们应该确保如果VPN设备发生故障，我们仍然能够访问到生产机房。

15.3.17 让优秀的人做工具/平台

许多互联网公司都有基础平台的技术部门，专门负责开发一些基础平台、工具和服务，提供给各个应用研发团队使用。但这往往是一个短期内难以见到效益的事情，许多时候，业务的发展一般，对于一些实现，简单的三板斧就搞定了，自然不需要用到更高效、更具扩展性的产品，所以，对于业务规模不大的公司来说，更多的时候，是在做一些技术储备的事情。基础平台部门往往是伴随着公司的高速发展而壮大的，研发出来的产品和服务如果没有使用，自然就得不到改进，然后就更加没有人使用，这样可能会导致一个恶性循环。这个时候往往是考验高层的决心的时候，是否坚持仍然保留适当比例的底层平台开发人员呢？

应用软件的研发与平台、工具的研发毕竟是不一样的。如果基础不老，其实业务的风险更大。集中人力和时间做一些平台和工具，其实是节省成本的。当然，前提是，你确实有一批高素质的工程师。

我觉得关于这点，大家应该学一学硅谷的一些公司，让优秀的人去做平台和工具，并提供最好的待遇，给予足够的尊重，对于他们的衡量标准也应该不同。

15.3.18 要有分工，每个角色都很重要

实际的大规模数据库机器的运维，离不开训练有素的工程师，他们需要有许多知识、经验和技巧，也必然需要分工，比如有开发数据库运维平台的、专门操作数据库的、专门进行调优的、专门进行源码优化的。我们的团队可能还有项目经理、质量管控、文档工程师、成本分析、培训教育等各个专业领域的人。他们的价值不能被低估，他们在自己专业领域发挥得越出色，团队的总产出就会越高，为什么笔者在此要强调，他们的价值不能被低估呢，因为在现实中，一些非实际运维工作的角色往往不被看重，比如网络安全、质量管理、流程推进等，但正是这些角色，一旦成为整个团队的瓶颈，将会极大地制约着整个团队的服务质量，在大规模化的运维中，他们的作用越显突出。

分工还有另外一层含义，所有需要了解的技术领域，都应该有相应的人在跟进，通过交流和分享，可以研究多得多的知识。

15.3.19 其他团队应能轻松获取生产环境信息

许多公司都存在的一些问题是，运维的生产系统管得太死，导致研发人员不能得知项目的真实运行情况，运维人员的顾虑是，不能让研发直接访问生产环境，如果给了研发人员生产服务器的权限，就会有隐患，但这个问题其实也很好解决，现在的开源监控系统功能强大，一般情况下，是够用的，可以让研发也能通过数据库的监控工具、监控平台直观地看到生产服务器的运行情况。对于生产环境的数据查询，如果有机密数据，那么经过一些处理，也是可以方便地进行访问的。现实中，我们的运维数据，并不仅仅只是提供给研发和测试人员使用的，实际上，我们可以将其转换成更适合其他角色理解的描述方式，让产品、运营等人员也能清楚地知道项目的整体情况，比如服务器的利用率如何，还有多少扩容空间，新活动耗费了多少增加的资源，单台服务器允许多少人在线。所有这些，都需要运维平台越来越成熟。

15.3.20 由独立的系统处理代码性能问题

对于一些难以解决的架构和代码问题，我们需要一套独立的系统来跟踪和处理。因为运维故障处理系统记载的问题，很容易就会被遗忘了。

15.3.21 运维人员应介入产品开发的初期

运维人员应该从产品的设计阶段就跟进，这意味着从一开始就要考虑可靠性、扩展性、维护性和监控。研发人员可能会更多地考虑到功能的扩展，运维人员可能会更偏向于多集群、冗余这些运维架构的考虑。研发人员可能不熟悉硬件的性能或架构，从而导致过度设计，而运维人员熟悉硬件，可以给予研发人员更专业的意见。运维人员要跟进机器的申请和采购，及时通知研发人员采购的进度。任何服务在上线之前，都应该预先部署好监控，运维工程师可以协助研发人员设计监控接口或提供相应的规范让研发人员设计监控，运维人员也要撰写相关的操作文档或向研发人员提供文档的规范。通过运维人员介入软件生命周期的各个阶段，最终形成符合运维标准的文档和产品。

15.3.22 关注安全

初创小公司或中小公司，在安全上往往没有投入人力资源，而是更多地依赖于研发工程师或运维工程师的经验和习惯。在没有独立安全团队的时候，我们要遵循一些安全的经验法则，比如配置文件的独立，研发人员不应该接触到生产环境的密码等机密信息，研发人员也应该尽量避免直接介入生产环境的部署和调试。要有良好的规范，要审查代码，避免给生产环境带来隐患。运维人员应该关注安全方面的信息，在你的网站已经有了巨大商业价值的时候，更需要注重安全。

15.3.23 关注配置管理

配置管理是指对不断变化的软硬件资源进行识别、记录和管理。这方面的内容读者可以参考ITIL相关的图书。现实中存在的一个问题，许多公司存在信息孤岛，从而导致运维、研发等团队的信息不一致。甚至运维部门内部也各自有各自的服务信息记录，无法对服务信息进行标准化和共享，这点将大大阻碍运维平台的建设。

15.3.24 对优先级进行管理

线上业务，应该是有优先级的。我们应该按照紧急程度和影响范围进行分级。对于核心业务，应该有经验丰富的工程师进行管理。如果核心业务发生故障，短时间内解决不了问题，那么我们可能要对事故处理进行“升级”，由经验更丰富的工程师进行处理或由更有权限、掌握更多资源信息的人进行处理。

15.3.25 不要为了优化而优化

不要为了优化而优化，如果不是必须要优化的，就不要去优化。优化肯定要有目标，否则你无法衡量你的优化效果。不要为优化而优化，这样可以减少成本，避免问题的发生。

15.3.26 不要过早优化

过早优化是一切罪恶的根源。我们应该忽略一些微小的不足。比如，对于Web服务器的响应，我们可能只需要关注99%的响应就可以了。对于其他的1%，你应该有一个意识，它可能是因为什么原因而产生的。这样在问题被放大之前，你就能知道有什么办法可以去解决，现实中，知道什么时候优化属于过早优化是高级工程师和初级工程师的一个重要区别。

15.3.27 要有知识分享系统

虽然互联网上几乎有无穷无尽的知识，以及各种解决方案，但对于许多有价值的信息，需要在公司内部积累和分享。比如一些故障处理过程和分析经验，一些公司内部项目的设计，以及新人如何逐步了解各种工作上的知识等。Wiki为知识的积累和分享提供了一个极佳的形式，笔者曾经就在原公司内部撰写了许多文章。通过撰写文档，可以大大提高分享的效率。当然，有了Wiki，系统还需要进行推广使用，我们有必要鼓励内部员工进行文档的撰写和分享。

文档的目的是为了让信息流通更有效率，从而提高工作效率。它应该属于一项需要不断提高的技能，比较常见的问题是，如果业务发展很快，或者产品迭代开发频繁，那么文档往往就不是最新的，许多错误没有被修正，久而久之，IT人员丧失了撰写文档的动力，这里面有工作压力的因素，也有自身技能的因素。我们有必要训练撰写各种文档的技巧，比如，一些需要频繁修改且很容易过时的内容，也许不值得记录或选择另外的形式进行发布会更好。阅读一份错漏百出的文档比没有文档的后果更严重。

15.3.28 参加业内技术论坛

应该多参与业内的交流，一些商业性质的数据库大会，都可以考虑参与。不要惧怕分享自己的经验，对于一些方案，也许其他公司能有更好的解决方案，如果你分享了经验，同行们也会分享经验。从某种角度上看，我们和同行也是竞争者的关系，但是如果你需要发展，就要看看业内的竞争对手在做什么？要跳出公司的格局去看待技术和管理问题。参与业内的技术论坛，也是一种招聘的方式，通过认识更多人，扩大影响力，吸引更多人加入自己所在的公司。参加各种技术论坛也是关注行业技术趋势的一种手段，运维人员应该清楚技术的走向，清楚什么样的产品组合、什么样的框架会更适合于公司的业务。对于这些认识，肯定需要和行业内的人进行交流的。

15.3.29 必须开周会

许多管理者低估了周会和例会的重要性。如果经常不重视周会，那么整个团队可能就会变得松散，没有凝聚力。周会有一个重要的作用就是讨论分工。随着机器规模的扩大，人员的增加，团队管理者需要分工明确，责任到人。对于各种服务和机器，都能找到对应的负责人及后备的负责人。

周会也可以讨论彼此的工作进度，对于未达成或延迟的工作要一起交流对策。

了解小团队内部其他人的工作状态及其他团队的工作情况，传达一些上层的信息，这些都是非常重要的事情。周会也可以用来探讨一些技术问题，交流彼此的研究方向，互相分享，日常的交流分享是不能替代在专门的会议上探讨问题的，因为每个人的工作饱和度都不一样，个性也不一样，固定一段时间进行正式的交流并成为习惯是值得推荐的沟通方式。

15.3.30 积极支持队友，和团队一起成长

一些IT人员因为忙于自己的工作，当团队成员咨询问题或技术建议时，往往不会予以理睬。我能理解IT公司，特别是一些互联网公司，工作强度高的特点，但是如果想要以后工作得更轻松，更重要的是提高整个团队的输出。资深的工程师有必要预留自己的时间用于指导初中级工程师，这也是工作的一部分。如果一个团队，每个人都能预留一定的时间用于内部支持，我相信团队会成长得更快。毕竟每个人都是会互相影响的，在互相帮助的氛围下，工作也会更愉悦，更能够享受在团队工作的乐趣。

15.3.31 从公司的利益出发

我们在做选择和决策的时候，要考虑到公司的商业利益，要考虑到公司确实有这个需求，而不是为了丰富自己的履历，为了挑战自己。简单地说，我们需要的是我们确实需要的，而不是自己想要的。一些互联网公司，在早期使用了.NET的架构，而在发展过程中，发现LAMP的架构更适合企业的发展，那么在选择新的平台、工具和开发方式的时候，也许不得不做一些残酷的选择，为了企业的利益，一些.NET的研发人员可能要面临转型或被淘汰。

15.3.32 确保每个人都是可以被替换的

要确保每个人都是可以被替代的。否则，因为意外变故，很可能导致工作陷入被动。这些需要文档、流程和规范的支持，需要培养备用角色，也需要持续招聘。持续招聘，并不是意味着随时招人，而是需要做好准备，在要招人的时候，有各种渠道去招人。

15.3.33 不要受绩效束缚

关键绩效指标（KPI）是指用于评测组织中与关键目标或关键成功因素相关的那些指标，许多公司在到了一定的规模之后，都把KPI考核作为一项主要的管理工具。

一个事实是，绩效是一种工具，人却是复杂的，管理人是复杂的事情，需要考虑的事情很多，很难靠绩效这个工具来简化所有的问题。我们知道，许多东西在量化之后，就显得比较好管理。有一些职位，比如一些公司高层，或者销售人员，比较好量化一些指标，对于他们来说，量化指标，往往是看得见的数字，但对于一些其他职位，可能就很难量化指标了。本质上，这是一个复杂的问题，仍然需要各种社会工程学的配合。

绩效的设计应该是帮助个人发展，帮助人赢得尊重的，而不是用于桎梏个人的。有人也许会说个人的价值观和公司的价值观有冲突了怎么办？但凡一个好的公司，往往是具备包容性的，而员工如果发现个人的价值观和公司的价值观严重冲突，不能妥协的话，那么还是建议走人的好，继续在一起，对于双方都是损失。

绩效应该随需而变。绩效往往会演变为制定出来的计划。既然是计划，那么就可能会因为市场的变化、竞争对手的变化，而不得不做修正，如果仍然固守旧有的指标，埋头苦干，那么可能会陷入战略上的错误和方向错误，再怎么努力没有用。

推荐大家看一本书《赢》，看看通用的管理大师杰克·韦尔奇是如何看待绩效的。虽然他运用了绩效造就了伟大的文化，但同样有一个不容忽视的背景是，他花了很多年创立了坦诚沟通的企业文化。如果没有坦诚、没有沟通，绩效可能就会成为破坏企业文化的杀手。我们在推动工作进展的时候，不是去考虑对公司是否真的有帮助，而是主要去考虑自己的绩效；自己现有的工作成果，工作输出，决定了自己后续的工作方向，这是一个非常不好的倾向。有时人会有一种执念，自己付出了很多努力，在某件事情上付出了许多时间和人力，就会舍不得放弃，但这样完全没有必要，最好的选择是果断止损，如果有成本更适合的、更有挑战性的、对企业更有效益的事情，那么应该马上换方向。

15.3.34 不断优化流程设计

应该有意识地优化流程设计以提高工作效率和服务质量。随着公司业务的发展，运维部门的不断扩张，如果缺乏合理的流程或缺乏高层次的人才，那么往往会出现人数增多了，效率反而下降了。为什么效率会下降，主要是因为随着公司规模的扩大，所管理和维护的资源急剧膨胀，出于安全和其他的一些考虑，设计了各种各样的流程，以便得到正确的执行结果，但其实这些流程往往会导致效率下降，部门内的沟通成本也会越来越高，这都需要我们对流程本身建立反馈和优化的机制，有意识地不断优化流程。

15.3.35 要了解一些财务知识

许多运维人员不懂财务知识，甚至没有成本意识。这也是有原因的，公司并没有对他们进行一些基础培训，而且许多时候，他们也不清楚各种资源的成本。但是如果你需要为公司谋取最大的利益，那么你就应该对于各种资源、产品和服务的成本有一个大概的认识。管理者有必要让员工了解一些信息。作为一家公司，特别是作为一家上市公司，获取利润是需要着重考虑的，对于成本的支出需要慎重。对于成本的慎重也不是说一定要压缩和节省各种开支，而是说，对于成本的支出，你需要提供足够的数据支撑，并且能够跟随公司的业务发展做出调整和优化。

管理者需要会预算管理，公司的运维部门往往是公司最大的成本支出部门。运维人员申请资源、编制预算的时候，更多地是从技术的角度来考虑，而管理决策层大都不太懂技术，他们更多地是从商务价值的角度去评估预算报告。那么，运维团队的负责人应该使用公司决策层能够理解的语言编制预算报告，需要解释各种预算项目所带来的商务价值回报，IT预算应该服从于商务需求所驱动的费用增长。IT部门不应该只是成本消耗的部门，它更应该是创造价值的部门。

15.3.36 了解其他领域

我们应该了解运维之外的领域，比如产品、运营和市场，了解了其他领域，更有利和其他团队沟通，也有利于开拓自己的视野。以前的公司、部门往往是按照功能垂直划分的，有时难以形成合力，随着管理水平的提高，现代化的公司组织，往往更扁平化，更注重实效，更注重流程优化，决策权也被逐步授予低层次的员工，这个时候，基层员工要有更全面的能力。了解其他的领域，有助于你有更全面的视野，通盘考虑问题，进行流程优化，做出合理的决策。

互联网技术发展得很快，以前MySQL领域的人才很稀缺，但是这些年已经得到了很大的普及，各大公司的数据库运维发展得很快，以前一个人维护几十个库就够了，但以现在的运维技术，可以让一个DBA维护上千台机器，上万个实例。可以看到的是，数据增长得很快，但DBA的需求增长得相对更慢，每一个DBA所要负责的数据库越来越多。

MySQL的入门门槛不高，很容易就能熟悉，一般的中小项目的数据库，资深研发人员或高级运维工程师也可以兼任，所以许多公司并不需要一个专职的DBA，而且许多项目从应用到后端，都托管在云上了，这进一步降低了数据库DBA的需求，未来DBA的重心应该是向开发倾斜，更多地与应用开发部门协作，从后端为程序员提供帮助和指导，更多地向业务靠拢。当然，首先你要确保数据库运维的标准化和自动化。



小结 本章是运维篇的最后一章，讲述了规模化运维需要熟悉的一些知识，介绍了一些运维规则。运维的管理是一个很广的范畴，所以我仅仅列举了一些和数据库运维更相关的值得遵循的规则，任何管理都是相通的，如果大家希望了解更多的管理知识，可以找一些专门的管理领域的图书来阅读。

第五部分 性能调优与架构篇

本篇将为读者介绍性能调优的一些背景知识和理论，然后介绍一些工具的运用，最后介绍从应用程序到操作系统、到数据库、到存储各个环节的优化。

性能调优是一个高度专业的领域，它需要一定的方法论做指导，我们需要有一定的背景知识和方法论做引导，才能提出正确的问题，正确的问题往往意味着有解决问题的可能性，这也是我们在处理各种事务的时候最难知道的。提出正确的问题是一种能力，也是可以训练出来的。本篇将花大量篇幅叙述各种调优方法，并分享笔者从业多年来的一些经验和意识，目的是和大家沟通有无，启发大家更有效率、更智能地解决性能问题。

本书将主要侧重于如何解决问题，而不会深入讲解理论，现实中，大家主要还是依赖经验法则，较少用到理论。但是，一些通用的理论，大家有必要熟悉。作为DBA，性能调优不应该是我们的主要工作，如果对此不是特别有兴趣，那么只要清楚我们所要掌握的能解决问题的知识就可以了。日常运维中，如何保证不出现性能问题，或在性能问题出现之前就将隐患处理掉，才是更值得看重的，对于运行良好的系统，不存在各种高难度的问题要你去处理。

由于架构和性能优化的相关性很大，因此也合并在本篇一并讲述。

第16章 基础理论和工具

本章首先讲述性能调优的一些概念和理论，然后介绍一些工具的使用，最后介绍一些性能调优的方法。

16.1 性能调优理论

16.1.1 基础概念

对于一些概念，可能不同的人有不同的解释，基本上没有一个很严格的标准答案，笔者将在此阐释个人理解的一些基本概念，以方便大家在阅读本书时，对照概念，理解我所讲述的内容。

资源（resource）：物理服务器的功能组件，一些软件资源也可以被衡量，比如线程池、进程数等。系统的运行，需要各种资源，对于资源列表的确定，我们可以凭借对系统的了解来确定，也可以通过绘制系统的功能块图的方式来确定要衡量的资源。

常见的物理资源如下所示。

·CPU、CPU核数（core）、硬件线程（hardware thread）、虚拟线程（virtual thread）

·内存

·网络接口

·存储设备

·存储或网络的控制器

·内部高速互联

负载（load）：有多少任务正在施加给系统，也就是系统的输入，要被处理的请求。对于数据库来说，负载就包括了客户端段发送过来的命令和查询。

负载如果超过了设计能力，往往会导致性能问题。应用程序可能会因为软件应用的配置或系统架构导致性能降低，比如，如果一个应用程序是单线程的，那么无疑它会受制于单线程架构，因为只能利用一个核，后续的请求都必须排队，不能利用其他的核。但性能下降也可能仅仅是因为负载太多了。负载太多将导致排队和高延时，比如，一个多线程应用程序，你会发现所有的CPU都是忙碌的，都在处理任务，这个时候，仍然会发生排队，系统负载也会很高，这种情况很可能是施加了过高的负载。

如果在云中，你也许可以简单地增加更多的节点来处理过高的负载，在一般的生产应用中，简单地增加节点有时解决不了问题，你需要进行调优和架构迭代。

负载可以分成两种类型：CPU密集型（CPU-bound）和I/O密集型（I/O-bound）。

·CPU密集型指的是那些需要大量计算的应用，它们受CPU资源所限制，也有人称为计算密集型或CPU瓶颈型。

·I/O密集型指的是那些需要执行许多I/O操作的应用，例如文件服务器、数据库、交互式shell，它们期望更小的响应时间。它们受I/O子系统或网络资源所限制。

对于CPU密集型的负载，可以检查和统计那些CPU运算的代码，对于I/O密集型的负载，可以检查和统计那些执行I/O操作最多的代码。这样就可以更有针对性地进行调优。我们可以使用系统自带的工具或应用程序自己的性能检测工具来进行统计和分析。

对于吞吐率，很显然，数据库支持的简单查询的吞吐率会比复杂查询的吞吐率大得多，其他应用服务器也是类似的，简单的操作执行得更快，所以对于吞吐，我们也需要定义我们的系统应处理何种负载。

利用率（utilization）：利用率用于衡量提供服务的资源的忙碌程度，它是基于某一段时间间隔内，系统资源用于真正执行工作的时间的百分比。即，

利用率=忙的时间/总计时间

利用率可以是基于时间的，比如CPU的利用率：某颗CPU的利用率或整体系统的CPU利用率。比如对于磁盘的利用率，我们可以使用iostat命令检查%util。

利用率也可以是基于容量的，它可以表示我们的磁盘、内存或网络的使用程度，比如90%的磁盘空间被使用，80%的内存被使用，80%的网络带宽被使用等。

可以用高速公路收费站的例子来进行类比。

利用率表现为当前有多少收费亭正在忙于服务。利用率100%，就表示所有的收费亭都正在处理收费，你找不到空闲的收费亭，因此你必须排队。那么在高峰时刻，可能许多时候都是100%的利用率，但如果给出全天的利用率数据，也许只有40%，那么如果只关注全天的这个利用率数据就会掩盖一些问题。

往往利用率的高位会导致资源饱和。利用率100%往往意味着系统有瓶颈，可以检查资源饱和度和系统性能加以确定。该资源不能提供服务的程度被标识为它的饱和度，后文有资源饱和度的详细解释。

如果是检测的粒度比较大，那么很可能就会掩盖了偶尔的100%的峰值，一些资源，如磁盘，在60%的利用率的时候，性能就开始变差了。

响应时间（response time）：也叫延迟，指操作执行所需要的耗时。它包括了等待时间和执行时间，优化执行时间相对简单，优化等待时间则复杂多了，因为要考虑到各种其他任务的影响，以及资源的竞争使用。对于一个数据库查询，响应时间就包括了从客户端发布查询命令到数据库处理查询，以及传输结果给客户端的所有时间。延迟可以在不同的环节进行衡量，比如访问站点的装载时间，包括DNS延迟、TCP连接延迟和TCP数据传输时间。延迟也可以在更高的级别进行理解，包括数据传输时间和其他时间，比如从用户点击链接到网页内容传输，并在用户的电脑屏幕上渲染完毕。延迟是以时间做量度来衡量的，可以很方便地进行比较，其他的一些指标则不容易衡量和比较，比如IOPS，你可以将其转化为延迟来进行比较。

一般情况下，我们衡量性能主要是通过响应时间，而不是使用了多少资源，优化本质上是在一定的负载下，尽可能地减少响应时间，而不是减少资源的占用，

比如降低CPU的使用。资源的消耗只是一个现象，而不是我们优化的目标。

如果我们能够记录MySQL在各个环节所消耗的时间，那么我们就可以有针对性地进行调优，如果我们可以将任务细分为一些子任务，那么我们就可以通过消除子任务、减少子任务的执行次数或让子任务执行得更有效率等多种手段来优化MySQL。

伸缩性（scalability）：对于伸缩性，有两个层面的意思。一是，在资源的利用率不断增加的情况下，响应时间和资源利用率之间的关系，当资源利用率升高时，响应时间仍然能够保持稳定，那么我们就说它的伸缩性好，但是如果资源利用率一旦升高，响应时间就开始劣化，那么我们认为其伸缩性不佳。二是，伸缩性还有一层意义，表征系统不断扩展的能力，系统通过不断地增加节点或资源，处理不断增长的负载，同时依然能够保持合理的响应时间。

吞吐率（throughput）：处理任务的速率。对于网络传输，吞吐率一般是指每秒传输的字节数，对于数据库来说，指的是每秒查询数（QPS）或每秒事务数。

并发（concurrency）：指的是系统能够并行执行多个操作的能力。如果数据库能够充分利用CPU的多核能力，那么往往意味着它有更高的并发处理能力。

容量（capacity）：容量指的是系统可以提供的处理负荷的能力。我们在日常运维中有一项很重要的工作就是容量规划，即确保随着负荷的增长，我们的系统仍然能够处理负荷，确保服务良好和稳定。容量也指我们的资源使用极限，比如我们的磁盘空间占用，在磁盘空间到达一定的阈值后，我们可能还要考虑扩容。

饱和（saturation）：由于负荷过大，超过了某项资源的服务能力称为饱和。饱和度可以用等待队列的长度来加以衡量，或者用在队列里的等待时间加以衡量。超过承载能力的工作往往处于等待队列之中或被返回错误，比如CPU饱和度可用平均运行队列（runq-sz）来衡量，比如可以使用iostat命令输出的avgqu-sz指标衡量磁盘饱和度。比如内存饱和度可以用交换分区的一些指标来衡量。

资源利用率高时，可能会出现饱和，图16-1是一个资源利用率、负载、饱和之间关系的说明图，在资源利用率超过100%后，任务不能马上被处理，需要排队，饱和度就开始随着负载的增加线性增长。饱和将导致性能问题，因为新的请求需要排队，需要时间进行等待。饱和并不一定要在利用率100%的时候才会发生，它取决于资源操作的并行度。

饱和不一定能被发现，生产环境监控系统、监控脚本时存在一个容易犯的错误，那就是采样的粒度太粗，比如每隔几分钟进行采样，可能就会发现不了问题，但问题却会发生在短时间的几十秒内。突然的利用率的高峰也很容易导致资源饱和，出现性能问题。

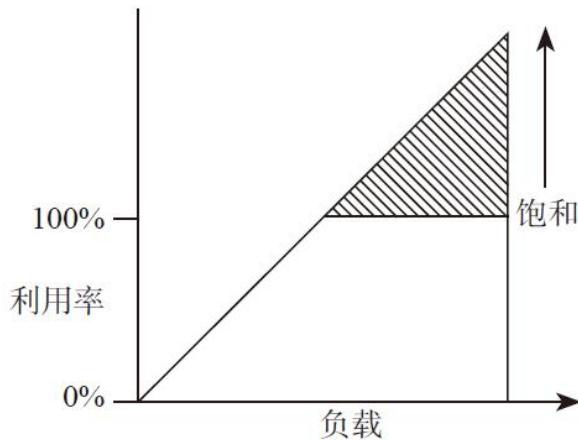


图16-1 资源利用率、负载、饱和之间的关系

我们需要熟悉以上概念，并了解它们之间的关系，一般来说，随着负载的上升，吞吐率也将上升，吞吐曲线开始时会一直是线性的，我们的系统响应时间在开始的一个阶段会保持稳定，但是到达某个点后，性能就会开始变差，响应时间变得更长，以后随着负载的继续增加，此时我们的吞吐率将不能再继续增长，甚至还会下降，而响应时间也可能会变得不可接受。有一种例外情况是，应用服务器返回错误状态码，比如Web服务器返回503错误，由于基本上不消耗资源，难以到达极限，所以返回错误码的吞吐曲线会保持线性。

对于性能的看法其实比较主观，一个性能指标是好还是坏，可能取决于研发人员和终端用户的期望值。所以，如果我们要判断是否应该进行调优，那么我们需要对这些指标进行量化，当我们量化了指标，确定了性能目标时，这样的性能调优才更科学，才更容易被理解和沟通一致。

以下将简要叙述三个基础理论：阿姆达尔定律、通用扩展定律和排队论。

16.1.2 阿姆达尔定律

阿姆达尔定律（Amdahl's law）是计算机科学界的一项经验法则，因IBM公司的计算机架构师吉恩·阿姆达尔而得名。吉恩·阿姆达尔在1967年发表的论文中提出了这个重要定律。

阿姆达尔定律主要用于发现当系统的部分组件得到改进，整体系统可能得到的最大改进。它经常用于并行计算领域，用来预测应用多个处理器时理论上的最大加速比。在性能调优领域，我们利用此定律有助于我们解决或缓解性能瓶颈问题。

阿姆达尔定律的模型阐释了我们在现实生产中串行资源争用时候的现象。图16-2分别展示了线性扩展（linear scaling）和按阿姆达尔定律扩展的加速比（speedup）。图16-2中的曲线是符合阿姆达尔定律的加速比曲线。在一个系统中，不可避免地会有一些资源必须要串行访问，这就限制了我们的加速比，即使我们增加了并发数（横轴），但取得的效果并不理想，难以获得线性扩展的能力（图16-2中的直线）。

以下介绍中，系统、算法、程序都可以看作是优化的对象，笔者在此不会加以区分，它们都有串行的部分和可以并行的部分。

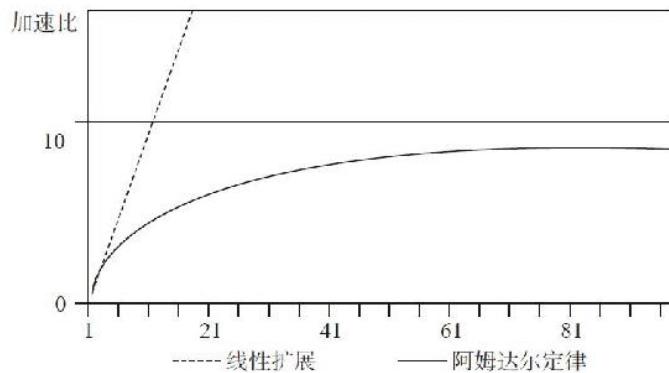


图16-2 阿姆达尔定律下的加速比对比线性扩展下的加速比

在并行计算中，使用多个处理器的程序的加速比受限制于程序串行部分的执行时间。例如，如果一个程序使用一个CPU核执行需要20个小时，其中的部分代码只能串行，需要执行1个小时，其他19个小时的代码执行可以并行，那么，如果不考虑有多少CPU可用来并行执行程序，最小的执行时间也不会少于1个小时（串行工作的部分），因此加速比被限制为最多20倍（20/1）。

加速比越高，证明优化效果越明显。

阿姆达尔定律可以用如下公式表示：

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{T(1)\left[B + \frac{1}{n}(1-B)\right]} = \frac{1}{B + \frac{1}{n}(1-B)}$$

其中，

S(n): 固定负载下，理论上的加速比。

B: 串行工作部分所占比例，取值范围为0~1。

n: 并行线程数、并行处理节点个数。

以上公式具体说明如下。

加速比=没有改进前的算法耗时T(1)/改进后的算法耗时T(n)。

我们假定算法没有改进之前，执行总时间是1（假定为1个单元）。那么改进后的算法，其时间应该是串行工作部分的耗时（B）加上并行部分的耗时(1-B)/n，由于并行部分可以在多个CPU核上执行，所以并行部分实际的执行时间是(1-B)/n

根据这个公式，如果并行线程数（我们可以理解为CPU处理器数量）趋于无穷，那么加速比将与系统的串行工作部分的比例成反比，如果系统中有50%的代码需要串行执行，那么系统的最大加速比为2。也就是说，为了提高系统的速度，仅增加CPU处理器的数量不一定能起到有效的作用，需要提高系统内可并行化的模块比重，在此基础上合理增加并行处理器的数量，才能以最小的投入得到最大的加速比。

下面对阿姆达尔定律做进一步说明。阿姆达尔这个模型定义了固定负载下，某个算法的并行实现相对串行实现的加速比。例如，某个算法有12%的操作是可以并行执行的，而剩下的88%的操作不能并行，那么阿姆达尔定律声明，最大加速比是 $1 / (1 - 0.12) = 1.136$ 。如上公式中的n趋向于无穷大，那么加速比 $S=1/B=1 / (1 - 0.12)$ 。

再例如，对于某个算法，可以并行的比例是P，这部分并行的代码能够加速S倍（S可以理解成CPU核的个数，即新代码的执行时间为原来执行时间的1/S）。如果此算法有30%的代码可以被并行加速，即P等于0.3，这部分代码可以被加速2倍，即S等于2。那么，使用阿姆达尔定律计算其整个算法的加速比如下。

$$\frac{1}{(1-P) + \frac{P}{S}} = \frac{1}{(1-0.3) + \frac{0.3}{2}} = 1.176$$

以上公式和前一个公式是类似的，只是前一个公式的分母是用串行比例B来表示的。

再例如，某项任务，我们可以分解为4个步骤，P1、P2、P3、P4，执行耗时占总耗时百分比分别是11%、18%、23%和48%。我们对它进行优化，P1不能优

化，P2可以加速5倍，P3可以加速20倍，P4可以加速1.6倍。那么改进后的执行时间计算如下。

$$\frac{0.11}{1} + \frac{0.18}{5} + \frac{0.23}{20} + \frac{0.48}{1.6} = 0.4575$$

总的加速比是 $1/0.4575=2.186$ 。我们可以看到，虽然有些部分加速比有20倍，有些部分有5倍，但总的加速比并不高，略大于2，因为占时间比例最大的P4部分仅加速了1.6倍。

图16-3演示了并行工作部分的比例不同时的加速比曲线，我们可以观察到，加速比受限制于串行工作部分的比例，当95%的代码都可以进行并行优化时，理论上的最大加速比会更高，但最高不会超过20倍。

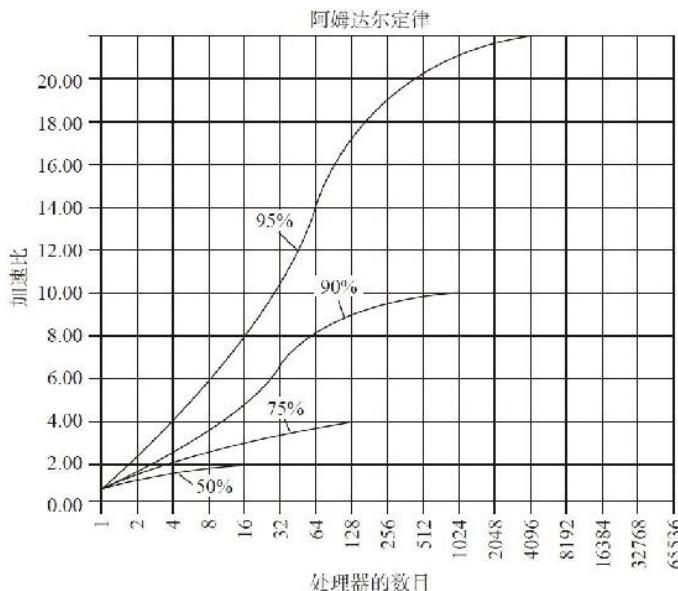


图16-3 并行工作部分的比例不同（50%、75%、90%、95%）时的加速比

阿姆达尔定律也用于指导CPU的可扩展设计。CPU的发展有两个方向，更快的CPU或更多的核。目前看来发展的重心偏向于CPU的核数，随着技术的不断发展，CPU的核数在不断地增加，目前我们的数据库服务器配置四核、六核都已经比较常见了，但有时我们会发现虽然拥有更多的核，当我们同时运行几个程序时，只有少数几个线程处于工作中，其他的并未做什么工作。实践当中，并行运行多个线程往往并不能显著地提升性能，程序往往并不能有效地利用多核。在多核处理器中加速比是衡量并行程序性能的一个重要参数，能否有效降低串行计算部分的比例和降低交互开销决定了能否充分发挥多核的性能，其中的关键在于：合理划分任务、减少核间通信。

16.1.3 通用扩展定律

可扩展性指的是，我们通过不断地增加节点来满足不断增长的负载需求，这样的一种能力。可是，很多人提到了可扩展性，却没有给它一个清晰的定义和量化标准。实际上，系统可扩展性是可以被量化的，如果你不能量化可扩展性，你就不能确保它能够满足需要。USL (universal scalability law, 通用扩展定律) 就提供了一种方式，让我们可以量化系统的可扩展性。

USL，即通用扩展定律，由尼尔·巩特尔博士提出，相对比阿姆达尔定律，USL增加了一个参数 β 表示“一致性延迟”(coherency delay)。图16-4是它的模型图，纵轴表示容量，横轴表示并发数。

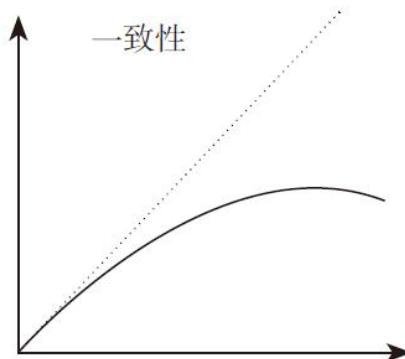


图16-4 USL模型图

USL可以用如下公式进行定义。

$$C(N) = \frac{N}{1 + \alpha((N-1) + \beta N(N-1))}$$

其中，

$C(N)$: 容量。

$0 \leq \alpha, \beta < 1$ 。

α : Contention, 争用的程度, 由于等待或排队等待共享资源, 将导致不能线性扩展。

β : Coherency, 一致性延迟的程度, 由于节点之间需要交互以使数据保持一致, 因此会带来延迟。为了维持数据的一致性, 将导致系统性能恶化, 即随着N的上升, 系统吞吐率反而会下降。当这个值为0时, 我们可以将其看作是阿姆达尔定律。

N: Concurrency, 并发数, 理想情况下是线性扩展的。如果是衡量软件的可扩展性, 那么N可以是客户端/用户并发数, 在固定的硬件配置下(CPU数不变), 不断增加客户端/用户, 以获取性能吞吐模型, 我们的压力测试软件, 如LoadRunner、sysbench即为此类。如果是衡量硬件的可扩展性, 那么N可以是CPU的个数, 我们不断增加CPU的个数, 同时保持每颗CPU上的负载不变, 即如果每颗CPU施加100个用户的负载, 每增加一颗CPU, 就增加100个用户, 那么一台32个CPU的机器, 需要3200个用户的并发负载。

下面我们来看看图16-5到图16-8所示的4个图, 对应在不同负载下容量(吞吐能力)的变化。

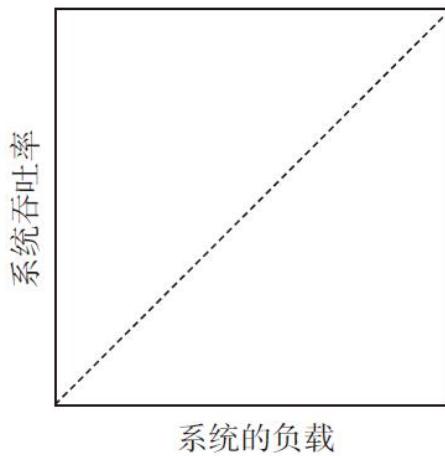


图16-5 $\alpha=0, \beta=0$ 时的容量变化

图16-5中, $\alpha=0, \beta=0$, 此时, 随着负载的升高, 系统吞吐是线性上升的, 即我们所说的线性扩展, 这是很理想化的一种情况, 每份投入必然会获得等值回报, 但很难无限进行下去, 性能模型的前面部分可能会表现为线性扩展。

图16-6中, $\alpha>0, \beta=0$, 此时对于共享资源的争用将导致性能曲线不再线性增长。

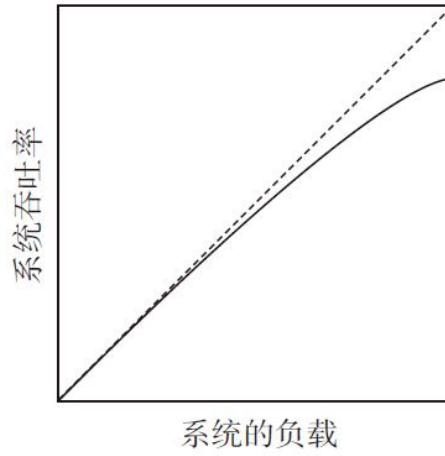


图16-6 $\alpha>0, \beta=0$ 时的容量变化

图16-7中, $\alpha>0, \beta=0$ 。此时共享资源的争用大大增加, 我们将看到一种“收益递减”的现象, 即我们的持续投入资源(比如金钱)变大, 但是所取得的收益都越来越小。

图16-8中, $\alpha>0, \beta>0$ 。此时 β 参数开始影响我们的性能曲线, 我们除了共享资源的争用, 还需要应对系统内各个节点的通信、同步状态的开销。此时性能曲线将会变差, 回报趋向于负值。

USL应用很广，如压力测试工具结果分析，对磁盘阵列、SAN和多核处理器及某些类型的网络I/O建模，分析内存颠簸、高速缓存未命中导致的延时等场景。由于它的应用范围很广，所以也称之为通用扩展定律。

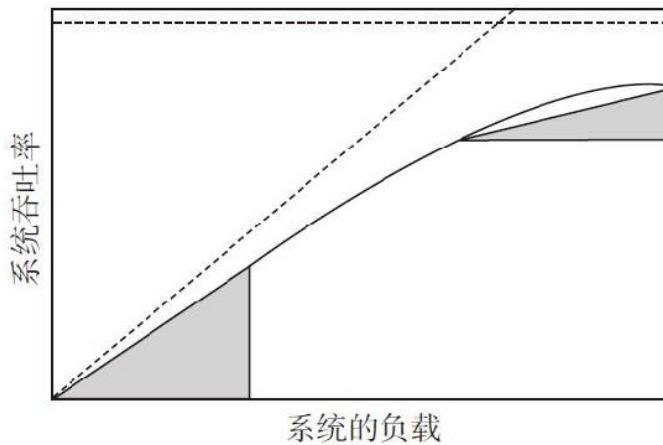


图16-7 $\alpha>0$ 、 $\beta=0$ 时的容量变化

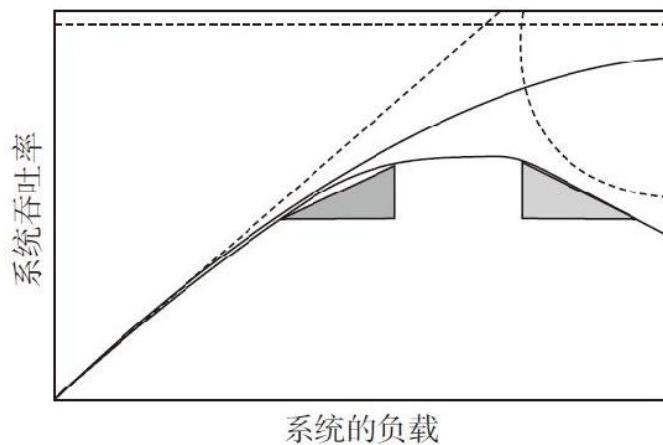


图16-8 $\alpha>0$ 、 $\beta>0$ 时的容量变化

USL一些具体的应用场景如下。

(1) 模拟压力测试

以下例子，如图16-9所示，将不断增加虚拟用户（横轴表示的Virtual users），记录其吞吐率（纵轴表示的Throughput），然后通过绘制的图形得到性能吞吐的模型。

(2) 检测错误的测量结果

有时我们进行测试，会发现我们的测量输出结果不符合模型，这时我们需要审视下，是否我们的测量方式存在问题或受到了其他因素的干扰？需要找出是什么原因导致的非预期的行为。

(3) 性能推断

如果扩展性很差，我们可以通过公式和图得知是 α （对共享资源的争用）还是 β （一致性延迟）应该承担更大的责任。

(4) 性能诊断

USL公式虽然简单易用，但普通人也许无法从中找到解决问题的思路和方法。因为所有的信息都被浓缩为2个参数 α 和 β 。然而，应用程序开发者和系统架构师可能依据这些信息，就能轻易地找到问题症结所在。

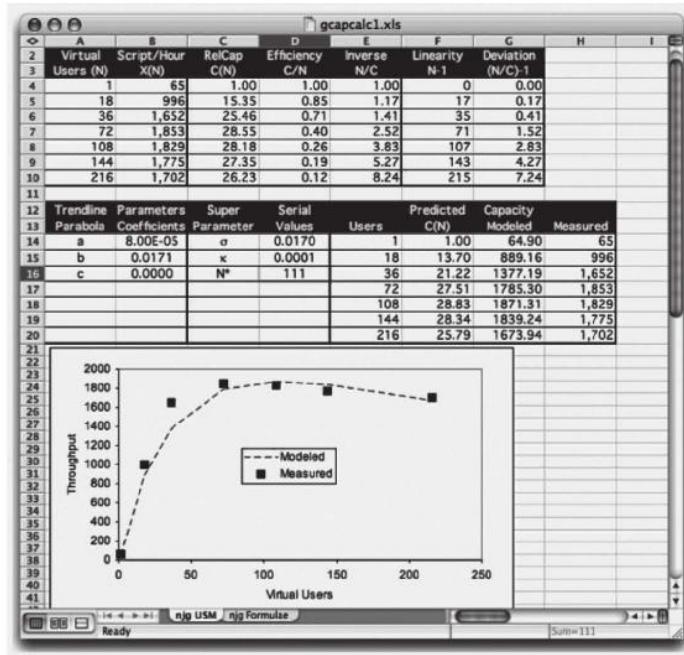


图16-9 模拟压力测试

(5) 对生产环境收集的性能数据进行分析

对生产环境的性能数据（图形）进行分析，可以让我们确定合适的工作负载（比如并发线程数、CPU个数）。

(6) “扩展区”（scalability zone）概念的应用

我们看下图16-10，我们绘制不同场景下的性能曲线，这些曲线定义了可扩展区域Async msging、Sync waiting、Sync thrashing。程序的性能点图跨越了多个区域。在超过15个并发的时候，性能曲线进入另外一个区域Sync waiting，扩展性变差，这是因为“同步排队”（synchronous queueing）的影响所导致的。

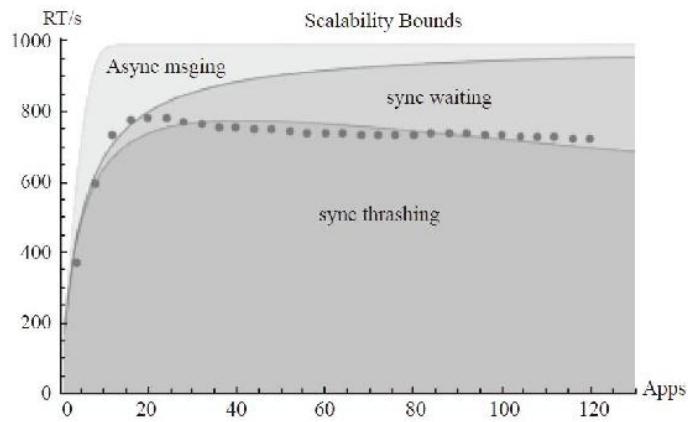


图16-10 可扩展区域

16.1.4 排队论

(1) 排队论的历史

排队论（queueing theory）起源于20世纪初的电话通话。Agner Krarup Erlang，一个在丹麦哥本哈根电话交换局工作的工程师，通过研究人们打电话的方式，发明了人们需要等待多久的公式，厄朗发表了一篇著名的文章“自动电话交换中的概率理论的几个问题的解决”。随着以后的发展，排队论成为数学中一门重要的学科，20世纪50年代初，大卫·坎达（David G.Kendall）对排队论做了系统的研究，使排队论得到了进一步的发展。

(2) 定义

排队论也称为随机服务系统理论、排队理论，是数学运筹学的分支学科。它是研究服务系统中排队现象随机规律的学科。排队论广泛应用于电信、交通工程、计算机网络、生产、运输、库存等各项资源共享的随机服务系统和工厂、商店、办公室、医院等的设计。

排队是我们每个人都熟悉的现象。因为为了得到某种服务必须排队。有一类排队是有形的，例如在售票处等待买票的排队，加油站前汽车等待加油的排队等；还有一类排队是无形的，例如电话交换机接到的电话呼叫信号的排队，等待计算机中心处理机处理的信息的排队等。为了叙述的方便，排队者无论是人、物或信息，以后都统称为“顾客”。服务者无论是人或事物，例如一台电子计算机也可以是排队系统中的服务者，以后都统称为“服务台”。

排队现象是我们不希望出现的现象，因为人在排队至少意味着是在浪费时间；物的排队则说明了物资的积压。但是排队现象却无法完全消失，这是一种随机现象。顾客到达间隔时间和为顾客服务时间的随机性是排队现象产生的主要原因。如果上述的两个时间都是固定的，那么我们就可以通过妥善安排来完全消除排队现象。

排队论是研究排队系统在不同的条件下（最主要的是顾客到达的随机规律和服务时间的随机规律）产生的排队现象的随机规律性。也就是要建立反映这种随机性的数学模型。研究的最终目的是为了运用这些规律，对实际的排队系统的设计与运行做出最优的决策。

(3) 排队论的一般模型

图16-11是排队论的一般模型图。

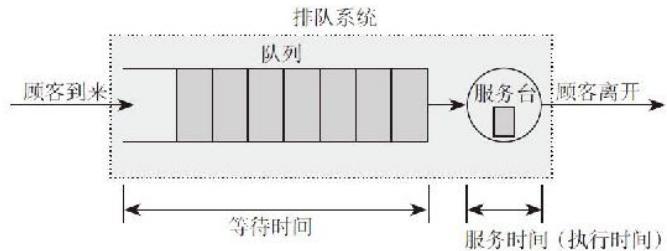


图16-11 排队论模型

其中，服务台用于服务队列中的顾客，可以多个服务台并发工作。

图16-11中的排队系统，各个顾客从顾客源出发，随机地来到服务机构，按一定的排队规则等待服务，直到按一定的服务规则接受服务后离开排队系统。

对于一个服务系统来说，如果服务机构过小，以致不能满足要求服务的众多顾客的需要，那么就会产生拥挤现象而使服务质量降低。因此，顾客总是希望服务机构越大越好，但是，如果服务机构过大，人力和物力方面的开支就会相应地增加，从而就会造成浪费，因此研究排队模型的目的就是要在顾客需要和服务机构的规模之间进行权衡和决策，使其达到合理的平衡。

(4) 理论归纳

在计算机领域，许多软硬件组件都可以模型化为排队系统。我们可以使用排队理论分析排队现象，分析队列的长度、等待时间、利用率等指标。

排队理论基于许多数学和统计理论，比如概率理论、随机过程理论、Erlang-C公式（Erlangs C formula）、李特尔法则。

以下简单介绍排队论的一些理论。详细的内容可参考Neil J.Gunther的图书《The Practical Performance Analyst》。

李特尔法则（Little's law）

李特尔法则可以用如下公式来表示。

$$L = \lambda W$$

这个公式定义了一个系统中的平均访问请求数=平均到达速率×平均服务时间

比如，我们有一个系统，平均到达速率是10000次请求/s，每个请求需要花费0.05s来处理，即平均服务时间为0.05s，那么根据李特尔法则，服务器在任何时刻都将承担 $10000 \times 0.05 = 500$ 个请求的业务处理。如果过了一段时间，由于客户端流量的上升，并发的访问速率达到20000次请求/s，这种情况下，我们该如何改进系统的性能呢？根据李特尔法则，我们有如下两种方案。

- 1) 提高服务器的并发处理能力，即 $20000 \times 0.05 = 1000$ 。
- 2) 减少服务器的平均服务时间，即 $W = L/\lambda = 500/20000 = 0.025s$ 。

排队论表示法

我们可以用肯德尔表示法（Kendall's notation）来对排队系统进行分类，肯德尔表示法可使用如下的简化形式：

A/S/m

其中，

A: 到达的规则，即到达的时间间隔的分布，可能是随机的、确定型的或泊松分布等其他分布方式。

S: 服务规则，即指服务时间的分布，可能是固定的或指数的等其他分布方式。

m: 服务台个数，一个或多个。

表示顾客到达的间隔时间和服务时间的分布常用的约定符号分别如下。

M: 指数分布，在概率论和统计学中，指数分布（Exponential Distribution）是一种连续概率分布。指数分布可以用来表示独立随机事件发生的时间间隔，比如旅客进机场的时间间隔、中文维基百科新条目出现的时间间隔，等等。

D: 确定型（Deterministic）。

G: 一般（General）服务时间的分布。

一些常见的排队系统模型具体如下。

·M/M/1：表示顾客相继到达的间隔时间为指数分布、服务时间为指数分布、单服务台。

·M/M/c：表示顾客相继到达的间隔时间为指数分布、服务时间为指数分布、多服务台。

·M/G/1：表示顾客相继到达的间隔时间为指数分布、服务时间为一般服务时间分布、单服务台。

·M/D/1：表示顾客相继到达的间隔时间为指数分布、服务时间为确定型时间分布、单服务台。比如我们的旋转磁盘可用此模型进行分析。

16.2 诊断工具

我们需要熟悉Linux下常用的诊断性能的工具，确切地说，我们需要在平时不使用这些命令的时候，就能够熟练应用它，这样我们在实际诊断性能问题的时候，才可以快速使用它们，而不是事到临头才去学某个命令应该如何使用。

我们进行性能调优的首要目的是需要找到系统的瓶颈所在。最常见的瓶颈是内存、I/O或CPU。Linux提供了一系列的工具来检查系统和查找瓶颈。一些工具揭示了系统的总体健康状态，一些工具则提供了特定的系统组件信息。使用这些工具将是一个好的起点，有助于我们确定性能调优的方向。

现实中，真正出现性能问题时，往往会有许多现象发生，发现一个现象并不难，难的是定位问题的根源，是什么因素的影响最大。当你具备了知识，能够熟练使用各种工具，了解数据库、操作系统、硬件等各种组件的机制，通过检查各种工具和命令输出的数据，你对找到问题的根源所在将会越来越有经验。

使用工具要避免只使用自己熟悉的工具，因为工具在不断地进化中，所以，如果有了更好的工具，那么花一些学习成本也是值得的。

对于性能的优化，我们往往有许多种工具可以选择，这也会造成一些困扰，因为不同工具的功能有重叠，甚至大部分都有重复，这样不仅浪费了资源，也让用户的学习成本变得更高，因为可能你要熟悉许多工具而不只是一种。

16.2.1 OS诊断工具

sar、vmstat、iostat都是工具包sysstat（the system monitoring tool）里的命令，如果你的系统中没有这些命令，那么你需要安装sysstat包。

本节对sar会做比较详细的介绍，因为其他命令收集的信息与它类似，因此本节将不对其他命令做详细说明，仅仅列出一些需要关注的要点。

1.sar

sar（system activity reporter，系统活动情况报告）命令是系统维护的重要工具，主要用于帮助我们掌握系统资源的使用情况，可以从多方面对系统的活动进行报告，报告内容包括：文件的读写情况、系统调用的使用情况、磁盘I/O、CPU效率、内存使用状况、进程活动及与IPC有关的活动等。

sar通过cron定时调用执行以收集和记录信息，默认情况下，Linux每10分钟运行一次sar命令来收集信息，如果你认为时间跨度太长，不容易发现性能问题，你也可以更改调度任务的间隔，修改/etc/cron.d/sysstat即可。

```
cat sysstat
# run system activity accounting tool every 10 minutes
*/10 * * * * root /usr/lib64/sa/sa1 1 1
```

然后重启crond生效。

```
/etc/init.d/crond restart
```

sar命令的常用格式如下。

```
sar [ options... ] [ <interval> [ <count> ] ]
```

`sar`如果不加参数，则默认是读取历史统计信息，你可以指定`interval`和`count`对当前的系统活动进行统计。其中参数的具体说明如下。

`interval`为采样间隔，`count`为采样次数，默认值是1。

`options`为命令行选项，`sar`命令常用的选项分别如下。

`-A`: 所有报告的总和。

`-u`: 输出CPU使用情况的统计信息。

`-v`: 输出inode、文件和其他内核表的统计信息。

`-d`: 输出每一个块设备的活动信息，一般添加选项`-p`以显示易读的设备名。

`-r`: 输出内存和交换空间的统计信息。

`-b`: 显示I/O和传送速率的统计信息。

`-c`: 输出进程的统计信息，每秒创建的进程数。

`-R`: 输出内存页面的统计信息。

`-y`: 终端设备的活动情况。

`-w`: 输出系统交换活动的信息，即每秒上下文切换次数。

如下是一些`sar`使用的例子。

(1) CPU资源监控

例如，每10s采样一次，连续采样3次，观察CPU的使用情况，并将采样结果以二进制的形式存入当前目录下的文件`test`中，需要键入如下命令：

```
sar -u -o test 10 3
```

输出项说明如下。

`-CPU`: all表示统计信息为所有CPU的平均值。我们可以使用`sar -P n`查看某颗CPU。

`%user`: 显示在用户级别运行和使用CPU总时间的百分比。

`%nice`: 显示在用户级别，用于nice操作，所占用CPU总时间的百分比。

`%system`: 在核心级别（kernel）运行所占用CPU总时间的百分比。

`%iowait`: 显示用于等待I/O操作所占用CPU总时间的百分比。

`%idle`: 显示CPU空闲时间所占用CPU总时间的百分比。

1) 若`%iowait`的值过高，则表示硬盘存在I/O瓶颈。

2) 若`%idle`的值很高但系统响应很慢时，有可能是CPU正在等待分配内存，此时应加大内存容量。

3) 若`%idle`的值持续低于10，则系统的CPU处理能力相对较低，表明系统中最需要解决的资源是CPU。

(2) 查看网络的统计

语法是`sar -n KEYWORD`。

KEYWORD常用的值及说明具体如下。

`-DEV`: 显示网络设备统计，如`eth0`、`eth1`等。

`-EDEV`: 显示为网络设备错误统计。

`-NFS`: 显示NFS客户端活动统计。

`-ALL`: 显示所有统计信息。

如下命令可查看网络设备的吞吐，数据每秒更新一次，总共更新5次。

```
[root@localhost ~]# sar -n DEV 1 5
```

输出项说明。

·第一字段：时间。

·IFACE：设备名。

·rxpck/s：每秒收到的包。

·txpck/s：每秒传输的包。

·rxbt/s：每秒收到的所有包的体积。

·txbt/s：每秒传输的所有包的体积。

·rcmp/s：每秒收到的数据切割压缩的包的总数。

·txcmp/s：每秒传输的数据切割压缩的包的总数。

·rxmcst/s：每秒收到的多点传送的包。

可以使用grep命令对输出进行过滤，命令如下。

```
sar -n DEV 2 5 | grep eth1
```

如果想知道网络设备错误报告，也就是用来查看设备故障的。应该用EDEV命令；比如下面的例子。

```
sar -n EDEV 2 5
```

(3) 内存分页监控

例如，每10s采样一次，连续采样3次，监控内存分页，命令及输出结果如下。

```
sar -B 10 3
10:45:04 AM  pgpgin/s pgpgout/s      fault/s majflt/s
10:45:14 AM    606.19     3648.35    13893.21     0.00
10:45:24 AM    626.17     3726.67    525.97      0.00
10:45:34 AM    557.36     3734.53     1.50      0.00
Average:       596.60     3703.17    4810.10     0.00
```

输出项说明如下。

·pgpgin/s：表示每秒从磁盘或SWAP置换到内存的字节数（KB）。

·pgpgout/s：表示每秒从内存置换到磁盘或SWAP的字节数（KB）。

·fault/s：每秒钟系统产生的缺页数，即主缺页与次缺页之和（major+minor）。

·majflt/s：每秒钟产生的主缺页数，这会导致将数据从磁盘加载到内存，因此需要留意。

(4) I/O和传送速率监控

例如，每10s采样一次，连续采样3次，需要键入如下命令。

```
sar -b 10 3
```

输出项说明如下。

·tps：每秒钟物理设备的I/O传输总量。

·rtps：每秒钟从物理设备读入的数据总量。

·wtps：每秒钟向物理设备写入的数据总量。

·bread/s：每秒钟从物理设备读入的数据量，单位为块/s。

·bwrtn/s：每秒钟向物理设备写入的数据量，单位为块/s。

(5) 进程队列长度和平均负载状态监控

例如，每10s采样一次，连续采样3次，监控进程队列长度和平均负载状态，命令如下。

```
sar -q 10 3
```

输出项说明如下。

-runq-sz: 运行队列的长度（等待运行的进程数）。

-plist-sz: 进程列表中进程（processes）和线程（threads）的数量。

-ldavg-1: 最后1分钟的系统平均负载（system load average）。

-ldavg-5: 过去5分钟的系统平均负载。

-ldavg-15: 过去15分钟的系统平均负载。

(6) 系统交换活动信息监控

例如，每10s采样一次，连续采样3次，监控系统交换活动信息，命令如下。

```
sar -W 10 3
```

输出项说明如下。

-pswpin/s: 每秒系统换入的交换页面（swap page）数量。

-pswput/s: 每秒系统换出的交换页面数量。

(7) 设备使用情况监控

例如，每10s采样一次，连续采样3次，报告设备使用情况，需要键入如下命令。

```
# sar -d 10 3 -p
```

其中，参数-p可以打印出sda、hdc等易读的磁盘设备名称，如果不使用参数-p，设备节点则有可能是dev8-0、dev22-0这样的形式。

输出项说明如下。

-tps: 每秒从物理磁盘I/O的次数。多个逻辑请求会被合并为一个I/O磁盘请求，一次传输的大小是不确定的。

-rd_sec/s: 每秒读扇区的次数。

-wr_sec/s: 每秒写扇区的次数。

-avgrq-sz: 发送到设备的请求的平均大小，单位为扇区。

-avgqu-sz: 磁盘请求队列的平均长度。

-await: 从请求磁盘操作到系统完成处理，每次请求的平均消耗时间，包括请求队列的等待时间，单位是毫秒。

-svctm: 系统处理每次请求的平均时间，不包括在请求队列中消耗的时间。

%util: I/O请求占CPU的百分比，比率越大，说明越饱和。

1) avgqu-sz的值较低时，设备的利用率较高。

2) 当%util的值接近100%时，表示设备带宽已经占满。

(8) 查看历史统计信息

有时我们希望能够看到历史性能统计信息，可以进入目录/var/log/sa，使用sar-fsaXX命令查看历史数据，例如，

```
sar -f sa22
```

默认将显示整天的数据。我们可以加上-s选项指定特定时间段的数据，例如，

```
sar -q -f sal3 -s 14:00:00 | head -n 10
```

以上命令将只显示13日14点之后的load的统计数据，且只显示最前面的10条记录。

(9) 输出inode、文件和其他内核表的统计信息

```
sar -v 10 3
```

输出项说明如下。

·dentunusd: 目录高速缓存中未被使用的条目数量。

·file-nr: 文件句柄（file handle）的使用数量。

·inode-nr: 索引节点句柄（inode handle）的使用数量。

要想判断系统的瓶颈问题，有时需要将几个sar命令选项结合起来。

·怀疑CPU存在瓶颈，可用sar-u和sar-q等来查看。

·怀疑内存存在瓶颈，可用sar-B、sar-r和sar-W等来查看。

·怀疑I/O存在瓶颈，可用sar-b、sar-u和sar-d等来查看。

2.iostat

iostat是I/O statistics（输入/输出统计）的缩写，iostat工具将对系统的磁盘操作活动进行监视。它的特点是汇报磁盘活动的统计情况，同时也将汇报出CPU的使用情况。iostat有一个弱点，那就是它不能对某个进程进行深入分析，只能对系统的整体情况进行分析。

iostat的语法如下。

```
iostat [ options... ] [ <interval> [ <count> ] ]
```

举例如下。

```
iostat -x sda 1  
iostat -txm 10 3
```

参数及说明分别如下。

·-t: 打印汇报的时间。

·-x: 默认显示所有设备。

·-m: 统计信息显示每秒多少MB而不是默认的每秒多少块。

输出项解析如下。

·avgqu-sz: 对于在线OLTP业务，应该大概近似于MySQL的页块大小，如果你看到这个值远远大于你的MySQL实例页块（16KB），那么可能存在一些其他的非数据库I/O负荷，或者你的数据库因为某些原因（如预读）导致检索数据的效率不高。

·svctm: 服务时间。

·await: 平均等待时间。

磁盘I/O请求包含服务时间（svctm）和等待时间（await），svctm一般小于10ms，我们要重点关注await。

·%util: 磁盘利用率。

在磁盘利用率达到100%的时候，意味着存在I/O瓶颈，这个时候，I/O达到饱和，此时的吞吐率我们可以用如下公式来衡量：

$$(r/s+w/s)*svctm = %util$$

此时，吞吐率(r/s+w/s)就是svctm的倒数。注意，此公式仅在磁盘利用率达到100%的时候成立。

3.vmstat

vmstat这个工具提供了系统整体性能的报告，它能对进程、内存、页交换、I/O、中断及CPU使用情况进行统计并报告信息。vmstat的输出类似如下。

```

vmstat 2 222
procs -----memory-----swap-- ----io---- --system-- ----cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
1 0 0 27724752 1103508 26829932 0 0 0 26 0 0 3 0 96 0 0
2 0 0 27725012 1103508 26829940 0 0 0 136 7293 10504 5 0 95 0 0
1 0 0 27724244 1103508 26830532 0 0 0 4096 7620 11758 6 1 93 0 0

```

我们一般关注r、b、id及si、so。

(1) procs部分

r: 表示当前有多少进程正在等待运行，如果r是连续的大于在系统中的CPU的个数，则表示系统现在运行得比较慢，有数的进程正在等待CPU。

b: 表示当前有多少进程被阻塞。

(2) memory部分

swpd、free、buff、cache这4项展示了内存是如何使用的，swap列显示了swap被使用的数量，free列显示了目前空闲的内存，buffer列显示了buffer所使用的内存，cache列显示了page cache所使用的内存，cache越大，表明有越多的内存用于cache文件。

(3) swap部分

si: 每秒从磁盘交换到swap的内存。

so: 每秒从swap交换到磁盘的内存。

si、so正常情况下应该等于0，如果持续不为0，那么很可能存在性能问题。

(4) io部分

bi: 从块设备读入数据的总量（读磁盘）。

bo: 块设备写入数据的总量（写磁盘）。

bi、bo的变化反映了我们磁盘读取和写入的速率。

(5) system部分

in: 每秒中断次数，包括时钟。

cs: 每秒上下文切换次数。

(6) CPU部分

这些列展示了不同CPU完成不同任务的CPU时间百分比。我们由此可以得知，CPU是否真正在做事，还是处于空闲或等待状态。很高的sy值，表明可能有过度的系统调用或系统调用效率不高。

us: 运行非内核代码所花费的时间百分比，即进程在用户态使用的CPU时间百分比。

sy: 运行内核代码所花费的时间百分比，即进程在系统态使用的CPU时间百分比。

id: 空闲的CPU时间百分比。

wa: 等待I/O的CPU时间百分比。

st: 从虚拟机偷取的CPU时间百分比。

4.oprofile

oprofile是Linux内核支持的一种性能分析机制，是一套低开销的工具集合，简单易用，适合于在实际的系统中分析程序的性能瓶颈。它可以工作在不同的体系结构上，包括MIPS、ARM、IA32、IA64和AMD。内核2.6的发行版一般都内置了这个工具。

通过oprofile这个工具，开发人员可以得知一个程序的瓶颈在哪里，进而指导代码优化。

oprofile的基本原理是进行抽样统计。处理器定时中断，oprofile可在这个时候记录哪些代码正在执行。往往耗时更长的代码被取样的次数更多，通过这种方式，我们可以发现哪个可执行程序或哪个function消耗的CPU最多，进一步分析即可找到可供调优的代码片段。

系统的运行，往往就是在处理各种事件，比如CPU指令、磁盘I/O、网络包、系统调用、库调用、应用程序事务、数据库查询，等等。性能分析和往往分析和研究这些事件的统计，例如每秒操作的次数、每秒传输的字节数、平均延时、有时重要的细节信息在统计中可能会被忽视掉，这个时候对每类事件单独进行统计会更有助于你了解系统的行为。

oprofile支持两种采样(sampling)方式：基于事件的采样(eventbased)和基于时间的采样(timebased)。

基于事件的采样是oprofile只记录特定事件的发生次数。这种方式需要CPU内部有性能计数器(performance counter)。

基于时间的采样是oprofile借助OS时钟中断的机制，每个时钟中断时oprofile都会记录一次(采一次样)，引入此种采样方式的目的在于提供对没有性能计数器的CPU的支持，其精度相对于基于事件的采样要低。因为要借助OS时钟中断的支持，对禁用中断的代码oprofile不能对其进行分析。

虽然说基于事件的方法在理论上更精确，但在大部分简单的场景下，基于时间的方法也能工作得很好，所以，如果你的CPU在基于事件的性能诊断中存在异常的情况，那么就使用基于时间的方法好了。

许多人不知道如何有效地使用oprofile来进行性能优化，这里将介绍一些基本的用法。为了及时反映软硬件的发展，支持不同的CPU，oprofile版本更新得比较频繁，目前版本(截至2014年10月6日)是1.0.0版本。由于主要的发行版内置的oprofile一般是较低的版本，因此如下所做的介绍，都将基于较低的版本0.9.4。

```
opcontrol --version
opcontrol: oprofile 0.9.4 compiled on Nov 22 2011 12:03:03
```

我的生产环境是RHEL 5.4，如果需要oprofile内核，则需要安装内核符号信息，命令如下。

```
rpm -i kernel-debuginfo-common-2.6.18-164.el5.x86_64.rpm
rpm -i kernel-debuginfo-2.6.18-164.el5.x86_64.rpm
```

安装好的vmlinu在在这里。

```
/usr/lib/debug/lib/modules/2.6.18-164.el5/vmlinu
```

oprofile包含有一系列的工具集，这些工具默认在路径/usr/bin之下，工具及说明分别如下。

1) op_help: 列出可用的事件，并带有简短的描述。

2) opcontrol: 控制oprofile的数据收集。2012年，oprofile 0.9.8开始引入operf工具，将替换旧的基于opcontrol的工具，允许非root用户也可以进行性能监测。

opcontrol的配置默认在/root/.oprofile/daemonrc下，可以使用demsg查看oprofile使用的是哪一种模式。

3) opreport: 对结果进行统计输出。一般存在两种基本形式。

```
opreport-f
```

```
opreport-l`which oprofiled`2>/dev/null|more
```

4) opannotate: 产生带注释的源文件/汇编文件，源语言级的注释需要在编译源文件时加上的调试符号信息的支持。

5) opgprof: 产生与gprof相似的结果。

6) oparchive: 将所有的原始数据文件收集打包，从而可以在另一台机器上进行分析。

7) opimport: 将采样的数据库文件从另一种abi外部格式转化为本地格式。

基本步骤具体如下。

1) opcontrol -no-vmlinu#不对内核进行性能分析

或者 opcontrol -vmlinu=/boot/vmlinu-`uname -r`#对内核进行性能分析

默认的配置存放在/root/.oprofile/daemonrc中

默认的采样的文件存放在/var/lib/oprofile/samples/中

也可以指定默认数据存放的地方，命令如下。

```
opcontrol --no-vmlinu --session-dir=/home/me/tmpsession
opcontrol --start --session-dir=/home/me/tmpsession
```

在开始收集采样数据前可回顾下我们的设置，运行opcontrol-status。

2) 清除上一次采样到的数据。

```
opcontrol --reset
```

3) 开始收集信息。

```
opcontrol --start
```

4) 运行程序，施加负荷。

5) dump出收集的数据，然后可以继续运行或关闭oprofile。

```
opcontrol --dump
```

我们可以随时运行命令opcontrol-reset清除我们当前会话的采样数据，重置计数器。

6) 停止数据收集，且kill掉daemon进程。

```
opcontrol --shutdown
```

默认将数据放在/var/lib/oprofile/samples下，可以使用opcontrol-reset清理文件。

7) 输出统计报告。

```
oprofile -f [--session-dir=dir]
```

查看系统级别的报告：oprofile--long-filenames。

查看模块级别的报告，oprofile image:进程路径-l，命令如下。

```
oprofile -l image:/bin/myprog,/bin/myprog  
oprofile image:/usr/bin/*
```

查看源码级别的报告，opannotate image:进程路径-s。

输出的报告类似于图16-12的形式。

The screenshot shows a terminal window displaying an oprofile report. The output is as follows:

```
CPU: Intel Architecture Performance Counter Driver  
Counted CPU CLE UNHALTED events (Clock cycles when not halted, with a unit mask of 0x30 (No unit mask) count 10000)  
CPU_CLE_UNHALT...  
samples | %  
37675 51.5% mysqld  
16058 26.8% smooth_plugin.so.3.0.0  
16014 7.74% mysqld  
1519 2.65% 11:00-2.6.0.so  
6239 1.04% libpthread-2.3.so  
1381 1.23% opencored  
291 0.04% bash  
61 0.02 1d-2.5.so  
40 0.02 libcrypt.so.1.5.6  
12 0.0010 11:00-2.6.0.so  
9 0.0016 11:00-2.6.0.so  
6 0.0010 11:00-2.6.0.so  
6 0.0010 libpcap-3.2.7.so  
6 0.0010 httpd  
5 8.4e-04 gawk  
5 8.4e-04 grep  
5 8.4e-04 readelf,readelf  
3 0.0e-04 lsmod
```

图16-12 oprofile工具的一个输出报告

其中，第一列是收集的采样数据的统计次数，第二列是耗费的时间百分比，第三列是进程名。

oprofile还可以观测事件列表命令如下。

```
opcontrol --list-events
```

如下是一个完整的示例。

```
opcontrol --start --no-vmlinux --separate=kernel #启动收集  
程序运行中，此时  
oprofile收集数据  
opcontrol --status #显示状态  
opcontrol -h #关闭  
oprofile  
oprofile -f | more #显示报告  
oprofile image: /usr/local/mysql-5.1.58-linux-x86_64-glibc23/bin/mysqld  
oprofile -l image: /usr/local/mysql-5.1.58-linux-x86_64-glibc23/bin/mysqld | more
```

注意事项

·不建议在虚拟机里利用oprofile来测试性能。

·调试的内核最好是原生内核。

·使用oprofile定位CPU密集型的场景是合适的，但对于某些I/O密集型或是低负载类型的场景就会有些无能为力，这时可以借助其他工具进一步定位性能瓶颈。

5.free

free命令用于显示系统的自由内存和已经被使用的内存。free指令显示的内存的使用情况包括实体内存、虚拟的交换文件内存、共享内存区段及系统核心使用的缓冲区等。

语法： free[-bkmotV][-s]

参数及说明分别如下。

-b: 以Byte为单位显示内存使用情况。

-k: 以KB为单位显示内存使用情况。

-m: 以MB为单位显示内存使用情况。

-o: 不显示缓冲区调节列。

-s: 持续观察内存使用状况。

-t: 显示内存总和列。

-V: 显示版本信息。

如下是free命令的输出。

```
free
total        used       free      shared  buffers   cached
Mem:   65966584     65787112     179472          0    443532  15532932
-/+ buffers/cache: 49810648  16155936
Swap:  16779884        316   16779568
```

其中各项说明如下。

-Mem: 表示物理内存统计。

-/+buffers/cache: 表示物理内存的缓存统计。

-Swap: 表示硬盘上交换分区的使用情况，这里我们不去关心。

·系统的总物理内存：65966584（64GB），但系统当前真正可用的内存大小并不是第一行free标记的179472KB，它仅代表未被分配的内存。以下我们将逐行解释输出。

第1行 Mem

-total: 表示物理内存总量。

-used: 表示总计分配给缓存（包含buffers与cache）使用的数量，但其中可能有部分缓存并未实际使用。

-free: 未被分配的内存。

-shared: 共享内存。

-buffers: 系统已分配但未被使用的buffers数量。

-cached: 系统已分配但未被使用的cache数量。

buffer指的是作为buffer cache的内存，即块设备的读写缓冲区。cache指的是作为page cache的内存，即文件系统的cache。如果cache的值很大，则说明cache住的文件数很多。如果频繁访问到的文件都能被cache住，那么磁盘的读I/O必定会非常小。但是过大的文件cache可能会影响到内存的使用效率，导致操作系统上其他进程的内存不够大，甚至还会使用到swap空间。

total=used+free

第2行 -/+buffers/cache

-used: 也就是第一行中的used-buffers-cached，也是实际使用的内存总量。

-free: 未被使用的buffers与cache和未被分配的内存之和（见第一行buffers、cached、free），这就是系统当前实际可用的内存。

第2行所指的是从应用程序的角度来看，对应用程序来讲，buffers/cache是等同可用的，当程序使用内存时，buffers/cache会很快地被使用。从应用程序的角度来说，可用内存=系统free memory+buffers+cached。

第1行Mem是对操作系统来讲的。buffers/cache都是属于被使用的，所以它认为free只有179472。

我们一般理解的free输出应该从应用程序的角度去理解，应该关注第二行的free输出，也就是16155936KB。因为那些buffers和cache是可能被重用的。

```
-/+ buffers/cache: 49810648 16155936
```

6.top

能够实时显示系统中各个进程的资源占用状况，类似于Windows的任务管理器。它不断更新最新情况直至用户结束程序。默认情况下，可列出消耗CPU资源最多的十多个进程。你也可以交互式地键入不同的键按其他选项进行排序，比如按【M】键可列出占用最多内存的几个进程，按【I】键可切换显示各CPU的使用率或整体使用率等，按【K】键可终止某个进程，按空格键可重新刷新屏幕输出。

以下是一个系统运行top命令的例子。

```
top - 17:10:08 up 497 days, 40 min, 1 user, load average: 0.19, 0.26, 0.26
Tasks: 433 total, 2 running, 430 sleeping, 0 stopped, 1 zombie
Cpu(s): 0.5%us, 0.2%sy, 0.0%ni, 99.1%id, 0.2%wa, 0.0%hi, 0.1%si, 0.0%st
Mem: 65966584k total, 65791212k used, 175372k free, 444868k buffers
Swap: 16779884k total, 316k used, 16779568k free, 15532032k cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
19462 mysql 15 0 29.4g 29g 9996 S 16.2 46.4 111787:00 /usr/local/mysql/bin/mysqld --defaults-file=/my.cnf
24901 nemo 15 0 13024 1368 812 R 0.3 0.0 0:00.16 top -c
 1 root 15 0 10368 640 544 S 0.0 0.0 5:12.51 init [3]
 2 root RT -5 0 0 0 0 S 0.0 0.0 1:39.38 [migration/0]
 3 root 34 19 0 0 0 S 0.0 0.0 50:46.99 [ksoftirqd/0]
 4 root RT -5 0 0 0 S 0.0 0.0 0:00.02 [watchdog/0]
```

第一列显示的是当前时间、系统运行时间（up time）、使用者（users）数目和平均负载（load average）。可以按键切换是否显示。

```
top - 17:10:08 up 497 days, 40 min, 1 user, load average: 0.19, 0.26, 0.26
```

平均负载的三个数值分别表示在平均过去1分钟、5分钟和15分钟，可运行或不可中断状态的进程数目。平均负载为1.0表示一个CPU被占用所有时间。如果计算机有多个CPU，则平均负载的参考值亦会成倍数增长。例如一个双CPU4核的计算机，所有CPU所有时间被完全占用时的平均负载应该为 $1.0 \times 2 \times 4 = 8.0$ 。

第二列显示任务（task）信息，任务表示一个进程或多线程进程中的某个线程，任务信息包括任务总数、运行中（running）、睡眠中（sleeping）、已停止（stopped）和不能运行（zombie）的进程数目，zombie就是僵尸进程。可以按【t】键切换和下一列CPU状态列是否同显示。

```
Tasks: 433 total, 2 running, 430 sleeping, 0 stopped, 1 zombie
```

第三列显示CPU状态，包括以下信息。

·us (user)：用户空间（user space）占用CPU的百分比。

·sy (system)：核心空间（kernel space）占用CPU的百分比。

·ni (nice)：nice值比一般值0大（优先级较低）的进程占用CPU的百分比。

·id (idle)：CPU空闲时间百分比。

·wa (iowait)：CPU等待的百分比。当值过高时（如超过30%），表示系统的存储或网络I/O性能存在问题。

·hi (H/W Interrupt)：CPU处理硬件中断时间的百分比。除非光驱不断检查是否有光盘外，此值一般不会太高。

·si (S/W Interrupt)：CPU处理软件中断时间的百分比，此值一般不会太高。

·st (Steal)：在如Xen等的虚拟环境下，CPU运作虚拟机器时间的百分比。太高，则表示可能需要停止一些虚拟机器。

```
Cpu(s): 0.5%us, 0.2%sy, 0.0%ni, 99.1%id, 0.2%wa, 0.0%hi, 0.1%si, 0.0%st
```

第四列和第五列分别显示内存和交换空间（swap space）的使用率。可以按【M】键切换是否显示。

```
Mem: 65966584k total, 65791212k used, 175372k free, 444868k buffers
Swap: 16779884k total, 316k used, 16779568k free, 15532032k cached
```

其他一些示例如下。

默认情况下，top是交互式（interactive）的输出，会一直在屏幕上刷新，如果我们需要获取top的输出，那么我们可以使用批处理模式。例如如下示例。

```
top -b -d 5 -n 5
```

其中各参数及说明分别如下。

-b: 批处理模式操作。

-d: 刷新时间间隔。

-n: 交互次数，即输出几次。

如下例子可指定某个或某几个进程的top输出。

```
top -p 4360,4358
```

如下例子可指定某个用户的top输出。

```
top -u garychen
```

7.dstat

对比其他工具，dstat更强大，可观察性也更强，dstat可综合显示各种系统资源的使用情况，如磁盘、网络、CPU、内存等。

以下是运行dstat的一个示例。

```
dstat 2 10
You did not select any stats, using -cdngy by default.
---total-cpu-usage--- -disk/total- -net/total- ---paging--- ---system---
usr sys idl wai hiq siq| read wrt| recv send| in out | int csw
 4 0 95 0 0 0|2269B 641k| 0 0 | 0 0 | 12867 5261
12 0 87 0 0 0| 0 134k| 552k 3655k| 0 0 | 6787 11k
19 0 80 0 0 0| 0 1096k| 608k 3724k| 0 0 | 6924 12k
 6 0 94 0 0 0| 0 1182k| 501k 3246k| 0 0 | 6351 8553
15 1 84 0 0 0| 0 206k| 576k 4241k| 0 0 | 6800 12k
21 1 78 0 0 0| 0 504k| 597k 6403k| 0 0 | 7053 14k
12 0 87 0 0 0| 0 304k| 461k 5518k| 0 0 | 6229 9607
18 1 81 0 0 0| 0 328k| 511k 6113k| 0 0 | 6651 12k
16 1 83 0 0 0| 0 266k| 580k 6359k| 0 0 | 7116 14k
16 0 83 0 0 0| 0 188k| 673k 7052k| 0 0 | 7452 11k
14 0 86 0 0 0| 0 144k| 532k 6051k| 0 0 | 6557 8724
```

上面输出中total-cpu-usage部分的hiq、siq分别为硬中断和软中断的次数。

上面输出中system部分的int、csw分别为系统的中断（interrupt）次数和上下文切换（context switch）。

若要将结果输出到CSV文件可以加--output filename。我们可以使用绘图工具对此文件进行画图。如果内核支持，还可以使用--top-io-adv参数查看最消耗I/O的进程。

8.netstat

netstat命令用于显示各种网络相关的信息。

常见的参数及说明分别如下。

-a: all，显示所有选项，默认不显示LISTEN相关。

-t: tcp，仅显示TCP相关选项。

-u: udp，仅显示UDP相关选项。

-n: 不显示别名，能显示为数字的全部转化成数字。

-l: 仅列出有正在Listen（监听）的服务状态。

-p: 显示建立相关链接的程序名。

-r: 显示路由信息，路由表。

-e: 显示扩展信息，例如uid等。

-s: 按各个协议进行统计。

-c: 每隔一个固定时间，执行该netstat命令。

下面将列举几个示例进行说明。

我们可以使用netstat-tlpn显示目前正在监听TCP协议端口的MySQL服务。

以下命令将每隔一秒输出一次网络信息，检测MySQL服务是否已经起来并监听端口了。

```
netstat -tlpnc |grep mysql
```

以下命令将查看连接到MySQL服务端口的IP信息，并按连接数进行排序。

```
netstat -nat | grep ":3306" |awk '{print $5}'|awk -F: '{print $1}'|sort|uniq -c|sort -nr|head -20
```

9.mtr

mtr是一个功能强大的网络诊断工具，综合了ping和traceroute的功能。它可以帮助系统管理员诊断网络异常，并提供友好的网络状态报告以供分析。

以下将介绍mtr的数据是如何产生的，如何解读mtr的报告，以及如何诊断异常。

mtr使用“ICMP”包来测试网络争用和传输，mtr的工作原理是：启动mtr时，它通过发送不断增长TTL（生存时间）的ICMP包来测试本机和目标主机的连通性。TTL控制了ICMP包要经过多少“跳”才会被返回，例如，主叫方首先发出TTL=1的ICMP数据包，第一个路由器将TTL减1得0后就不再继续转发此数据包，而是返回一个ICMP超时报文，主叫方从超时报文中即可提取出数据包所经过的第一个网关地址。然后又发出一个TTL=2的ICMP数据包，可获得第二个网关地址，依次递增TTL便获取了沿途所有网关的地址。

mtr连续发送不断增长TTL（生存时间）的ICMP包以收集中间路由器的信息，包括各种连接、响应能力、状态等信息，这就允许mtr能够打印出中间路由器的响应率和响应时间直到目标主机。丢包率或响应时间的突然上升可能意味着这个网络存在问题。

我们可以把mtr看作一个单向的衡量网络质量的工具，从本机到目的主机和从目的主机到本机往往走的是不一样的网络路径，本机到目的主机没有丢包，但目的主机到本机却可能会丢包。从本地不同的主机到目的主机也可能走不一样的网络路径。所以，如果诊断出是网络问题，那么建议在两个方向上都使用mtr进行测试验证，如果可以，多验证一些主机之间的通信。

示例如下。

```
mtr --report www.google.com
```

添加--report表示发送10个包到目的主机www.google.com，然后生成报告。如果不加--report选项，那么mtr会一直在交互模式中运行，交互模式会不断刷新报告以反映最新的信息。一般情况下--report选项生成的报告就已经足够了。

如何阅读报告。

如下是一个对google的mtr报告。

```
% mtr --no-dns --report google.com
HOST: deleuze          Loss%   Snt    Last     Avg   Best Wrst StDev
 1. 192.168.1.1        0.0%   10     2.2    2.2   2.0   2.7   0.2
 2. 68.85.118.13       0.0%   10     8.6   11.0   8.4  17.8   3.0
 3. 68.86.210.126      0.0%   10     9.1   12.1   8.5  24.3   5.2
 4. 68.86.208.22       0.0%   10    12.2   15.1  11.7  23.4   4.4
 5. 68.85.192.86       0.0%   10    17.2   14.8  13.2  17.2   1.3
 6. 68.86.90.25       0.0%   10    14.2   16.4  14.2  20.3   1.9
 7. 68.86.86.194       0.0%   10    17.6   16.8  15.5  18.1   0.9
 8. 75.149.230.194     0.0%   10    15.0   20.1  15.0  33.8   5.6
 9. 72.14.238.232      0.0%   10    15.6   18.7  14.1  32.8   5.9
10. 209.85.241.148     0.0%   10    16.3   16.9  14.7  21.2   2.2
11. 66.249.91.104      0.0%   10   22.2   18.6  14.2  36.0   6.5
```

我们从报告中可以看到，如上的mtr经过了11跳（hops），一般我们使用术语“跳”的计数来标识报告中的问题，“跳”指的是因特网上网络包到达目的地所经过的节点和路由器。

输出项说明如下。

·**Loss%**: 每一跳的丢包率。

·**Snt**: 发送的数据包数量。

·**Last**: 最后一个包的响应时间（毫秒）。

·**Avg**: 所有包的平均响应时间（毫秒），一般情况下，我们更关注这个响应时间。

·**Best**: 最短的响应时间（毫秒）。

·**Wrst**: 最长的响应时间（毫秒）。

·StDev: 响应时间的标准差。StDev越大，表示各个包的响应时间差异越大。如果差异很大，我们可能需要审视下Avg的平均值是否可靠，看看Best和Wrst，确认Avg是否能够代表真实的延时时间，是否有被其他的因素所干扰。

一般情况下，我们可以把以上的输出报告分解为三个部分，前几跳往往是本地ISP，最后的两三跳往往是目的主机的ISP，中间的一些跳是网络传输经过的一些路由器。对于本地ISP和目的主机ISP的网络问题，我们往往可以及时反馈，得到处理，但对于一些中间结点的路由器出现的问题，往往ISP自身也无力去解决。

下面我们通过4个例子来说明如何分析报告。

示例1：当我们阅读报告时，我们一般关注的是丢包率和延时。如果我们在任何一跳看到有超过一定百分比的延时（比如，超过5%），那么那个路由器就可能存在问题，但也存在这样一种情况，一些ISP限制了ICMP包，我们看到了很高的丢包率，但实际上并没有发生丢包，例如下面的例子。

```
root@localhost:~# mtr --report www.google.com
HOST: ducklington      Loss% Snt Last Avg Best Wrst StDev
1. 63.247.74.43        0.0% 10  0.3  0.6  0.3  1.2  0.3
2. 63.247.64.157       50.0% 10  0.4  1.0  0.4  6.1  1.8
3. 209.51.130.213      0.0% 10  0.8  2.7  0.8  19.0  5.7
4. aix.prl.atl.google.com 0.0% 10  6.7  6.8  6.7  6.9  0.1
5. 72.14.233.56        0.0% 10  7.2  8.3  7.1  16.4  2.9
6. 209.85.254.247      0.0% 10  39.1 39.4 39.1 39.7  0.2
7. 64.233.174.46        0.0% 10  39.6 40.4 39.4 46.9  2.3
8. gw-in-f147.le100.net 0.0% 10  39.6 40.5 39.5 46.7  2.2
```

我们看到第2跳的丢包率高达50%，但实际上并没有丢包。判断是否真正丢包，可以看下后续的跳，如果后续的跳显示没有丢包，那么就没有丢包，而是因为ISP限制了ICMP的速率。

示例2：ICMP速率限制和丢包可能会同时出现，这个时候，丢包率应该选择后续的跳中最低的丢包率，这个最低的丢包率才代表实际的丢包率，例如下面的例子中，丢包率是40%，而不是60%。

```
root@localhost:~# mtr --report www.google.com
HOST: localhost          Loss% Snt Last Avg Best Wrst StDev
1. 63.247.74.43        0.0% 10  0.3  0.6  0.3  1.2  0.3
2. 63.247.64.157       0.0% 10  0.4  1.0  0.4  6.1  1.8
3. 209.51.130.213      60.0% 10  0.8  2.7  0.8  19.0  5.7
4. aix.prl.atl.google.com 60.0% 10  6.7  6.8  6.7  6.9  0.1
5. 72.14.233.56        50.0% 10  7.2  8.3  7.1  16.4  2.9
6. 209.85.254.247      40.0% 10  39.1 39.4 39.1 39.7  0.2
7. 64.233.174.46        40.0% 10  39.6 40.4 39.4 46.9  2.3
8. gw-in-f147.le100.net 40.0% 10  39.6 40.5 39.5 46.7  2.2
```

示例3：有时目的主机的不正确配置也会导致丢包，比如目的主机有防火墙Drop掉了ICMP包，例如下面的例子中，我们可以看到最后一跳是100%的丢包率。

```
root@localhost:~# mtr --report www.google.com
HOST: localhost          Loss% Snt Last Avg Best Wrst StDev
1. 63.247.74.43        0.0% 10  0.3  0.6  0.3  1.2  0.3
2. 63.247.64.157       0.0% 10  0.4  1.0  0.4  6.1  1.8
3. 209.51.130.213      0.0% 10  0.8  2.7  0.8  19.0  5.7
4. aix.prl.atl.google.com 0.0% 10  6.7  6.8  6.7  6.9  0.1
5. 72.14.233.56        0.0% 10  7.2  8.3  7.1  16.4  2.9
6. 209.85.254.247      0.0% 10  39.1 39.4 39.1 39.7  0.2
7. 64.233.174.46        0.0% 10  39.6 40.4 39.4 46.9  2.3
8. gw-in-f147.le100.net 100.0 10  0.0  0.0  0.0  0.0  0.0
```

示例4：有时路由器出于某种原因没有正确配置，或者是对其有特殊设置，例如下面的例子中，我们可能会看到许多问号，但网络质量良好，并没有发生丢包。

```
root@localhost:~# mtr --report www.google.com
HOST: localhost          Loss% Snt Last Avg Best Wrst StDev
1. 63.247.74.43        0.0% 10  0.3  0.6  0.3  1.2  0.3
2. 63.247.64.157       0.0% 10  0.4  1.0  0.4  6.1  1.8
3. 209.51.130.213      0.0% 10  0.8  2.7  0.8  19.0  5.7
4. aix.prl.atl.google.com 0.0% 10  6.7  6.8  6.7  6.9  0.1
5. ???
6. ???
7. ???
8. ???
9. ???
10. ???                 0.0% 10  0.0  0.0  0.0  0.0  0.0
```

注意事项

跨IDC的网络，本身就不是很稳定，特别是超长距离的横跨太平洋、大西洋的网络，中间经过的节点很多，很可能会出现波动或堵塞，导致延时很高，如果网络丢包率不是很高（比如大于10%），那么你不需要过分地关注，从架构上进行设计，让用户尽可能地访问本地的网络，提高用户体验，比去解决跨IDC的高延时和不稳定的网络会更具实践性。

网络的质量也和本地的连接、负载有关系，所以测量的时候也要留意这些因素的影响。

10.strace

strace是一个简单易用的工具，用于跟踪一个进程的系统调用或信号产生的状况，它最简单的用法就是跟踪一个可执行文件的执行，记录程序运行过程中的系统调用。

通过使用参数-c，它还能对进程中所有的系统调用做一个统计分析。它也能筛选出特定的系统调用，以下是一些示例。

1) 查找程序启动的时候加载了哪些配置文件。

```
$ strace php 2>&1 | grep php.ini可以查看加载了哪个  
php.ini文件.
```

如果想只筛选某个系统调用则可以使用如下命令。

```
$ strace -e open php 2>&1 | grep php.ini
```

2) 有时程序没有权限打开文件，它并不会提示你详细的信息，这时我们就可以用strace来判断是否存在权限的问题。

```
$ strace -e open,access 2>&1 | grep your-filename
```

3) 对于一些很消耗资源的进程，我们有时会想知道它们正在做什么？知道了pid后，可以使用-p参数来查看。

```
$ strace -p 15427
```

4) 加-c参数进行统计分析，可以查看哪些操作占据了最多资源。

```
$ strace -c -p 11084
```

监视一段时间后，按【Ctrl+C】键退出，会输出统计报表供你分析。

16.2.2 MySQL诊断工具

1. MySQL自带工具

MySQL自带了一些工具，大都是为管理的目的而发布的一些工具，如mysql、mysqladmin、mysqldump，具体的使用方法，请参考前面的章节。如果我们需要获得更全面的信息，进行更准确的诊断，那么更好的选择是使用一些第三方工具，如Percona工具包。

一般我们可以通过如下几种方式来收集信息。

·通过使用mysql、mysqladmin执行一些查询和命令来获取当前的全局状态变量。

·查询information_schema库下面的表，information_schema库保存了一些数据库的元信息，如连接信息。

·MySQL 5.5版本后新增了performance_schema库，这个库下的动态性能视图主要用于收集MySQL的性能信息，比如锁、事件等信息。基于事件的调优是一个方向，未来performance_schema会越来越成熟，将成为很重要的调优手段。

2.Percona工具包详解

(1) 介绍和安装

Percona工具包是Percona公司的一个性能诊断工具集，由于MySQL自带的性能诊断工具很匮乏，所以很多时候需要借助第三方的工具来协助诊断和定位问题，作为一名DBA，有必要熟练使用Percona工具包里的部分工具，以便更有效地管理数据库和诊断问题。以下将简要介绍Percona的安装方式和一些常用的工具。

本书所介绍的内容是基于Percona toolkit 2.1的版本。由于开源工具发展得比较快，可能这本书中所举的例子已不再适用。

由于是第三方工具，虽然有Percona公司的支持，许多工具也经过了大量用户的验证，但是仍然可能存在一些风险因素。强烈建议在使用以下所介绍的工具之前，仔细阅读官方文档，并经过自己的测试，验证其确实是可行的，才投入到生产环境中使用。

我们可以下载源码包、二进制包或RPM包进行安装。你需要确保Perl已经安装了模块DBI和DBD:mysql。可以很容易地下载到单独的工具，下载命令如下。

```
wget percona.com/get/TOOL(工具名  
)
```

以下以RHEL 5.464位为例简要说明如何安装Percona工具。

```
1) root  
[root@db1000 pkgs]# perl --version  
This is perl, v5.8.8 built for x86_64-linux-thread-multi  
cpan> install Time::HiRes  
2) 确认  
perl已经安装了  
DBI,DBD:mysql  
Time::HiRes  
3) 安装
```

```
percona-toolkit下载源码包，解压之，进入解压缩目录
```

```
perl Makefile.PL  
make  
make test  
make install默认安装在  
/usr/bin/ F  
ls -l /usr/bin/pt-*
```

Percona工具也有其配置文件，配置文件的语法简单直接，配置文件的规则具体如下。

·每行的格式可以是：option=value或option，等于号两边的空格被忽略。

·--表示选项解析结束，后面的行都是程序的附加参数。

·忽略空行。

·空格#“空格”+“#”表示后面的是注释内容。

Percona工具读取配置文件的顺序具体如下。

- 1) /etc/percona-toolkit/percona-toolkit.conf: 全局配置。
- 2) /etc/percona-toolkit/TOOL.conf: 可以指定某个工具的具体配置，TOOL是工具名。如pt-query-digest
- 3) \$HOME/.percona-toolkit.conf: 这是用户下的一个全局配置。
- 4) \$HOME/.TOOL.conf: 这是用户下的某个工具的具体配置。

也可以在命令行中指定配置文件，如--config/path/to/file，注意不要有=号，必须把--config参数放在命令行的最前面。--config表示不读取任何配置文件。

Percona工具包一般支持使用DSN语法来设置如何去连接MySQL。DSN的英文全称是DATA SOURCE NAME，一个DSN是一个逗号分隔的key=value形式的字符串，例如：h=host1,P=3306,u=bob。DSN对于大小写敏感，中间不允许有空格，如果有空格的话，必须要用引号引起来。

也有部分工具不支持DSN的方式连接数据库，它们自身提供了连接数据库的参数，如：“--host”、“--user”、“--password”。

一些工具可同时使用DSN和“--host”、“--user”、“--password”之类的参数。

一些标准的key及其说明分别如下。

·A: 字符集。

例如，A=utf8表示在连接时SET NAMES UTF8

·D: 数据库名。

·F: 设置mysql客户端库读取的配置文件，如果不进行设置，那么就读取标准配置文件，如/etc/my.cnf、\$HOME/.my.cnf。

·h: 主机名或IP。

·p: 密码。

·P: 端口。

·S: socket file。

·u: 数据库账号。

有些工具还会附加一些其他的key，这里就不赘述了。

可以通过设置环境变量PTDEBUG=1，启用Percona工具的Debug功能。命令的Debug信息会输出到STDERR，例如，输出Debug信息到一个文件，命令如下。

```
PTDEBUG=1 pt-table-checksum ... > FILE 2>&1
```

(2) pt-query-digest

pt-query-digest是最应该被掌握的一个工具。它可以分析MySQL的各种日志，如慢查询日志、general日志，也可以分析SHOW PROCESSLIST的输出。配合tcpdump我们还可以对线上数据库流量进行采样，实时监控数据库流量，及时发现性能问题。

其基本用法如下。

```
pt-query-digest /path/to/slow.log > /path/to/keep/report_file
```

如果你有大量的数据库节点，可以考虑把pt-query-digest的分析报告写入数据库，以方便检索和绘图。

输出的结果报表类似如下，以下截取了报告的部分内容。

```
# Overall: 565 total, 22 unique, 0.00 QPS, 0.00x concurrency
# Time range: 2012-09-22 18:33:43 to 2012-10-16 10:45:31
# Attribute          total      min      max      avg      95%    stddev   median
# ======          ======      ==      ==      ==      ==      ==      ==      ==
# Exec time         123s     503ms     15s      2s       7s      2s      1s
# Lock time        53ms      31us     145us    94us    119us    17us    93us
# Rows sent        1.67k          0        20      3.02     9.83     4.12     0.99
# Rows examine    616.77M   72.90k    12.03M   1.09M    6.61M   2.02M   245.21k
# Query size      139.49k        25      381    252.81   346.17   70.94   234.30
# Profile
# Rank Query ID           Response time   Calls   R/Call Apdx   V/M Item
# ====== ======           ======   ==      ==      ==      ==      ==      ==
# 1 0xBE5D289C750F172A 308.6929 25.0%    40    7.7173 0.00 0.08  SELECT ccc tbl_eee tbl_ddd bbb
# 2 0x5C898C5E065DD204 149.4144 12.1%   105   1.4230 0.50 0.00  SELECT tbl_ddd info tbl_eee tbl_ddd
# 3 0x6F05415421300718 136.7381 11.1%    97    1.4097 0.50 0.00  SELECT tbl_ddd info tbl_eee tbl_ddd
# 4 0x2E9AE41A4D2149A1 123.0681 10.0%    22    5.5940 0.00 0.02  SELECT ccc tbl_eee tbl_ddd bbb
# 5 0xAF556BC27138443 121.9603 9.9%     73    1.6707 0.50 0.00  SELECT tbl_ddd info tbl_eee tbl_ddd
# 6 0xD07F224EF598BD9A 105.0456 8.5%     16    6.5653 0.00 0.23  SELECT ccc(tbl_eee)tbl_ddd bbb
# 7 0xC22F9709F846BB4E 99.1936 8.0%     73    1.3588 0.50 0.00  SELECT tbl_ddd info tbl_eee tbl_ddd
# 8 0x4CAD792BF4A54CE9 53.7477 4.4%      4    13.4369 0.00 0.17  SELECT tbl_fff(tbl_eee)tbl_ddd
# 9 0x347319A37AC29893 39.1390 3.2%      69    0.5672 1.00 0.00  SELECT tbl_fff pt_game_base_score
# 10 0x7EF77B274F1C37D3 27.2826 2.2%      4    6.8207 0.00 0.00  SELECT ccc(tbl_eee)tbl_ddd bbb
# 11 0x8383B2CB219358F3 16.7553 1.4%     18    0.9308 0.97 0.00  SELECT tbl_iii(tbl_hhh)tbl_eee
# MISC 0xMISC           51.5793 4.2%      44    1.1723 NS 0.0 <11 ITEMS>
# Query 1: 0.00 QPS, 0.00x concurrency, ID 0xBE5D289C750F172A at byte 120071
# This item is included in the report because it matches --limit.
# Scores: Apdex = 0.00 [1.0]*, V/M = 0.08
# Query_time sparkline: | ^ |
# Time range: 2012-09-25 11:01:09 to 2012-10-16 10:14:31
# Attribute          pct      total      min      max      avg      95%    stddev   median
# ======          ======      ==      ==      ==      ==      ==      ==      ==
# Count             7        40
# Exec time        25      309s      7s      10s      8s      9s      802ms      7s
# Lock time        6       4ms      59us    110us    89us    103us    12us    91us
# Rows sent         2       40      1       1       1       1       1       0       1
# Rows examine    45 281.88M    6.73M    7.29M    7.05M    6.94M   183.33k    6.94M
# Query size        5      7.19k      184      184      184      184      0      184
# String:
# Hosts
# Users          db_user
# Query_time distribution
#   lus
#   10us
#   100us
#   1ms
#   10ms
#   100ms
#   10s+
# Tables
#   SHOW TABLE STATUS LIKE 'ccc'\G
#   SHOW CREATE TABLE `tbl_eee`\G
#   SHOW TABLE STATUS LIKE 'tbl_ddd'\G
#   SHOW CREATE TABLE `bbb`\G
#   SHOW TABLE STATUS LIKE 'ggg'\G
# EXPLAIN /*!50100 PARTITIONS*/
select ... from ....
```

输出格式的解释具体如下。

Rank: 所有查询日志分析完毕后，此查询的排序。

Query ID: 查询的标识字符串。可以搜索这个字符串以快速定位到慢查询语句。

Response time: 总的响应时间，以及总占比，应优化占比较高的查询，对于比例较小的查询一般可以忽略，不进行优化。

Calls: 查询被调用执行的次数。

R/Call: 每次执行的平均响应时间。

Apdx: 应用程序的性能指数得分，响应时间越长，得分越低。

V/M: 响应时间的方差均值比。可说明样本的分散程度，这个值越大，往往是越值得考虑优化的对象。

Item: 查询的简单显示，包含了查询涉及的表。

对于报告中的如下输出，我们可以利用偏移量到慢查询日志里定位具体的sql语句。

```
# Query 1: 0.00 QPS, 0.00x concurrency, ID 0xBE5D289C750F172A at byte 120071
```

定位方法如下。

```
tail -c +120071 /path/to/slow.log. | head
```

查询响应时间的分布，这里使用了很形象的表示方式，如下所示。

```
# Query_time distribution
#   lus
#   10us
#   100us
#   1ms
#   10ms
```

```
# 100ms
#   ls #####
# 10s+ #
```

可以看到，有许多查询响应时间在1秒到10秒之间。

对于一些TOP SQL，我们可以使用EXPLAIN工具分析执行计划，进行调优。

对于更新语句，此工具会帮你改写成可以使用EXPLAIN工具的SELECT语句。

其他的一些使用方式示例如下。

1) 分析SHOW PROCESSLIST的输出。

```
./pt-query-digest --processlist S=/path/3307/mysql.sock,u=root,p=dxwd\* --interval 5 --run-time 5m
pt-query-digest --processlist h=host1 --print --no-report
```

2) 分析tcpdump的输出。

分析执行SQL的频率，一般在高峰期取样，一定要记得关闭tcpdump，因为生成的文件可能会很大。

首先运行命令。

```
nohup tcpdump -i eth1 port 3306 -s 65535 -x -nn -q -ttt > dbxx_sql_new.log &
```

过一段时间后，如1分钟，终止tcpdump任务。

然后使用pt-query-digest进行分析。

```
./pt-query-digest --type=tcpdump --watch-server 12.12.12.12:3306 dbxx_sql_new.log > report_to_developer.rtf
```

对生产环境的采样可以采取如上的方法，比如每分钟抓取5秒的网络包，然后把分析结果入库。利用监控系统及时发现问题，通知DBA或研发人员线上的性能问题。

3) 把分析过的SQL记录到历史信息表中，就可以知道分析的SQL最后出现的时间，如果已经解决掉了，就不用再优化了。具体步骤如下。

第一步，在存放优化信息的数据库中创建一个用户用于存放信息。

```
GRANT CREATE,SELECT,INSERT,UPDATE,DELETE ON ptool.* to ptool@'%' IDENTIFIED BY 'ptool';
```

第二步，执行如下命令分析慢查询日志。

```
/home/mysql/scripts/pt-query-digest --create-review-table --reviewh=13.13.13.13,P=3305,u=ptool,p=ptool,D=ptool,t=query_review --create-review-history-table --
review-history h=13.13.13.13,P=3305,u=ptool,p=ptool,D=ptool,t=query_review_history --report-all /path/to/log3307/slowquery.log
```

以上命令如果是第一次运行，则会将信息存储到指定的表中，以后再次运行时，如果表中的某条SQL的reviewed_by列已有设定值，那么此工具就不会显示标记了的SQL。如果要显示所有SQL，那么需要使用选项--report-all。

如果有--report-all和query_review表（表中记录的SQL及你添加的意见等信息），那么生成的报告里将带有你检查过的SQL的意见，这点会很有用。

第三步，可以运行如下命令查询Top SQL。

```
SELECT * FROM `query_review_history` WHERE ts_max > '2012-09-20 00:00:00' ORDER BY ts_cnt DESC , query_time_sum DESC LIMIT 3;
```

可以按照checksum（数据表里的是十进制的显示，报告里的是十六进制的显示方式）去数据表中查询对应的SQL，记录自己的优化意见，命令如下。

```
SELECT * FROM `query_review` WHERE CHECKSUM=0xB76366269B6B4973;
```

4) 报告不记录到历史信息表中，只记录简单的信息。

```
/home/mysql/scripts/pt-query-digest --create-review-table --review h=13.13.13.13,P=3305,u=ptool,p=ptool,D=test,t=query_review /path/to/log3307/slowquery.log
```

存放信息的表需要我们手动建立，或者添加选项--create-review-table。每次重新运行以上命令时，都会重新更新表内的值。比如最早、最近出现的时间。

5) 使用pt-query-digest分析通用日志和二进制日志。

分析通用日志示例如下。

```
pt-query-digest --type genlog general.log > /tmp/xxx.log
```

分析二进制日志示例如下。

```
pt-query-digest --type binlog \
--group-by fingerprint \
--limit "100%" \
--order-by "Query_time:cnt" \
--output report \
--report-format profile \
/tmpp/xxx.log
```



注意 /tmp/xxx.log日志是文本形式的二进制日志。

(3) pt-stalk

pt-stalk的语法格式如下。

```
Usage: pt-stalk [OPTIONS]
```

即使我们有了完备的性能收集程序，对于一些突然的性能波动也仍然会难以捕捉，如果是偶发性的性能问题，几天才发生一次，那么持续地对系统收集大量信息不仅会显得没有必要而且还会耗费太多资源。pt-stalk这个工具有助于解决此类问题，它可以在一定的条件下被触发，用于收集系统信息。我们可以查询SHOW GLOBAL STATUS命令结果绘制的图形，查看是否某个变量发生了突变，然后选择这个变量做一个启动pt-stalk的条件，收集足够的信息。比较好的一个衡量的阈值是Threads_running，比如，我们可以将某个生产环境Threads_running的阈值设置为20，pt-stalk每秒监控status的变量Threads_running，如果连续5秒都超过20，那么就可以开始收集统计信息了。注意，触发的阈值不能设置得太高，因为会导致不能发现故障，或者导致故障发生的真正原因已经过去，当然，阈值也不能太低，因为可能会误报警。

pt-stalk是一个后台程序，默认我们可以通过文件/var/log/pt-stalk.log，查看pt-stalk的运行状态。如下命令将检查pt-stalk的日志。

```
tail -f /var/log/pt-stalk.log
2013_07_09_10_24_04 Check results: status(Threads_running)=55, matched=yes, cycles_true=1
2013_07_09_10_25_03 Check results: status(Threads_running)=51, matched=yes, cycles_true=1
2013_07_09_10_26_04 Check results: status(Threads_running)=44, matched=yes, cycles_true=1
2013_07_09_10_28_04 Check results: status(Threads_running)=62, matched=yes, cycles_true=1
2013_07_09_10_28_05 Check results: status(Threads_running)=57, matched=yes, cycles_true=2
2013_07_09_10_29_03 Check results: status(Threads_running)=46, matched=yes, cycles_true=1
2013_07_09_10_29_04 Check results: status(Threads_running)=56, matched=yes, cycles_true=2
```

pt-stalk将收集的数据默认放在目录/var/lib/pt-stalk下，你可以使用参数--dest指定你希望存放数据的目录。你还可以使用--notify-by-email参数指定邮件报警联系人。

如下示例中，pt-stalk运行在后台（--daemonize），监视SHOW GLOBAL STATUS中的Threads_running状态，如果Threads_running的值超过了64，就将状态信息记录到日志里。pt-stalk每秒钟检测一次Threads_running值，如果连续5次满足触发条件，就开始收集主机和MySQL信息。我们使用--dest指定存放数据的目录，使用--disk-pct-free来定义磁盘的剩余空间阈值，如果剩余空间小于20%，则不再进行数据收集。--iterations可限制收集数据的次数，默认情况下pt-stalk会永久执行。--collect-tcpdump表示还要调用tcpdump收集网络包信息。

```
pt-stalk --pid /path/to/pt-stalk.pid --dest /path/to/data --disk-pct-free 20 --log /path/to/log/pt_stalk.log --collect-tcpdump --function status \
--variable Threads_running --threshold 64 \
--iterations 2000 --notify-by-email=garychen@db110.com --daemonize --user=root --password=your_password -S /tmp/mysql.sock
```

除了常用的threads_running，我们还可以使用SHOW PROCESSLIST的输出值触发pt-stalk，例如“--function processlist--variable State--match statistics--threshold 20”表示，show processlist输出中State列的值为statistics的线程数如果超过20则触发收集。

性能故障时刻，我们应该尽可能地收集操作系统和MySQL的信息，不仅要收集正在执行的任务信息，还要收集正在等待资源的信息，因为我们并不能确定是执行慢还是等待了太多资源。这个工具还可以调用tcpdump来收集网络包信息，然后我们再调用pt-query-digest进行分析，命令如下。

```
tcpdump -r 2013_04_30_18_20_48-tcpdump -nn -x -q -ttt |pt-query-digest --type tcpdump --watch-server ip:port >report.rtf
```

其他的一些参数及说明如下。

--run-time：触发收集后，该参数将指定收集多长时间的数据。默认是30秒。

--sleep：为防止一直触发收集数据，该参数指定在某次触发后，必须sleep一段时间再继续观察并触发收集。默认是300秒。

--cycles：默认情况下pt-stalk只有连续5次观察到状态值满足触发条件时，才会触发收集。

有了数据之后，我们就可使用pt-sift对收集的数据进行分析，这个工具将帮助我们分析pt-stalk收集到的信息，它会自动下载其他需要用到的工具。

(4) pt-sift

pt-sift的语法格式如下。

```
pt-sift FILE|
PREFIX|
DIRECTORY
```

如果存在/var/lib/pt-stalk，则默认读取/var/lib/pt-stalk下的所有文件，否则读取当前目录下的文件。

如果是非默认目录，则请指定自己的工作目录，命令如下。

```
./pt-sift /path/to/data
```

如果是指定文件名或前缀，则它会到默认目录里去查找。例如如下命令。

```
./pt-sift /var/lib/pt-stalk/2012_09_07_00_00 #在默认目录
/var/lib/pt-stalk里查找
```

```
以
2012_09_07_00_00为前缀的文件分析。
./pt-sift /var/lib/pt-stalk/2012_09_07_00_00_13 #在默认目录里查找
以
2012_09_07_00_00_13为前缀的文件分析。
```

如下是pt-sift的一个输出，这里仅显示磁盘信息。

```
===== db1000 at 2012_09_07_00_00_13 DEFAULT (6 of 6) =====
--diskstats--
#ts device rd_s rd_avkb rd_mb_s rd_mrg rd_cnc rd_rt wr_s wr_avkb wr_mb_s wr_mrg wr_cnc wr_rt busy in_prg io_s qtime stime
(29) sdb1 1637.9 16.4 26.3 1% 4.4 2.6 614.4 26.6 16.0 85% 0.9 0.2 63% 0 2252.3 0.8 0.1
sdb1 0% 35% .25% 20% .30% 20% 25% 20% .15% .5% .10% .5% .10% 15% 5%
--vmstat--
r b swpd free buff cache si so bi bo in cs us sy id wa st
r 16 223328 96528 131056 11066620 0 0 109 387 0 0 6 2 92 1 0
2 0 223328 96728 123476 10941780 13 0 27233 19077 17420 50895 10 4 73 14 0
wa 0% 20% 15% .25% .30% 25% 30% .25% 10% 5% 0% .5% 0% . . . . .5% .
--innodb--
txns: 66xACTIVE (9s)
16 queries inside InnoDB, 552 queries in queue
Main thread: sleeping, pending reads 42, writes 0, flush 0
Log: lsn = 2234, chkp = 2233, chkp age = 1
Threads are waiting at:
40 trx/trx0trx.c line 213
8 trx/trx0trx.c line 1591
5 lock/lock0lock.c line 3592
4 trx/trx0trx.c line 722
1 trx/trx0trx.c line 940
1 srv/srv0srv.c line 2101
1 lock/lock0lock.c line 4835
1 btr/btr0sea.c line 947
Threads are waiting on:
1 S-lock on RW-latch at 0x2aaab72410b8 created in file btr/btr0sea.c line 139
--processlist--
State
404 Sending data
105 statistics
93
42 Updating
10 cleaning up
Command
557 Query
103 Sleep
1 Binlog Dump
--stack traces--
No stack trace file exists
--oprofile--
No oreport file exists
```

(5) pt-align

pt-align的语法格式如下。

```
pt-align [FILES]
```

可以把其他工具的输出格式化为按列排齐。

如aaa.log包含了如下内容。

```
DATABASE TABLE ROWS
foo bar 100
long_db_name table 1
another_long_name 500
```

可以使用pt-align输出转换为如下形式，现在文字是对齐的。

```
pt-align aaa.log
DATABASE      TABLE      ROWS
foo           bar        100
long_db_name  table      1
another_long_ name     500
```

pt-align也可正确地处理空白字符（如空格、TAB），我们可以用它来格式化和vmstat、iostat的输出，移除一些不需要显示的内容。

(6) pt-archiver

pt-archiver的语法格式如下。

```
pt-archiver [OPTION...] --source DSN --where WHERE
```

pt-archiver可将MySQL数据库中表的记录归档到另外一个表或文件中，也可以直接进行记录的删除操作。

工作原理： pt-archiver工具能够智能地选择表上的索引，从源表中分批次找出符合WHERE条件的数据，根据要求把表数据归档成csv格式或将表数据插入到归档表中，然后删除源表中的数据（默认删除）。

工作过程：从源表SELECT数据，插入到新表或归档到文件，然后删除源表的数据。通过这种方式，保证只有归档成功了才删除源表的数据。

这个工具可用于迁移数据，可以减少对线上OLTP应用的影响，它可以小批量地把OLTP数据库上的数据导入OLAP数据库中。也可以将它写入一个文件中，方便使用LOAD DATA INFILE命令导入数据。我们还可以用它来实现增量的删除操作。



注意 默认归档数据的同时会删除生产库上的数据，因此在生产环境中使用时一定要慎重。

如下将介绍一些示例。

从OLTP数据库归档表tbl到OLAP数据库，并且归档到一个文件中。

```
pt-archiver --source h=oltp_server,D=test,t=tbl --dest h=olap_server  
--file '/var/log/archive/%Y-%m-%d-%D.%t'  
--where "l=1" --limit 1000 --commit-each
```

删除子表的孤立数据，这部分数据在父表中不存在关联的信息。

```
pt-archiver --source h=host,D=db,t=child --purge  
--where 'NOT EXISTS(SELECT * FROM parent WHERE col=child.col)'
```

将test库的userinfo表中id小于10000的记录归档到/home/mysql/tmp/userinfo_archive_20131010.log文件中。

```
pt-archiver --source h=host,D=test,t=userinfo --user=root --password=your_password --file '/home/mysql/tmp/userinfo_archive_20131010.log' --where "id<=10000" --commit-each
```

其中的参数及说明如下。

--limit: 控制导出数据归档的粒度，默认是1。

--commit-each: 控制在每批次归档数据的时候提交。

其他的一些选项及说明如下。

--no-delete: 不删除源表的数据。

--progress: 每进行一次归档或删除，都显示所耗时间的信息，并可以据此预估总时间。

--statistics: 结束的时候给出统计信息，包括开始的时间点、结束的时间点、查询的行数、归档的行数、删除的行数，以及各个阶段所消耗的总的时间和比例，便于后续以此进行优化。

--columns: 需要导出哪些列，列名用逗号隔开。

--sleep: 在前后两次导出之间sleep（休息）的时间。避免给服务器造成较大的压力。如果同时设置了--commit-each选项，那么提交和刷新文件的操作应在sleep之前发生。

--primary-key-only: 只选择主键列。对于删除表数据的场景，该选项可以避免取回整行数据，因此也更高效。

--ignore或--replace选项: 使用insert ignore或replace语句可代替insert语句。

(7) pt-config-diff

pt-config-diff的语法格式如下。

```
pt-config-diff [OPTION...] CONFIG CONFIG [CONFIG...]
```

它的功能是检查配置文件和服务器变量之间的差异。DBA有时变更MySQL全局变量，但忘记了同步修改配置文件，这样可能会导致隐患，因为有时重启后，又使用了旧的配置，另外，这个工具可以用来在迁移或重新搭建环境的时候，确保新的迁移环境的配置和原来的生产环境的配置一致。

如下将介绍一些示例。

对比host1和host2的变量，可用如下命令。

```
pt-config-diff h=host1 h=host2
```

检查本地实例和本地配置文件，可用如下命令。

```
pt-config-diff u=root, P=3306, S=/tmp/mysql.sock, p=password /etc/my.cnf
```

对比两个配置文件，可用如下命令。

```
pt-config-diff /etc/my-small.cnf /etc/my-large.cnf
```

(8) pt-show-grants

pt-show-grants的语法格式如下。

```
pt-show-grants [OPTION...] [DSN]
```

它是导出权限的工具。方便我们在配置主从、迁移数据库的时候比对权限。有时我们并不想通过导出导入系统库mysql来实现权限的配置，使用这个工具可以生成更友好的赋权语句，我们在目标库上直接执行即可。利用这个工具也可以方便地收回权限。

如下将介绍一些示例。

1) 导出权限，可用如下命令。

```
pt-show-grants u=root, P=3306, S=/tmp/mysql.sock, p=password
```

2) 查看每个用户的权限生成revoke收回权限的语句如下。

```
pt-show-grants --host='localhost' --user='root'  
--password='password' --revoke
```

(9) pt-summary

pt-summary的语法格式如下。

```
pt-summary
```

这个工具仅收集系统信息，它不是一个用于调优诊断的工具。它可以生成一个友好的报告，可展示系统的平台、CPU、内存、磁盘、网络、文件系统、进程等各种信息，让你对基础环境有一个很好的概览。

pt-summary会运行许多命令去收集系统的状态和配置信息，先将这些信息保存到临时目录的文件中去，然后运行一些Unix命令对这些结果做格式化，建议用root用户或有权限的用户运行此命令。

示例如下，这里只截取了报告的部分内容。

```
$ ./pt-summary  
# Percona Toolkit System Summary Report #####  
# Date | 2014-10-10 01:55:37 UTC (local TZ: CST +0800)  
Hostname | db1000  
Uptime | 182 days, 19:36, 1 user, load average: 1.82, 1.26, 0.90  
Platform | Linux  
Release | CentOS release 6.4 (Final)  
Kernel | 2.6.32-358.el6.x86_64  
Architecture | CPU = 64-bit, OS = 64-bit  
Threading | NPTL 2.12  
SELinux | Disabled  
Virtualized | No virtualization detected  
# Processor #####  
Processors | physical = 2, cores = 12, virtual = 24, hyperthreading = yes  
Speeds | 21x1200.000, 1x1600.000, 2x2101.000  
Models | 24xIntel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz  
Caches | 24x15360 KB  
# Memory #####  
Total | 126.0G  
Free | 1.8G  
Used | physical = 124.2G, swap allocated = 16.0G, swap used = 231.4M, virtual = 124.4G  
Buffers | 215.2M  
Caches | 17.2G  
Dirty | 135072 kB  
UsedRSS | 120.4G  
Swappiness | 60  
DirtyPolicy | 20, 10  
DirtyStatus | 0, 0
```

(10) pt-mysql-summary

pt-mysql-summary的语法格式如下。

```
pt-mysql-summary [OPTIONS] [-- MYSQL OPTIONS]
```

其中，“-”后的参数是传递给MySQL的。

这个工具用于收集MySQL信息，并生成友好的报告给我们阅读。这个工具的主要功能是对MySQL的配置和STATUS信息进行汇总。它所生成的报告可以告诉我们，当前系统上存在哪些MySQL实例，主机的一般信息，对SHOW PROCESSLIST的一些分析总结，以及变量的设置。并对STATUS状态变量进行采样，显示各种状态变量的增量变化。

如下例子将汇总本地MySQL服务器的信息。

```
pt-mysql-summary -- --user=root --password='password' --socket=/tmp/mysql.sock
```

其他参数如下。

--sleep：采样GLOBAL STATUS时，间隔多少秒。默认是10秒。

(11) pt-fifo-split

pt-fifo-split的功能是模拟切割文件并通过管道传递给先入先出队列（FIFO）而不用真正地切割文件。

当InnoDB使用load data的方式导入一个巨大的文件时，会创建一个很大的事务，产生很多UNDO。如果异常回滚的话，会很耗时，可能会远远超过导入数据的时间，所以更合理的方式是分批导入数据，那么如何在不切割数据文件的情况下达到分批导入数据的目的呢？使用Unix fifo即可。

例1：如下是一个每次读取一百万行记录的范例。

```
pt-fifo-split --lines 1000000 hugefile.txt
while [ -e /tmp/pt-fifo-split ]; do cat /tmp/pt-fifo-split; done
```

例2：每次读取一百万行，指定fifo文件为/tmp/my-fifo，并使用LOAD DATA命令导入数据。

```
CREATE TABLE load_test (
    col1 bigint(20) NOT NULL,
    col2 bigint(20) default NULL,
    key(col1),
    key(col2)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

一个窗口：

```
pt-fifo-split infile.txt --fifo /tmp/my-fifo --lines 1000000
```

另一个窗口：

```
while [ -e /tmp/my-fifo ]; do
    mysql -e "set foreign_key_checks=0; set sql_log_bin=0; set unique_checks=0; load data local infile '/tmp/my-fifo' into
    table load_test fields terminated by '\t' lines terminated by '\n' (col1, col2);"
    sleep 1;
done
```

如果在mysql命令提示符下使用LOAD DATA导入数据将会出现乱码，请设置SET character_set_database=字符集，或者修改LOAD DATA命令，添加character set语句，具体如下。

```
mysql -e "set foreign_key_checks=0; set sql_log_bin=0; set unique_checks=0; load data local infile '/home/mysql/db110.data' into table test.db110 character set
gbk fields terminated by '\t' lines terminated by '\n';"
```

(12) pt-duplicate-key-checker

这个工具的功能是查找重复的索引和外键。这个工具会将重复的索引和外键都列出来，并生成删除重复索引的语句。其原理是检查SHOW CREATE TABLE的输出，查找重复或冗余的信息。冗余指的是索引的字段是其他索引的最左部分。

示例如下。

```
./pt-duplicate-key-checker --user=root --password=password --socket=/tmp/mysql.sock
```

(13) pt-slave-delay

pt-slave-delay的语法格式如下。

```
pt-slave-delay [OPTION...] SLAVE-HOST [MASTER-HOST]
```

SLAVE-HOST[MASTER-HOST]可以使用DSN语法。

这个工具的用途是设置从服务器滞后于主服务器的时间。MySQL同步在快速的网络中是毫秒级的，如果有误操作，从库很快就变更了，对于一些频繁进行，不是经过严格测试的升级，可能会带来风险。可考虑配置一个延迟复制的副本，以改善故障情况下的可恢复性。

MySQL5.6版本已经支持延迟复制，如果是5.1版本，可以用pt-slave-delay来设置延迟。

工作原理：通过启动和停止复制SQL线程来设置从服务器落后于主服务器的指定时间。默认是基于从服务器上relay日志的二进制日志的位置来判断，因此不需要连接到主服务器。

检查主库传输过来的日志的位置信息（可以用SHOW SLAVE STATUS命令查看relay日志），对比本地已经应用的日志的位置信息，就能知道延迟的时间了。

如果IO线程不落后主服务器太多的话，这个检查方式就能工作得很好。一般是通过--delay和--delay加--interval来控制的。--interval间隔多久检查一次，默认设置是1分钟检查一次，即每隔1分钟检查一次延迟，通过不断启动和关闭复制SQL线程来保持主从一直延迟固定的时间。

正常运行时，如果关闭了数据库，那么这个工具会每隔15秒重试一次连接，连续几次之后，如果还是不能连接，那么就会异常退出。

```
./pt-slave-delay u=xxxx,S=/tmp/mysql.sock,p=password --log /path/to/log/delay.log --daemonize
```

以上命令将以daemon模式在后台运行，从库将保持一直滞后主库1小时。

--delay：设置延迟时间，默认为1小时。

--interval：设置检查点频率，每次检查间隔多久，默认是1分钟（1m）。

--delay和--interval可选的时间允许使用不同的单位，如，s秒、m分、h小时、d天。

--log：日志，可以检查日志输出从而了解其原理和运行机制。

```
pt-slave-delay --delay 1m --interval 15s --run-time 10m slavehost
```

以上命令将运行这个工具10分钟（默认是永久运行的）。从库保持一直滞后主库1分钟，每次检查间隔15秒，因此理论上是延迟1分钟15秒。

如果正在运行这个工具，而且这个工具已经停止了复制SQL线程，那么当我们按【Ctrl+C】退出这个工具时，它会友好地退出，意即它会启动复制SQL线程，并恢复现场。

连接MySQL的用户需要如下权限：PROCESS、REPLICATION CLIENT、SUPER。

如果启动出错，则会报错如下：“Had to create DBD:mysql:dr::imp_data_size unexpectedly”，这时建议升级Perl。

(14) pt-online-schema-change

pt-online-schema-change的语法格式如下。

```
pt-online-schema-change [OPTIONS] DSN
```

长期以来困扰DBA的一个问题是，MySQL在线修改表结构的能力很弱，对于大表修改表结构将会很耗时，还会影响到服务。为了减少对服务的影响，可能需要进行主从切换，或者选择特定的时间进行升级。MySQL新的版本5.6和5.7增强了数据库在线修改表结构的功能，对于早期的版本，我们可以试试pt-online-schema-change这款工具。

这个工具的功能是实现在不锁表的情况下修改表结构。这点对于在线应用很重要，这样在修改表结构的同时，数据库还可以继续提供读写服务。

工作原理：创建一个和原表表结构一样的新表，新表为空数据，对新表进行表结构修改，然后从原表中复制数据到新表，当数据复制完成以后就进行新旧表的切换，新表被命名为原表的名字，默认动作会将原表删除。在复制数据的过程中，任何在原表中所做的更新操作都会更新到新表，因为这个工具会在原表上创建触发器，触发器会捕获原表上的更新，并将它们更新到新表。

上述过程中，原表复制数据到新表是分批分批复制记录到新表的，也有相关的参数可以控制负载，如--max-load。

示例如下。

向表sakila.actor中添加一个列。

```
pt-online-schema-change --alter "ADD COLUMN c1 INT" D=sakila,t=actor
```

更改表的引擎为InnoDB。

```
pt-online-schema-change --alter "ENGINE=InnoDB" D=sakila,t=actor
```

注意事项有如下几点。

- 表需要有主键或唯一索引。
- 如果有外键，请仔细阅读官方文档。
- 需要确保原表中之前没有触发器。
- 利用此工具修改表结构，建议先进行备份。
- 切换新旧表的时候，会导致连接中断，需要确保应用中有重连的机制。
- 如果已经有长事务在操作这个表，那么这个工具可能会因为等不到锁而超时退出。
- 可能导致复制延时。
- 推荐使用独立表空间，以便释放空间。

(15) pt-kill

pt-kill的语法格式如下。

```
pt-kill [OPTIONS] [DSN]
```

kill掉满足某些条件的MySQL查询。

pt-kill获取SHOW PROCESSLIST的信息，对信息进行过滤，打印满足条件的连接，或者kill掉这些连接。主要的目的是kill掉那些严重消耗资源的查询，以保障服务的正常运行。pt-kill也可以检查运行SHOW PROCESSLIST命令的输出文件，打印满足条件的连接，这种情况下，不需要连接MySQL去kill掉连接。

参数及其说明如下。

--busy-time: 匹配运行时间超过busy-time的查询。这些查询的SHOW PROCESSLIST输出的Command列为Query，Time列大于busy-time指定的值，才会被匹配。

--victims: 指定哪些匹配的查询会被kill掉或被打印。有如下三个值。

· oldes: 默认值，kill掉运行时间最长的查询。

· all: kill掉所有查询。

· all-but-oldest: kill掉了运行时间最长的查询之外的所有查询。有时我们并发了许多同样的查询，这时我们只需要确保最长运行时间的那条查询能够执行成功即可，这种情况下，我们可以使用这个选项。

--kill: kill匹配条件的连接。

--print: 打印kill语句，并不会真正地kill掉连接。如果--kill和--print都指定了，那么不仅要kill掉匹配的连接，也要打印被kill掉的连接。

pt-kill的工作过程具体如下。

pt-kill命令kill连接需要4个步骤。了解这4个步骤，有助于你理解这个工具的使用，并能准确选择要kill掉的连接。4个步骤具体如下。

- 1) 分组查询到不同的类别。--group-by选项可控制分组。默认情况下，这个选项没有值，所有查询都将被分组到一个默认的类中。
- 2) 进行匹配。每个类别都要进行匹配。首先，查询会被不同的查询匹配选项过滤，如--match-user。然后，查询会被不同的类匹配选项过滤，如--query-count。
- 3) victim (kill候选者) 选择。如果一些查询被过滤出来，那么它可以被kill掉，--victims控制哪些查询会被kill掉。一般情况下，你可能会选择kill掉运行时间最长的查询，或者希望kill掉所有匹配到的查询。
- 4) 对选择的查询执行操作。如kill、print。

如下将介绍一些示例。

kill运行时间超过60s的查询，默认情况下kill最长时间的查询，命令如下。

```
pt-kill --busy-time 60 --kill
```

仅仅打印运行时间超过60s的查询，而不是kill掉它们，命令如下。

```
pt-kill --busy-time 60 --print u=root,S=/tmp/mysql.sock,p=password
```

每隔10s检查一次sleep状态的所有连接，并kill掉它们，注意参数--victims all，命令如下。

```
pt-kill --match-command Sleep --kill --victims all --interval 10
```

打印所有的登录连接，命令如下。

```
pt-kill --match-state login --print --victims all
```

检查SHOW PROCESSLIST输出的文件，查看哪些连接匹配条件，命令如下。

```
mysql -e "SHOW PROCESSLIST" > proclist.txt  
pt-kill --test-matching proclist.txt --busy-time 60 --print
```

kill掉运行时间超过120s且运行时间最长的那个连接，命令如下。

```
./pt-kill --busy-time 120 --match-command Query --match-db db_name --match-user user_name --kill --print u=root,S=/tmp/mysql.sock,p=password
```

--match-db和--match-user限定了数据库名和用户名，以免误操作。

kill掉运行时间超过120s的所有连接，命令如下。

```
./pt-kill --busy-time 120 --match-command Query --match-db db_name --match-user user_name --victims all --kill --print u=root,S=/tmp/mysql.sock,p=password
```

kill掉运行时间超过600s且正在创建临时表的所有查询，命令如下。

```
./pt-kill --busy-time 600 --match-command Query --match-db db_name --match-user user_name --match-state "Copying to tmp table" --victims all --kill
```

(16) pt-visual-explain

这个工具将格式化EXPLAIN出来的执行计划，并按照Tree方式输出，以方便阅读。

语法格式如下。

```
pt-visual-explain [OPTION...] [FILE...]
```

示例如下。

```
pt-visual-explain <file containing explain output>  
pt-visual-explain -c <file containing query>  
mysql -e "explain select * from mysql.user" | pt-visual-explain
```

(17) pt-slave-restart

这个工具的作用是检查MySQL的复制状态，处理指定的MySQL复制错误。比较常用的使用方式是，忽略指定的MySQL错误号，重新启动复制SQL线程。建议不要滥用这个工具，仅在你明确知道复制错误可以忽略的时候，才使用这个工具忽略掉复制错误。如果错误太多导致了严重的数据不一致，那么建议重建整个从库。

语法格式如下。

```
pt-slave-restart [OPTIONS] [DSN]
```

常用参数及其说明如下。

--verbose：可以显示复制错误。

--error-length：显示复制错误的长度。

--daemonize：后台模式。

--log：当是daemonize模式时，输出日志到这里。

--error-numbers：匹配了错误号才处理，错误代码代号之间使用逗号进行分隔，对应SHOW SLAVE STATUS输出里的last_errno。

--error-text：匹配了错误文本才处理，对应SHOW SLAVE STATUS输出里的last_error。

--run-time: 运行多久才退出， 默认以秒为单位， 其他单位s=seconds、m=minutes、h=hours、d=days。

--skip-count: 当重新启动slave时， 应跳过多少条语句， 默认值是1。

--sleep: 每次检查复制状态的间隔休眠时间。

--until-master: 重启slave， 直到从库应用日志到指定的主库日志位置。

如下例子将检查MySQL服务器的复制， 跳过错误代码为1062的复制错误。

```
pt-slave-restart --verbose --error-numbers=1062 --run-time=60 u=root,S=/tmp/mysql.sock,p=password
```

(18) pt-diskstats

pt-diskstats是一个交互式的监控系统I/O的工具， 这个工具类似于iostat， 但显示的信息更具可观察性， 它也可以分析从其他机器收集来的数据。

其实现的原理是读取/proc/diskstats中的数据进行展示。

语法格式如下。

```
pt-diskstats [OPTION...] [FILES]
```

按q可退出。按“?”显示帮助。

常用参数及其说明如下。

--interval: 默认为1， 设置对/proc/diskstats采用的间隔。

--iterations: 运行多少次， 默认情况下是永久运行。

示例如下所示。

```
./pt-diskstats --interval=5  
./pt-diskstats --interval=2 --iterations 10
```

如图16-13所示， 是pt-diskstats的一个输出。

Device	rd_s	rd_avkb	rd_mb_s	rd_mrg	rd_cnc	rd_rt	wr_s	wr_avkb	wr_mb_s	wr_mrg	wr_cnc	wr_rt	busy	ios_s	qtime	stime
/dev/sda	0.0	4.0	0.0	25	0.0	10.0	2.5	9.8	0.0	592	6.0	0.5	0.5	0.0	0.0	0.0
/dev/sdb	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	688	6.0	0.0	0.0	0.0	0.0	0.0
/dev/sdc	0.0	0.0	0.0	76	0.0	0.0	0.0	0.0	0.0	624	6.0	0.0	0.0	0.0	0.0	0.0
/dev/sdd	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	624	6.0	0.0	0.0	0.0	0.0	0.0
/dev/sde	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	624	6.0	0.0	0.0	0.0	0.0	0.0
/dev/sdf	0.0	0.0	0.0	25	0.0	0.0	0.0	0.0	0.0	513	6.0	0.0	0.0	0.0	0.0	0.0
/dev/sdg	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	568	6.0	0.0	0.0	0.0	0.0	0.0
/dev/sdh	0.0	0.0	0.0	15	0.0	0.0	0.0	0.0	0.0	903	1.4	0.0	0.0	0.0	0.0	0.0
/dev/sdi	0.0	5.0	0.0	14	0.0	0.0	55.0	100.0	0.0	303	1.4	0.0	0.0	0.0	0.0	0.0

图16-13 pt-diskstats的输出图例

图16-13中的输出项及其解释如下。

rd_s: 每秒读的次数， 此数值是每秒平均读次数， 表征了每秒实际发送到底层物理设备的读请求的次数。

rd_avkb: 每次读的平均大小， 单位是KB（千字节）。

rd_mb_s: 每秒读多少MB。

rd_mrg: 请求在发送给底层实际物理设备前， 被I/O调度合并的百分比。

rd_cnc: 读操作的平均并发。

rd_rt: 读操作的平均响应时间， 以毫秒为单位。

wr_s、wr_avkb、wr_mb_s、wr_mrg、wr_cnc、wr_rt对应于rd_*相关的解释。

busy: 对应iostat的%util。

in_prg: 正在进行请求的数目。

ios_s: 物理设备的平均吞吐量， 即IOPS（每秒IO）， 它是rd_s和wr_s的总和。

qtime: 平均排队时间， 即请求在被发送到物理设备前，在调度队列里等待的时间。

stime: 平均服务时间， 即实际物理设备处理请求的平均时间。

注意块设备（block device）和物理设备（physical device）的区别。

·块设备，如`/dev/sda1`，应用程序以文件系统的方式访问它，逻辑I/O发生在此。

·物理设备指的是底层的实际物理设备，比如磁盘、RAID卡，物理I/O发生在此。

我们所说的队列指的是与块设备相关的队列，队列保存读写请求直到这些请求被实际发送给物理设备为止。

(19) pt-deadlock-logger

`pt-deadlock-logger`是一个死锁检测工具，适用于InnoDB引擎，可以提取和记录InnoDB的最近的死锁信息。

语法格式如下。

```
pt-deadlock-logger [OPTIONS] DSN
```

工作原理：检测死锁（通过`SHOW ENGINE INNODB STATUS\G;`），然后直接打印死锁信息，或者指定`--dest`参数将信息存入`test`库下的一个表`test.deadlocks`中。

可使用参数`--run-time`或`--iterations`来确定执行时间。

`--iterations`: 迭代检查多少次，如`--iterations 4--interval 30`表示检测4次，每次间隔30s。

`--run-time`: 运行此工具多久时间。这个参数的优先级比`iterations`更高，如`--run-time 1m`。

`--interval`: 检测死锁的频率，默认是30s。

`--dest`: 指定存储死锁信息的数据库表，需要预先创建表。

如下将介绍一些示例。

检测死锁，输出至屏幕，检测10次即可，命令如下。

```
pt-deadlock-logger u=root,S=/tmp/mysql.sock,p=password --iterations 10
```

检测死锁，并把死锁信息存入`test`库的表中，命令如下。

```
pt-deadlock-logger u=root,S=/tmp/mysql.sock,p=password --dest D=test,t=deadlocks
```

检测死锁，并把死锁信息存入另外一个数据库中，命令如下。

```
pt-deadlock-logger SOURCE_DSN --dest DEST_DSN,D=test,t=deadlocks
```

运行4个小时，每次间隔30秒，在后台运行，检查死锁信息中，命令如下。

```
pt-deadlock-logger SOURCE_DSN --dest D=test,t=deadlocks --daemonize --run-time 4h --interval 30s
```

注意频繁调用`SHOW INNODB STATUS`，也可能对生产系统产生影响，对于负载较重的MySQL数据库，建议每次间隔大于30s以上。

(20) pt-table-checksum

`pt-table-checksum`这个工具的目的是在线检测MySQL的主从一致性。

数据库的主从由于如下原因可能会出现不一致。

·从库被误写了。

·数据库主机宕机导致MyISAM表损坏。

·数据库实例崩溃后，我们指定了不准确的日志点重新进行同步。

·基于语句的复制。

·一些Bug，特别是一些非核心的功能，比如存储过程，可能会导致复制出错，从而导致主从数据的不一致。

我们需要确保主从数据是一致的。有时我们在做了迁移或升级之后，也希望能有一个工具来确认主从数据的一致性。`pt-table-checksum`这个时候就可以派上用场了。

该工具的工作原理具体如下：

这个工具通过对比主从数据内容的CRC值来判断数据的一致性。它可以分批次地校验数据，以减少对生产的影响，它把一张表分为若干个trunk，如果一张表有300万行，分为100个trunk，那么每个trunk就是有3万行，它会锁定这个trunk，进行CRC值的计算。

运行此工具时，如果有权限将会自动创建如下的表格。

```
CREATE TABLE checksums (
    db          char(64)      NOT NULL,
    tbl         char(64)      NOT NULL,
    chunk       int           NOT NULL,
    chunk_time  float         NULL,
    chunk_index varchar(200)  NULL,
    lower_boundary text        NULL,
    upper_boundary text        NULL,
    this_crc   char(40)      NOT NULL,
    this_cnt   int           NOT NULL,
    master_crc char(40)      NULL,
    master_cnt  int           NULL,
    ts          timestamp     NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (db, tbl, chunk),
    INDEX ts_db_tbl (ts, db, tbl)
) ENGINE=InnoDB;
```

pt-table-checksum在主库上生成REPLACE INTO语句，然后通过复制传递到从库。类似如下的语句，是基于语句级别的复制，将这条语句复制到从库，从库会执行这条语句，可以知道，从库上的这条记录保存了从库内容的checksum值。checksum默认采用crc32值。

```
REPLACE INTO `percona`.`checksums` (db, tbl, chunk, chunk_index, lower_boundary, upper_boundary, this_cnt, this_crc)
SELECT
    `db_name`,
    `table_name`,
    '1',
    NULL,
    NULL,
    NULL,
    COUNT(*) AS cnt,
    COALESCE(LOWER(CONV(BIT_XOR(CAST(CRC32(CONCAT_WS('#', `id`, `name`, CONCAT(ISNULL(`name`)))) AS UNSIGNED)), 10, 16)), 0) AS crc
FROM `percona`.`garychen`
```

以上的crc列，简单说明一下，它会把一个trunk里面的每一行数据的所有字段都拼成一个String，然后对String取32位校验码，然后对这个trunk内所有计算好的校验码进行异或操作，从十进制转换成十六进制。

在主库中UPDATE更新master_src的值。运行命令类似如下的语句。

```
UPDATE `percona`.`checksums`
SET chunk_time = '0.000563', master_crc = '31012777', master_cnt = '4'
WHERE db = 'db_name' AND tbl = 'table_name' AND chunk = '1'
```

这个操作，同样会被复制到从库。

由上述示例可以得知，通过检测从库上的this_src和master_src列的值可以判断复制是否一致。命令类似如下的语句。

```
SELECT db, tbl, SUM(this_cnt) AS total_rows, COUNT(*) AS chunks
FROM percona.checksums
WHERE (
    master_cnt <> this_cnt
    OR master_crc <> this_crc
    OR ISNULL(master_crc) <> ISNULL(this_crc))
GROUP BY db, tbl;
```

可以使用pt-table-checksum-explain查看工具是如何检查表的。

它可以检测到从库，并自动连接它们。自动连接从库有多种办法，可以使用--recursion-method来指定，默认是PROCESSLIST，即通过检查SHOW PROCESSLIST里的输出来判断从库，然后去连接对应的从库。host方法是利用SHOW SLAVE HOSTS的输出信息判断从库，但这种方法在低版本中支持得不好。你也可以建立一个表存储从库的信息，通过检索这个表来连接从库进行数据比对。

它仅针对一台服务器执行checksum操作，一次只针对一个表进行checksum操作。如果显示有很多表，那么--replicate-check-only可仅显示存在差异的表。

如果被完全终止了，可以使用--resume选项继续执行，你也可以使用Ctrl加C退出。

chunk的大小也是可以动态调整的，调整的依据是checksum操作在一定时间内可以完成。

示例如下。

```
./pt-table-checksum h=ip, P=3306, u=test_gary, p=password
--databases=db_name --tables=tbl_name
--recursion-method=processlist
```

其他选项及说明具体如下。

--max-lag: 最大延迟，超过这个就等待。

--max-load: 最大负载，超过这个就等待。

--databases: 只检查某些库。

--tables: 只检查某些表。

注意事项有如下几点。

主从数据库的schema应该一致，否则复制可能会失败。

使用的时候应选择在业务低峰期运行，因为运行的时候会造成表的部分记录被锁定。虽然操作是对trunk逐个进行的，但是它会对每个trunk做SELECT FOR UPDATE，这样做主要是担心做checksum的时候会有写入，所以各个trunk都不适合太大。

pt-table-checksum提供了多种手段以确保尽量不会对生产环境造成影响，你可以使用--max-load来指定最大负载，如果达到最大负载，就暂停运行。你也可以设置超时时间innodb_lock_wait_timeout。

如果发现有不一致的数据，则可以使用pt-table-sync工具来进行修复。

如果表中没有主键或唯一索引，或者没有合适的索引，或者处于其他不适合检查的情况下，那么工具可能会忽略这个表。

(21) pt-table-sync

这个工具可以高效地进行数据表的同步。

语法格式如下。

```
pt-table-sync [OPTION...] DSN [DSN...]
```

其工作原理具体如下。

对比主从之间的差异，在主库上执行数据的更改（使用REPLACE INTO语句或DELETE语句），再同步到从库上。对于主库上存在，从库上不存在的数据，执行REPLACE INTO语句。对于从库上存在，主库上不存在的数据，执行DELETE语句。在主库上执行修改是基于主库现在的数据，所以REPLACE INTO语句不会更改主库上的数据。

这个工具会更改数据，所以如果需要使用这个数据在不同的MySQL库之间进行数据同步，那么建议先进行数据备份。主从实例可能会因为一些误操作或软硬件异常而导致数据出现不一致的问题，而使用这个工具可以修复主从之间的数据差异。使用这个工具修复主从库的不一致问题，必须先保证被修复的表上有主键或唯一键。

常用参数及其介绍具体如下。

--execute: 执行变更。

--print: 仅打印变更语句，可以把--execute参数换成--print先查看会变更什么数据。

--replicate: 指定一个同步列表，--replicate指定的表里存储了需要同步的表的信息。这个表实际上就是pt-table-checksum工具生成的校验信息，我们可以先利用pt-table-checksum工具进行校验，然后利用校验结果进行同步。

--sync-to-master: 指定从库，同步到主库。

--ignore-databases: 忽略同步的数据库列表，以逗号分隔。

--ignore-engines: 忽略同步的引擎列表，以逗号分隔。

--ignore-tables: 忽略同步的表，以逗号分隔。

如下将介绍一些示例，我们假设host1是主库，host2是从库，端口为3306。

1) 先使用pt-table-checksum进行校验，默认将校验结果存储在percona.checksums中。

```
./pt-table-checksum --user=user --password=password --host=host1 --port=port --databases=db_name --tables=tbl_name --recursion-method=processlist
```

2) 根据校验结果，修复从库中的数据。

```
./pt-table-sync --execute --replicate percona.checksums --sync-to-master h=host2,P=3306,u=user,p=password
```

3) 修复后，使用第步骤1) 的语句重新校验一次。

注意：使用pt-table-sync的风险比较大，对于生产环境，建议使用pt-table-checksum进行校验，如果有数据不一致的问题，则考虑重建从库；如果要使用pt-table-sync进行数据同步，则建议仔细阅读官方文档，了解它的限制和可能产生的影响，以避免影响生产环境或修复数据不成功。

(22) pt-query-advisor

这个工具可以利用一些规则来分析慢查询日志或general日志。以了解生产环境中SQL的撰写是否规范。如下将介绍一些示例。

分析慢查询日志，并给予建议。

```
pt-query-advisor /path/to/slow-query.log
```

输出的报告中包含了3种级别的提示，note级别、warn级别和critical级别，本书将摘录一些提示，具体如下。

note级别：应该使用“table as别名”的方式，而不是“table别名”的方式，因为使用as可读性更好。

别名不要和原来的表名一样。

INSERT INTO语句应该显式地指定列名，如INSERT INTO tbl(col1,col2)VALUES...。

日期/时间值需要用引号括起来。如WHERE col<='2012-02-12'

应该使用“<>”而不是“!=”。因为“!=”不标准。

warn级别：参数尽量不使用前导通配符，比如LIKE%name这样的方式是用不到索引的，MySQL的索引一般是前缀索引。

SELECT语句如果没有WHERE条件，则有可能会导致检索太多的记录。

多表查询GROUP BY或ORDER BY子句的列不在同一个表中会导致使用临时表（temporary table）和文件排序（filesort）。

不要使用SQL_CALC_FOUND_ROWS，这种方式检索数据效率很低，会需要检索所有的记录以确定记录的总数。

critical级别：不要混合ANSI标准的JOIN方式和MySQL中的JOIN语法。否则，容易导致用户混淆。

可以使用ON或USING语句来指定一个简单的连接。被连接的列在ON或USING子句中列出，而WHERE子句中可以列出附加的选择条件。

(23) pt-mext

该工具用于查看SHOW GLOBAL STATUS信息。可以增量（-r选项）显示。逐列进行显示，如下示例将每隔10秒执行一次SHOW GLOBAL STATUS，执行4次，并将结果合并到一起查看。

```
pt-mext -r -- mysqladmin ext -uroot -p111111 -i 10 -c 4
```

如图16-14所示的是运行pt-mext命令的一个输出。在此笔者对输出信息进行了过滤，仅显示特定的操作统计信息。

Event	Value	Value	Value
Com_delete	0	0	0
Com_update_all	0	0	0
Com_insert	17824475	163867	164%
Com_create_table	0	0	0
Com_select	1677586	0	0
Com_update	0	0	0
Com_update_multi	0	0	0
Com_change_table	252238	0	0

图16-14 pt-mext输出结果图

(24) pt-upgrade

该工具用于在多个MySQL实例上执行查询，并比较查询结果和耗时。这个工具在进行数据库版本升级的时候会很有用。

pt-upgrade的语法格式如下。

```
pt-upgrade [OPTION...] DSN [DSN...] [FILE]
```

如下将介绍一些示例。

查看慢查询在不同主机实例上的运行效果，命令如下。

```
pt-upgrade h=host1 h=host2 slow.log
```

比较host2和host1的结果文件，命令如下。

```
pt-upgrade h=host1 --save-results host1_results/ slow.log  
pt-upgrade host1_results/ h=host2
```

(25) pt-find

`pt-find`命令可以查找MySQL中的表，并执行一些操作，这个工具类似于我们操作系统下的`find`命令。默认的操作是打印数据库名和表名。

如下将举例一些示例。

打印一天以前创建的表，注意，仅对MyISAM有效，命令如下。

```
pt-find --ctime +1 --engine MyISAM
```

查找InnoDB引擎的表，并转化为MyISAM表，命令如下。

```
pt-find --engine InnoDB --exec "ALTER TABLE %D.%N ENGINE=MyISAM"
```

查找test库和junk库中的空表，并删除之，命令如下。

```
pt-find --empty junk test --exec-plus "DROP TABLE %s"
```

查找大于5GB的表，命令如下。

```
pt-find --tablesize +5G
```

对所有表都按照数据占用空间大小（数据+索引）进行排序，命令如下。

```
pt-find --printf "%T\t%D.%N\n" | sort -rn
```

把数据表的size信息存放到表中，命令如下。

```
pt-find --noquote --exec "INSERT INTO sysdata.tblsize(db, tbl, size) VALUES('%D', '%N', %T)"
```

16.3 调优方法论

16.3.1 性能调优的误区

许多人犯的错误可能仅仅只是因为自己熟悉一些工具、知道一些命令，就到处使用它，而不管是什么场合。特别是初级工程师，因为了解的命令和方法有限，他们更容易这样做。

还有些人喜欢去网上寻找答案，但是一些问题的解决方案，特别是涉及一些商业厂商的软硬件，网上并没有标准的答案，只有原厂工程师才掌握处理这些问题的方案，而且，如果是紧急处理性能问题，上网查找很可能费力不讨好。

有些人不清楚性能问题到底出现在哪，会尝试修改不同的参数配置，然后查看效果怎么样，这样的调优不但耗时，还可能给生产系统带来隐患甚至导致生产环境异常。

有些人往往把问题归咎于其他环节、其他团队。比如，怀疑是网络的问题，怀疑是数据库的问题，如果你怀疑是其他团队、其他环节出了问题，那么你应该说明下自己是如何分析、如何查找和为什么怀疑的，这样可以更有助于沟通，而不是浪费了其他部门的时间和精力。

所有以上出现的问题，都是因为没有一定的规则指引，没有按照一定的方法论来处理问题。方法论包括了一些定量分析和确认疑问的方法，标识了哪些是最重要的性能问题。它可以帮助性能分析者定位问题，告诉我们应该从哪里开始，应该通过哪些步骤来定位问题。对于初学者，可以按照方法论逐步来定位问题，对于专家和熟手，可以将方法论作为一个检查列表，确认我们没有忽视一些细节。

16.3.2 调优指引

当发生性能问题时，我们需要知道从哪里开始我们的分析，应该收集什么样的数据，我们应该如何分析这些数据。如下是一些调优指引。

1) 首先，我们需要定义调优的目标。

在调优之前，我们需要设定我们的目标，比如：资源使用率、延时、吞吐率。要依据你的业务、服务协议等级和服务标准量化你的性能调优所能达到的效果，例如：平均响应时间小于5ms，99%的响应时间小于10ms。确定调优的目标之后，我们再通过各种方式找到系统的瓶颈所在，然后优化它们。

2) 我们需要了解我们的数据流，了解我们的物理部署，这样我们才能有意识地针对整个系统进行调优。

3) 影响服务性能的主要因素从大到小大致是：架构和设计、应用程序、硬件、数据库和操作系统。高性能的服务是设计出来的，而不是调优出来的，并且，如果你的架构设计良好，那么不需要怎么调优，调优也会更加容易。

4) 大规模、高性能的服务往往不是一蹴而就的，需要在后期不断持续地迭代优化，甚至调整架构。一个优秀的架构师有一个很重要的素质，那就是：在合适的时间以合理的成本介入进行调整优化。

5) 性能调优有两个方向，一个是让工作做得更快，一个是让工作做得更少。针对数据库来说，就是要尽量减少对数据库的访问，使用最快的路径访问数据。

6) DBA需要从系统资源使用和应用访问的双重角度去考虑问题，应用程序开发人员往往更关注程序的负载情况，而系统管理员往往更关注资源的使用情况。作为一名DBA，需要能够同时从这两个角度去考虑问题。

从资源使用的角度进行分析，我们常用的一些指标有IOPS、吞吐率、利用率和饱和度。

从工作负载的角度进行分析，我们常用的指标有吞吐率和延时。对工作负载的分析要求我们熟悉工作负载的属性，具体负载做了什么？比如对于数据库访问，包括客户端主机/IP、数据库名、数据表名、查询字符串，通过熟悉这些情况，我们可以确定哪些工作是不需要做的，是可以消除的，哪些工作是可以加快的。我们还可能需要深入到应用的代码细节里去优化。

7) 性能调优是一种取舍，我们需要意识到性能调整是有权衡取舍的，性能好、成本低、快速交付这三个目标往往不可兼得。许多公司的项目选择了成本低和快速交付，而把性能问题留待以后去解决，但是一旦你的架构存在问题，你将很难进行调优，反而不得不付出昂贵的调整成本。在许多互联网公司，更普遍存在的一种情况是项目的时间是固定的，甚至要赶在对手发布之前进行发布，如果选择了成本低、性能好的决策，那么如期交付往往很难做到，这个时候如果仍然按照原定的进度计划交付软件，则往往意味着软件和服务的稳定性会下降。

在物理组件之间也有权衡取舍，比如有些应用CPU很空闲，内存很紧张，那么可以使用CPU压缩数据以节省内存的使用。

参数的调整也有权衡取舍，比如文件系统的块大小，较小的块，接近应用的访问I/O大小，更适合随机访问的应用负载。而较大的块，更适合一些大的操作，比如备份。比如，小的网络缓冲区可以减少每个连接的内存开销，使系统更好地扩展，而大的网络缓冲区则可以提升网络吞吐率。

8) 性能调优越靠近任务处理的环节就越有效，因为它们更了解数据，对于施加负荷的应用程序而言，针对应用程序自身的调优最有效。对于一个普通的基于数据库的网站服务而言，我们的软件栈是应用程序→数据库→系统调用→文件系统→存储。对存储设备进行优化可以提升I/O能力，但这是对最底层的优化，它的改善反映到更上层的应用的改善，往往是打了折扣的，比如存储设备的性能提升了1倍，也许应用的性能才提升20%。造成这种现象的根本原因是越远离应用层，越不熟悉数据是如何存取的，所以针对存储设备所做的优化，相对于应用程序自身的调优，可能对于应用的性能改善并没有太大的用处。

9) 参数的调整可能会马上见效，但很可能随着软硬件环境的变更，又变得不再适合，甚至还会对性能起反作用。所以，如果不是必须要调整，那么就不要去调整。更少的参数配置，使用默认的参数配置也意味着更少的维护成本。

网上有许多调优的建议和分享。这些分享往往都是基于特定的环境的，或者是基于特定的场合，而在之后，他们又有了调整，却往往没有继续分享，一些调优的建议，特别是互联网上的文章，以讹传讹，很可能存在某种隐患，为了保持系统的稳定，除非我们确定必须要更改，否则建议不要修改。更合适的策略是，关注业内的专家，看看他们的分享，看看哪些参数是在我们的系统上进行修改的。做一些记录，一旦有需要，我们再采纳也不迟。

10) 我们不仅要标识出哪些问题可能是需要调优的，也要衡量修复这些问题所带来的收益。如果修复一个问题，对于整体系统的贡献不大，那么投入时间和精力去调优可能不是一个好的主意。

16.3.3 调优步骤

调优的具体步骤如下。

1) 收集信息。

如果你是一个支持工程师，或者是一个独立的数据库咨询顾问，那么当有性能问题发生时，你需要和客户确认一些信息。

如下是要问的一些基本问题，通过这些问题，你往往可以很快就能确定原因，或者确定一个合适的解决方案，或者确定下一步应该怎么做。

·你为什么认为有性能问题，你是如何判断的？

·系统之前是好的吗？最近有做过什么软硬件的变更吗？业务流量、负载有变化吗？

·问题可以使用延时或运行时间来描述吗？

·影响到了其他的用户和程序吗？

·软硬件的环境是什么样的？版本/配置/参数是什么样的？

如果我们的线上有性能问题，那么我们需要收集负载信息和资源使用情况。

我们还可能需要检查配置，比如出现网络问题，比如有时网卡工作在100MB的模式下而不是1000MB的模式，比如RAID阵列坏了一块盘，比如操作系统、应用程序、固件的版本有变化，比如因为程序配置错误，访问了远程资源而不是本地资源。

2) 分析问题。

在问完这些问题之后，我们大致有了诊断的思路。这时我们可以推测有可能是哪些因素导致了性能问题。

3) 验证性能问题的原因。

之后我们再进行一些验证测试工作来验证我们的想法。比如MySQL的InnoDB buffer pool对性能的影响很大，我们假定是InnoDB buffer pool不够才导致的性能恶化，我们可以部署一个新的环境，增大InnoDB buffer pool，进行压力测试，这个时候，我们会发现性能有所改善，从数据库的状态信息我们也可以发现物理读减少了许多。

通过不断地猜测和验证，我们可以逐步缩小范围，定位到真正的导致性能恶化的因素上。

4) 进行调整。

5) 观察性能调整的效果。

16.3.4 调优的方法

(1) USE方法

有一种调优的方法叫USE方法。其基本思路是，对于每项资源，检查其错误、利用率和饱和度。

在资源利用率很高或趋向饱和时，如果出现瓶颈，性能降低，那么这种情况下，USE的分析方法将会最有效。

我们首先检查错误，对于错误，需要仔细就调查其产生原因。错误很可能会导致性能降低，如果错误是不可以被重现的，那么你可能不能及时发现错误。

然后我们检查资源利用率，检查饱和度，在检查资源利用率和饱和度的过程中，我们逐步缩小范围，最终定位到问题所在。

(2) 延迟分析法

找出响应最慢的环节，这个环节将被再次细分，再找出最影响时间的因素，并不断循环，直到最终解决问题。



小结 本章介绍了一些性能调优的基础概念、理论和常用的诊断工具。综合知识、工具、经验、意识，我们才可能成为一名性能调优专家。其中一些工具和命令的解释是摘录自网上的信息，笔者做了一些修正，但仍然可能有错漏之处，建议不熟悉命令的读者，好好阅读man帮助，这是笔者认为学习命令的最好的一种方法。Percona Toolkit包含了很多工具，平时我们经常使用的可能只有寥寥几个工具。笔者没有对所有工具做实际验证，读者应该牢记一点，如果工具可能修改生产环境中的数据，那么就一定要慎重使用，建议首先备份好数据。

第17章 应用程序调优

本章将主要讲述应用程序调优的一些方法和步骤，应用程序调优的领域很广，本章主要关注的是涉及数据库方面的调优。

在进行性能分析之前，我们先要熟悉应用的角色，它是什么版本的，做什么的，它是什么类型的应用，它是如何配置的，是否有相关的官方和社区支持，比如Bug库、邮件组。我们了解的信息越全面，就越有助于我们进行诊断和调优。

17.1 程序访问调优

如果能够满足以下几个方面的要求，那么程序的访问调优会更顺利。

17.1.1 好的架构和程序逻辑

最好是能够通过架构层面尽量避免性能问题的发生。如果你的物理部署无法满足预期的负载要求，或者应用软件的功能架构无法充分利用计算资源，那么，你无论怎么“调优”都无法带来理想的性能提升和扩展性。

生产实践中的性能问题更多地归根于系统的架构设计和应用程序的程序逻辑。运行较长时间之后，MySQL经过了高度优化，性能往往已经很好了，由于数据库的查询只占据了总体响应时间的很小一部分，优化数据库对于整体用户体验的改善并无太大用处，而更改业务逻辑往往是最直接、最有效的。

一个应用的功能模块图对于我们的调优将会很有帮助，通过查看应用的模块图和物理部署图，从数据流、数据交互的角度去理解和分析问题，往往能够发现架构中存在的问题。

1.缓存

互联网应用往往有多级缓存，比如用户访问网站，可能要经过浏览器缓存、应用程序缓存、Web服务器缓存、Memcached之类的缓存产品、数据库缓存等。缓存可以加速我们从更慢的存储设备中获取数据，可以改善用户体验，提高系统吞吐。对于多级缓存来说，越是靠近用户，越是靠近应用，就越有效，也就是说，缓存要靠近数据的使用者，靠近工作被完成的环节，显然，在浏览器的缓存中读取图片会比到Web服务器中读取图片高效得多。到Squid缓存服务器中获取数据比实际去后端的Web服务器中获取数据要高效得多。

缓存的存在应该限定为保护不容易水平扩展的资源，如数据库的大量读取，或者提升用户体验，或者在靠近用户的城市部署图片缓存节点。如果资源易被水平扩展，那么添加缓存层可能不是一个好主意。

缓存的设计目标是，用更快的存储介质存储更慢的存储介质上的数据，以加速响应。比如把磁盘的数据存放在内存中。我们这里仅仅关注被放置在数据库前端的缓存产品，比如Redis、Memcached等产品。我们所使用的缓存产品主要是为了扩充读能力，对于写入，则并没有多少帮助。

MySQL有InnoDB缓冲区，但是其更多地只是属于数据库的一个组件，它的功能是把热点数据缓存在内存中，如果要访问数据，则存在解析开销，也可能需要从磁盘中去获取数据，因为缓存中的内容会因为不常被使用而被剔除出缓存。由于InnoDB缓冲的最小单元是页，而不是基于记录，因此要缓存一条记录，可能要同时缓存许多不相干的记录，这样就会导致内存缓存的利用率比较差。

而对于Memcached之类的记录级的缓存来说，因为应用程序有目的性地缓存了自己所需要的数据，所示其效率一般来说是要高于基于页的InnoDB缓存。而且分布式的Memcached集群可以配置成一个超大的缓存，相对于单个实例内的InnoDB缓存，其扩展性也无疑好得多了。

现实中，由于Memcached的引入可能会导致开发的复杂度上升，所以在项目初期，往往并没有引入Memcached等缓存产品，一般的单机MySQL实例也可以扛得住所有流量，当项目规模扩大之后，读请求的处理可能会成为瓶颈，这个时候可以选择增加从库，或者使用Memcached等缓存产品来突破读的瓶颈。

缓存的指标有缓存命中率、缓存失效速率等。

缓存命中率指命中次数与总的访问次数的比值。

缓存失效速率指每秒的缓存未命中次数。

缓存命中率和性能的关系如图17-1所示。

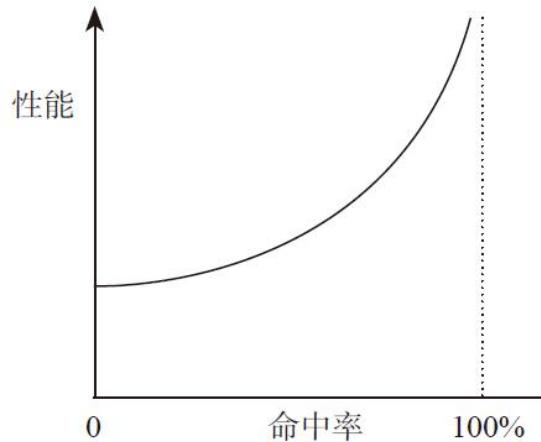


图17-1 缓存命中率和性能的关系

由图17-1可以得知缓存命中率和性能的关系，98%~99%命中率的性能提升远远大于10%~11%命中率的性能提升。这种非线性的图形，主要是因为缓存命中(cache hit)和缓存未命中(cache miss)所访问的存储的速度差异比较大而形成的，比如内存和磁盘。由于这样一个非线性的图形，我们在模拟缓存故障的情况下要注意缓存的命中率，如果缓存的命中率不高，那么即使缓存挂了，对后端数据库的冲击也不会很大，但是如果缓存命中率很高，那么如果缓存挂了，可能就会对后端的数据库造成很大冲击。

缓存失效效率(cache miss rate)，即每秒的非命中次数，由于没有命中缓存，此时需要从更慢的存储上去获取数据。这个指标比较直观，有利于我们分析当应用没有命中缓存时，我们的存储系统能否承受冲击。如果缓存命中率显得很高，但是每秒缓存未命中的次数也很高，那么性能一样会很差。

当我们使用缓存产品，我们要清楚以下几点。

·缓存的内容。

·缓存的数据量。

·设定合适的过期策略。

·设置合适的缓存粒度，建议对单个记录、单个元素进行缓存而不是对一个大集合进行缓存，如果要将整个集合对象数据进行缓存的话，获得其中某个具体元素

的性能将会受到严重的影响。

为了提高吞吐率，减少网络回返，建议一次获取多条记录，如Memcached的mget方法。

稳定的性能和快速的性能往往一样重要。我们在设计缓存的时候，要考虑到未命中的时候，生成结果的代价，如果会导致偶尔访问的用户响应慢，那么请不要牺牲这部分很小比例的用户。

一般来说，Memcached属于被动缓存，我们也可以采取主动缓存的策略，预生成一些访问最多的，生成代价最昂贵的内容到Memcached中。

由于序列化和反序列化需要一定的资源开销，当处于高并发高负载的情况下，可能要消耗大量的CPU资源，对于一些序列化的操作一定要慎重，尤其是在处理复杂数据类型时，可能序列化的开销会成为整个系统的瓶颈。

注意事项具体如下。

如果缓存挂了怎么办？或者因为调整缓存，清空缓存，对数据库产生了冲击怎么办？

假设你的业务逻辑是，如果缓存挂了，就去后端的数据库中获取数据。那么很可能短时间内的流量远远超过了数据库的处理能力，导致数据库不能提供服务。所以就需要考虑对于后端数据库的保护，你可能需要对你的应用服务降级使用，即关闭掉不重要的模块，以确保核心功能，或者对数据库进行限流。更友好的方式是，在应用程序里通过锁的机制控制对数据库的并发访问。

限流指的是应用对请求有排队机制，如果队列超过了一定的长度就会触发限流，就会随机抛弃掉一些请求。

2. 非结构化数据的存储

不要在数据库里存储非结构化的数据，如视频、音乐、图片等，可以考虑把这些文件存储在分布式文件系统上，数据库中存储地址即可。

3. 隔离大任务

批量事务一般应该和实时事务相分离，因为MySQL不太擅长同时处理这两类任务。

有时我们会运行一些定时任务，这些任务很耗费资源，我们需要注意调度，减少对生产环境的影响，比如更新Sphinx索引，需要定期去数据库中扫描大量记录，可能短时间内会造成数据库负荷过高的问题。

对数据库进行的一些大操作，我们可以通过小批量操作的方式减少操作对生产系统的影响，比如下面的这个删除大量数据的例子。

```
delete * from table_name where ctime < '2014-12-12'.
```

执行SQL会删除千万级别的记录，由于删除的记录过多，可能会导致执行计划变为全表扫描，从而导致不能写入数据，影响生产环境。

优化方案具体步骤如下。

- 1) 程序每次获取1万条符合条件的记录，SQL为“SELECT id FROM table_name WHERE ctime<'2014-12-12' limit 0,10000”。
- 2) 根据主键id删除记录。每批100条，然后线程休眠100毫秒，直到删除完步骤1) 中查询到的所有id。
- 3) 重复步骤1) 和步骤2)，直到执行“SELECT id FROM table_name WHERE ctime<'2014-12-12' limit 0,10000”返回空结果集为止。

休眠100ms是为了限制删除的速率，减少操作对生产环境的影响。

有时这些大操作无法完全消除，但又占据了大量的资源，这时我们就可以通过系统资源控制的方式对应用进行限制。

4. 应用程序相关数据库优先注意事项

以下将列举一些应用程序相关的数据库优化注意事项。

· 检查应用程序是否需要获取那么多的数据，是否必须扫描大量的记录，是否做了多余的操作。

· 评估某些操作是应该放在数据库中实现还是在应用中实现？不要在数据库中进行复杂的运算操作，比如应用就更适合做正则的匹配。

· 应用程序中是否有复杂的查询？有时将复杂的查询分解为多个小查询，效率会更高，可以得到更高的吞吐率。

· 有时框架中会使用许多无效的操作，比如检测数据库连接是否可用。应该设置相关参数减少这类查询。

17.1.2 好的监控系统和可视化工具

解决性能问题如果是临时的、紧急的，特别是在生产繁忙的时候，你往往会觉得束手无策，因为在压力之下，可能会遗漏一些问题，或者没有得到最好的解决方案。系统应该能够及时预警，在性能问题爆发之前就能够发现问题。所以，你应该有一个好的监控系统。能够监控到数据流各个环节的延时响应。

应用服务需要考虑维护性，可以进行性能统计，了解哪些操作占据了最多资源，通过可视化工具检查性能，可以让我们能够观察应用正在做什么事情，如何优化应用以减少不需要的工作。

17.1.3 良好的灰度发布和降级功能

系统越来越复杂，各个组件之间互相调用，可能某个模块会导致整个系统不能提供服务。我们有必要区分核心的业务和非核心的业务，理清各种模块之间的关系，如果能够有针对性地停止或上线功能/模块/应用，屏蔽掉性能有问题的模块，保证核心基础功能的正常运行，那么我们的整体系统将会更加稳健，可以更快地从故障中恢复。所以，你最好有良好的灰度发布和降级功能，这点对大系统尤为重要。

17.1.4 合理地拆分代码

进行架构调整常用的一个技术是垂直拆分，这里不会严格区分拆分代码、垂直拆分和拆库。

对于复杂的业务，如果出现了因代码而导致的性能问题，那么可以先从物理上隔离服务再考虑代码优化，如部署更多独立的Web服务器或数据库副本以提供服务。将数据库数据拆分到独立的实例（垂直拆分）或增加读库。

垂直拆分需要慎重，因为跨表的连接会变得很难。所以还是要看业务逻辑，如果表之间的关联很多很紧密，那么可能拆分数据库就不是一个好的方案。

拆分业务之后，需要把用户引导到新的程序或服务上。有诸多方式可以使用，如更改域名、应用前端分发、302跳转，或者有些客户端有更多的功能，可以接受云端的指令，修改访问不同功能的域名。

拆分代码需要慎重，因为分离的多套代码，可能会需要更多的接口调用，需要更多的交互，增加了复杂性，应该视后续发展而定。如果代码之间的划分并不是很清晰，一个需求来了，要互相提供接口，更改多套代码，那样往往就增加了复杂性，会影响开发效率。所以具体拆分代码还是要看看后续的项目发展，想清楚应该如何拆分。

许多时候，是业务优先的，因此要先保证业务，增加更多的资源用于突发的负荷。对于数据库中数据的拆分，可能也不是一步到位的，需要兼顾业务的正常运行，因为对数据的拆分，往往需要先进行代码/模块的拆分。可以考虑的一种措施是，克隆一份数据的副本，先用拆分出来的模块读写副本，等代码稳定后，再删除不需要的数据。

模块降级是需要考虑的，模块的各自监控也是需要的。这样在发生故障的时候可以及时关闭一些出问题的模块。保证核心的业务服务。升级的时候，也可以进行灰度升级，逐步打开一些功能开关。

17.2 应用服务器调优

数据库的前端一般是应用服务器，如果应用服务器得到了优化，也可以减少数据库的压力，使得整个系统的性能更好、可扩展性更好。应用服务器的调优是一个很大的主题，本章节只是介绍一些基本的指引。

在调优之前，必须要清楚应用服务器的响应过程，细分为各个阶段，哪个阶段耗费的时间最多，就首先从那个部分着手进行优化。

每个应用服务器都有许多参数配置，我们在进行调优的时候，应尽量逐个对参数加以调整优化，这样可以更好地衡量调整的效果，如果参数优化没有效果，那就恢复原来的配置。

修改参数的数值，可以逐步调整参数的值，如果一次性调整得过多，那么可能你得不到一个最优的配置，或者推翻你自己的判断，逐步调整参数，你有更大的可能性最终得到一个性能良好的系统。

在应用程序所在的软硬件环境发生变动的时候，你应该重新审视以前所做的优化配置，看它们是否依然能够工作。

你应该清楚应用服务器的一些参数会如何影响到数据库的负载，清楚哪些参数会导致数据库的连接数增加，由于一些连接池或框架的行为，大量的连接会导致过多地检测数据库的命令，因此也需要加以留意。



小结 应用程序调优往往是最有效的方式，通过减少不必要的工作或让工作执行得更快，我们可以大幅度地提升性能。应用程序访问调优更多的是软件架构的范畴。如果读者有兴趣，建议多阅读一些软件架构方面的著作。

第18章 MySQL Server调优

本章将为读者介绍针对MySQL Server的优化，这也是DBA最熟悉的领域之一。首先我们介绍MySQL的主要参数，然后，讲述常见硬件资源的优化。我们假设读

者已经具备了足够的基础知识，所以，本章将更多的针对一些特定的主题进行叙述。

18.1 概述

衡量数据库性能的指标，一般衡量数据库的性能有两个指标：响应时间和吞吐率。响应时间又包括等待时间和执行时间。我们进行优化的主要目的是降低响应时间，提高吞吐率。

下面我们来看下MySQL是如何执行优化和查询的？

大致的步骤如下所示。

- 1) 客户端发送SQL语句给服务器。
- 2) 如果启用了Query Cache，那么MySQL将检查Query Cache，如果命中，就返回Query Cache里的结果集，否则，执行下一个步骤。
- 3) MySQL Parser解析SQL，MySQL优化器生成执行计划。
- 4) MySQL查询处理引擎执行此执行计划。

MySQL性能优化显然是要对以上的部分环节或所有环节进行优化，尽量降低各个环节的时间，以提高吞吐率。对于性能的优化，正确的策略是衡量各个环节的开销，优化开销大的环节，而不是使用网上的一些所谓的参数调优和脚本调优，因为他们并不是针对你的特定情况而进行的调优，只是一些泛泛的建议，往往帮助不大。

对于客户端来说，发送SQL语句的开销一般很小，如果是响应缓慢的网络，网络延时较高，那么可以考虑使用长连接或连接池等手段进行加速，或者一次发送多条语句，或者使用存储过程等手段减少网络包的往返次数。

本书主要聚焦于后面的3个步骤，我们需要关注Query Cache是如何加速响应；如何进行查询优化，生成良好的执行计划；实际查询处理过程中对于I/O、CPU、内存等资源的使用是怎样的。我们需要尽量确保高效地利用资源，突破资源的限制。

之前的开发篇和运维篇章已经讲述了许多基础知识，这里不再赘述，本章我将主要从系统资源和MySQL参数设置的角度，讲述一些我们需要关注的优化点。

18.2 MySQL的主要参数

本节将列举一些主要的参数，下面将详细介绍各个参数。

1.innodb_buffer_pool_size

一个简单的策略是如果数据库很大，远远超过内存，那么应设置尽可能大的缓冲池（buffer pool）。如果数据库较小，一般来说，缓冲池的大小设置为稍大于数据库的10%就可以了，大于10%是因为MySQL不只是缓存数据页，还有一些额外的开销。如果我们使用ps命令检查MySQL实际占用的内存，就会发现实际分配的内存会比我们设定的内存要大一些。

更合适的策略是衡量你的热点数据的大小，如果设置的缓冲区能容纳绝大部分的热点数据，没有产生过多的物理读，那么这个设置就是比较合理的设置。需要注意的是，不要设置得过大，因为我们需要给操作系统和其他程序预留内存，增加的负荷也会导致更多的内存使用，我们还可能需要预留内存以用于文件缓存，比如InnoDB的日志文件、MySQL的二进制日志文件等。

要注意32位系统的内存限制，超过了内存限制，可能会导致实例崩溃，系统宕机。

一般计算MySQL需要多少内存比较难，也难以预测，比较可靠的方式是查看目前的生产环境的内存消耗。

MySQL所消耗的内存还和连接数有关，每个连接所消耗的内存总量将依赖于负载，如果查询很复杂，那么它会消耗更多的内存，如果只是简单的查询，平时基本上是sleep的状态，那么它实际占用的内存就很少，所以，对于连接数多的业务，你应该实际观察下，操作系统下实际占用的内存和你设置的InnoDB缓冲池之间的区别，以衡量是否要调整设置，设置一个更安全一点的参数值，以免连接数暴涨消耗了过多的内存。

如下的图18-1描述了存储的访问层次。

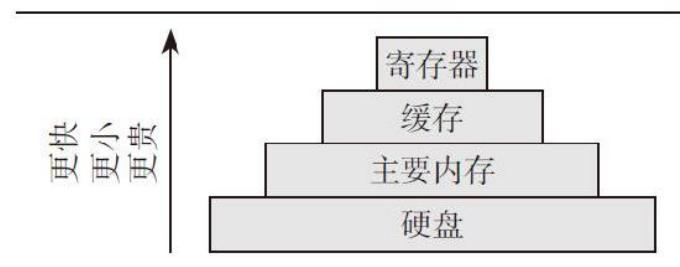


图18-1 存储的访问层次

对于图18-1中所示的架构，上一层应尽可能缓存下一层的“热点”数据，也就是说，我们需要平衡好内存和磁盘的成本，尽量避免磁盘访问，对于MySQL来说，内存的主要部分就是InnoDB缓冲池，优化好了InnoDB缓冲池的访问，就成功了一大半，一般采取的策略是利用空间和时间的局部性，频繁地访问，应该尽可能去访问内存而不是磁盘，由于数据的访问可能会很复杂，也许1%的缓存未命中（cache miss）需要几十乃至上百GB的内存来避免，而不仅仅是cache miss对应的数据大小，所以我们还要充分理解数据库的物理设计和逻辑设计，设计出合理的方案，减少cache miss导致的物理读。

需要清楚的一个道理是，一般而言，数据库系统的专用存储系统比操作系统的存储系统高效得多。InnoDB缓冲池就是如此，而MyISAM仍然是需要OS来缓存数据的，加速访问，所以往往表现得不那么好。

2.innodb_flush_method

这个选项只在Unix系统上有效。如果这个选项被设置为fdatasync（默认值），那么InnoDB将使用fsync()来刷新数据和日志文件。如果被设置为O_DSYNC，那么InnoDB将使用O_DSYNC来打开并刷新日志文件，但使用fsync()来刷新数据文件。如果指定了O_DIRECT（在一些GNU/Linux版本上可用），那么InnoDB将使用O_DIRECT来打开数据文件，并使用fsync()来刷新数据和日志文件。笔者的建议是设置innodb_flush_method=O_DIRECT，因为这样设置可以避免双重缓冲，让数据库跳过文件系统缓冲直接和设备进行交互，需要留意的是，如果你的磁盘做了RAID，那么你必须使用带电池的RAID卡。

3.innodb_log_file_size

日志组里每个日志文件的大小。早期的版本中，在32位的计算机上，日志文件的合并大小必须小于4GB。默认是5MB。不太好去确定innodb_log_file_size这个参数的大小，早期MySQL版本的配置文件里建议的InnoDB缓冲大小的25%是没有什么道理的，首先InnoDB缓冲不一定就设置对了，而且InnoDB事务日志一般和你的日志写入量、写入频率有关系，和你的缓冲池大小不存在必然的关系。

MySQL的灾难恢复分为redo和undo两个过程，redo即找到日志文件里记录的已经更改了但是并未写入数据文件的记录，然后应用这些日志。undo即回滚那些没有提交的操作，undo的时候，数据库已经可以访问了，但是undo的那部分数据还不能更改。

MySQL在切换事务日志的时候，可能会进行一次“check point”的操作，将部分数据写入到磁盘，也就是说，要确保我们的缓存里比日志还旧的数据写入了磁盘。其他时刻也可能发生“check point”。如果数据库宕机，MySQL重新启动，那么，它会去事务日志里找到“check point”的标记信息。在这个“check point”标记之前的数据，我们可以认为都已经写入到了磁盘。那么，我们就只需要执行这个“check point”之后的所有操作，应用这些日志到数据库即可。

事务日志不能过小，否则可能会导致性能问题。事务日志是循环写的，先写第一个日志文件，再写第二个日志文件，然后又会去写第一个日志文件，而在覆盖旧的日志之前，需要确保我们的缓存里比日志还旧的数据已经写入磁盘。如果事务日志过小，那么磁盘的I/O操作就可能会变得很频繁，因为MySQL必须写入一些脏数据到数据文件中。一次性刷新大量数据，可能会导致性能下降。

事务日志也不能太大了，因为这个时刻，我们的“check point”会不怎么频繁，那么MySQL的灾难恢复可能需要更长的时间，因为它需要应用更多的日志。生产环境的恢复速度将取决于应用日志的进度，一个1GB的事务日志，如果要全部应用，有可能需要应用半个小时以上来执行恢复。

我们可以配置事务日志可以写入半个小时到1个小时的日志。这样对于大部分应用已经足够了。太小了，会频繁切换日志；太大了，可能会导致故障恢复的时间过长。我的经验值是256~512MB。

日志的写入量可以查看变量innodb_os_log_written。我们可以每隔一分钟查看一次，统计每分钟写入的日志量。下面的命令将会每分钟检查一次日志的写入量。

```
mysqladmin extended -uroot -pxxxxxxxx -r -i 60 -c 3 |grep "innodb_os_log_written"
```

如果由上面的命令得知每分钟写入量为10MB，那么我们配置可以连续写45分钟的日志。默认有2个日志，那么每个日志的大小=10*45/2=225，大约等于256MB，那么我们可以配置innodb_log_file=256MB。

如果得出的结论是InnoDB日志需要几个GB那么大，那么很可能是不正常的，你要深究为什么会写入这么大的日志，为什么有这么多/大的变更，你可能需要在应用层就规避这种情况。

4.innodb_flush_log_at_trx_commit

当innodb_flush_log_at_trx_commit被设置为0时，日志缓冲将每秒一次被写到日志文件中，并且对日志文件进行磁盘操作的刷新，但是在事务提交时不进行任何操作。当这个值为1（默认值）时，在每个事务进行提交时，日志缓冲将被写到日志文件，且把对日志文件的变更刷新到磁盘中。当设置为2时，在每个事务进行提交时，日志缓冲将被写到文件，但不会对日志文件进行到磁盘操作的刷新，对日志文件的刷新每秒发生一次。我们可以看到，设置为2比设置为0更安全。

我们的生产环境一般推荐设置为innodb_flush_log_at_trx_commit=2，因为它可以兼顾效率和一定的安全性，理想情况下，最多可能丢失1秒的事务。如果设置为1，则对于性能的影响会很大，因为每次提交事务，都会伴随着磁盘I/O的操作，需要把数据刷新到磁盘，I/O可能会成为瓶颈，对于高安全性的数据，在能够满足I/O性能的前提下，可以考虑将其设置为1。

5.sync_binlog

这个参数是设置，每当写了sync_binlog次二进制日志后，把日志实际刷新到磁盘中，默认值是0，不与硬盘同步。

绝大部分公司的生产环境普遍使用的是auto commit模式（自动事务提交），每次写一个语句，就会写一次二进制日志，如果不是自动事务提交，那么每个事务将写入一次二进制日志。

如果设置为1，那么最多丢失1条记录（事务），这是最安全的选择。

生产环境的推荐设置是8~20，这样可以兼顾效率和安全，如果设置为1，你可能会碰到I/O瓶颈，你需要选用更好的SSD设备，或者使用带电池的RAID卡来缓解I/O瓶颈，优化文件系统也是一个选项，ext4和xfs就比ext3的表现要好得多。

生产实践证明，sync_binlog会对事务吞吐率有比较大的影响。事务日志、数据文件、二进制日志文件是需要同步的。数据库可以看作一个巨大的同步机，各个组件之间存在复杂的通信和同步等待，如果sync_binlog操作较慢，那么可能对整个系统的吞吐率造成严重的影响。

有一个相关的参数innodb_support_xa我们需要了解。innodb_support_xa设置为1时，这个变量允许InnoDB支持XA事务，即Distributed(XA)Transactions分布式事务，MySQL部分支持XA事务。一般互联网公司的业务不需要分布式事务，而且应该尽量避免，那么，是不是要禁用innodb_support_xa呢？并不是像一些人理解的那样，没有分布式事务，就不要这个特性，MySQL内部会使用XA来协调存储引擎和二进制日志，以确保灾难恢复功能工作正常，所以应该开启它。MySQL存储引擎各自独立，互不知晓其他引擎的状态，因此，跨引擎的事务可以看作一个分布式的事务，且需要一个第三方来协调它，这个第三方就是MySQL Server。我们可以把“二进制日志”看作一个“存储引擎”。MySQL Server需要协调二进制日志的写入和InnoDB事务的写入。

生产环境为了安全和复制，必须开启binlog和innodb_support_xa。如果将sync_binlog参数设置为1，那么存储引擎和二进制日志需要完全同步。如果二进制日志所在的磁盘存在性能问题，那么也会影响到我们的事务提交。生产繁忙的系统，有时经常会看到许多commit慢查询，就是因为二进制日志的写入瓶颈导致了InnoDB事务的提交缓慢。

6.innodb_thread_concurrency

InnoDB试着在InnoDB内部保持操作系统线程的数量少于或等于这个参数给出的限制。官方建议是将其设置为处理器数目加磁盘数之和，对于高并发事务，也许你应该把这个值设置得更大一些。对于一些资源等待异常的情况，后来的事务会被已经在等待队列中的事务卡住，你可以通过临时增大这个值，让更多的事务并发执行。

7.innodb_max_dirty_pages_pct

这是一个范围从0到100的整数。默认是90。InnoDB中的主线程试着从缓冲池写数据，使得脏页（没有被写的页面）的百分比不超过这个值。可以运行如下命令进行修改。

```
SET GLOBAL innodb_max_dirty_pages_pct = value;
```

生产环境建议将其设置为更小的值：50~75。

8.read_buffer_size

每个线程连续扫描时为扫描的每个表分配的缓冲区的大小（字节）。如果进行多次连续扫描，可能还需要增加该值，默认值为131072。只有当查询需要的时候，才分配read_buffer_size指定的全部内存。

9.read_rnd_buffer_size

排序后，按照排序后的顺序读取行时，则通过该缓冲区读取行，以避免搜索硬盘。将该变量设置为较大的值可以改进ORDER BY的性能。但是，这是为每个客户端分配的缓冲区，因此你不应该将全局变量设置为较大的值。相反，只为需要运行大查询的客户端更改会话变量即可。

10.sort_buffer_size

每个排序线程分配的缓冲区的大小。增加该值可以加快ORDER BY或GROUP BY操作。查询需要排序的时候（如fisort）才分配sort_buffer_size指定的内存，不要设置得过大，否则小的排序也需要大的内存。

在我们确定需要进行大的排序操作的时候，我们可以在会话级别定义大的排序sort_buffer_size。

11.myisam_sort_buffer_size

当运行REPAIR TABLE命令修复表、运行CREATE INDEX命令创建索引或运行ALTER TABLE命令修改表结构时，排序过程中需分配的缓冲区，可以在会话级别进

行设置。

12.query_cache_size

为缓存查询结果分配的内存的数量。默认值是0，即禁用查询缓存。请注意即使将query_cache_type设置为0也将分配query_cache_size设置的内存。重新定义大小会清除原来缓存的结果集。对于写操作很频繁的应用，可以禁用它，以消除失效Query Cache的开销，这样可能获得性能上的提升。禁用的办法是设置query_cache_size=0。建议生产环境中将其设置为64MB~256MB，不要太大，对于绝大部分业务，256MB就已经足够了。

如果要启用Query Cache，那么需要同时设置query_cache_type=1。

13.join_buffer_size

用于完全连接（当不使用索引的时候使用连接操作）的缓冲区的大小。

给不能利用索引的连接使用的。多表连接需要多个join buffer。所以一个查询可能要用到多个join_buffer_size。

14.max_connections

允许的并行客户端连接数目。

15.max_connect_errors

如果中断与主机的连接超过了该数目，则该主机会阻塞后面的连接。你可以用FLUSH HOSTS语句解锁锁定的主机。默认值太小了，可以设置在5000以上。

16.skip-name-resolve

不要解析客户端连接的主机名。只使用IP。如果你要使用该项，那么授权表中的所有Host列值必须为IP号或localhost。生产环境中必须设置这个参数，否则反向解析缓慢时，会导致MySQL连接缓慢，出现严重的性能问题。

18.3 MySQL内存优化

18.3.1 如何避免使用swap

这里我们仅仅讨论Linux系统下的swap（交换）。其他系统，如Solaris，会有一些区别。

简单地说，swap指的是将最近不常使用的内存移动到下一级存储里（硬盘），在需要的时候，再载入到主内存中。

swap空间一般是指我们磁盘上的预先配置的一个分区，也可以是文件，用于将内存中的数据交换到磁盘上。物理内存和swap空间之和就是我们可用的虚拟内存的大小。当我们的内存不够了或应用程序消耗了太多的内存，操作系统会把不需要立即使用的数据传输到磁盘，以释放内存空间，如果以后需要了，再从磁盘上复制回内存，这样一个过程也称为交换（swap out/swap in）。通过这样一个交换的动作，增加了实际可用的内存，可以提高系统的吞吐能力，但是数据的交换如果太频繁，就会大大增加磁盘的延时时间，可能会导致严重的性能问题。一般来说，数据库负载，需要尽量避免使用到swap。我们可以使用free、vmstat、sar等命令查看swap使用的统计信息。

通过free命令，如果我们看到了一小部分swap空间被使用，那么这一般是正常的，不需要额外关注，我们需要关注的是是否有正在进行的swap in/swap out操作。

一些人建议将swap分区设置为物理内存的大小，对于Linux系统来说，这个建议有一定的意义，为了不浪费过多的硬盘空间，建议使用如下的策略。

·如果MEM<2GB，那么SWAP=MEM×2，否则SWAP=MEM+2GB。

·对于内存非常大的系统，如32GB、64GB，我们可以使用0.5×内存大小。

MySQL避免使用swap的一些方法如下。

(1) 设置memlock

可在参数文件中设置memlock，将MySQL InnoDB buffer锁定到内存，永不使用swap，但这是有风险的。如果内存不够大，MySQL会被操作系统的OOM机制杀掉。如果因为物理内存故障导致内存总量变少，那么它可能还会导致系统无法顺利启动，因为MySQL会不断申请内存。

(2) 使用大内存页

可以设置MySQL使用Linux系统的大内存页（操作系统和MySQL都需要设置）。Linux系统的大内存页是不会被交换出去的。

(3) 设置vm.swappiness

可以设置`vm.swappiness=0`, 以减少使用swap的可能。

`swappiness`参数, 它可以在运行时进行调优。这个参数决定了, 将应用程序移动到交换空间而不是移动到正在减少的高速缓存和缓冲区中的可能性, 降低`swappiness`可以提高交互式应用程序的响应能力, 但是会降低系统的总体吞吐量。

(4) 禁用NUMA或调整NUMA

在生产环境中你可能会碰到在没有内存压力的情况下, 也发生swap in/swap out的情况, 导致不定时出现的性能问题。尤其是在使用了大的buffer pool size的情况下, 这一般是因为使用了NUMA技术, 需要考虑禁用NUMA或更改程序分配内存的方式, `numactl`命令可以实现这个目的, 使用方式为: `numactl--interleave all command`, 例如, `/usr/bin/numactl--interleave=all mysqld`, 详情请参考18.3.2节NUMA。



注意 不要去禁用swap, 并不是所有内核在swap分区被禁用的情况下都能工作得很好, 这可能会导致服务异常, 某个服务在禁用swap的时候能够工作得很好, 并不代表所有程序都能很好地工作。而且内存不够的概率更高了, 当使用了过多的内存时, 程序更容易被操作系统的OOM机制杀掉。我们需要意识到, swap分区为我们处理问题留了一个缓冲, 给我们争取到了处理问题的时间, 所以我们不要把swap分区设置得过小, 相对于你所获得的收益, “浪费”一些磁盘空间是值得的。

18.3.2 NUMA

从系统架构来说, 目前的主流企业服务器可以分为3类: SMP (Symmetric Multi Processing, 对称多处理架构)、NUMA (Non-Uniform Memory Access, 非一致存储访问架构) 和MPP (Massive Parallel Processing, 海量并行处理架构)。下面我们来看下SMP和NUMA架构。

1.SMP

如图18-2所示的是一个SMP系统。

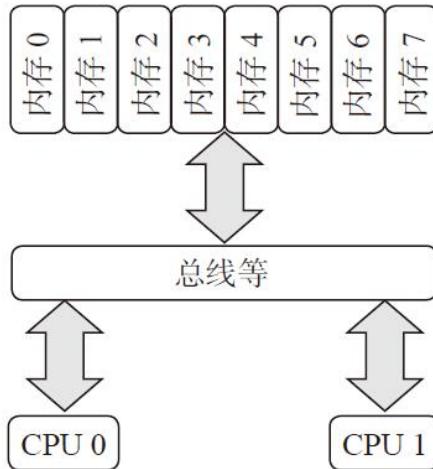


图18-2 SMP系统

在这样的系统中, 所有的CPU共享全部资源, 如总线、内存和I/O系统等, 多CPU之间没有区别, 均可平等地访问内存和外部资源。因为CPU共享相同的物理内存, 每个CPU访问内存中的任何地址所需要的时间也是相同的, 因此SMP也被称为一致存储器访问结构 (Uniform Memory Access, UMA), 尤其是在和NUMA架构对比的时候。对于SMP服务器而言, 每一个共享的环节都可能是瓶颈所在。由于所有处理器都共享系统总线, 所以当处理器的数目增多时, 系统总线的竞争冲突也会加大, 系统总线成为了性能瓶颈, 所以其扩展性有限, 这种架构已经被逐步淘汰, 但在CPU内部还有应用, 单个CPU的所有核共享访问该CPU的本地内存。

2.NUMA

如图18-3所示的是NUMA系统。

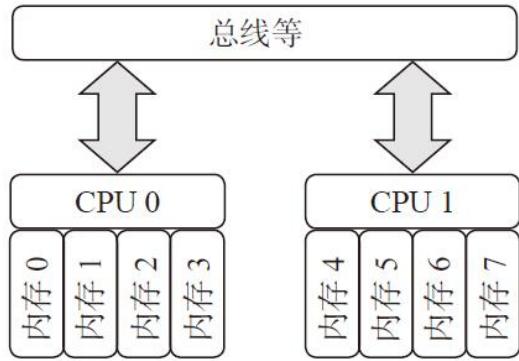


图18-3 NUMA系统

在这种架构中，每颗CPU有自己独立的本地内存，CPU节点之间通过互联模块进行连接，访问本地内存的开销很小，延时比访问远端内存（系统内其他节点的内存）小得多。这也是非一致存储访问NUMA的由来。

综上所述可以得知，NUMA对内存访问密集型的业务更有好处，NUMA系统提升了内存访问的局部性，从而提高了性能。

关于CPU信息，我们可以查看/proc/cpuinfo。对于NUMA的访问统计，我们可以使用numastat命令进行检查，也可以查看/sys/devices/system/node/node*/numastat文件。如图18-4所示，NUMA使用了default策略，这将导致内存分配的不均衡，numastat命令的输出如下。

	node0	node1
numa_hit	14589985197	11205826989
numa_miss	2485743	357480322
numa_foreign	357480322	2485743
interleave_hit	5134768	6247209
local_node	14576807167	11175787269
other_node	15663773	387520042

图18-4 NUMA使用default策略，numastat命令的输出

各项输出的含义如下。

·numa_hit：在此节点分配内存而且成功的次数。

·numa_miss：由于内存不够，在此节点分配内存失败转而在其他节点分配内存的次数。

·numa_foreign：预期在另一个节点分配内存，但最终在此节点分配的次数。

·interleave_hit：交错分布策略分配内存成功的次数。

·local_node：一个运行在某个节点的进程，在同一个节点分配内存的次数。

·other_node：运行在其他节点的进程，在此节点分配内存的次数。

在Linux上NUMA API支持4种内存分配策略，具体如下。

·缺省（default）：总是在本地节点分配（分配在当前线程运行的节点上）。

·绑定（bind）：分配到指定节点上。

·交织（interleave）：在所有节点或指定的节点上交织分配。

·优先（preferred）：在指定节点上分配，失败后在其他节点上分配。

绑定和优先的区别是，在指定节点上分配失败时（如无足够内存），绑定策略会报告分配失败，而优先策略会尝试在其他节点上进行分配。强制使用绑定有可能会导致前期的内存短缺，并引起大量换页。

我们可以检查程序具体的内存分配信息，假设pid是mysqld的进程ID，通过查看/proc/pid/numa_maps这个文件，我们可以看到所有mysqld所做的分配操作。各字段的显示如下。

```
2aaaaad3e000 default anon=13240527 dirty=13223315
swapcache=3440324 active=13202235 N0=7865429 N1=5375098
```

各字段及其解析如下。

·**2aaaaad3e000**: 内存区域的虚拟地址。实际上可以把这个当作该片内存的唯一ID。

·**default**: 这块内存所用的NUMA策略。

·**anon=number**: 映射的匿名页面的数量。

·**dirty=number**: 由于被修改而被认为是脏页的数量。

·**swpcache=number**: 被交换出去，但是由于被交换出去，所以没有被修改的页面的数量。这些页面可以在需要的时候被释放，但是此刻它们仍然在内存中。

·**active=number**: “激活列表”中的页面的数量。

·**N0=number and N1=number**: 节点0和节点1上各自分配的页面的数量。

我们可以使用**numactl**命令显示可用的节点。

```
numactl --hardware
available: 2 nodes (0-1)
node 0 size: 64570 MB
node 0 free: 8556 MB
node 1 size: 64640 MB
node 1 free: 1982 MB
node distances:
node   0      1
0:    10    20
1:    20    10
```

如上命令告诉我们，系统有两个CPU节点：node0、node1。每个节点分配了64GB的内存。

distance衡量了访问内存的成本，系统认为访问本地节点内存的成本是10，访问远端内存的成本是20。

NUMA架构存在的一个问题是，对于NUMA架构，Linux默认的内存分配方案是优先在请求线程当前所处的CPU的本地内存上尝试分配空间，一般是node0。如果内存不够，系统就会把node0上已经分配的内存交换出去，以释放部分node0的内存，尽管node1上还有剩余的内存，但是系统不会选择向node1去申请内存。显然，swap的成本远比访问远端内存的成本高，这将导致不定时地出现性能问题。

解决办法具体如下：

1) 关闭NUMA。

如果是单机单实例，则建议关闭NUMA，关闭的方法有如下两种。

·硬件层，在BIOS中设置关闭。

·OS内核，启动时设置**numa=off**。

可用类似如下的方式进行修改。

```
[root@db1000 ~]# cat /proc/cmdline
ro root=LABEL=/ rhgb quiet
vi /etc/grub.conf
kernel /vmlinuz-2.6.18-164.el5 ro root=LABEL=/ rhgb quiet numa=off
```

确认NUMA是否关闭，检查**numactl--show**的输出信息。

```
[root@db1000 home]# /usr/bin/numactl --show
policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
cpubind: 0
nodebind: 0
membind: 0
```

关闭之前这个命令会显示多个节点的信息，输出结果如下所示。

```
policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
cpubind: 0 1
nodebind: 0 1
membind: 0 1
```

而关闭之后则只会显示一个节点的信息，**nodebind**项只有一个值0。我们也可以检查启动信息**dmesg|grep-i numa**。

2) 使用**numactl**命令将内存分配策略修改为**interleave**（交叉）或绑定CPU。

可通过修改单实例启动脚本**mysql.server**或多实例启动脚本**mysqld_multi**，例如，修改**mysqld_multi**脚本（MySQL 5.1）320行

将\$com="\$mysqld"更改为\$com="/usr/bin/numactl- interleave all\$mysqld";

也可以修改启动脚本和参数，绑定MySQL的各个实例到固定的CPU节点，笔者更推荐使用这种方式。下面的例子，在节点0的CPU上运行名为program的程序，并且只在节点0和1上分配内存。

```
numactl --cpubind=0 --membind=0,1 program
```

下面的例子，在节点1上运行\$MYSQLD程序，只在节点内分配内存。

```
numactl --cpunodebind=1 --localalloc $MYSQLD
```

3) 设置参数memlock。

MySQL进行初始化启动的时候，就已经预先把InnoDB缓冲池的内存锁住了，即设置参数memlock等于1，设置这个参数，也有一定的风险，如果内存不够，可能会导致系统启动不正常，因为MySQL Server会不断申请内存。

4) 使用大内存页。

还有一些其他的辅助手段。

配置vm.zone_reclaim_mode=0使得内存不足时倾向于向其他节点申请内存。

echo-15>/proc/<pid_of_mysqld>/oom_adj，将MySQL进程被OOM_killer强制kill的可能性调低。

18.4 MySQL CPU优化

系统的性能一般取决于系统所有组件中最弱的短板，CPU、内存、I/O、网络都可能会成为瓶颈所在。现实中，一般是CPU瓶颈或I/O瓶颈，I/O瓶颈也可能是由于内存不够所导致的。

CPU的瓶颈一般是大量运算和内存读取所导致的，比如加密操作、索引范围查找、全表扫描等。生产环境中出现CPU瓶颈往往是因为大量的索引范围查找或连接了太多表。I/O瓶颈往往是因为内存已经不能保存住数据库的热数据，因此读写操作必须访问实际的物理磁盘，从而导致过多的物理读。

实际生产环境中，更多的会碰到I/O瓶颈，而不是CPU瓶颈，你可以使用top或mpstat判断数据库服务器是否存在CPU瓶颈。

由于MySQL在多CPU主机上的扩展性有限，不能充分利用多CPU的主机，所以生产中可能会在同一个主机上部署多个实例。有时我们会绑定MySQL实例到某个CPU节点上。

如果想要优化性能，那么我们更倾向于选取速度更快的CPU，而不是增加CPU。从理论上来说，如果操作比较集中于一些资源对象，瓶颈多是因为锁和队列等待，那么这个时候应该选取更强劲的CPU。而如果操作分散于诸多不相干的资源上，那么并发程度可以更高，可以倾向于使用更多的CPU，但能否使用更多的CPU、并发多线程执行操作，还要受制于存储引擎的设计。就目前来说，InnoDB的扩展性还是不佳。

下面我们来看看CPU的高级特性。

PC Server上有一种节能模式，一般是处于关闭的状态，这种电源管理技术可以在负载低的时候，调低CPU的时钟速度，降低能耗，但这种技术并不能和突发的负荷协作得很好，有时会来不及调整时钟以响应突然的高并发流量。

还有另外一种电源管理技术，它通过分析当前CPU的负载情况，智能地完全关闭一些用不上的核心，而把能源留给正在使用的核心，并使它们的运行频率更高，从而进一步提升性能。相反，需要多个核心时，应动态开启相应的核心，智能调整频率。这样，就可以在不影响CPU的TDP（热功耗设计）的情况下，把核心工作频率调得更高。这种加速技术可能会破坏我们的性能规划，因为系统的行为并不是“线性”的了。

18.5 MySQL I/O优化

18.5.1 概述

我们的生产环境一般是OLTP应用，I/O瓶颈一般来自于随机读写，随机读的消除和写的缓解主要靠缓存，所以我们要确保MySQL的缓冲区能够缓存大部分的热点数据。当然，也没有必要缓存所有的热点数据，可以接受一定的缓存未命中（cache miss）。注意，传统的一个调优方法是基于命中率进行调优，更靠谱的方案是基于缓存未命中情况进行调优，虽然有时命中率很高了，但只要缓存未命中次数达到一定的频率，你就会碰到I/O瓶颈。

数据库引擎比操作系统或RAID更了解数据，能够更高效地访问数据，文件系统和RAID层面的预读要关掉，因为它们帮不上什么忙，应该交给数据库以更智能地判断数据的读取。

内存的随机读写速度比硬盘的随机读写速度快了几个数量级，所以如果有I/O的性能问题，那么添加内存会是最简便的方案。数据库缓冲是调优的重点，我们需要确保数据库缓冲能够缓存大部分的热点数据，理论上来说，如果数据库缓冲已经不够了，那么文件系统或RAID缓冲也没有什么用，因为它们要小得多，且不了解

数据，缓存应该考虑在更接近用户的地方进行优化，由于应用比数据库更了解数据，所以对于高并发的业务，客户端/应用程序的本地内存或缓存服务（如Memcached）会比MySQL更有效率，提供更好的扩展性。

顺序读写无论是在内存还是在磁盘中，都比随机读写更快。一般是不用考虑特殊的缓存策略。对于机械硬盘，由于磁盘的工作原理，顺序读写的速度比随机读写速度快得多，我们需要着重优化随机读写，尽量减少随机读，以提高吞吐。对于SSD，虽然顺序读写也很快，相对而言，随机读写并没有差太多，而且优化随机读写也不是那么迫切，但是还是有必要优化大量随机读写的SQL，因为随着访问量的上升，贡献大量随机读写的SQL，将会很快导致整个系统出现瓶颈。

随机读写往往来自质量不高的SQL，这些SQL往往是因为索引策略不佳或表连接过多，从应用层优化或进行索引优化，会更有效果，也更具可行性。

文件碎片也可能会导致更多的随机I/O，尽管数据库是顺序访问数据的，但是I/O却不是顺序的，MySQL自身并没有提供工具来检查数据文件是否碎片很多，我们也不建议频繁地进行表的重建和优化，但是在进行了大批量数据操作之后，比如大量删除数据之后，在不影响服务的前提下，优化一下表（OPTIMIZE TABLE）还是可取的。

对于OLAP应用，I/O调优和OLTP有些相似，也是要先考虑应用调优和SQL调优，尽量减少I/O操作，如果必须要执行大量的I/O操作，那么应该尽量将其转换为顺序读写。

18.5.2 选择合适的I/O大小

一般来说，MySQL的块大小是操作系统块的整数块，你可以通过命令`getconf PAGESIZE`来检查操作系统的块大小，更大的I/O大小，意味着更大的吞吐，尤其是对于传统的机械硬盘，一次更大的I/O，意味着不需要进行多次I/O，可以减少寻道的时间。对于数据库，由于往往是一些随机记录的检索，因此并不需要一次性读取大量的记录，所以一次I/O不需要太大。许多人把默认的数据库的块大小调整为8KB，以获得更高的性能。

18.5.3 日志缓冲如何刷新到磁盘

对于数据库的I/O性能调整，需要在性能和数据的安全性上求得平衡。如果生产环境有严重的I/O性能问题，那么它往往是由程序的不良设计造成的。一个应用级别的SQL调整，可能就能解决了问题。而从操作系统的I/O层面可能就会无解。

InnoDB使用了数据缓冲和事务日志，数据缓冲大小、日志大小、日志缓冲、InnoDB如何刷新数据和缓冲，都会对性能产生影响。

InnoDB的脏数据并不是马上写入数据缓冲（数据文件）的，而是会先写日志缓冲（日志文件），将脏数据暂时保留在数据缓冲区中，这是一种常见的数据库持久化的技术，这些日志记录了数据变更，可以用来做故障恢复。数据的读写一般是随机读写，而日志的写入，是顺序写入，日志写入的效率高得多，通过延缓数据的持久化，可以将数据更高效率地写入到磁盘中。

InnoDB在缓冲区满的情况下会将日志缓冲区刷新到磁盘，一般不需要调整日志缓冲区的大小（`innodb_log_buffer_size`），除非有很多有BLOB字段的记录，`innodb_log_buffer_size`的大小默认是1MB，建议是1~8MB。

我们通过配置`innodb_flush_log_at_trx`来控制如何将日志缓冲刷新到磁盘，`innodb_flush_log_at_trx`的值可设置为0、1、2，默认为1。

设置为1的情况下，每个事务提交都要写入磁盘，这是最安全的做法，而设置为其他值时，可能会丢失事务。一般机械磁盘受磁盘旋转和寻道的限制，最多只能达到几百次IO/每秒，所以这个设置会严重降低事务并发，如果你数据库的安全性要求很高，那么设置`innodb_flush_log_at_trx`为1，这时你可能要把日志文件放在更好的磁盘设备上，如SSD设备或带电池的磁盘阵列上。

如果将`innodb_flush_log_at_trx`设置为2，那么每次事务提交时会将日志缓冲写到操作系统缓存中，但不实际刷新到磁盘中，每秒再刷新日志缓冲到磁盘中，这样做可以减轻I/O负荷，如果不存在极端的情况，理论上宕机最多只会丢失最近1秒的事务。

如果`innodb_flush_log_at_trx`设置为0，那么每秒都会将日志缓冲写到日志文件中，且将日志文件刷新到磁盘，但在事务提交的时候并不会将日志缓冲写到日志文件中，一般不建议将其设置为0，在设为0时，如果mysqld进程崩溃，那么停留在日志缓冲区的数据将被丢失，因此你会丢失事务。当为2时，进程虽然会崩溃，但每次事务提交，都写入了日志，只是暂时没有被刷新到磁盘，所以不会丢失事务，因为操作系统负责把这些数据写入文件，当然，如果宕机了，那么你的数据还是会被丢失的。

设置为1将会更安全，但每次事务提交时都会伴随磁盘I/O，受机械硬盘的寻道和旋转延迟限制，可能会成为系统瓶颈，在确认可以满足I/O性能的前提下，可将其设置为1。

建议在生产环境中将`innodb_flush_log_at_trx`设置为2。

18.5.4 事务日志

如果日志文件里记录的相关数据并未写入数据文件，那么这个日志文件是不能被覆盖的。日志文件如果过小，那么可能会过多地检查点操作，增加I/O操作，而如果日志文件过大，则会增加实例崩溃的恢复时间。一般建议在生产系统中将其大小设置为256~512MB，以平衡恢复时间和性能。你也可以定量分析实际需要的事

务日志大小，方法是衡量一段时间（如0.5~2个小时）内写入的日志记录（`innodb_os_log_written`）的大小，你所分配的多个日志的总计大小应能确保保留此段时间的日志。

事务日志一般没有必要和数据文件分离，除非你有许多（20+）盘。如果只有几个盘，却专门使用独立的盘来存放二进制日志、事务日志，则有些浪费。在有足够的盘的情况下，磁盘I/O分离才有意义，不然成本就会太高了，且无法充分利用有限的资源。建议将日志文件和数据文件放在同一个盘/卷的另一个原因是日志文件和数据文件放在一起，可以做LVM快照。

18.5.5 二进制日志

如果分离二进制日志和数据文件，可能会带来一点性能上的提升，但分离的主要目的不是性能，而是为了日志的安全。如果没有带电池的RAID卡，那么分离就是有必要的。如果有带电池的RAID卡，那么一般情况下就没有必要进行分离，即使有许多顺序日志写入，RAID卡也可以合并这些操作，最终只会看到不多的一些顺序I/O。

如果将二进制日志存放在独立的盘上，那么即使我们的数据文件损坏了，我们也可以利用备份和日志做时间点恢复。

18.5.6 InnoDB如何打开和刷新数据、日志文件

InnoDB有几种方式和文件系统进行交互，默认是以`fdatasync`的方式读写文件的，生产环境中推荐设置为`O_DIRECT`。以下将简单介绍这两种方式。

·`fdatasync`: 默认InnoDB使用`fsync()`刷新数据和日志。使用默认设置没有什么问题，但也许发挥不了你硬件的最高性能。

·`O_DIRECT`: 对于数据文件，MySQL Server也是调用`fsync()`刷新文件到磁盘的，但是不使用操作系统的缓存和预读机制，以避免双重缓冲，如果你有带电池的RAID卡，则可以配合这个选项一起使用。注意RAID卡需要开启写缓存，默认策略是Write Back。

18.5.7 InnoDB共享表空间和独立表空间

InnoDB表空间不仅可以存储表和索引数据，还有UNDO（可以理解为数据前像）、`insert buffer`、`doublewrite buffer`等其他内部数据结构。

目前有两种表空间的管理方式，共享表空间和独立表空间。默认的是共享表空间的管理方式，InnoDB表空间的管理比较简单，并没有Oracle那样丰富的特性。如果使用默认的共享表空间的话，数据和索引就是放在一起的，所有数据都存储在`innodb_data_file_path`参数设置的数据文件里。

我们可以通过`innodb_data_file_path`设置多个InnoDB数据文件，一般将最后一个文件设置为可自动扩展的，以减少数据文件的大小，你也可以将数据文件分离到不同的磁盘中。由于数据文件不能收缩，所以使用共享表空间存在一个严重的问题是空间的释放。如果你增加了数据文件，那么你还需要重启数据库实例，这些都加大了管理开销。

当自动扩展的数据文件被填满之时，每次扩展默认为8MB，我们可以调整为更大的值，如32MB、64MB，这个选项可以在运行时作为全局系统变量而改变。因为每次分配小空间，代价都会比较大，所以预分配一个较大的文件是有道理的。

另一种方式是独立表空间，我们需要将`innodb_file_per_table`设置为1。

这个选项可以将每个InnoDB表和它的索引存储在它自己的文件中，由于每个表都有自己的表空间，所以又称为独立表空间。UNDO、各种数据字典等其他数据仍然存储在共享表空间内。你可以通过操作系统命令比较直观地看到数据大小，也方便删除表释放空间，所以许多有经验的DBA都设置MySQL实例为独立表空间，从而可以更方便地释放空间和减少文件系统的I/O争用。

InnoDB也支持在裸设备上存储，通过这种方式，你也许可以得到少许的性能提升，但由于管理难度比较大，因此很少有人使用这种方式管理数据库文件。

18.5.8 UNDO暴涨的可能性

有时我们的共享表空间会暴涨，其实是由于UNDO空间发生了暴涨，UNDO空间暴涨的原因主要有如下两点。

·存在长时间未提交的事务，因为未提交的事务需要使用发布查询时刻的UNDO的数据，所以共享表空间内的这部分UNDO数据不能被清除，将会积累得越来越多。

·也许是负载太高，清理线程还来不及清除UNDO，这种情况下，性能将会急剧下降。

18.5.9 关于`doublewrite buffer`

InnoDB可使用`doublewrite buffer`来确保数据安全，以避免块损坏。`doublewrite buffer`是表空间的一个特殊的区域，可顺序写入。当InnoDB从缓冲池刷新数据到磁盘时，它首先会写入`doublewrite buffer`，然后写入实际的数据文件。InnoDB检查每个页块的校验和，以判断是否坏块，如果写入`doublewrite buffer`的是坏块，那么显然还没

有写入实际数据文件，那么就用实际数据文件的块来恢复doublewrite buffer。如果写入了doublewrite buffer，但是数据文件写的是坏块，那么就用doublewrite buffer的块来重写数据文件，这也是MySQL灾难恢复的一个基本步骤。如果操作系统本身支持写入安全，不会导致坏块，那么我们可以禁用这个特性。

18.5.10 数据库文件分类

可以考虑把二进制日志文件、InnoDB数据文件的物理文件分布到不同的磁盘中，这样做主要考虑的是把顺序I/O和随机I/O进行分离。你也可以把顺序I/O放到机械硬盘上，把随机I/O放到SSD上，如果有带电池的RAID卡且开启了写缓存，那么顺序I/O的操作一般是很快的。具体如何放置文件，还需要综合考虑性能、成本和维护性等多个因素。笔者的做法是，如果没有性能问题，就把所有文件都放在一个盘上，这样维护起来将会更方便。

如下是按照顺序I/O和随机I/O对数据库文件做了下分类。

(1) 随机I/O

- 表数据文件 (*.ibd)：启用了独立表空间 (innodb_file_per_table=1)。

- UNDO区域 (ibdata)：UNDO里存储了数据前像，MySQL为了满足MVCC，需要读取存储在UNDO里的前像数据，这将导致随机读，如果你要运行一个需要很长时间的事务或一个时间很长的查询，那么可能会导致很多随机读，因为长事务或未提交的事务将有更多的可能性读取前像数据。

(2) 顺序I/O

- 事务日志 (ib_logfile*)。

- 二进制日志 (binlog.xxxxxxx)。

- doublewrite buffer(ibdata)。

- insert buffer(ibdata)。

- 慢查询日志、错误日志、通用日志等。

18.5.11 何时运行OPTIMIZE TABLE

有些人会建议定时运行一些OPTIMIZE TABLE之类的命令，以优化性能，这点与Oracle类似，也总会有些人建议你定时运行重建索引的操作。一般来说，除非在进行了大量会影响数据分布的操作之后，比如删除了大量的数据、导入数据等，一般情况下是不需要重整表的。定时地运行OPTIMIZE TABLE命令不现实，还可能会导致生产系统的不可用。

OPTIMIZED TABLE命令会优化InnoDB主键的物理组织，使之有序、紧凑，但是其他索引仍然会和以前一样未被优化。哪一个索引对性能更重要呢？也许从来没有基于主键的查询条件。其实，数据、索引的分布也是需要一个过程的，随着时间的演变，自然而然会达到一个平衡。强制优化之后，过一段时间，它又会回到原来的不好不坏的状态。

所以MySQL 5.1的官方文档中才会建议：如果您已经删除了表的大部分，或者如果您已经对含有可变长度行的表（含有VARCHAR、BLOB或TEXT列的表）进行了很多更改，则应使用OPTIMIZE TABLE。

18.5.12 MySQL磁盘空间

磁盘空间如果出现瓶颈，往往是因为数据库规划失误，前期没有进行足够的调研，也有小部分原因是业务发展得太快了，数据呈现爆炸式增长。大部分业务，一般预留1到2年的数据增长空间就已经足够了，如果你预计数据未来会有一个海量的规模，那么提前进行分库分表则是有必要考虑的。

你需要尽可能地了解占据数据库总体空间比重较大的一些数据，清楚哪些表是可以被清理或归档的，许多情况下，我们并不需要这么多的数据，或者许多数据是不需要保留很久的，是完全可以清除的，你越了解数据，就越能够和研发团队一起制定合理的数据保留策略。

在系统上线之前，你就需要制订好将数据进行批量清理和归档的方案，可以使用定期任务删除数据，你也可以利用分区表删除旧的历史数据。

当数据库实例的数据变得很大，单台机器已经很难保存所有数据的时候，你可以考虑将实例、数据库分离到其他的机器。

由于处理器和高速缓存存储器速度的提升超过磁盘存储设备速度的提升，许多业务将受磁盘空间所累。一些业务拥有海量数据，但大部分都是冷数据，你又不能进行简单的归档处理，这个时候数据压缩就派上用场了。

目前的数据库主机，CPU资源往往过剩，数据压缩可以减少数据库的大小，减少I/O和提高吞吐量，而压缩仅仅只会消耗部分CPU成本。MySQL 5.5开始提供了InnoDB表压缩的功能，在MySQL 5.6中InnoDB表压缩的功能得到了进一步的完善，真正可以用于生产环境了。

对于真正海量高并发的应用，内存为王，你应该在内存中尽可能地保证热点数据和索引，更多的索引和数据可以放在一个内存块中，那么查询的响应也将更

快，表是压缩的也意味着你需要更少的存储空间和更小、更少的I/O操作。对于MySQL 5.5、5.6，你需要配置为独立的表空间才能使用表的压缩功能，对于MySQL 5.7，你也可以不使用独立表空间。

由于固态硬盘一般比传统机械硬盘要小，且成本更高，所以压缩对固态硬盘尤其有意义。

不同的内容压缩率将会不一样，如果你需要将表修改为压缩表，那么你需要在更改之前进行测试验证，以确认压缩率和转换表的时间，一般来说，设置KEY_BLOCK_SIZE为8KB可以适用于大部分情况，8KB意味着将每个页压缩为8KB，你也可以将标准的16KB页压缩为4KB或2KB，但可能会导致过多的性能损耗而压缩率并不能得到提升。



小结 本章讲述了调优将会涉及的MySQL参数及在使用MySQL的过程中，内存、CPU、I/O的优化。笔者不推荐读者对生产环境的参数做大的调整，也不推荐使用各种不常用的手段去优化硬件资源的利用率，压榨硬件的性能。保持一个维护性更好的数据库，使用通用的参数，可以让工作变得更简单些，笔者认为这才是更重要的。但是，作为DBA，一定要熟悉各种调优的手段，因为你可能会碰到极端的场景。

第19章 操作系统、硬件、网络的优化

本章将介绍操作系统和硬件的性能优化，对于硬件，我们主要讲述CPU、内存、磁盘阵列及固态硬盘。任何优化，首先都需要有足够的数据支持，对于操作系统下性能数据的收集，这里将不再赘述，请参考前面章节的相关内容。

19.1 基本概念

如下是需要了解的一些基本概念。

(1) 什么是进程

进程可以简单地理解为程序加数据，程序本身只是指令、数据及其组织形式的描述，进程才是程序（那些指令和数据）的真正运行实例。若干进程都有可能与同一个程序有关系，且每个进程都可以用同步（循序）或异步（平行）的方式独立运行。用户下达运行程序的命令之后，就会产生进程。同一程序可以产生多个进程（一对多的关系），以允许同时有多位用户运行同一程序，却不会发生冲突。

进程在运行时，状态会发生改变，如新生、运行、等待、就绪、结束等，各状态的名称也可能会随着操作系统的不同而不同。

(2) 什么是线程

线程是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。一个进程可以有许多线程，每条线程并行执行不同的任务。使用多线程技术（多线程即每一个线程都代表一个进程内的一个独立执行的控制流）的操作系统或计算机架构，同一个程序的平行线程，可在多CPU主机或网络上真正做到同时运行（在不同的CPU上）。

多线程技术可以让一个进程充分利用多个CPU。同一进程中的多条线程将会共享该进程中的全部系统资源，如虚拟地址空间、文件描述符和信号处理等。但同一进程中的多个线程也都有各自的调用栈（call stack）、寄存器环境（register context）和线程本地存储（thread local storage）。在多核或多CPU上使用多线程程序设计的好处是显而易见的，即提高了程序的执行吞吐率。

(3) 什么是内核调度

内核调度将把CPU的运行时间分成许多片，然后安排给各个进程轮流运行，使得所有进程仿佛在同时运行，内核需要决定运行哪个进程/线程，哪个需要等待，选择要在哪个CPU核上运行线程。内核运行于一个特殊的CPU态，内核态。拥有完全的权限访问设备，内核将仲裁对设备的访问，以支持多任务，避免用户进程访问彼此的空间而破坏数据，除非显式被允许访问。用户程序一般运行在用户态，它们通过系统调用的方式执行一些限制权限的操作，例如I/O操作。

(4) 什么是虚拟内存

虚拟内存是计算机系统内存管理的一种技术。它使得应用程序认为它拥有连续的、可用的内存（一个连续的、完整的地址空间），而实际上，它通常是被分隔成多个物理内存碎片，还有部分被暂时存储在外部磁盘存储器上，在需要时将会进行数据交换。与没有使用虚拟内存技术的系统相比，使用这种技术的系统将使得大型程序的编写变得更容易，对真正的物理内存（例如RAM）的使用也更有效率。

对虚拟内存的定义是基于对地址空间的重定义的，即把地址空间定义为“连续的虚拟内存地址”，以此来“欺骗”程序，使它们以为自己正在使用一大块的“连续”的地址。

对于每个进程或内核而言，它们操作大块的虚拟内存，而实际虚拟内存到物理内存的映射，是由我们的虚拟内存管理系统来实现的，也就是说，虚拟内存给进程或内核提供了一个它们独有的几乎无限内存的视图。

对于操作系统的优化，主要是对一些内核参数及文件系统进行优化。由于默认的Linux内核参数已经基本够用，因此本书将只关注文件系统的优化。

19.2 文件系统的优化

文件系统是一种向用户提供底层数据访问的机制。它将设备中的空间划分为特定大小的块（扇区），一般每块有512B。数据存储在这些块中，由文件系统软件来负责将这些块组织为文件和目录，并记录哪些块被分配给了哪个文件，以及哪些块没有被使用。以下仅针对文件系统和MySQL相关的部分做一些说明。

常用的文件系统有ext3、ext4、XFS等，你可以检查Linux系统的/etc/fstab文件，以确定当前分区使用的是什么文件系统，ext3即第三代扩展文件系统，是一个日志文件系统，低版本的Linux发行版将会使用到这种文件系统，它存在的一个问题是删除大文件时比较缓慢，这可能会导致严重的I/O问题。ext4即第四代扩展文件系统，是ext3文件系统的后继版本。2008年12月25日，Linux 2.6.29版公开发布之后，ext4成为了Linux官方建议的默认的文件系统。ext4改进了大文件的操作效率，使删除大的数据文件不再可能导致严重的I/O性能问题，一个100多GB的文件，也只需要几秒就可以被删除掉。

文件系统使用缓存来提升读性能，使用缓冲来提升写性能。在我们调整操作系统和数据库的时候，要注意批量写入数据的冲击，一些系统会缓冲写数据几十秒，然后合并刷新到磁盘中，这将表现为时不时的I/O冲击。

默认情况下，Linux会记录文件最近一次被读取的时间信息，我们可以在挂载文件系统的时候使用noatime来提升性能。为了保证数据的安全，Linux默认在进行数据提交的时候强制底层设备刷新缓存，对于能够在断电或发生其他主机故障时保护缓存中数据的设备，应该以nobarrier选项挂载XFS文件系统，也就是说，如果我们使用带电池的RAID卡，或者使用Flash卡，那么我们可以使用nobarrier选项挂载文件系统，因为带电池的RAID卡和FLASH卡本身就有数据保护的机制。还有其他的一些挂载参数也会对性能产生影响，你需要衡量调整参数所带来的益处是否值得，笔者一般不建议调整这些挂载参数。它们对性能的提升并不显著。

你可能需要留意文件系统的碎片化，碎片化意味着文件系统上的文件数据块存放得不那么连续，而是以碎片化的方式进行分布，那么顺序I/O将得不到好的性能，会变成多次随机I/O。

所以在某些情况下，使用大数据块和预先分配连续的空间是有道理的，但你也需要知道，文件碎片是一个常态，最开始的表没有什么碎片，但随着你更新和删除数据，数据表会变得碎片化，这会是一个长期的过程，而且在绝大多数情况下，你会感觉不到表的碎片对于性能的影响，因此，除非你能够证明表的碎片化已经严重影响了性能，否则不建议进行表的整理，比如运行OPTIMIZE TABLE命令。

Direct I/O允许应用程序在使用文件系统的同时绕过文件系统的缓存。你可以用Direct I/O执行文件备份，这样做可以避免缓存那些只被读取一次的数据。如果应用或数据库，已经实现了自己的缓存，那么使用Direct I/O可以避免双重缓存。

许多人期望使用mmap的方式来解决文件系统的性能问题，mmap的方式有助于我们减少一些系统调用，但是，如果我们碰到的是磁盘I/O瓶颈，那么减少一些系统调用的开销，对于提升整体性能/吞吐的贡献将会很少。因为主要的瓶颈，主要花费的时间是在I/O上。许多NoSQL的数据库使用了mmap的方式来持久化数据，在I/O写入量大的时候，其性能急剧下降就是这个道理。

一般来说，文件系统缓存，对于MySQL的帮助不大，可以考虑减小文件系统缓存，如vm.dirty_ratio=5。

我们推荐在Linux下使用XFS文件系统，它是一种高性能的日志文件系统，特别擅长处理大文件，对比ext3、ext4，MySQL在XFS上一般会有更好的性能，更高的吞吐。Red Hat Enterprise Linux 7默认使用XFS文件系统。Red Hat Enterprise Linux 5、6的内核完整支持XFS，但未包含创建和使用XFS的命令行工具，你需要自行安装。

19.3 内存

我们需要了解CPU、内存、固态硬盘及普通机械硬盘访问速度的差异，比如内存为几十纳秒(ns)，而固态硬盘大概是25μs(25000ns)，而机械硬盘大概是6毫秒(6000000ns)，它们差得不是一两个数量级，机械硬盘对比内存差了五六個数量级，所以内存访问比磁盘访问要快得多，所以总会有许多人想尽办法优化数据的访问，尽量在内存当中来访问数据。

内存往往是影响性能最重要的因素，你应该确保热点数据存储在内存中，较少的内存往往意味着更多的I/O压力。许多应用一般是有热点数据的，且热点数据并不大，可以保存在内存中。对于MySQL来说，应将innodb_buffer_pool_size设置得大于我们的热点数据，否则可能会出现某个MySQL实例InnoDB的缓冲不够大，从而产生过多的物理读，进而导致I/O瓶颈。

数据库服务器应该只部署数据库服务，以免被其他程序影响，有时其他程序也会导致内存压力，如占据大量文件的系统缓存，就会导致可用内存不够。

19.4 CPU

现实世界中，CPU的技术发展得很快，一颗CPU上往往集成了4/6/8个核，由于多核很少会全部利用到，所以一般会在生产机器上部署多实例，以充分利用CPU资源。还可以更进一步，使用CPU绑定技术将进程或线程绑定到一个CPU或一组CPU上，这样做可以提升CPU缓存的效率，提升CPU访问内存的性能。对于NUMA架构的系统，也可以提高内存访问的局部性，从而也能提高性能。

CPU利用率衡量的是在某个时间段，CPU忙于执行操作的时间的百分比，但是，许多人不知道的是，CPU利用率高并不一定是在执行操作，而很可能是在等待内存I/O。CPU执行指令，需要多个步骤，这其中，内存访问是最慢的，可能需要几十个时钟周期来读写内存。所以CPU缓存技术和内存总线技术是非常重要的。

我们对CPU时钟频率这个主要的指标可能有一些误解。如果CPU利用率高，那么更快的CPU不一定能够提升性能。也就是说，如果CPU的大部分时间是在等待锁、等待内存访问，那么使用更快的CPU不一定能够提高吞吐。

关于容量规划。

对于访问模式比较固定的应用，比如一些传统制造业的生产系统，则比较容易对CPU进行容量规划，可以按照未来的访问请求或访问客户端数量，确定CPU需要扩容的幅度，你可以监控当前系统的CPU利用率，估算每个客户端/每个访问请求的CPU消耗，进而估算CPU 100%利用率时的吞吐，安排扩容计划。由于互联网业务，负荷往往变化比较大，多实例有时会导致CPU的容量模型更为复杂，我们更多地依靠监控的手段提前进行预警，在CPU到达一定利用率，负载到达一定阈值时，进行优化或扩容。

如何选购CPU。

对于企业用户来说，CPU的性能并不是最重要的，最重要的是性价比，新上市的CPU往往价格偏贵，一般来说建议选择上市已经有一定时间的CPU。而对于大规模采购，你需要衡量不同CPU的价格及测试验证实际业务的吞吐，进而能够得出一个预算成本比较合适的方案，可能你还需要综合平衡各种其他硬件的成本以确定选购的CPU型号。

19.5 I/O

19.5.1 概述

I/O往往是数据库应用最需要关注的资源。作为数据库管理人员，你需要做好磁盘I/O的监控，持续优化I/O的性能，以免I/O资源成为整个系统的瓶颈。本节将讲述一些硬件维护人员需要了解的磁盘硬件知识，并对它的规划和调整做一些介绍。

一些基础概念的介绍如下。

·逻辑I/O：可以理解为是应用发送给文件系统的I/O指令。

·物理I/O：可以理解为是文件系统发送给磁盘设备的I/O指令。

·磁盘IOPS：每秒的输入输出量（或读写次数），是衡量磁盘性能的主要指标之一。IOPS是指单位时间内系统能处理的I/O请求数量，一般以每秒处理的I/O请求数量为单位，I/O请求通常为读或写数据操作的请求。OLTP应用更看重IOPS。

·磁盘吞吐：指单位时间内可以成功传输的数据数量。OLAP应用更看重磁盘吞吐。

实践当中，我们要关注的磁盘I/O的基本指标有磁盘利用率、平均等待时间、平均服务时间等。如果磁盘利用率超过60%，则可能导致性能问题，磁盘利用率往往是大家容易忽视的一个指标，认为磁盘利用率没有达到100%，就可以接受，其实，磁盘利用率在超过60%的时候，就应该考虑进行优化了。对于磁盘利用率的监控，在生产中，往往也会犯一个错误，由于监控的粒度太大，比如10分钟、2分钟一次，因此会没有捕捉到磁盘高利用率的场景。

Linux有4种I/O调度算法：CFQ、Deadline、Anticipatory和NOOP，CFQ是默认的I/O调度算法。在完全随机的访问环境下，CFQ与Deadline、NOOP的性能差异很小，但是一旦有大的连续I/O，CFQ可能会造成小I/O的响应延时增加，数据库环境可以修改为Deadline算法，表现也将更稳定。

如下命令将实时修改I/O调度算法：

```
echo deadline > /sys/block/sdb/queue/scheduler
```

如果你需要永久生效，则可以把命令写入/etc/rc.local文件内，或者写入grub.conf文件中。

19.5.2 传统磁盘

传统磁盘本质上是一种机械装置，影响磁盘的关键因素是磁盘服务时间，即磁盘完成一个I/O请求所花费的时间，它由寻道时间、旋转延迟和数据传输时间三部分构成。

一般读取磁盘的时候，步骤如下。

- 1) 寻道：磁头移动到数据所在的磁道。
- 2) 旋转延迟：盘片旋转将请求数据所在的扇区移至读写磁头下方。
- 3) 传输数据。

一般随机读写取决于前两个步骤，而大数据顺序读写更多地取决于第3)个步骤，由于固态硬盘消除了前两个步骤，所以在随机读写上会比传统机械硬盘的IOPS高得多。

优化传统磁盘随机读写，也就是优化寻道时间和旋转延迟时间，一些可供考虑的措施有缓存、分离负载到不同的磁盘、硬件优化减少延时及减少震荡等。比如，操作系统和数据库使用的是不同的盘，我们需要了解读写比率，如果大部分是读的负载，那么我们加缓存会更有效；而如果大部分是写的负载，那么我们增加磁盘提高吞吐会更有意义。对于Web访问，由于本身就可能有几百毫秒的延时，那么100毫秒的磁盘延时也许并不是问题；而对于数据库应用，对于延时则要求很苛

刻，那么我们需要优化或使用延时更小的磁盘。

对于数据库类应用，传统磁盘一般做了RAID，那么RAID卡自身也可能会成为整个系统的瓶颈，也需要考虑优化。

19.5.3 关于RAID

几种常用的RAID类型如下。

·RAID0：将两个以上的磁盘串联起来，成为一个大容量的磁盘。在存放数据时，数据被分散地存储在这些磁盘中，因为读写时都可以并行处理，所以在所有的级别中，RAID 0的速度是最快的。但是RAID 0既没有冗余功能，也不具备容错能力，如果一个磁盘（物理）损坏，那么所有的数据都会丢失。

·RAID1：RAID 1就是镜像，其原理为在主硬盘上存放数据的同时也在镜像硬盘上写一样的数据。当主硬盘（物理）损坏时，镜像硬盘则代替主硬盘工作。因为有镜像硬盘做数据备份，所以RAID 1的数据安全性在所有的RAID级别上来说是最好的。理论上读取速度等于硬盘数量的倍数，写入速度有微小的降低。

·Raid10：指的是RAID1+0，RAID1提供了数据镜像功能，保证数据安全，RAID0把数据分布到各个磁盘，提高了性能。

·RAID5：是一种性能、安全和成本兼顾的存储解决方案。RAID 5至少需要3块硬盘，RAID 5不是对存储的数据进行备份，而是把数据和相对应的奇偶校验信息存储到组成RAID5的各个磁盘上。当RAID5的一个磁盘数据被损坏后，可以利用剩下的数据和相应的奇偶校验信息去恢复被损坏的数据。

几种RAID的区别如下。

1) RAID10理论上可以提供比RAID5更好的读写性能因为它不需要进行奇偶性校验。RAID 5具有和RAID 0相近似的数据读取速度，只是因为多了一个奇偶校验信息，写入数据的速度相对单独写入一块硬盘的速度略慢。

2) RAID10提供了更高的安全性。RAID5只能坏一块盘，RAID10视情况而定，最多可以坏一半数量的硬盘。

3) RAID5成本更低，也就是说空间利用率更高。RAID 5可以理解为是RAID 0和RAID 1的折中方案。RAID 5可以为系统提供数据安全保障，但保障程度要比镜像低而磁盘空间利用率要比镜像高，存储成本相对较便宜。

以上的区别是一些理论上的说明，实际情况可能还会因为算法、缓存的设计而不同。

我们是使用多个RAID，还是使用一个大RAID，将取决于我们是否有足够多的磁盘。如果我们有许多盘，比如超过10多块盘，那么我们使用多个阵列，是可取的；而如果你只有几块盘，比如6块盘，那么单独使用两块盘来做一个RAID1用于存放操作系统，就不太可取了。

RAID卡有两种写入策略：Write Through和Write Back。

·Write Through：将数据同步写入缓存（若有Cache的情况）和后端的物理磁盘。

·Write Back：将数据写入缓存，然后再批量刷新到后端的物理磁盘。

一般情况下，对于带电池模块的RAID卡，我们将写入策略设置为Write Back。写缓存可以大大提高I/O性能，但由于掉电会丢失数据，所以需要用带电池的RAID卡。

如果电池模块异常，那么为了数据安全，会自动将写入策略切换为Write Through，由于无法利用缓存写操作，因此写入性能会大大降低。一般的RAID卡电池模块仅仅保证在服务器掉电的情况下，Cache中的数据不会丢失，在电池模块电量耗尽前需要启动服务器让缓存中的数据写盘。

如果我们碰到I/O瓶颈，我们需要更强劲的存储。普通的PC服务器加传统磁盘RAID（一般是RAID 1+0）加带电池的RAID卡，是一种常见的方案。

在RAID的设置中，我们需要关闭预读，磁盘的缓存也需要被关闭。同样的，你需要关闭或减少操作系统的预读。

19.5.4 关于SSD

SSD也称为固态硬盘，目前SSD设备主要分为两类，基于PCI-E的SSD和普通SATA接口的SSD，PCI-E SSD卡性能高得多，可以达到几十万IOPS，容量可以达到几个TB以上，常用的品牌有Fusion-io，而普通的SSD虽然才几千IOPS，但性价比更好，常用的品牌有Intel等。PCI-E相对来说稳定性、可靠性都更好，由于I/O资源的极大富余，可以大大节省机架。普通SSD，基于容量和安全的考虑，许多公司仍然使用了RAID技术，随着SSD容量的增大和可靠性的提升，RAID技术不再显得重要甚至不再被使用。

由于许多公司的SSD的I/O资源往往运行不饱和，因此SSD的稳定、性能一致、安全、寿命显得更重要，而性能可能不是最需要考虑的因素。依据笔者的使用经验，许多SSD的设备故障，其原因并不在于SSD设备本身，而在于SSD设备和传统电器组件之间的连接出现了问题，主机搭载传统机械硬盘的技术已经非常成熟，而在主机上搭载SSD，仍然需要时间来提高其可靠性。所以我们在选购主机的时候，SSD在其上运行的可靠性也是一个要考虑的因素。我们对于磁盘RAID也应该加以监控，防止因为磁盘RAID异常而导致数据文件损毁。

传统的机械硬盘，瓶颈往往在于I/O，而在使用了固态硬盘之后，整个系统的瓶颈开始转向CPU，甚至在极端情况下，还会出现网络瓶颈。由于固态硬盘的性能

比较优越，DBA不再像以前那样需要经常进行I/O优化，可以把更多的时间和精力放在业务逻辑的设计上，固态硬盘的成本降低了，也可以节省内存的使用，热点数据不一定需要常驻内存，即使有时需要从磁盘上访问，也能够满足响应的需求了。传统的I/O隔离和虚拟化难度较高，很重要的原因是I/O资源本身就比较紧缺，本身就很紧缺的资源，难以进行分割和共享，而高性能的PCI-E SSD卡使得虚拟化更可能落地。

传统的文件系统已经针对传统的机械磁盘阵列有许多优化，所以想在其上再做一些软件层的优化和算法设计，很可能会费力不讨好，但是如果是SSD设备，则另当别论，用好了SSD设备，可能可以大大减少SSD设备的故障率，充分利用它的潜能，随着固态硬盘大规模的使用，未来将需要在文件系统和数据库引擎上都做出相应的优化，以减少使用SSD的成本。

19.6 网络

对于数据库应用来说，网络一般不会成为瓶颈，CPU和I/O更容易成为瓶颈。网络的瓶颈一般表现为流量超过物理极限，如果超过了单块网卡的物理极限，那么你可以考虑使用网卡绑定的技术增加网络带宽，同时也能提高可用性，如果是超过了运营商的限制，那么你需要快速定位流量大的业务，以减少流量，而请运营商调整带宽在短时间内可能难以完成。

网络瓶颈也可能因为网络包的处理而导致CPU瓶颈。交换机和路由器通过微处理器处理网络包，它们也可能会成为瓶颈，对于主机来说，如果对于网络包的处理没有一个CPU负载均衡策略，那么网卡流量只能被一个CPU处理，CPU也可能会成为瓶颈。

网络端口，也可能会成为瓶颈所在，不过这种情况很少见，即使是有大量短连接的场合。首先你需要优化连接，减少短连接，或者使用连接池，如果实在优化不下去了，可以考虑修改系统的内核参数net.ipv4.ip_local_port_range，调整随机端口范围，或者减少net.ipv4.tcp_fin_timeout的值，或者使用多个逻辑IP扩展端口的使用范围。

在进行网络优化之前，我们需要清楚自己的网络架构，了解你应用的网络数据流路径，比如是否经过了DNS服务器，你需要使用网络监控工具比如Cacti监控流量，在超过一定阈值或有丢包的情况下及时预警。

你需要意识到，跨IDC的网络完全不能和IDC内网的质量相比，且速度也可能会成为问题，跨IDC复制，其实本质上是为了安全，是为了在其他机房中有一份数据，而不是为了实时同步，也不能要求必须是实时同步。你需要确保应用程序能够处理网络异常，如果两个节点间距离3000英里^[1]，光速是186000英里/秒，那么单程需要16毫秒，来回就需要32毫秒，然后节点之间还有各种设备（路由器、交换机、中继器），它们都可能影响到网络质量。

所以，如果你不得不进行跨IDC的数据库同步，或者让应用程序远程访问数据库，那么你需要确保你的应用程序能够处理网络异常，你需要确认由于跨IDC网络异常导致的复制延时不致影响到业务。由于网络异常，Web服务器可能连接数暴涨、挂死、启动缓慢（由于需要初始化连接池），这些都是潜在的风险，你需要小心处理。

当有新的连接进来时，MySQL主线程需要花一些时间（尽管很少）来检查连接并启动一个新的线程，MySQL有一个参数back_log来指定在停止响应新请求前在短时间内可以堆起多少请求，你可以将其理解为一个新连接的请求队列，如果你需要在短时间内允许大量连接，则可以增加该数值。Linux操作系统也有类似的参数net.core.somaxconn、tcp_max_syn_backlog，你可能需要增大它们。



小结 本章介绍了文件系统及硬件的一些知识。读者平时应该关注资源的使用情况，并跟踪硬件的发展。通过对操作系统的观察，如资源的使用情况和报错日志，在某些情况下更容易发现程序的性能问题。本书介绍的许多知识都只是“泛泛而谈”，笔者自己也很少对操作系统或硬件进行调优，读者如果有兴趣深入研究操作系统和硬件，那么建议多阅读相关领域的专门著作。

[1] 1英里=1.609千米。——编辑注

第20章 可扩展的架构

本章将为读者讲述可扩展的架构相关的知识和技术。可扩展的架构意味着这个架构伸缩性好，我们可以用更多的节点来提高吞吐率，而性能不会下降到不可接受的范围。互联网世界飞速发展，数据量、访问量对比过去有了爆炸式的增长，数据库比整个系统的其他组件受到的挑战更大。一般来说我们可以通过增加Web服务器来提高处理能力，但我们很难简单地通过增加数据库服务器的节点来提高吞吐。

20.1 做好容量规划

做好容量规划，也就是收集足够的信息，看系统如何处理负载，如果负载增加时，系统应该如何扩展。

容量规划往往基于我们对于业务的理解，业务发展得如何，我们的应用需要怎样的性能目标？通过研究资源的限制和影响的因素，制订自己的容量规划。

研究资源限制的方式是，首先，我们要衡量服务的请求数，监视其增长速率，然后衡量硬件和软件的资源使用情况，监控其增长速率，然后可以把请求数的增长和资源的使用映射起来，推断在目前的资源限制下（看哪个资源会最先到达瓶颈），还能提供的最大服务请求，或者可以使用工具进行压力测试以判断最大服务请求，如果可能，应该用真实的业务流量进行压测，可以考虑tcpdump这样的能够重放流量的工具。

如上的方法，难点之一在于如何将业务的指标转换为应用服务器的访问请求，再转换为数据库的QPS，你需要熟悉业务，或者说需要数据库的维护人员和研发

人员、产品人员一起探讨，确定未来的业务流量的增长会对数据库产生什么影响。系统可能会很复杂，难以得出结论，但是，如果你留有一定的余量，技术团队也富有经验，那么在一般情况下，是不会有太大的问题的。因为你将会有足够的时间处理突然增加的负荷，对于大规模的和复杂的系统，则需要有更多的设计考虑，监控和记录各种接口的调用情况，设计各个子系统的容量，做到更自动化的预警和扩容。

研究影响的因素是指，研究数据量增长、事务吞吐等会影响到资源使用的因素，现实中，特别是互联网应用，往往低估了数据量的增长和事务的增长。所以前期要尽可能多地收集信息，了解你的实例所处的环境，做一些扩展性的规划，比如考虑是否要进行分片设计，是否要做负载均衡的规划等。

还需要考虑故障情况下的系统使用，对于海量高并发的应用，要注意其他组件的失效影响，要清楚现实系统中可能包括了各种组件，有负载均衡设备、Web服务器、应用服务器、Cache服务器（如Memcached、Redis）、消息队列、DNS服务器、Proxy等各种组件。所有的组件都需要监控和记录数据，特别是对缓存的监控要到位，要监控缓存性能的变化，统计命中率、失效率，缓存失效对数据库的冲击很大，所以你必须考虑到缓存命中率下降或缓存宕机的影响，留有一定的余量是必须的。确保当你的缓存节点挂掉一个或多个时，你的整个系统，还能继续提供稳定的服务。如果超过了系统能够提供服务的能力，那么你应该有措施和机制来保护数据库系统，以避免雪崩等连锁反应。

以上主要叙述的是随着应用规模不断增加的情况下的扩容考虑，现实中，我们还需要关注磁盘空间、内存等单一资源的扩展，由于它们相对比较简单，因此这里将不再赘述。

20.2 扩展和拆分

在互联网的架构设计中，一个关键的衡量指标就是系统的可扩展性，也称为伸缩性，指的是系统不断增加其承载能力的能力。在业务的高速发展，IT系统不应该成为整个公司的瓶颈。

扩展可简单地分为以下两类。

1. 向上扩展（scale up）

相对而言，这是更简单的方式，你应该优先使用该方式，一般情况下向上扩展就足够了，但如果向上扩展的代价太大了，那么它就不可取了。向上扩展总是有极限的，比如，你可以不断地增加CPU节点，但由于CPU节点之间还需要进行通信，因此CPU缓存一致性会越来越难以保证，等到了一个极限，即使你增加了CPU节点，系统的吞吐也不可能上升，甚至可能还会下降。

目前MySQL能充分利用的资源是有限制的，比如主机使用的是256 GB内存、32核、一个PCI-E Flash卡一般就足够了，如果超过了这个硬件成本，即使有性能上的提升，但成本上可能就不合算了。随着软硬件技术的发展和硬件价格的变化，你需要寻找一个性价比良好的方案合理搭配软硬件。市场上的主流产品，不仅能够做到更大规模的生产，而且成本也更低，一般是更值得考虑的。如果硬件规格过高或过新，你可能不得不付出更昂贵的成本，性价比反而不好。

对于一些应用类型，比如复杂的查询，即使再昂贵的硬件也无济于事，因为性能更受限制于体系架构，你的体系架构无法充分利用到你的硬件资源。所以笔者建议，在做扩展之前，要先确认你已经很难进行软件架构的优化了，比如，你是否一定要访问这么多数据？是否可以减少访问？是否可以归档和清理数据。

2. 横向扩展

横向扩展也称为水平扩展（scale out），是指通过副本、垂直拆分、水平拆分的方式，把不同的数据放到不同的节点中，我们所说的节点指的是物理部署的MySQL实例。

副本比较好理解，一般是指增加数据库的从库（复制副本），通过分担一部分读的流量到从库上，扩展整个系统的处理能力，也就是我们常说的读写分离，20.3节我们将详述读写分离。

垂直拆分指的是按功能模块划分数据，采用这种方式的多个数据库之间的表结构不一样。比如一个电商网站，可能有库存管理的数据，也有用户相关的数据，它们属于不同的功能，可以拆分到不同的节点。

对于更微观的层级，数据表可以按字段划分数据：大字段和字段访问频率相对于表中其他低很多的字段更适合从表中垂直分拆出去，通过减少I/O资源消耗来达到优化性能的目的。

水平拆分（sharding）指的是将同一个表中的数据进行分片保存到不同的数据库中，一般实现为数据库内的分表设计，这些数据库中的表的结构一般都是相同的。当内存、磁盘空间、磁盘I/O、网络、CPU等资源受限制了，我们就需要拆分，以获得持续稳定的服务能力。数据库节点的处理能力都是有限制的，进行水平拆分主要是需要扩展写的能力。读的能力相对比较容易扩展，我们可以通过缓存、副本等进行扩展，但写的能力，单个主机（节点）可能难以处理，这个时候，你需要进行水平拆分，将数据按照一定的规则，分配到不同的节点。

如果预知数据会有一个爆炸式的增长，那么可以直接从单节点过渡到分片（sharding）结构。而不是经过许多垂直拆分，导致架构变得复杂、难看。

市场中已经有一些数据库中间层产品，或者一些云数据库，宣称能够将数据透明地拆分到不同的节点中，可完美地实现水平扩展，现实使用中，这样的产品可能会有各种限制，而且效率不高。通过自己手动地分片数据，让应用层到具体的节点去获取数据会更高效，因为我们要知道一个原则，应用层才真正地了解数据，知道数据应该如何查询和处理，没有人比你自己更熟悉自己的应用。

进行数据分片设计的一个重要步骤是确定sharding key，常见的有用户ID，比如，我们可以通过对用户ID进行散列，把数据分布到不同的节点中。

大部分应用，使用一些简单的算法进行分库分表的路由，准备10倍到20倍的扩展，1到2年的扩展即可，也就是说，我们把分库分表的逻辑直接写在程序里，使用mod、crc32等散列算法即可。许多人还使用key分段的方法来做容量扩展及负载均衡，笔者不是很推荐采用这种方式，因为可能会导致数据不均衡和热点数据不均衡的问题，调整起来也会很困难。

如果预见到数据和负载会有爆炸式的增长，那么更值得推荐的方式是设计一张路由表，里面存储数据到后端物理节点的路由信息。这样的实现方法有一个好处，我们可以达到一个效果，前端有许多逻辑的DB，而后端是有限的一些物理节点。这种方式更方便我们进行迁移和维护。路由表不一定要保存在数据库里，你也可以设计一个高可用的服务来存储这个路由信息。

如下是设计和维护分片需要注意的一些要点。

- 1) 确保最重要最频繁的查询访问尽可能少的节点，不仅要方便存储数据，还要确保数据读取方便高效。
- 2) 要避免单个节点的资源超过限制，要有足够的监控和预警措施，各分片的数据也可能会不均衡，你需要及时发现这种情况，并进行调整。
- 3) 分片数量要合适，要有利于负载均衡和进行维护，比如修改表结构。分片数量也不能过多，过多可能会导致一些异常问题，比如，一个简单的查询要访问多个分片响应会变得很慢。需要说明的一点是，分片可以让失效的数据保持在某个范围内，但同时分片将导致更多的硬件错误。虽然按照分片的逻辑，把数据拆分到多个节点上很美好，如果可能发生故障，每次只会有一部分服务异常，但节点出问题的概率也大大增加了，而实际生产环境中单台主机宕机的概率是很小的，所以不要过多地分片，你应该在资源可能短缺的情况下，为了扩容考虑的目的而考虑分片。

节点的个数和分片的数量有赖于DBA和研发、产品等团队一起讨论确定，互联网应用数据的容量规划在很多情况下都不是很清晰，这是现实，所以应尽可能地沟通信息，掌握足够的数据是有必要的。

- 4) 分片之后，表之间的连接会比较困难，你需要尽量避免连接，或者采用更合适的方案来组合数据，比如，我们可以设定多个sharding key，使用不同的sharding key冗余多份数据，以做到减少连接，更高效地访问数据。我们也可以使用缓存、统计表等手段来减少连接。
- 5) 热点数据可能会导致性能问题，可能需要调整sharding key或算法，将数据切割为更小的分片，如果对于复制延时要求不高，也可以利用从节点来扩展读的能力。
- 6) 应确保节点的快速恢复，比如对于单点故障，你可以自动路由到正常的节点，或者是提供开关降级服务，比如提供一个只读的节点，用于保障部分功能可用。

注意，可扩展并不是要构建一个完美的扩展架构，而是应用程序真的需要进行扩展时，才考虑扩展，我们要预先规划，即使在以后规模变得很大的时候，系统也仍然能够提供合格的服务。

20.3 读写分离

读写分离指的是，通过增加一些节点，扩展读的能力。这些节点可以是主节点的全部内容的副本或部分内容的副本，也可以是缓存产品。读写分离一般配合负载均衡产品一起使用。

对于读多写少（非更新查询为主）的负载，特别适合做读写分离。需要留意的是，要保证用户感知到自己所做的变更有效即可，用户在很多情况下并不需要知道其他用户的改变。如果用户对于数据的一致性要求在某个时刻很高，那么这部分数据，建议不要使用读写分离，MySQL的复制可能会出现延时，无法满足业务的需要。你可以采取变通的方式，比如，在用户修改了内容之后，临时强制用户访问主节点，以获取一致性的数据，在过一段时间之后，再让用户访问副本的数据，一般在此时，副本的数据已经同步到最新状态了。

读写分离往往和负载均衡技术配合使用。负载均衡软硬件产品有F5、HAProxy及一些自己设计的MySQL Proxy代理等，负载均衡可以更高效地利用硬件，你可以设置权重，分配更多的流量给性能更好的机器，负载均衡产品一般还有故障检测、自动冗余切换等功能，这可以大大提高机器的可用性。

读写分离技术的一个难点在于延时的影响，你需要有一个手段来确保你没有读取到太旧的数据，写操作和一些不能容忍延时的查询，需要指向主库。对于数据延时敏感度高的数据，你需要定义延时的阈值，通过自动或手工的方式处理延时数据对于用户体验的影响。

你可以通过监控SHOW SLAVE STATUS里的输出Seconds_behind_master的方式判断是否有延时，但这种方式不太可靠。监控复制滞后（replication lag）更稳健的方式是通过心跳表的方式。

我们很难确保MySQL的延时，因为网络波动、复制异常、性能问题等都可能导致复制中断，而往往需要人工来进行干预，毕竟有能力开发专用的Proxy代理的公司很少，所以，不建议使用读写分离，采用读写分离一般是基于一个前提，主库已经出现了读瓶颈，如果出现了读瓶颈，那么使用缓存一般是更有效、更成熟的解决方案。

由于没有好的读写分离的方案，如果你一定要使用读写分离，那么推荐应用程序自身实现读写分离，把读的流量指向负载均衡产品或Proxy代理。

20.4 切勿过度设计

过度设计，指的是你的设计过度地考虑了未来的一些需求，或者根本就是想象出来的需求。

现实中，我们对于数据的量化往往做得还不够。我们要设计一个可扩展的架构，但由于没有进行足够的量化分析，往往导致了过度设计，比如采购了过多的硬件设备。可扩展的目标的设定很重要，你要根据业务的发展和自己业务的特点，定义可扩展的目标，在架构初期，可扩展往往并不是那么重要，尽快实现简单稳健的方案是最主要的，如果你留有一定的余量，可扩展性在大部分公司一般是不成为问题的。你需要靠经验和数据设定这个余量，因为一旦扩展不下去了，成本就会变得很高，可能还会需要重构。

你需要衡量是否要预先进行分片，现实中，往往会出现的一种情况是，研发人员不熟悉硬件性能极限和业务增长，扩展太多，比如分库太多，导致整个方案变得复杂和成本高昂，所以，如果需要进行分片，你一定要确认自己必须得这么做。如果你不熟悉硬件，那么你应该去请教对硬件熟悉的人员。如果研发人员不熟悉硬件，那么硬件维护人员和软件架构人员共同探讨是有必要的。

可扩展性是建立在数据规划的基础上的，所以你要熟悉你的业务，熟悉业务需要存取的数据的行为，我们在架构的初期，就应该确定数据的规模和访问的特点，可以考虑的一些因素具体如下。

- 各种操作的频率，比如读（SELECT）的次数和INSERT事务的次数。
- 峰值时刻的事务数。
- 查询是简单的还是复杂的，各自的比例如何。
- 并发连接数。
- 数据量，可以预估1~2年后的数据量。
- 数据的重要程度。
- 数据保留和清理的策略。
- 数据分片策略。

以上因素大部分是从运维的角度来考虑的，数据架构的时候，你还需要细化到具体的数据和具体的访问数据的行为，比如，你需要特别关注访问量最大、频率最高的一些查询，优先对这部分数据进行库表设计和应用程序优化。可以说，好的数据库是设计出来的，而不是调优出来的。

需要明确的是，互联网公司不同于传统企业，传统企业的业务类型比较固定，一般不会做变动，熟悉业务的系统架构人员、开发人员可以比较准确地进行性能规划，合理安排扩容的资源。但互联网公司业务变化大，许多时候是追求迭代发布，而不再要求性能规划、数据规划的精准度。我们应该确保的是，系统在一定时期内有足够的伸缩性，做好监控，确保系统有足够的性能余量可以支撑短期的负荷上升。传统行业的规划周期太长，而互联网行业规划个半年、1年也许就足够了，一般不用考虑2~3年之后的长期的扩展需求。

一般来说，我们已知的一些传统的性能规划的理论，并不适合于互联网公司不断变化和发展的应用，利用简单的基准测试来进行性能规划也是不切实际的，因为很难模拟真实的负载。比如我们经过基准测试，可以证明在一个新的硬件架构上，可以支撑起比目前的负载高10倍的访问量。但实际上，如果现实中真的有了10倍的访问量的时候，许多情况就已经发生了变化，如流量、用户、数据、应用复杂度、关联数据的交互、热点数据大小，甚至应用的核心功能也已经发生了大的变化。所以，我们需要做到的是确认我们的系统能够满足未来一段时间的产品发展，有足够的余量，而不用去规划一个长期的目标，当然，这需要足够的数据收集和量化工作，我们做不到完全精准，但是你不能有数量级别的差异。

20.5 可扩展的方法

确保可扩展有许多方法，如果具体到某一个产品，那么也要有各种各样的手段，这里主要从更宽泛的角度去讨论一下可扩展的方法，比如静态内容、动态内容、网络应该如何做到可扩展，以及可扩展的一个重要手段：解耦。了解其他环节、其他领域的知识，将有助于我们确定调优的方向，合理地进行数据库的优化。

20.5.1 优化静态内容、动态内容

首先要分离静态内容和动态内容，分离了静态内容和动态内容之后，我们才可以分别进行优化，选择更适合的应用服务器，比如Nginx更适合静态文件，而Apache相对来说更适合动态内容。某些静态文件还可以压缩传输。

完全静态化是不现实的，往往需要通过模板的方式，我们有必要了解我们所维护的项目的静态化策略。不同的应用服务器适合处理不同的内容，尤其对于海量流量的应用，用更合适的产品来处理特定的内容，会更有规模效应。

静态内容优化的主要的技术是CDN技术，CDN的目的是将网站的内容发布到最接近用户的网络位置，使用户可以就近取得所需的内容。

动态内容优化的一些方法和指引规则具体如下。

·计算复用：计算复用指的是，通过一些编程技巧，可以重复利用之前的计算结果，加快执行效率。计算复用并不适合应用于复杂的算法操作，在日常的许多编程中，都可能碰到，如果有一些操作频繁的执行，又和上下文无关，那么可能是需要考虑计算复用的。

·使用缓存：缓存系统缓存了程序处理的结果，它可以减少对后端的调用。

·同样的内容不要产生两次：因为数据是可以被缓存的，无论缓存在服务器还是客户端，我们都不需要重复产生相同的内容，因为这样会浪费系统资源

·仅在数据发生改变时，重新生成内容：有时我们想生成一些静态文件，提高访问效率，相对于重新生成所有的文件，仅重新生成数据发生了改变的页面，是成本更低的方式。

·将系统切割为更小的组件，分离频繁变更的部分和不经常变动的部分。

·减少对数据库的调用：相对于应用服务器，数据库的可扩展性更低，减少对数据库的调用，可以让数据库没那么可能成为整个系统的瓶颈所在。

·对于缓存产品，如Memcached，需要留意缓存策略，比如，超时的设置或设置得过小，会导致数据库的压力。更有效率的做法是，在数据内容发生改变的时候，才通知缓存失效。

·对于一些变化非常频繁的内容，几乎没有缓存的必要，这个时候有必要和产品经理、用户体验设计师、前端和后端的研发人员一起来考虑问题，是否可以减少变化的可能性。

20.5.2 网络优化

关于网络优化的参数，这里就不展开讲了，笔者很少调优网络相关的内核参数。下面将叙述一些网络相关的注意事项，具体如下。

·我们需要了解数据流，清楚我们的网络架构，这样有助于我们进行分析，在哪些环节可能存在网络问题，哪些环节可以优化。比如用户最开始发起访问，一般要有DNS查询的环节，那么我们是否可以让用户选择最近的DNS服务器呢？我们是否可以调度用户访问最近机房的服务器呢？我们是否需要配置一个反向代理来加速用户访问呢？

·应用服务器处出现网络瓶颈的可能性远大于数据库，数据库一般网络流量很小。

·现实中，优化网络的行为很少，这个主要是因为绝大部分项目在还远远没有到达网络瓶颈的时候就暴露出架构的问题了。

·跨IDC的网络质量不能和内网质量相比，对于跨IDC的网络，两个节点之间来回往往需要几十毫秒，节点之间的各种设备（路由器、交换机等）都可能影响到网络质量，运维比研发人员要更加意识到跨IDC网络质量对于整体系统的影响。

20.5.3 解耦

解耦是确保可扩展的架构的最重要的技术之一。

许多知名的网站和服务，采取的都是一种松散耦合的服务导向的模型，比如Twitter、Amazon等。这种松散耦合的服务，可以尽快发布新特性。小型的团队可以自己制定决策，发布面向用户的变更，而不依赖于其他团队。

解耦的方法具体如下。

1) 异步。

异步指的是，有些操作并不需要马上去做，而是可以延迟到以后再做，因为这并不会影响用户的体验。“异步”的字面意思可能会导致混淆，“异步”并不是说一定要把工作推迟到以后去完成（尽管这可能会发生），异步技术一般使用了队列，实际上队列往往不会被积压，它们会处理得很快，它的本质目的是为了解耦，因为有些操作并不需要等待另外一些操作，可以“异步”地、并发地进行。

2) 把业务逻辑分解为更小的部分，分而治之。隔离那些可以异步操作的部分。

通过把系统分解为更小的部分，我们可以做到如下几点。

·简化问题，原来一个复杂的逻辑，我们可以将其分解为一些更简单的问题。

·故障隔离，针对更小的系统，我们可以针对性地设计处理策略，某个子系统的故障不会影响到其他的子系统，其他子系统仍然可以正常地运行。

·分解我们的方法、策略和实现，这个比较好理解，分解为更小的部分，我们的复杂的方法、策略和实现就变成了更小的问题。

·简化我们的设计，我们把复杂的问题分解为简单的问题，那么设计也会变得相对简单得多了。

·更好地建立性能模型，因为能够更准确地衡量影响性能的因素，从而可以简化容量规划。容量规划其实不是一件容易的事情，你首先得有一个性能模型，如果影响性能的因素有很多，那么这个性能模型会很复杂，难以建立，或者不太准确；现在我们分解了问题，因此我们可以建立一系列简单的模型，然后就可以综合

这些简单的性能模型得到我们最终的性能模型。

3) 使用消息队列。

这个和上面所说的异步，是结合在一起的，利用消息队列可以很好地异步处理数据传送和存储，我们把需要完成的工作的信息用队列进行传送，这样就可以实现异步幕后处理队列了。也就是说，我不想现在就做某件事情，而是告诉其他人去做这件事情，这样可以加快我做事的效率。这也是我们上面所说的异步。

互联网应用大量地使用了消息队列。消息队列不仅被用于系统内部组件之间的通信，同时也被用于系统跟其他服务之间的交互。当你频繁地向数据库中插入数据、频繁地向搜索引擎提交数据时，就可以采取消息队列来异步插入。另外，还可以将较慢的处理逻辑、有并发数量限制的处理逻辑，通过消息队列放在后台进行处理，例如FLV视频转换、发送手机短信、发送电子邮件等。

消息队列的使用可以增加系统的可扩展性、灵活性和用户体验。非基于消息队列的系统，它的运行速度将取决于系统中最慢的组件的速度（也就是短板效应）。而基于消息队列，可以将系统中的各个组件解除耦合，这样系统就不再受到最慢组件的束缚，各组件之间可以异步运行，从而可以以更快的速度完成各自的工作。除此之外消息队列还可以抑制性能波峰的产生，在瞬时业务高峰产生时可保持性能曲线的平滑。

消息队列有许多解决方案，有许多正在广泛使用的产品，许多互联网公司基于自身的业务特点，设计了满足内部需求的消息队列。消息队列可能会有单点，所以，你也需要确认你的架构已经解决了单点问题。

20.6 使用云数据库

云计算，或者说云平台，更多的是属于水平扩展的范畴，系统建立在更小的虚拟系统之上，所以，一开始你不需要购买庞大的系统，可以从很小的购买预算开始，你可以以更小的粒度进行扩展，而传统的方式可能需要添加整整一台机器。由于更具有弹性，所以一开始可以从很小的规模开始，因此不像传统行业那样，需要精细的容量规划。云服务商有时还可以提供高级的特性，可以按负载动态扩容和释放（需要配合可靠的监控）节点。

由于可以比较方便地增加和减少节点，性能优化很快就可以见效，因为优化了性能，就只需要更少的节点，而对于传统行业来说，你投资了设备，那么即使你优化了性能，这些设备也是闲置的，不能立马起到降低成本的作用。

本质上来说，系统是不是可扩展的，很大程度上并不取决于使用的产品，而在于你的软件架构，所以在云上，我们仍然可以使用传统数据库，而依然保持良好的扩展性。而这也属于比较通用的做法。国内外的云服务商一般都提供了MySQL云。

如果不考虑维护成本，我们对比国内的IDC托管主机和云计算的成本，可以得出结论，云平台的成本比自己租用主机的费用高出不少，一些初创小公司在项目初期，为了节省维护和部署的成本，可以使用商业公司的云服务，而在规模扩大之后，你就需要考虑性价比是否合适，一般来说，对于大批量的节点购买，云服务商会提供更大的折扣。

使用云服务要考虑的一个问题是，针对云服务的调优（包括数据库）会变得很困难。由于多个实例可能位于同一台主机或同时申请使用了共享的资源，这种多租户的环境，可能会导致一些异常，比如虽然你的程序没有问题，但你的服务可能会受到其他租户的影响，调优诊断也会变得很困难，因为可能你所使用的资源难以监控，传统的许多调优方式在云中可能不太适用。随着云服务的成熟和普遍使用，随着监控和诊断平台的完善，云服务的调优会变得越来越简单。

一些云服务的升级和维护，也可能导致你的服务出现异常。云服务商一般提供99.5%的可用性保证，而基于传统的方式托管机器，运维得当，4个9（99.99%）的可用性也是可以轻易达到的。

一般售卖的云主机，主要依据内存的大小进行定价，其他资源，比如CPU，随着内存的增加，也会得到相应的扩展。

一些公司可能部分服务器自己托管在IDC，部分使用云服务，这样一个比较折中的想法，因为对于突发的负荷，可以使用云服务的弹性扩容能力来处理。



小结 本章为读者介绍了可扩展架构的一些知识，部分内容严格来说，并不属于数据库的范畴，但它们和数据库的可扩展性息息相关，而且，我认为一个好的DBA，应该熟悉这方面的内容。DBA应该有容量规划的能力，知道常用的拆分数据的手段，审慎使用读写分离的技术，并且能够和研发人员一起探讨可扩展的一些方法。可扩展要考虑的因素很多，在进行架构调优的过程中，你必须综合考虑性能和成本，做一些取舍。

关于云平台的使用，笔者的经验不多，读者如果需要使用云服务，可以考虑亚马逊、阿里云等服务商。

第21章 高可用性

本章将为读者介绍单点故障的处理策略，以及单点故障最主流的解决方案：MySQL数据库切换。

21.1 概述

可用性定义为系统保持正常运行时间的百分比，高可用可以理解为系统可用时间的百分比很高，也就是说服务可用的时间很高，数据没有丢失，也没有其他异常。比如，一个未预热的数据库突然承受大流量的冲击，最开始的阶段，响应时间可能会很长，这个时候很难说服务可用。

我们一般用百分比来表示可用性，举例如下。

·99.999%的可用性表示全年5分钟故障时间。

·99.99%的可用性表示全年1小时故障时间。

·99.9%表示全年8小时故障时间。

提高可用性的成本可能会很高，其策略也可能会很复杂，因此我们需要尽可能地平衡“停机”（downtime）成本和减少“停机”时间的成本，衡量系统失败的概率和所造成的损失，对比投入的成本，不要过度设计。

我们可以分解系统的关键部分和非关键部分，这样可以让你更好地设计可用性策略，因为提高一个小系统的可用性会更容易。有时我们为了架构简单，不得不保留“单点”，那么我们可以使用更可靠的硬件和主机来尽量减少风险。

一般影响数据库服务可用性的主要因素是硬件、网络故障或性能问题、软件Bug等。如果使用了读写分离的架构，还可能因为复制延时或复制错误导致从库的数据滞后，数据不一致，这也会对可用性造成影响。生产人员的误操作，比如误删除了数据库文件和数据库表，都可能导致数据库服务的可用性下降。针对如上的影响因素，我们可以制定如下的一些措施，以提高可用性。

1) 上线或升级新的软硬件之前，充分做好测试验证。

2) 制定合理的备份策略，并定期恢复验证。

3) 严格按照流程规范操作数据库，隔离生产环境和测试、开发环境。

4) 架构力求简单、可靠，因为复杂的策略可能导致维护和处理问题变得困难，也很难实现高可用策略，记住，解决复杂问题的最好方法就是让复杂的问题不再出现。

5) 做好监控，尽量在问题爆发之前就能预警。

6) 应回顾和分析故障事件，尽量避免故障的再次发生。

高可用不仅仅是技术的问题，也是管理的问题，有正确的流程、规范和步骤，有良好的文档，有训练有素的维护人员，才可以减少故障发生的概率，并能在故障发生后快速恢复。

21.2 单点故障

单点故障（SPOF）指某个系统的一部分，如果它停止工作了，将会导致整个系统停止工作。在我们的架构设计中，要尽量避免单点故障。

要避免单点故障，我们首先应找到可能导致整个系统失效的关键的组件，综合评估，在满足我们可用性的要求下，应该如何避免单点故障，或者减少单点故障爆发的可能性。

一般我们靠增加冗余的方式来解决单点故障，冗余的级别和方式不一。从设备的角度，我们可以对主机的单个组件进行冗余，比如使用多个网卡，我们也可以对整个主机所有的关键部件进行冗余，在更高的级别上，我们可以对整个主机进行冗余，或者对整个IDC机房进行冗余。从组织管理的角度，我们还可以对维护数据库的人员进行冗余。MySQL的主从架构本质上也是增加一个冗余的从节点来提高可用性。

如下将详细介绍一些常用的解决单点故障的技术。

1) 使用负载均衡软硬件设备，比如对于一组读库，我们可以在前端放置一个负载均衡设备，以解决后端某个从库异常的故障，你可能还需要考虑负载均衡设备自身的高可用性。

2) 使用共享存储、网络文件系统、分布式文件系统或复制的磁盘（DRBD）。

传统的数据库产品，如Oracle RAC使用的是基于SAN的共享存储存放数据，数据库的多个实例并发访问共享存储存取数据，应用通过配置在数据库主机上的虚拟IP访问数据库，如果某个数据库主机宕机，其他数据库实例接管虚拟IP，那么应用仍然可以访问到数据库。

MySQL官方介绍了一个实现网络RAID的方案DRBD。也有人使用网络文件系统NFS或分布式文件系统存储共享的数据库文件。

使用共享存储是比较传统的做法，由于成本比较高，且共享存储自身可能也会成为单点，因此互联网架构中很少使用这类方案，有些人为了确保主库数据的安全性，把二进制日志放到共享存储中，这也是一种可以接受的做法。

使用网络RAID，即DRBD虽然是可行的，但现实中用得并不多，主要原因在于目前的SSD已经足够快了，DRBD自身会成为整个系统的瓶颈，而且会导致主机的浪费，因为只有一半的主机可用。因此作为折中的方案，可以只用DRBD复制二进制日志。

笔者没有使用过NFS存储共享的数据库文件，网络文件系统难以实现高吞吐，NFS更适用的场景是存放一些共享的备份文件。有些人选择使用分布式文件系统来存放数据库文件，由于分布式文件系统本身的复杂性，你需要考虑它的维护成本及团队人员的技能等因素，如果传统的方法可以存放数据文件，那么不建议使用这么一个“笨重”的方案。

3) 基于主从复制的数据库切换。

目前MySQL使用最多的高可用方案是MySQL数据库主从切换，也就是说，基于主从复制的冗余。通过对主库增加一个或多个副本（备库），在发生故障的情况下，把生产流量切换到副本上，以确保服务的正常运行。随着主机性能的发展，基于主机之间的高可用是主流也是趋势。21.3节会详细基于主从复制的数据库切换。

21.3 MySQL数据库切换

基于数据库复制架构的切换是目前MySQL高可用的主流解决方案。我们把数据库成双成对地设置成主从架构，应用平时只访问主库，如果主库宕机了，从库可以替补使用，且满足一定的条件，那么我们可以把应用的流量切换到从库，使服务仍然可用。

由于数据库切换依赖的是MySQL的主从复制架构，所以你需要深刻了解MySQL的复制原理和机制，确保MySQL的同步一直是可用的。你需要尽可能地保证数据已经同步到了从库，以免丢失数据。

数据库可以配置成主从架构，也可以配置成主主架构。我们建议使用主从架构，这是最稳健、最可靠的方案。有些人把数据库配置成主主架构的原因是，他们认为这样做可以更便于切换及回切。配置成主主架构的时候，你需要小心处理主键冲突等复制问题，在从库上进行操作时需要非常小心，因为错误的操作也会同步到主库。配置成主主架构只是为了方便切换，现实中，仍然需要确保仅有一个主库提供服务，另一个节点可作为备用。

除了单点故障，有时我们也可以为了其他目的进行切换，比如在大表中修改表结构，为了避免影响业务，临时把所有流量切换到从库。这种情况下，配置成主主架构会更方便。

为了简化容量管理，以确保切换数据库流量之后，数据库主机能够正常提供服务，应该确保主备机器的软硬件配置尽量一致。由于数据库从库的数据一般并未“预热”，热点数据也没有被加载到内存，所以在进行流量切换的初始时刻，可能会难以接受其性能，你可以预先运行一些SQL预热数据。

对于写入事务比较多的业务，在发生故障的情况下进行主从切换，可能会丢失数据和导致主从不一致，一般情况下，互联网业务的可用性会高于数据一致性，丢失很少的事务是可以接受的。一些数据也是允许丢失的，比如丢失一些评论是可以接受的，如果需要绝对的不能丢失数据，那么你的方案的实现成本会很高，比如为了确保不丢失主库的日志，你可能需要共享存储来存储主库的日志，还可能需要使用全同步或半同步的技术确保数据的变更已经被传送到了从库。

对于数据库的切换，我们有如下的一些方式。

1) 通过修改程序的配置文件实现切换。程序配置文件里有数据库的路由信息，我们可以修改程序的配置文件实现数据库流量的切换，在大多数情况下，我们需要重启应用。比如JAVA服务，默认配置下，我们需要重新启动应用服务。在服务非常多的情况下，也有把数据库配置信息存储在数据库中的。

2) 修改内网DNS。

我们可以在生产环境中配置内网DNS，通过修改内网DNS指向的数据库服务器的IP，实现主库在故障情况下的切换，这种方式，往往也需要重启应用服务。由于内网DNS可能不归属于DBA团队掌控，DNS服务器的维护和高可用也需要成本，而且更改内网DNS也需要时间，所以这种方式用得比较少。

3) 修改主机的hosts文件。

/etc/hosts里可以配置与数据库服务器的域名对应的IP，但是还不够理想。而且在有很多应用服务器的时候，维护一份统一的hosts文件的成本也会比较高。

4) 一些能够实现高可用的工具集，如MHA、MMM，它们用于监控数据库节点的存活状况，通过修改配置文件或漂移主库IP的方式来实现数据库的高可用。

MMM通过漂移虚拟IP的方式处理单点故障，但许多生产实践证明，其作为一套自动切换方案并不是很可靠，如果需要使用，建议只使用手动切换的功能。MHA是Peri编写的一套MySQL故障切换工具，支持通过修改全局配置和漂移虚拟IP两种方式处理单点故障，已经在许多生产环境中得到了验证，是值得考虑的方案。

你也可以自己编写脚本监控数据库节点的可用性，漂移虚拟机IP实现切换，需要留意的是，漂移IP的方式存在一个缺陷，其严重依赖硬件的可靠性，需要主机、网络设备的配合工作。在生产环境中，可能会因为网络硬件的原因导致虚拟IP不能正常漂移。

5) 对于大规模的数据库集群，需要更智能地处理单点切换，应该尽量不依赖自己无法控制的因素，我们可以使用独立的Proxy代理的方式实现单点切换。所有的流量都经过Proxy，Proxy智能地处理后端的数据库主节点宕机故障，需要留意的是，你还需要处理好Proxy自身的高可用性。实现Proxy的成本很高，一些互联网公司已经有自己成熟的数据库Proxy。理论上，Proxy是可以代理本地IDC的流量的，也可以代理其他IDC的数据库流量，但由于网络延时和安全的考虑，一般建议仅代理本地IDC的流量。如果需要配置跨IDC的数据库切换，更可靠的方案是，在应用层切换流量，也就是说，让用户去访问正常IDC的应用服务器。

6) 通过客户端、框架配合实现单点切换，相对于使用Proxy的方式，这种方式更轻量级。

21.4 跨IDC同步

有时我们需要部署IDC级别的冗余，在另一个IDC中部署数据库的从库，由于网络层的不稳定，你很难实现很高的可用性，除非你对数据的延时和数据的一致性要求不高。你需要意识到，距离越远，网络越不可靠；中间的环节越多，网络越不可靠，所以尽量不要进行跨越数据中心的实时操作。如果部署了跨IDC的数据库访问，比如部署了读写分离的架构，在一个IDC中集中地处理所有的写请求，把读请求分担到各个IDC，那么你需要在应用层友好地处理网络异常，或者复制问题导致的复制延时问题，如果有比较多的远程写入，那么还需要处理网络问题导致的写入失败。

不仅仅是主从同步，只要距离足够远，网络质量就难以得到保证，就需要留意同步对应用的影响，你可能需要尽可能地减少节点之间的数据交互，及时调度用户访问其他节点，甚至使用专用的高质量网络。



小结 本章主要介绍了MySQL保障高可用的方案，MySQL主从复制几乎是所有高可用方案的基础。读者应该熟悉MySQL的复制原理。对于跨IDC的数据同步，许多人可能没有意识到它的局限性，DBA应该尽量避免在生产环境中出现这种架构。

第22章 其他产品的选择

本章将为读者介绍其他的数据库产品，主要是NoSQL产品的选择。读者在熟悉MySQL之外，也应该了解其他的数据库产品。本章的目的是给读者一个引导，如何选择一些NoSQL产品，而不是推介或否定某些NoSQL产品，读者应该自己研究最新的稳定版本的NoSQL产品，确定是否符合生产环境的需要。在介绍NoSQL产品之前，有必要先了解一下列式数据库产品。

22.1 列式数据库产品

数据的存储可以简单地理解为，行式数据库，即把每行的数据串起来存储在数据库内，而列式数据库则是把每列的数据串起来存储在数据库内，行式数据库一般是不压缩的，而列式数据库，由于同一个列的数据被存储在一起，因此往往有重复值，数据可以大大压缩。

随着数据规模的扩大，MySQL在存储和分析海量数据方面变得越来越力不从心，虽然我们可以把数据切分到多个MySQL节点，但并不是每个业务的数据都适合分布式的MySQL存储的。

本质上，存储的方式应该是有利于查询和分析的。传统的关系数据库产品一般是以行的方式来存储数据，更适合于处理小批量的数据；而列式数据库则是以列的方式来存储数据的，更适合大批量的数据处理和查询。也就是说，侧重于OLTP的系统更适合使用行式数据库，对于OLTP联机事务处理系统来说，一般是随机读写，一次读取一小部分的数据，数据块一般比较小，几KB到十几KB不等，MySQL的一个块是16KB，容易一次I/O读取出来。由于行式数据库数据是按行存储的，每列数据分布在多块内，所以如果你要统计某列或修改某列，则需要把整行数据读取出来，读取磁盘的次数会比列式数据库多得多，因为列式数据库把一个列的数据都压缩存储在相邻的数据块之内，所以，侧重于OLAP的系统更适合于列式数据库。

选择使用行式数据库还是列式数据库，你需要在这中间寻求一个平衡，由于成熟的列式数据库产品一般是商业产品，比如Sybase IQ，价格比较昂贵。互联网公司很少使用列式数据库产品，更多的是依赖大规模的分布式数据处理和分析系统。

目前常用的基于MySQL的开源列式数据库产品为Infobright，Infobright是业界领先的成熟产品，其免费版本不能修改数据，只能使用“LOAD DATA INFILE”的方式导入数据，不支持INSERT、UPDATE和DELETE。收费版本比较昂贵，如果成本可以接受，作为OLAP系统的后端数据库将是一个很好的选择。

22.2 NoSQL产品的选择

22.2.1 概述

NoSQL产品发展得很快，这些年又不断有新的产品出现，但往往是昙花一现，一些上线的NoSQL产品由于不成熟，经常难于维护，甚至数据丢失，究其原因，往往是选择的NoSQL产品未充分考虑到运维的成本，由于工作性质的不同，程序开发者往往会更加重视功能需要，而忽视了数据库产品的一些基本指标，作为一个合格的软件架构师也应该关注产品的运维指标和总体拥有成本，能够从系统资源和应用访问的双重视角去考虑问题，以选择合适的数据库产品。

本节将主要阐述选择NoSQL产品的一些思路和方法学，并不会进行各种数据库产品的详细对比，大家可以按照自己的理解和侧重点编写程序或使用开源工具进行对比，对于NoSQL产品自带的性能评测工具，建议大家保持警惕，一般我们会假定软件厂商提供的性能监控工具是正确可靠的，但实际上，这些工具的输出可能是不准确的、不可信的。

选择NoSQL数据库产品需要考虑到许多因素，需要权衡取舍，但无论出于何种考虑，大规模的部署必然要求满足运维的一些基本指标，比如稳定性和可维护性。

一些有发展前景的NoSQL产品改进很快，所以可能会随着时间的演变，性能和稳定性都得到长足的发展，一些指标会得到很大的改善，你应该保持对市场上应用最广泛的一些产品的关注，了解其最新的发展。

本节将主要以最流行的MongoDB和Redis为例，讲述如何选择NoSQL产品，MongoDB和Redis都有良好的发展势头，各种功能特性得到不断完善，本书所讲述的MongoDB和Redis主要是基于2013年~2014年的版本，由于MongoDB和Redis发展得很快，因此一些论据、论点必然会过时，希望读者留意最新的版本，希望读者把本书对于这两个产品的说明仅仅作为了解NoSQL产品的方式，而不是真实产品的表现。

许多NoSQL产品是为了存取特定的数据而设计的，将它们都列在一起进行对比，没有多大意义，如果你需要选择和对比NoSQL产品，实际的比对建议限制在2种（或者最多3种）产品的对比上，如果同时进行对比的产品有很多，那么自身的结论也难有说服力。

选择NoSQL产品需要重点考虑的因素如下。

1. 灾难恢复性

灾难恢复性，即故障后数据的恢复性，我们可以通过模拟程序崩溃或主机宕机来验证灾难恢复性。可考虑如下三种场景：进程崩溃、操作系统崩溃和存储故障。

在崩溃后，测试验证能否正常启动数据库服务、数据文件是否损坏、数据是否完整、数据丢失了多少等？在存储异常的情况下，验证服务的稳定性，验证是否能及时发现硬件异常。

2. 可维护性

可维护性，顾名思义，是指对产品维护的难易程度。

3. 可靠性

衡量数据库系统操作的成功性。我们需确保操作结果符合系统设计的预期。

4. 高可用性

高可用性可以定义为系统保持正常运行时间的百分比。即，

$A_u = \text{up time} / \text{total time}$ 或

$A_u = (\text{total time} - \text{down time}) / \text{total time}$ 。

确定可容忍的停机时间是可行的。从这一点上来看，所需的可用性可以很容易地计算出来。

高可用往往不是单个数据库产品就可以决定的，这更多的是一个架构问题。对于MySQL和诸多其他开源数据库产品，都没有很通行的开源的高可用解决方案。各大公司在发展到一定规模之后，都采用了一些高可用的技术，目前数据库的高可用技术主要是从两个方向着手，一个方向是开发中间层，代理所有访问；另一个方向是应用程序框架/客户端实现高可用。

5. 高性能

我们衡量的高性能，应该是高访问压力下的高性能。性能一般可用响应时间来度量。

6. 可扩展性

可扩展性一般是指我们可以用更多的节点来提高吞吐率，同时性能不会下降到不可接受的范围，这更多地属于架构的范畴。

7. 资源利用

对于数据库来说，我们主要关注对I/O资源、CPU资源和内存资源的利用。

8. 功能特性实现

这里将简单列举一些示例，包括但不限于以下的一些功能和特性。

(1) dynamic schema

动态schema，也就是NoSQL产品的schema-less特性。MySQL在这方面不太具有优势，MariaDB在这方面有一些改进，据说MariaDB 10会有更多的改进。NoSQL产品在这方面往往更具优势。现实生产中，MySQL往往通过将记录存储为Key-Value类型来实现dynamic schema。

(2) automatic sharding

一些NoSQL数据库或多或少都实现了此类特性，传统关系型数据库不太容易实现。但NoSQL产品的auto-sharding特性，仍有待于大规模生产的测试验证。

(3) 锁的实现

不同的数据库产品对于锁的实现也不尽相同，锁的设计对于数据库的并发访问有很大影响，在这方面，传统数据库对于锁的实现较为复杂和高效，而NoSQL锁的实现往往比较简单，且粒度更大。

(4) 文件管理

本节主要关注数据文件的空间分配。

(5) 支持JOIN等复杂查询的能力

一般NoSQL产品实现了简单的Key-Value或一些改进的数据结构，但一般不会实现传统数据库“表”之间的“JOIN”。

9. 数据结构

一般传统的关系型数据库更适合存储一些结构化的数据，而NoSQL产品更适合存储一些非结构化的数据。在数据量小的时候，一般使用传统数据库就够了，传统数据库往往比较通用，但在数据规模很大的时候，许多公司都采用了SQL和NoSQL数据库并存的策略。

各种功能特性的选择，往往不可兼得，所以需要权衡取舍，比如，如果QPS或延时都很稳定，基本上没有变化，那么我们可以不用太关注使用资源的少许差异。如果数据安全性要求不高，允许丢失数据，那么我们可以采取最大性能的模式。

以下将详述上面的各个因素。

22.2.2 灾难恢复性

下面我们来对比下MySQL和一些NoSQL产品的灾难恢复机制。以MySQL和MongoDB为例。

(1) MySQL的数据库灾难恢复机制。

1) 靠预写日志（Write-ahead logging）来保障持久性。

也就是说，数据文件不会马上写入脏数据，而是先写日志，以后再批量刷新脏数据到数据文件以提高吞吐率。我们把这个日志叫作事务日志，数据库崩溃之后，可以找到事务日志的某个时间点，把这个时间点之后的日志运行一遍，然后回滚那些没有提交的日志。事务日志都是顺序写入的，可以设置参数来调整写日志的频率。

2) MySQL还有一个数据结构double write buffer（双写缓冲），可用于增强灾难恢复性。

数据文件随机读写，InnoDB最小的写入单元是16KB，如果由于故障或Bug只写了部分块，那么可能会导致坏块（data corruption）。InnoDB使用double write buffer来确保数据安全，避免块损坏。double write buffer是InnoDB表空间的一个特殊的区域，顺序写入。当InnoDB刷新数据时（从InnoDB缓冲区到磁盘），首先写入double write buffer，然后写入实际的数据文件。这样便可确保所有写操作的原子性和持久性。崩溃重启后，InnoDB将检查每个块（page）的校验和。判断是否坏块，如果写入double write buffer的是坏块，那么显然没有写入实际的数据文件，那么就用实际数据文件的块来恢复double write buffer，如果写入了double write buffer，但是数据文件写的是坏块，那么就用double write buffer的块来重写数据文件。

(2) NoSQL产品的灾难恢复机制。

常见的NoSQL产品的存储引擎有两类实现。一种是Memory-Mapped存储引擎，另一种是日志型的存储模型。

1) 许多NoSQL产品都采用了Memory-Mapped存储引擎，如Tokyo Tyrant、BDB、MongoDB等，MMAP方式是靠操作系统将数据刷新到磁盘的，用的是操作系统的虚拟内存管理策略。在某些场合可以提高吞吐率，毕竟写内存比写磁盘要快。主要弊端是数据库无法控制数据写入磁盘的顺序，这样就不可能使用预写日志（Write-ahead logging）来保障持久性。发生崩溃的情况下，数据文件可能被破坏，需要进行修复。

数据被破坏的过程大致如下。

Linux操作系统有定时刷新数据的机制，一般是几秒的间隔，I/O瓶颈严重时，还会自动调整。操作系统还有一个30秒的数据过期限制。具体的机制比较复杂，有兴趣的读者可以自己研究下。

生产环境中的大部分业务都有写入数据的场景，在宕机时刻，数据正在刷新的概率很高，此时的后果往往就是损坏数据。这种情况无法避免，它是由其实现机制决定的，宕机时刻，可能只刷新了部分数据，而刷新的这部分数据并不是按照数据库所期望的顺序写入的，这将破坏数据。如果每次写入数据时都强制刷新到磁盘，那样将是比较安全的，但这样做不是设计这类产品的本意，会严重影响效率。我们还可以用昂贵的小机来尽量确保不宕机，但这样做成本比较高，也不符合互联网公司采用相对廉价的PC集群这样一个趋势。

由以上可知，MMAP方式的数据库从理论上就注定了其灾难恢复性不佳。10gen公司宣称其增加了记录日志的功能，但只是改善了其原来比较差的灾难恢复性，按照传统数据库的标准，离真正的灾难恢复性还有比较远的距离，因为日志最多是把最近的操作重放一遍，但操作系统把数据库写坏的可能性一直存在。10gen公司选择这种方式也是有它的考虑的，官方认为单机可靠性并不是很重要，虽然对这点还存在争议。官方建议的是用复制和复制集（replication set）来保证持久性，意思是如果主库崩溃，就马上切换到正常的节点。如果主库宕机，重启会不成功，那么必须先进行修复，才能启动，修复的过程相当于把所有的数据全部导出来（跳过损坏的数据）再导进去，如果用户认为自己可以忍受数据损失，那么就没有关系。这是一种极度追求可用性的方案，缺点是难以保证数据安全。

目前存在争议的地方是，MongoDB在现实中要求许多冗余来保障安全，而实际上大部分的应用，单机就可以支撑了，单机的可靠性还是很重要的。不太可能使用3~4个节点的复制集来确保一个普通应用。且复制集并不是很成熟（2.4版本的投票机制存在问题）。

笔者所在的生产环境中曾出现过宕机破坏数据文件的情况（2013年），然后主机重启后MongoDB可以起来，但是数据文件里的部分块已经被破坏，由于MongoDB并没有做数据块的校验，因此不能及时发现这种数据破坏，很可能在运行一段时间后突然崩溃。所以宕机对于MongoDB来说还是很致命的，如果是重要的数据，为了以防万一，在宕机后，建议重建整个数据库。

10gen公司对此似乎一直保持着回避的态度，宣称的是“无单点故障”，其实也有“无奈”的成分，存储系统虽然在商业领域很成熟，但要通过一个开源公司来实现，其实是非常困难的。为了尽快适应市场需要，拿到风险投资，放弃灾难恢复性可能就成了必然选择，但这并不代表他们的做法是对的，新的稳健的存储引擎是必需的，这只是个成本问题。

2) 另一种是日志型的存储模型，数据文件是顺序添加的，如bigtable、couchdb。其他如HBASE、leveldb也是类似的实现。这种存储实现，可以保证在系统掉电后，数据文件尽量不被破坏，需要考虑的是灾难恢复后如何进行恢复，且不丢失数据。目前许多公司都是基于Google的bigtable模型来设计的，既保持了单机的持久性，又有优良的伸缩性。

3) 目前Redis的实现，有些特别，主流的快照持久化方式，是把内存中的数据定期dump到磁盘上（先dump到临时文件，然后mv，可以保证整个过程是原子性的安全操作），对于实例崩溃，电源掉电之类的故障，也能表现良好，只是会有数据丢失。Redis另有一种AOF的方式，通过回放日志来进行灾难恢复，它可以尽量减少数据丢失，但由于I/O资源消耗比较大，因此用的并不是很多。

22.2.3 可维护性

可维护性与维护服务的运作及在出现服务中断时尽快恢复等活动相关。许多NoSQL产品让人诟病的地方就是维护性比较差。缺少文档、缺少监控工具、诊断困难，这些都影响了维护性，运维人员显然没有积极性去使用维护性比较差的产品。MySQL发展了很多年，相关文档都已经比较完善，IT人员一般都学习过关系数据库的理论和SQL，比较容易上手。而理解NoSQL产品则不一定容易，也许会比较容易上手，但碰到问题后，很可能会一知半解，没有足够的文档或示例加以阐释。

可维护性在很大程度上取决于内部维护人员的技能，对于MySQL等常用的数据库，已经有许多人在学习和研究它，它有成熟的解决方案，合理的运维方案，出了问题后一般能够快速解决。可是NoSQL产品灾难恢复的解决方案可能还不成熟，也缺乏人去熟悉它的机制，出问题的概率更高。一些软件架构师选择NoSQL产品的理由是其强大的特性，但要让产品稳定运行于生产环境中，是需要有相关运维人员支持的，如果运维人员认为这个产品的使用者少、社区不活跃、太复杂，可能自身也没有积极性去研究和学习这个产品，这反过来会让这个产品的维护性变得更差。习惯的力量是很强大的，如果新的产品不够易用、难以理解、难以处理问题，这种情况下非要上某个产品，那么你可能是在和趋势作对。

NoSQL产品如果崩溃了，往往还需要Debug代码，而对于不熟悉源码的运维人员来说这将是一件很头痛的事情，所以，应该提供快速恢复或屏蔽错误的手段。

22.2.4 可靠性

对于用户的操作，数据库产品应按照约定成功执行。比如用户可能会犯错，存储介质可能会有异常，各种各样的系统错误都可能会发生，在这些情况下，我们的数据库系统仍然要保持功能的正常和完整。因此我们需要各种措施和手段来确保数据库产品的可靠性。

我们可以模拟各种用户错误，模拟各种底层硬件、操作系统的异常情况，来验证数据库系统的可靠性。

一般有两种级别的可靠性。

·应用依赖的可靠性，比如参考完整性。

·应用无关的可靠性，比如ACID特性。

许多数据库产品都支持ACID特性，但支持的程度可能不一样，MySQL InnoDB支持复杂的事务，事务操作可以同时修改多个表的数据，支持多语句操作，而MongoDB并不支持多文档事务，它仅支持单个文档的原子性操作，官方的解释是：一般情况下，这些就足够了，因为你可以把相关的数据都放到一个文档内，然后写这个文档。

可靠性，可以用如下两个指标来衡量。

1) 正确性：正确性指的是系统能够按照约定成功运行。对于复杂的数据库系统，可能很难用工具去校验系统的正确性，往往需要线上大规模用户的长期验证。关于正确性这方面，MySQL等传统数据库往往已经经历了10年以上的大量用户验证，因此相对来说它们更具正确性。

2) 可用性：系统能够正常工作的时间。我们可以统计一个系统能正常满足用户操作的时间比例，来衡量一个系统的可用性。

一般以上两者不可兼得，为了追求正确性，可能会降低可用性，比如银行、证券交易系统就需要很高的正确性，为了确保数据准确，宁可降低可用性、暂时停止服务。而提高可用性可能就会无法保证正确性，比如一些信息发布系统，虽然数据可能不是很准确，但并没有什么大不了的，用户更在乎的是系统可用。在做架构设计的时候，就需要权衡二者，确定合适的可靠性。

22.2.5 高可用性

本节内容更多考虑的是单机产品。对于只有少量机器的公司，单机虽然可能失效，但一般还在容忍的范围内，重要的是维护人员能够迅速地处理故障。设计体验良好的应用程序对于处理数据库的异常宕机会更好，给予用户友好提示，或者临时切换为只读，等待DBA干预是一般中小型公司推荐的做法。

而对于拥有大量机器的公司，如Facebook、Twitter，包括国内的许多公司都有各种数据库中间层（中间件）的方案，在一定的规模级别下，宕机的概率可能会导致运维工程师根本处理不过来，自然而然就催生了数据库单点切换，数据库中间层的一些方案。

MongoDB（2.4版本）可以配置为复制集（replication set）。但生产实践证明它的自动冗余切换的特性并没有宣传得那么好，官方的投票算法不够完善，难以解决复杂的网络硬件故障异常所导致的切换，所以其高可用性得打个折扣。

22.2.6 高性能

NoSQL产品的官方说明都是说它们具有高性能，对于许多人来说，这是一个好的卖点，对此我们应该有一个清楚的认识。官方的评测数据更多的是出于市场的需要，它属于产品生命周期的一个正常部分，是公司的一种宣传手段，公司需要一份评测数据来验证产品的先进性，证明比市场上另外一些产品更好，这样才会有源源不断的新用户加入，但如果将评测太当回事，就没有必要了。现实的情况是，商家发布的产品评测报告，本质上并不是一份基准测试报告，但它们却常常伪装成基准测试报告。商家的报告往往为了证明其比同类产品更优秀，但缺少上下文环境，这可能会导致用户产生误解。

我们衡量的高性能，应该是高访问压力下的高性能。各种测试报告往往专注于峰值吞吐数据，但是在现实世界中，我们是希望在合理的成本下达到期望值，同时还要求不降低服务质量，我们更关注峰值效率，显然基准测试报告很难满足此点。

如果我们使用缓存（如Memcached），那么我们就不用太担心读的压力。一些NoSQL产品和传统数据库产品都有缓存热点数据的功能，只是实现效率和成熟度的差别，但基于磁盘的数据库和基于内存的数据库有本质的区别。这里我们仅讨论基于磁盘的数据库系统，在海量数据高并发读的情况下，我们仍然需要缓存产品。因为单机的内存有限，无法缓存所有数据。

对于数据写入，起决定作用的是物理磁盘（存储），磁盘技术在其发展的几十年以来，一直是计算机的瓶颈所在，相对于CPU和内存技术的快速发展，磁盘技术一直没有太大的改观。所以不要期待换了数据库产品，就可能有什么特别的改善。

实践证明，单机的Key-Value产品性能并没有像传说中的那么高。为什么生产环境和网上的官方性能数据会出现那么大的差异呢？原因是部分的NoSQL产品是MMAP存储引擎的方式，因为写入数据时并不需要实时刷新到磁盘，而是操作系统批量地将数据刷新到磁盘，通过这种方式可以极大地提高吞吐率。这对于小的数据小的系统会很有效，但对于大量数据的场合，缓存一般都会耗尽，真正的性能将严重受制于磁盘的性能。如果做基于Key-Value的单机性能测试，传统的产品和NoSQL产品并没有什么区别，而且传统产品还会更胜一筹，因为毕竟传统数据库发展了很多年，各种优化、索引的技术更完备。

实际生产验证中，当数据库数据远远大于内存时，此时缓存已经不能存放所有的热点数据，那么数据库可能不得不去磁盘进行读取，miss rate（缓存未命中比率）指标可以用来衡量Cache（缓存）的效率，如果数据库能够尽可能地把热点数据放到缓存中（操作系统Cache或数据库Buffer），那么就可以减少miss rate，否则miss rate会上升，将可能导致磁盘I/O瓶颈，你的系统将受制于I/O瓶颈，MongoDB依赖于操作系统缓存数据，操作系统是不太可能知道数据的冷热情况的，所以其miss rate会表现得比MySQL差。注意，我们一般不关注数据库的缓存命中率，我们应该关注的是缓存未命中的比率。

生产实践还表明，MongoDB的INSERT速率会比InnoDB慢5倍以上，主要原因在于MongoDB写入了比其他数据库多得多的大数据（包括日志和数据文件）。

对比MongoDB，传统的数据库对于数据的处理往往做了更多的优化，比如InnoDB的insert buffer，数据文件也更紧凑，支持行锁，所以其有更高的性能，这点并不稀奇。

所以，如果我们测试NoSQL产品，发现其和传统产品有很大的区别时，应该检查下，是否有其他因素的影响（如OS、缓存、数据结构等）。

以上所说的是单机的基于磁盘的产品，如果是一个集群，从应用层sharding（分片），那么NoSQL集群的性能一般也会弱于传统数据库产品的集群。

以上并不是说NoSQL的性能就必然比传统的数据库产品差，因为有许多其他因素还没有考虑进去，数据结构及程序对于数据的操作方式也深刻地影响着性能。希望大家明白，衡量一个产品要考虑实际的物理限制。在底层硬件技术没有太大改变的情况下不要期待太高，性能大致差不多是更符合现实物理法则的。任何产品在自己的官方页面上，都可能会加上“高性能”几个字。

要提高I/O性能，更多的是其他设计层面上要考虑的，如：转换随机读写为顺序读写、进行压缩、访问存储方式（可参考http://en.wikipedia.org/wiki/Locality_of_reference）。例如，TokuMX对于数据页的写入是顺序I/O，顺序I/O的代价更小，从而TokuMX有更多的余量可用来处理读请求。

高性能还有一个潜台词，可以充分挖掘出硬件的能力，以及充分利用硬件的能力。如，在一台CPU资源很充足的机器上，支持压缩的数据库产品，往往可以获得比较好的性能。因为CPU不怎么饱和，而经过压缩写入更少的字节就意味着I/O写入的代价更小，性能可以得到提高，在大数据的情况下，数据能够被压缩不仅可以提高性能，也可以大大减少成本。本质上，这是把I/O瓶颈转换为CPU瓶颈。

其实高性能，并不是我们关注的重点，我们应该重点关注的是NoSQL产品的伸缩性（可扩展性）。

22.2.7 可扩展性

可扩展性，一般是指可以用更多的节点来提高吞吐率，性能不会下降到不可接受的范围。我们也称之为伸缩性。注意，伸缩性不等于性能，伸缩性更多的是满足吞吐率的要求。按照维基百科的定义：伸缩性指的是系统、网络、程序处理不断增长的负荷的能力。它能够满足不断增长的负荷，而自身的性能仍然尚可这样一种能力。

1) 一般的传统数据库，可能有些人会觉得可扩展性比较差，但对于海量数据，MySQL分库分表在目前来说仍然是最成熟的方法，从应用层分片不存在任何问题，前提是你需要有合理的数据规划。业内有各种分库分表的算法，一般可以预留几倍到10多倍的扩展，拆分起来的难度一般在可以容忍的范围内。对于绝大部分应用，简单的分库算法可以满足需求。如果真的有爆炸性的应用，可能需要百倍乃至千倍的扩展，那么在前端实现虚拟数据库就是一种可行的办法，虚拟数据库可以理解为前端有很多逻辑的数据库，后端是可变的一些物理数据库，虚拟数据库实现逻辑数据库到物理数据库的映射。这样的方式就像一个数据库代理软件，可以让应用更透明。典型的实现是在数据库中存储一个路由表。

大家可以参考下Facebook、Twitter的架构，它们都是把MySQL当作一个分布式的Key-Value存储，充分利用了MySQL的可靠、稳定和安全的特性。

2) NoSQL产品从理论上说，其可扩展性比传统数据库产品更好，但在现实生产环境中，NoSQL产品的可扩展性可能并没有那么好。大部分NoSQL产品并不是可扩展的，它们一般是属于单机的产品，有部分产品支持扩展，如Amazon的S3、Google的bigtable等。一些可扩展的技术目前仍然没有经过大规模的验证，尚不成熟，比如MongoDB的自动分片，所以和MySQL一样，从应用层分片往往是更值得考虑的、更稳健的方式。在前期的架构设计中，应该充分理解自己的业务，分片数据，尽量避免在后期又需要进行一些自动或手动的扩展。

需要注意的是：节点增多，宕机的可能性就会比较高。现实生产中，NoSQL产品的节点是需要具备单机可靠性的，或者如MongoDB，虽然不具备单机可靠性，但是会有一个复制集（replication set），保证自动故障冗余切换，否则，维护的成本会很高。

3) 一般情况下，我们不需要考虑读压力，也不用考虑读写分离，缓存产品（如Memcached）可以极大地缓解读的压力，我们主要担心的是写的I/O瓶颈。

对于简单查询，一般MySQL普通主机的QPS为几千，Key-Value的存储也是很稳健的，性能不会比NoSQL产品的差。这种能力，已经足够支持大部分应用了。所以我们在做架构设计的时候，要考虑清楚，我们的应用是否有这么高的并发访问量。如果一台机器能够满足1~2年的几倍增长需求，就不需要过度设计，不需要考虑到以后几十倍的可扩展性。过度设计将使得架构变得复杂，浪费资源。而且，即使碰到主机不能满足负荷的情况，我们也仍然可以通过向上扩展（scale up）的方式增加内存、CPU，使用SSD（SSD对比传统硬盘，随机读写有一个数量级别的I/O性能提升）来提升性能，能用硬件的方式解决就优先选用硬件的方式解决是更具实践性的方案。

4) 现实中的场景，有时是一些混合性的方案，同时使用了NoSQL产品和MySQL产品。比如新浪微博的Redis加MySQL技术方案，MySQL支持海量数据，Redis支持高性能及一些其他的特性。对于海量数据高性能的方案来说，这是更现实的选择。一般来说，同时支持海量数据和高性能的数据库，其实现代价会很大，或者说费力不讨好。在架构上，同时使用传统数据库和NoSQL数据库产品，是更值得采纳的方案。

5) 关于MySQL Cluster的方案，具体如下。

还有一种Oracle官方公布的伸缩性比较好的方案，MySQL Cluster，但很少有人精通，没有已知的大规模使用的案例。MySQL Cluster是share nothing的架构，把数据分布在各个节点的内存中。据说现在的新版本也能容许将部分数据放到磁盘上。这个产品所受的限制比较多，很复杂，除非你是这方面的专家，很熟悉它的特性和优化，否则不建议使用。

这种技术架构，可以有一定的扩展性，但实际上它可能对付不了不断增长的扩展性需求，因为一旦节点增多，各个节点需要互相通信，并保持同步，代价就会越来越大，节点会是有限的，这点和Oracle的RAC有点相似。而海量应用，其节点可能就会非常的多。

从成本上来说，内存也比较贵。把热点数据加载到内存是更经济的方法，用不着把所有数据都加载到内存里。基于磁盘的Key-Value海量存储仍然是性价比最佳的方式，如果纯粹想用内存数据库实现所有需求，可能性能很好，但成本将会是惊人的。

以上5点的说明，目的是澄清一些对于NoSQL产品可扩展性的误解。现实中，NoSQL产品往往没有宣传得那么好，但是，NoSQL产品毕竟代表了一个趋势，它突破了一些关系数据库产品的限制。DBA有必要利用NoSQL产品来增强整体系统的可扩展性。

可扩展性还可以理解为，随着数据规模的增大、机器设备的增多，业务规模的增大，我们的人手不用增加太多，我们的服务管理性仍然优良。不同阶段考虑的问题不太一样，当机器设备很少时，这个时候的维护成本主要是人的成本，当机器数目很大时，就有必要减少硬件、网络的成本了，这个时候，一个高性能、高可靠、高效的数据库产品就更值得看重了。随着数据的增加，数据库主机的增多，大规模运维必然会摆上日程，这方面，MySQL有比较成熟的解决方案，但对于一些开源产品来说，大规模的部署和自动化运维会有不少挑战，综合运维的成本会比较高。

22.2.8 资源利用

对于大规模服务的运维，需要考虑资源的利用效果。传统数据库产品已经发展了很多年，往往有了比较成熟的分配资源和控制资源的方式。而开源数据库产品，往往前期着重在功能实现上，可能会忽略对系统资源的使用。

对于资源的消耗，我们可以做一些数据库产品的测试，然后记录一些相关的数据指标，加以对比。例如，每秒的读/写次数、平均每个查询的物理读、每秒写入的数据、每秒读取的数据等。如下列出了一些指标，可供参考。

·DB-size: 一轮测试后数据库的大小。

·Bytes-per-doc: 平均每条记录/每个文档的长度。DB-size除以行数（文档数）。

·Write-rate: 使用iostat统计的平均每秒写入存储的字节数。

·Bytes-written: 本轮测试总计写入存储的字节数。

·Test-secs: 本轮测试所耗时间。

·TPS: 本轮测试的平均事务吞吐，各个数据库产品的事务应该类似，比如插入1000条记录/文档。

·Server: 所测试的数据库产品配置（比如是否启用了压缩，是否开启了日志，日志的刷新策略如何等）。

下面我们以MongoDB为例，说明其资源利用的不足之处。

1) MongoDB对于内存资源的利用。

MongoDB采用了MMAP的方式来操作数据文件，这将导致我们无法限制MongoDB进程所使用的内存容量，它会尽可能地申请所有空闲内存作为自己的缓存，虽然理论上在其他进程需要内存的时候，MongoDB可以释放出部分内存，但这样的资源管理方式并不太友好，可能会导致部署在同一台机器上的其他服务出现内存瓶颈。而且用操作系统来管理缓存，效率也比较差。

目前最好的部署办法只能是将其单独部署在一台服务器（虚拟机）上。MongoDB数据文件会比传统数据库存储大得多，会导致存储成本大大增加。而在文件比MySQL大得多的情况下，缓存数据的效果也会更差，因为缓存同样的数据，MongoDB所需要的内存要大得多。

2) MongoDB对于I/O资源的利用。

采用MMAP的方式其I/O效率比较差，容易导致主机达到I/O瓶颈，产生许多毛刺。MongoDB使用的是缓冲后批量刷新数据的方式，由于靠操作系统缓存来刷新数据，因此短时间内可能需要将大量数据写回到存储系统中。Linux并没有针对这种负载进行专门优化，写回期间，其他的I/O请求，比如数据读取或写日志，可能会受到影响，因为此时的I/O资源已经趋向饱和，资源竞争的现象也会加剧。

根本问题是，页面读取和日志写请求应该优先于脏页面的写回。传统的数据库存储引擎有专门的优化，能够尽可能地减少写数据对其他操作的影响，但MongoDB仅依赖于操作系统，难以避免资源的争用对自身的影响。如果查看MongoDB的性能曲线，可以观察到，在被修改的数据回写期间，数据库延时会有比较大的上升，对于InnoDB引擎，如果应用的写操作很频繁，那么也会存在因为检查点等操作导致短时间延时升高、吞吐率下降的问题，但绝大部分情况下会比MongoDB表现得更加稳定。

生产实践可证明，MongoDB写入的字节是InnoDB的4倍，TokuMX的20倍。这对于flash设备的寿命很不利。由于MongoDB的数据文件比InnoDB、TokuMX大得多，因为执行各种文件操作也会消耗更多的资源，缓存的命中率随着文件的增大可能也会下降，你将需要更多的物理读。

生产环境中，有些人会选择禁用journal日志，或者把日志放到内存文件系统（tmpfs）上，这样做性能往往能得到提高，但这是以牺牲安全性为代价的。

数据库往往是I/O密集型的负载，所以衡量一些常规的负载/查询消耗的磁盘读写次数可以大致评估这个产品的I/O效率。由于MongoDB自身并没有一个缓冲池，其主要是靠文件系统来实现读写数据的，因此MongoDB对比InnoDB，查询需要更多的物理读。MongoDB有一个类似LRU的算法用于预测哪些块在内存中，这可能会消耗过多的CPU。由于MongoDB主要靠文件系统来实现读写数据，所以需要重视对文件系统的调优，比如文件系统的预读参数。

3) 资源的利用效率也和负荷有关，由于InnoDB和TokuMX的表是聚簇索引（cluster index）组织数据的方式，数据是按照主键来排列的，那么如果基于主键查找或是主键的小范围查找，效率会最高。而MongoDB的表不是按聚簇索引的方式存储数据的，因此基于主键的查找显然需要更多的物理读。

4) 大规模运维的资源考虑。

一般来说MongoDB必须独占整个主机的资源，会影响到上面的其他实例，无法充分利用资源。大规模部署MongoDB，可能意味着硬件资源的极大浪费，磁盘空间、内存资源可能都存在巨大的浪费。许多人的生产实践都证明了MongoDB存储效率的不理想，有过多的写入。对于硬件成本，我们还是要关注下，尤其是对于有大量机器，使用固态硬盘的公司，存储成本很高。如果你能写入更少的字节，你就能购买更便宜的设备，你的设备就能够获得更长的寿命，你的备份预算也可以更少。

22.2.9 功能特性实现

如下叙述的是一些功能特性的实现，仅仅是一部分子集，读者可以考虑自己所需要评估的功能特性。

(1) 索引结构

B树: 传统的数据库一般使用B树或B树变种的索引结构，比如MySQL使用的是B+树，SQL Server使用的是标准的B-tree。从原理上来说，B系列树在查询过程中应该是不会很慢的，其主要问题出现在插入过程。B-Tree在插入的时候，如果是最后一个node（节点），那么速度将会非常快，因为是顺序写。但如果数据插入比较

无序、离散的时候，那么可能需要大量的随机I/O。MySQL和MongoDB使用的都是B树。在查询数据时可以获得比较良好的性能，但更新和插入数据的索引的开销会比较大。

fractal树： TokuDB实现了这种索引结构。这种索引结构极大地减少了随机写，一般情况下，更新、插入、查询数据均可获得良好的性能。但在顺序插入的时候性能不佳。

(2) dynamic schema

dynamic schema，也就是NoSQL产品的schema-less特性。MySQL在这方面不太具有优势，MariaDB在这方面有一些改进，NoSQL产品在这方面往往比较有优势。现实生产中，MySQL往往将记录存储为Key-Value类型来实现dynamic schema。Facebook就是这样做的。

(3) 锁

锁的实现，不同的数据库产品对于锁的实现不尽相同，在这方面，传统数据库对于锁的实现较为复杂和高效，而NoSQL对锁的实现一般比较简单，粒度也更大。如MongoDB对于并发写入的锁的粒度很粗，最开始是数据库级别的，后来，据官方介绍，MongoDB 2.8可支持文档级别的锁。

MongoDB（2.4版本）使用的是数据库级的reader-writer锁，不支持行锁，这个per-database RW-lock粒度太大，会导致并发性大降，如果所有文档（documents）都在一个数据库（database）里，那么并发的更新操作都需要申请数据库锁（database lock），很显然，这演化成了一个串行的操作，那么写操作次数的最高峰值将和单个线程操作设备的极限次数相近，即使磁盘设备有余量，也利用不上。也就是说，如果100个线程同时操作MySQL的某个表的某些数据，那么会产生竞争，如果把这些数据分离到多个表，则可以提高性能，而对于MongoDB，即使你把数据分离到了多个文档（如果仍然在同一个数据库里），也无助于性能的改善。

你可以选择每个集合（collection）一个数据库，这样可以减少互相等待，大大提高并发性能，但这种做法很奇怪，不符合正常的使用情况。

就目前的生产实践而言，MongoDB仍然有较大的性能提升空间，锁的实现显然是MongoDB的重要瓶颈根源之一。

(4) 事务

MongoDB不能严格地支持事务，可以理解为其支持单个文档的事务，它并不支持修改多个文档的原子性操作，官方的解释是，MongoDB的Documents模型可以嵌套，也许一个文档已经包含了所有需要原子性操作的数据了，这个还不能确定，对于数据库来说，如果不能支持多个对象的原子性操作，很难说得上其对事务的支持。对于海量数据来说，多个分片中，保证对事务的支持是很难的，往往不需要进行考虑，但是对于单个分片，仍然有一定的实现ACID特性的必要。

MongoDB的事务级别类似于我们所说的read uncommitted，read uncommitted是数据库理论中隔离级别最低的一种，这样做可以允许客户端在写操作还没有返回或还没有实际提交之前就看到，这样可能会出现数据显示和实际数据存储不一致的情况。比如我们刚查询到一批数据，但马上就宕机了，重启之后我们再查询，可能这部分数据已经丢失了。

(5) 文件管理

我们看下MongoDB的空间分配。

据笔者的生产实践，MongoDB占据的磁盘空间比MySQL大得多，可以理解为文档数据，如JSON这种格式，存在许多冗余数据，但空间占用大得有些不正常，甚至是传统数据库的三到四倍，不太契合工程实践，应该还有改善的余地。

1) MongoDB的每个库逻辑上包含许多集合（collection），物理上存储为多个数据文件，数据文件的分配是预先分配的，预分配的方式可以减少碎片，程序申请磁盘空间的时候将更高效，但MongoDB预分配的策略可能会导致空间的浪费。默认的分配空间的策略是：随着数据库数据的增加，MongoDB会不断分配更多的数据文件。每个新数据文件的大小都是上一个已分配文件的两倍（64MB、128MB、256MB、512MB、1GB、2GB、2GB、2GB），直到达到预分配文件大小的上限2GB。虽然2GB的阈值是可以调整的，但一般运维的时候往往不会进行调整，就这点来说，可能会导致大量空间的浪费。

对于磁盘空间的分配效率，笔者一直报以怀疑的态度，如果本身有I/O瓶颈，那么预分配一个2GB的文件，将可能导致服务出现严重的性能问题。预分配文件，可以减少碎片，提高程序申请空间的效率，但是是否有必要一次分配和初始化一个巨大的文件，这点还有待商榷。虽然对于预分配的机制，官方文档的说明是可以关闭的，但一般使用NoSQL产品时都会使用默认的配置，也建议使用默认的配置，因为默认配置往往经历了长久的考验，没有那么多Bug。

2) MongoDB的文档在数据文件中是连续存储的，这点不同于一些关系数据库的做法（它们会把长记录拆分为两部分，将溢出的那部分单独存放在另一处），如果没有预留足够的空间，那么更新可能会导致原有空间放不下新的文档。当更新迫使引擎在BSON存储中移动文档时，存储碎片将会导致意外的延迟。对此MongoDB官方给出了如下的解释。

“如果有足够的空间，在MongoDB中更新文档时，数据会在原地更新。如果更新后的文档大小大于已经分配的空间，那么文档会在一个新位置被重写。MongoDB最终会重用原来的空间，但这可能需要时间，而且空间可能会过度分配。

在MongoDB 2.6中，默认的空间分配策略将是powerOf2Sizes，这个选项从MongoDB 2.2开始就已经提供了。该设置会将MongoDB分配的空间大小向上取整为2的幂（比如，2、4、8、16、32、64，等等）。该设置会降低需要移动的文档的几率，并使空间可以得到更高效的重用，结果是更少的空间过度分配和更可预测的性能。用户仍然可以使用精确匹配的分配策略，如果不增加文档的大小，那么该策略将会更节省空间。”

以上是MongoDB官方的解释。

显然，这种策略将导致空间的浪费，特别是对于导入只读类型的数据。

3) MongoDB不支持数据文件的压缩，也不能回收空间。它所使用的碎片整理的策略，可能是在一个新的地方重写，而不是对旧的碎片进行整理和合并。

4) 不支持校验数据页。页面校验对于数据库是非常重要的，有助于识别存储设备异常、数据块异常。就这点来说，MongoDB存储的数据可能会不安全，也许某一天会导致实例崩溃。

(6) 支持JOIN等复杂查询的能力

一般的NoSQL产品都实现了简单的查询，一般不会实现传统数据库的“JOIN”。

这点对于未来的商业需求来说可能会导致存在隐患。因为如果数据是有关系的，那么就可能存在一些JOIN的需求。

(7) auto-sharding

一些NoSQL数据库或多或少地实现了此类特性，传统关系型数据库不太容易实现。但NoSQL产品的auto-sharding特性，仍然需要大规模生产的测试验证。

sharding一般是指，把数据分片分布到不同的节点（实例）。

当内存、磁盘、网络、CPU等资源受限制了，我们可能需要分片，以获得持续稳定的服务能力。NoSQL产品的auto-sharding往往是一个卖点。我们需要了解分片的一些常识和限制。

分片的一些要点具体如下。

1) 避免单个节点资源超过限制：比如Redis的内存超过了物理内存，性能急剧下降。比如，数据库的内存严重不够将导致索引需要频繁访问磁盘。

2) NoSQL一般不能JOIN，而MySQL JOIN则比较难。

3) 各分片数据可能不均衡。

4) 热点数据可能导致性能问题，可能需要调整sharding key或算法，切割为更小的分片，如果对于延时的要求不高，也可以利用从节点扩展读取的能力。

5) 节点的快速恢复。比如Redis在使用AOF持久化方式时，加载磁盘数据到内存时可能会慢到无法接受，MongoDB修复数据的速度可能会很慢。

6) 单点故障：可以自动路由到正常的节点，或者提供开关降级服务，或者可以提供一个只读的节点，保障部分功能可用。

7) 对于数据的容量规划还不是很清晰，有时这是现实，确实不清楚需求，需要不断调整。

8) 尽量模拟真实环境进行压力测试。

需要说明的一点是：分片可以让失效的数据控制在某个范围内，但同时分片将导致更多的硬件错误。

MongoDB（2.4版本）已知的问题是auto-sharding不太可靠，可能出现CPU瓶颈。即使NoSQL产品有auto-sharding功能，仍然建议软件架构师预先分片。

我们往往需要数据库产品能够通用，这样它才能适应未来不断增长的业务需求。MySQL是一个比较通用的数据库，而开源产品，比如MongoDB、Redis都比较特殊，更适合于特定的场景。我们评判一个产品功能是否完善，除了要满足数据操作的需求，还有其他的运维需要，比如，是否有完善的备份支持和监控支持，是否方便迁移和故障切换。开源产品在这方面功能一般不够完善，一些开源产品，比如LevelDB只是实现了数据库底层的一些功能，并不是一个完善的数据库产品，但可以作为其他数据库产品的一个基础，比如RocksDB就是基于LevelDB实现的。PostgreSQL作为一个功能强大的数据库产品，其功能丰富性还超过了MySQL，其更像是Oracle的开源版本，但在市场的流行程度都远不如MySQL，如果你要选择它，你要确认是否真的需要它的某些特性，有没有其他更好的替代方案。

22.2.10 数据结构

一般传统的关系型数据库更适合存储一些结构化的数据，而NoSQL产品更适合存储一些非结构化的数据。有些NoSQL的狂热支持者认为数据应该普遍使用NoSQL产品，移除关系型数据库的种种约束，比如不需要ACID，而一些传统数据库的拥趸者往往认为数据就应该整整齐齐地存放在数据表内。这两种极端都不可取。

在数据量小的时候，使用传统数据库一般就够用了，传统数据库往往比较通用，但在数据规模很大的时候，许多公司都采用了SQL和NoSQL数据库并存的策略。对于大规模的业务，我们需要在开发的便捷和运维的成本之间找到平衡点，选择合适的数据结构和合适的数据库产品。比如Redis里就存在一些适合开发人员“拿来就用”的数据结构，它们适合程序员的编程思维，因此用起来更舒服。

22.2.11 选择数据库产品的建议

如下是笔者对于数据库产品选择的一些建议。

1) 应该随着大流走。

我们先看一些业内公司的选择，比如Facebook、Twitter，大部分的应用使用的仍然是MySQL加Memcached。Facebook的MySQL主要用来存储Key-Value数据，在2008年，就已经有了1800台MySQL。Twitter曾经说要迁移到NoSQL上，但也只是说说，为什么许多公司仍然选择MySQL存储海量数据，是因为目前仍然缺少稳健的方案，谁也不愿意当小白鼠。

所以，我们应该跟着大流走，多研究NoSQL产品，在业内已经有了比较成功的应用之后，再考虑使用NoSQL产品。比如，新浪微博大规模使用Redis，其运维能力很强，就是一个值得借鉴的案例。一些公司，如Facebook、Yahoo、Baidu大规模使用Hadoop就是一个趋势。但是，某个数据库产品是否适用于自己的应用，仍然要仔细检查需求，看是否真正满足你的需要。如果有许多大公司选择使用某个数据库产品，那么随着时间的发展，这个产品必将变得更好、更完善，如果大公司也倾向于放弃这个产品，那么我们就需要审慎考虑，是否要继续跟进，是否会碰到一些产品限制或功能缺陷。

2) 选择产品有两个决定性的因素：是否有稳定的团队维护，开源社区是否活跃。目前的现实是大部分的NoSQL产品都是昙花一现。而那些趋势较好的产品大都满足这两个条件，比如Redis、MongoDB、HBase。MongoDB的迅猛发展就和10gen公司不遗余力地推广和响应社区有很大关系。Redis后面有VMware的支持，而HBase作为Apache的顶级项目，许多大公司都在使用和贡献patch。研究这些业内比较知名的产品，用更低的成本支撑应用才是正确的选择，而不是花费过多的精力研究不断出现的眼花缭乱的产品。

对于开源产品，内部有熟悉源码的团队也是必需的，即使社区再活跃也很难帮你马上定位到问题所在，有时候，你不能等待官方发布补丁，你必须自己解决问题，现实中，许多公司的解决方案往往比官方的解决方案还要早了好几年，这是因为大规模使用的公司往往比官方更早发现和提出问题，从而更快地拥有解决方案。比如Facebook和Google就对MySQL贡献了许多补丁。

3) NoSQL主要是为了满足扩展性而出现的，为了海量数据出现的，它不是一种通用的数据库产品。

4) 对于应用场景的界定，应该回到数据的本身上，看看我们的数据是否方便查询、管理和维护，一般来说目前的NoSQL产品在这点上比较薄弱，容易被滥用，如果数据本身是存在关系的，比如基于用户的数据，那么关系数据库仍然是更合适的选择，可以不断满足后面的商业需求，而不需要刻意使用NoSQL产品，许多情况下，其实我们并不清楚自己的需求和数据。

5) 关注基础运维指标。

NoSQL产品的部分初衷也许是认为这些数据存储起来很方便，再也不需要什么维护人员了，再也不用面对MySQL修改表结构的痛苦过程了，但事与愿违。许多人在谈论CAP，但是，如果稳定性、可靠性、可维护性等关键运维指标无法满足，CAP更多的只是纸上谈兵。如果底层存储实现、资源控制没有大的改进，那么某些NoSQL产品将会难以走远，运维这类产品就不是经验就可以解决的问题了。

6) NoSQL产品的安全也是一个因素，很多时候，我们必须相信内网是可靠的。如果这个产品暴露于外网，风险会更大，软件防火墙的防攻击能力欠佳。如果你碰到对安全不做考虑的一些NoSQL产品，则意味着数据的风险，你可能需要修改源码，增强安全机制。

7) 不要轻信宣传，许多信息，往往出于公司的广告宣传，或者编辑的博取“眼球效应”，往往不能代表实际的使用情况。一些产品，对外声称有许多公司在用，但也许只是某个公司的一个很小的项目在使用，而且现在也不一定还在使用，我们要知道，一个互联网公司，也许有成百上千个项目，某个项目使用了某个NoSQL产品一点也不稀奇。你应该多看一些业内人士的介绍，留意竞争对手的批评，考察其实际市场占用率，尽可能地从各种渠道获取信息。

总之，NoSQL产品，是为了特定的问题而出现的，虽然有各种NoSQL产品可以支持单机使用，Oracle也发布了memcache plugin响应需求，但如果在项目中普遍使用，成本是会大大增加的。目前的NoSQL产品，可靠性、可维护性大都欠佳，后期的升级、Debug、二次开发、维护成本都不容忽视。如果不能承受数据丢失的风险，那么你在选择的时候需要更慎重。

现实的场景是，许多公司的应用，即使架构再烂，也不会成为性能问题，也可以在后期解决，但如果你使用了处于测试阶段的NoSQL产品，可能一开始就会让你感到痛苦，你还得时刻面临未知的风险，软件人员认为有了问题可以Debug，通过更改代码来解决，而从运维的角度来看，这是一种很不好的倾向，如果是核心的关键数据，则意味着存在巨大的商业风险。

现实的问题是，我们有没有海量的数据，我们有没有可能碰到伸缩性的问题，我们的数据是否真的易于管理和维护？为什么许多应用NoSQL产品的公司绝大部分的应用仍然运行在传统的MySQL加Memcached架构下，这些都值得我们冷静思考。



小结 NoSQL产品往往作为MySQL的补充，在许多项目中被使用，甚至在一些项目中作为主要存储。MySQL DBA不应该局限于MySQL，也应该熟悉其他的数据库产品。本章为读者介绍了选择NoSQL产品需要考虑的一些因素，现实中，选择某个产品可能有许多非技术的因素，但这不是本书所能涵盖的。由于NoSQL数据库产品发展得很快，本章中介绍的一些NoSQL数据库产品的数据和行为很可能已经不再准确，请读者参考最新的资讯。

参考文献

[1] Michael Kofler.The Definitive Guide to MySQL 5(3rd ed)[M].New York:Apress,2005.

[2] Baron Schwartz,Peter Zaitsev,Vadim Tkachenko.High Performance MySQL:Optimization,Backups, and Replication(3rd ed)[M].New York:O'Reilly,2012.

- [3] Brendan Gregg. Systems Performance: Enterprise and the Cloud[M]. Upper Saddle River: Prentice Hall. 2013.
- [4] Bill Karwin. SQL Antipatterns: Avoiding the Pitfalls of Database Programming[M]. Raleigh: Pragmatic Bookshelf. 2010.
- [5] Luke Welling, Laura Thomson. PHP和MySQL Web开发[M]. 武欣, 等译. 北京: 机械工业出版社. 2011.