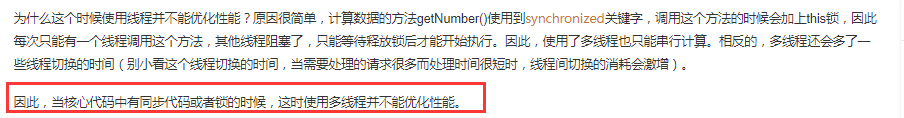
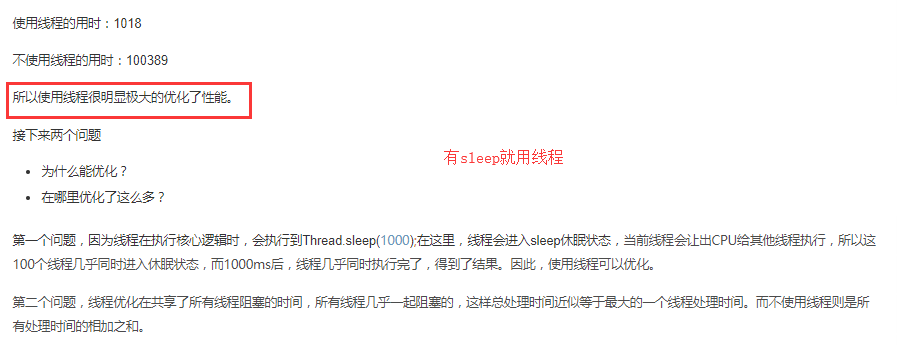
# 理解性能优化

## 容量评估，平均QPS,高峰QPS，单机QPS

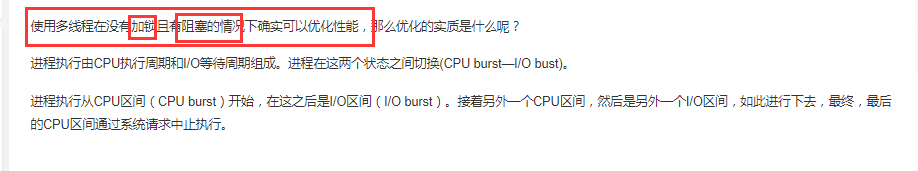


### 使用线程一定能优化吗





### 多线程优化的本质（使用多线程在没有加锁且有阻塞的情况下确实可以优化）

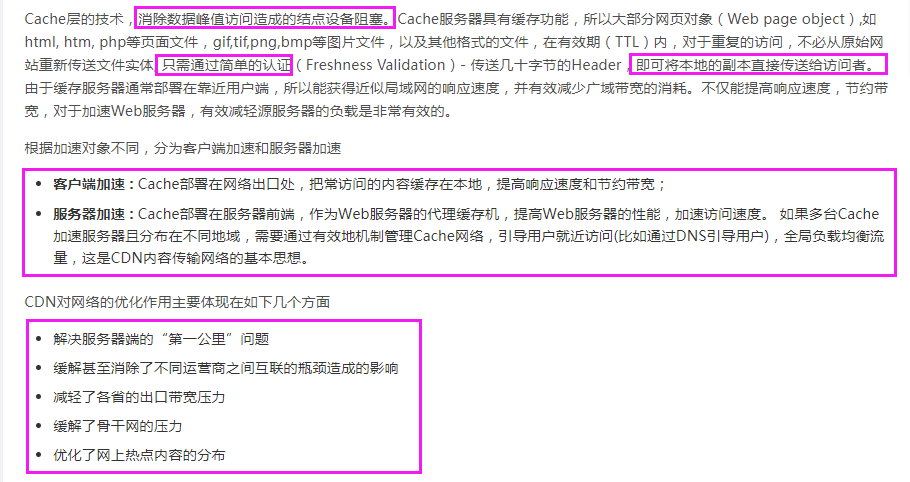


## CDN网络加速原理

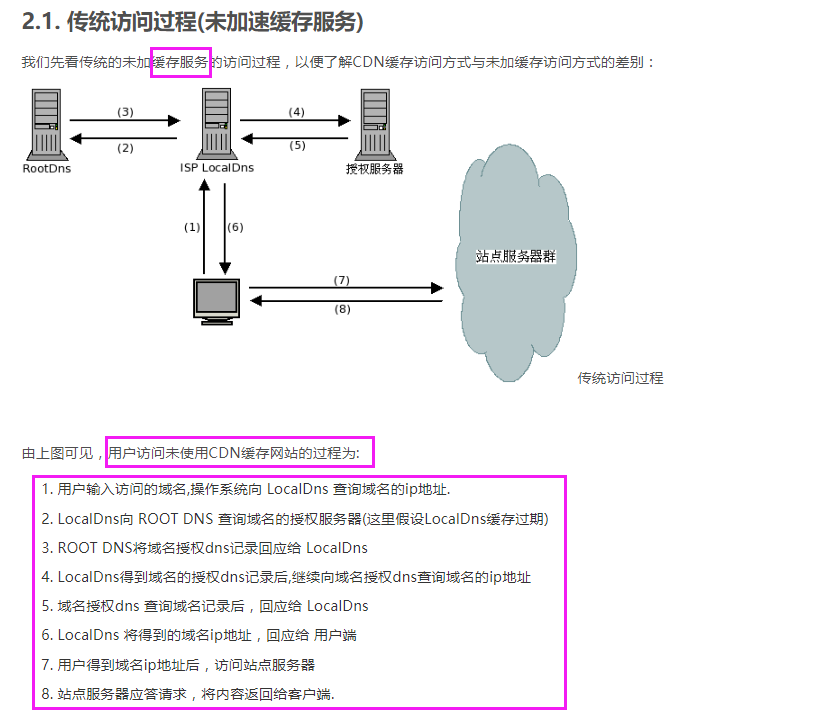
<https://blog.csdn.net/apricotCandy/article/details/88718498>

### cdn概述

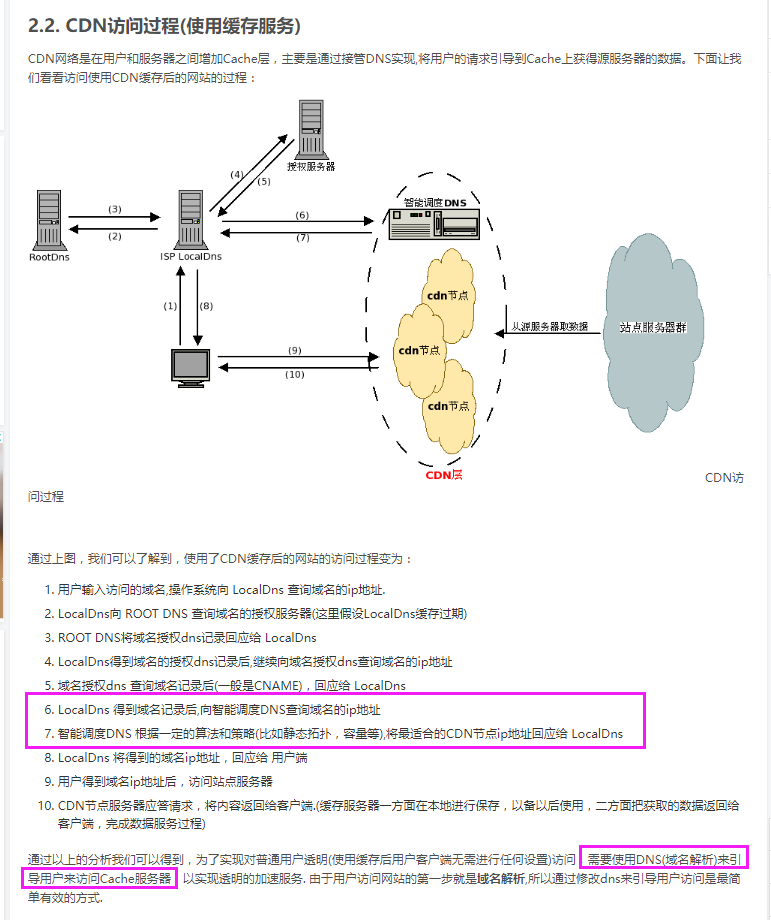
CDN的全称是Content Delivery Network，即内容分发网络。其目的是通过在现有的Internet中增加一层新的**CACHE(缓存)层**，将网站的内容发布到最接近用户的网络”**边缘**“的节点，使用户可以就近取得所需的内容，提高用户访问网站的响应速度。从技术上全面解决由于网络带宽小、用户访问量大、网点分布不均等原因，提高用户访问网站的响应速度。



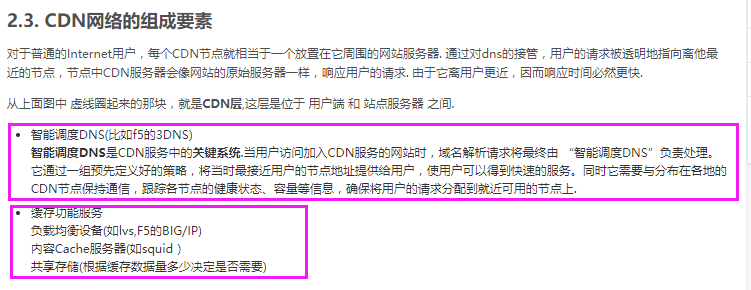
### 传统访问过程



### CDN访问过程（使用缓存服务）

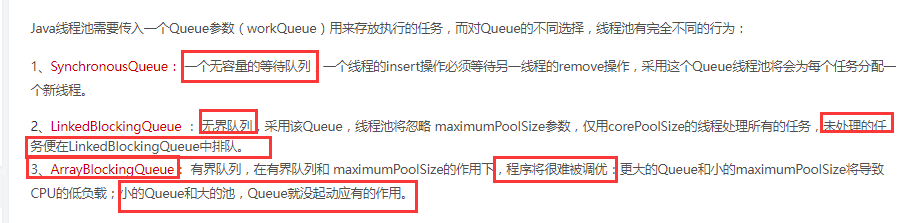


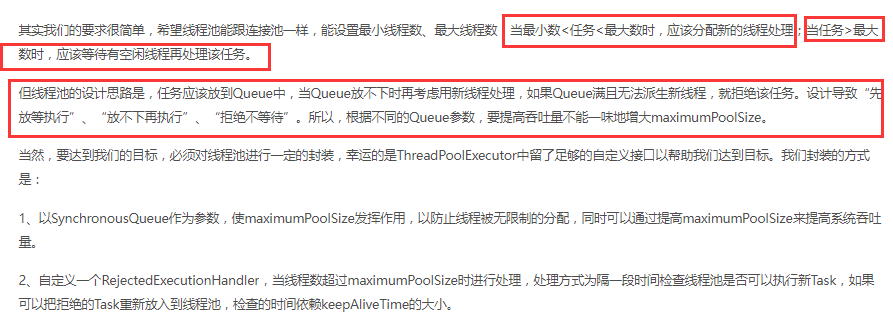
### CDN的组成要素



## 连接池的性能优化

### 1.线程池（解决用户响应时间长的问题）

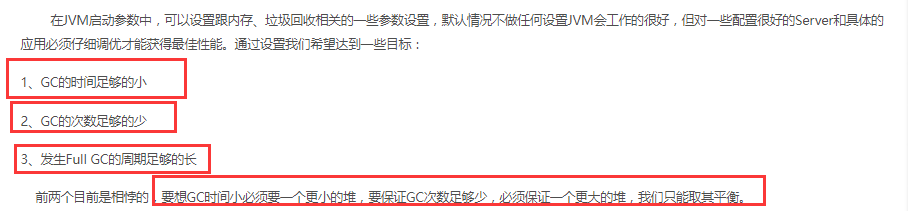


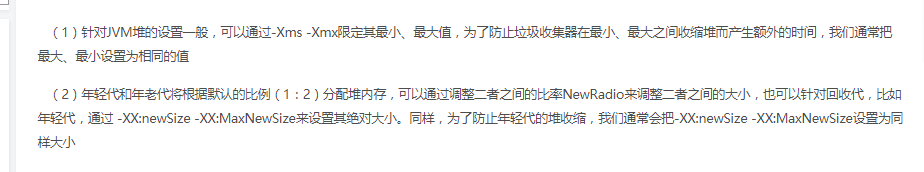


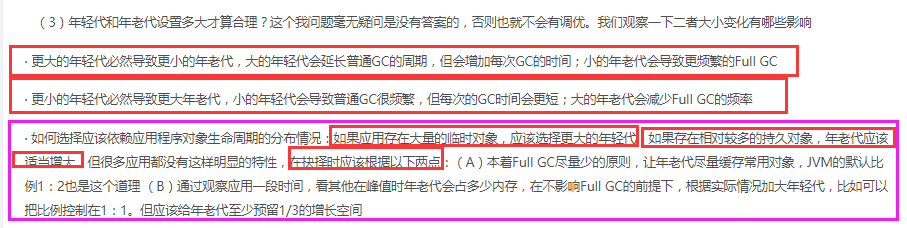
### 2.连接池（数据库连接池BasicDataSource）



### 3JVM启动参数：调整各代的内存比例和垃圾回收算法，提高吞吐量







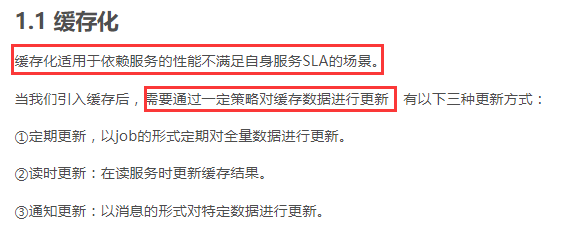


### 4程序算法：改进程序逻辑算法提高性能

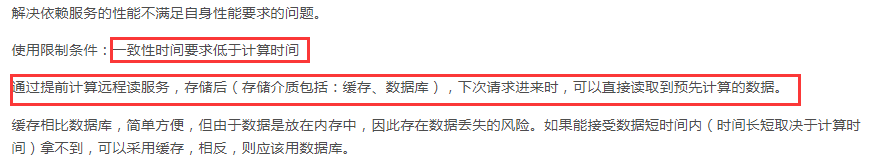
## 性能优化方法论

### 依赖远程读服务优化

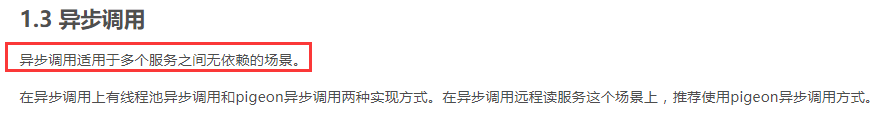
#### 1缓存化



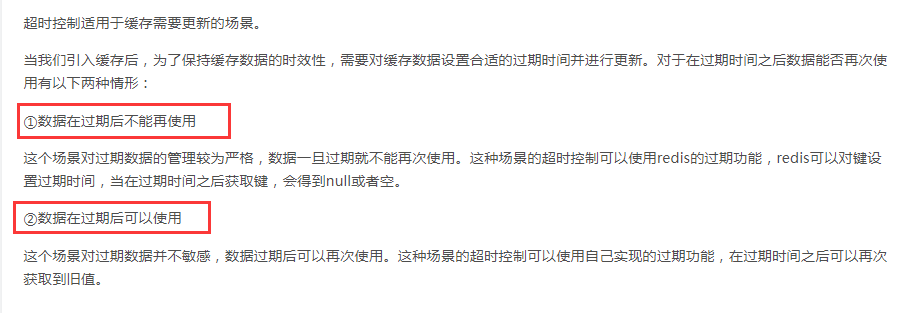
#### 2预计算



#### 3异步调用



#### 4超时控制



### 高并发写场景

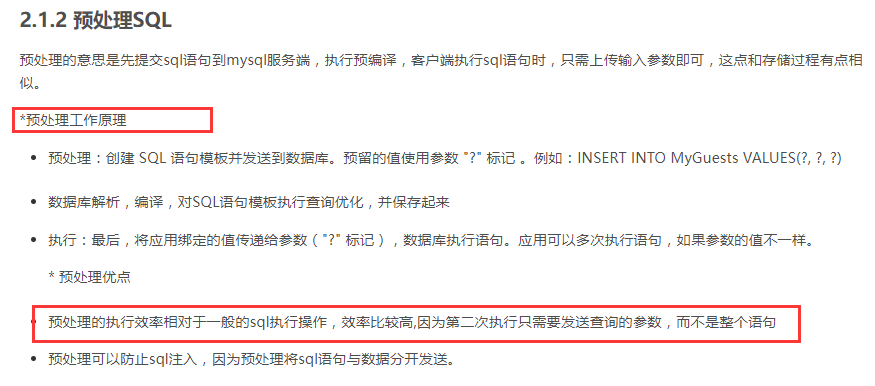
#### 1合并写

多条SQL写入场景，可采用合并写SQL的方式来提高SQL操作性能。具体方法如下：

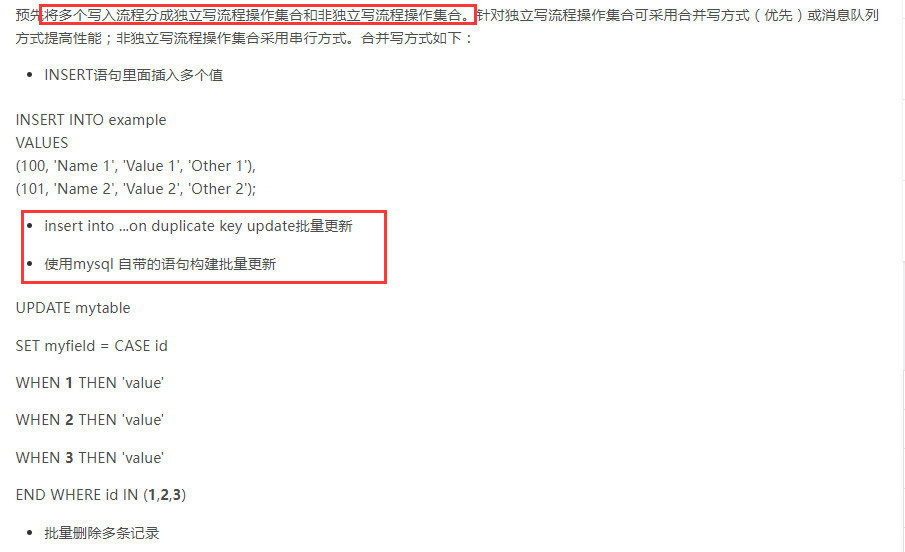
##### 1.1消息队列



##### 1.2预处理SQL



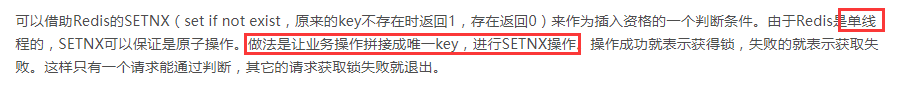
##### 1.3长流程写入操作



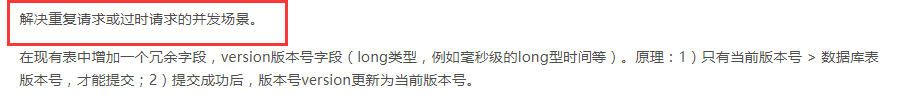
#### 2比较更新

高并发写场景，可采用比较更新策略（CAS），过滤不符合条件的SQL，保证数据一致性的同时，也提高了性能，避免死锁带来的死等待。具体方法如下：

##### 2.1基于redis



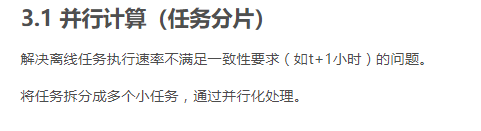
##### 2.2基于版本号



##### 2.3基于状态机条件更新



### 离线任务计算场景



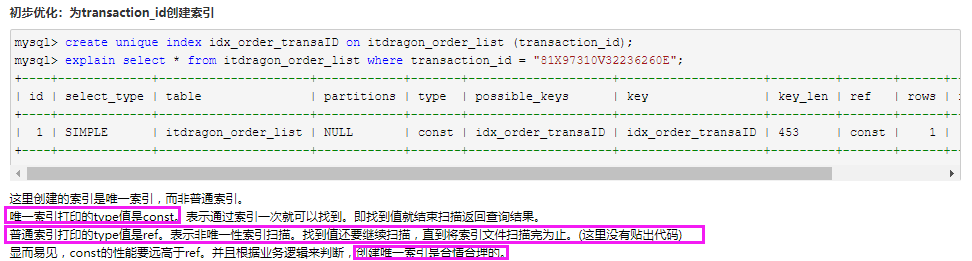
# MySQL调优

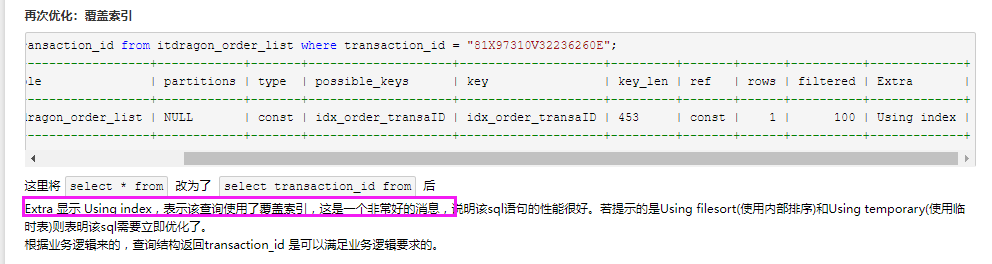
## 理解MySQL底层原理B+Tree机制

## 索引优化揭秘

### 场景一



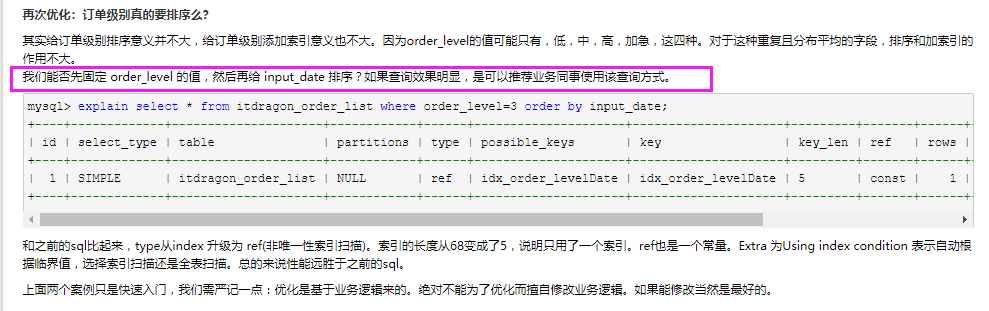




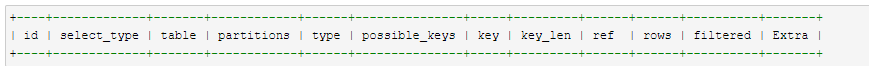
### 场景二







### explain分析sql语句



id

select 查询的序列号，包含一组可以重复的数字，表示查询中执行sql语句的顺序。一般有三种情况：

第一种：id全部相同，sql的执行顺序是由上至下；

第二种：id全部不同，sql的执行顺序是根据id大的优先执行；

第三种：id既存在相同，又存在不同的。先根据id大的优先执行，再根据相同id从上至下的执行。

select\_type

select 查询的类型，主要是用于区别普通查询，联合查询，嵌套的复杂查询

simple：简单的select 查询，查询中不包含子查询或者union

primary：查询中若包含任何复杂的子查询，最外层查询则被标记为primary

subquery：在select或where 列表中包含了子查询

derived：在from列表中包含的子查询被标记为derived（衍生）MySQL会递归执行这些子查询，把结果放在临时表里。

union：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，外层select将被标记为：derived

union result：从union表获取结果的select

partitions

表所使用的分区，如果要统计十年公司订单的金额，可以把数据分为十个区，每一年代表一个区。这样可以大大的提高查询效率。

type

这是一个非常重要的参数，连接类型，常见的有：all , index , range , ref , eq\_ref , const , system , null 八个级别。

性能从最优到最差的排序：system > const > eq\_ref > ref > range > index > all

对java程序员来说，若保证查询至少达到range级别或者最好能达到ref则算是一个优秀而又负责的程序员。

all：（full table scan）全表扫描无疑是最差，若是百万千万级数据量，全表扫描会非常慢。

index：（full index scan）全索引文件扫描比all好很多，毕竟从索引树中找数据，比从全表中找数据要快。

range：只检索给定范围的行，使用索引来匹配行。范围缩小了，当然比全表扫描和全索引文件扫描要快。sql语句中一般会有between，in，>，< 等查询。

ref：非唯一性索引扫描，本质上也是一种索引访问，返回所有匹配某个单独值的行。比如查询公司所有属于研发团队的同事，匹配的结果是多个并非唯一值。

eq\_ref：唯一性索引扫描，对于每个索引键，表中有一条记录与之匹配。比如查询公司的CEO，匹配的结果只可能是一条记录，

const：表示通过索引一次就可以找到，const用于比较primary key 或者unique索引。因为只匹配一行数据，所以很快，若将主键至于where列表中，MySQL就能将该查询转换为一个常量。

system：表只有一条记录（等于系统表），这是const类型的特列，平时不会出现，了解即可

possible\_keys

显示查询语句可能用到的索引(一个或多个或为null)，不一定被查询实际使用。仅供参考使用。

key

显示查询语句实际使用的索引。若为null，则表示没有使用索引。

key\_len

显示索引中使用的字节数，可通过key\_len计算查询中使用的索引长度。在不损失精确性的情况下索引长度越短越好。key\_len 显示的值为索引字段的最可能长度，并非实际使用长度，即key\_len是根据表定义计算而得，并不是通过表内检索出的。

ref

显示索引的哪一列或常量被用于查找索引列上的值。

rows

根据表统计信息及索引选用情况，大致估算出找到所需的记录所需要读取的行数，值越大越不好。

extra

Using filesort： 说明MySQL会对数据使用一个外部的索引排序，而不是按照表内的索引顺序进行读取。MySQL中无法利用索引完成的排序操作称为“文件排序” 。出现这个就要立刻优化sql。

Using temporary： 使用了临时表保存中间结果，MySQL在对查询结果排序时使用临时表。常见于排序 order by 和 分组查询 group by。 出现这个更要立刻优化sql。

Using index： 表示相应的select 操作中使用了覆盖索引（Covering index），避免访问了表的数据行，效果不错！如果同时出现Using where，表明索引被用来执行索引键值的查找。如果没有同时出现Using where，表示索引用来读取数据而非执行查找动作。

覆盖索引（Covering Index） ：也叫索引覆盖，就是select 的数据列只用从索引中就能够取得，不必读取数据行，MySQL可以利用索引返回select 列表中的字段，而不必根据索引再次读取数据文件。

Using index condition： 在5.6版本后加入的新特性，优化器会在索引存在的情况下，通过符合RANGE范围的条数 和 总数的比例来选择是使用索引还是进行全表遍历。

Using where： 表明使用了where 过滤

Using join buffer： 表明使用了连接缓存

impossible where： where 语句的值总是false，不可用，不能用来获取任何元素

distinct： 优化distinct操作，在找到第一匹配的元组后即停止找同样值的动作。

filtered

一个百分比的值，和rows 列的值一起使用，可以估计出查询执行计划(QEP)中的前一个表的结果集，从而确定join操作的循环次数。小表驱动大表，减轻连接的次数。

通过explain的参数介绍，我们可以得知:

1 表的读取顺序(id)

2 数据读取操作的操作类型(type)

3 哪些索引被实际使用(key)

4 表之间的引用(ref)

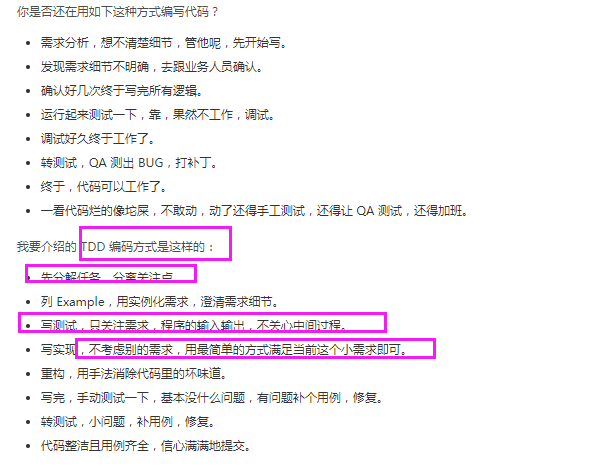
5 每张表有多少行被优化器查询(rows)

## SQL语句优化

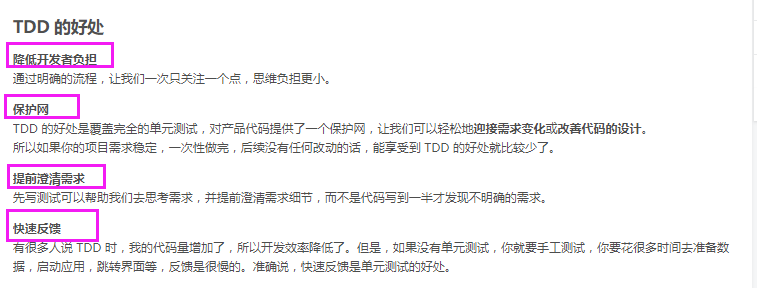
## SQL执行机制详解

# 测试理论

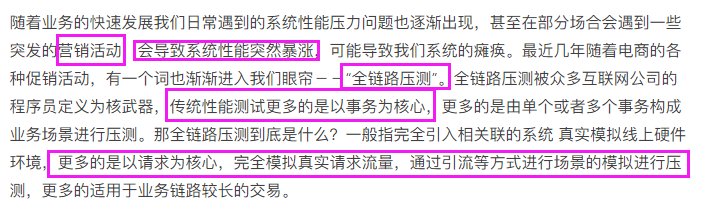
## 深度解析TDD测试驱动开发



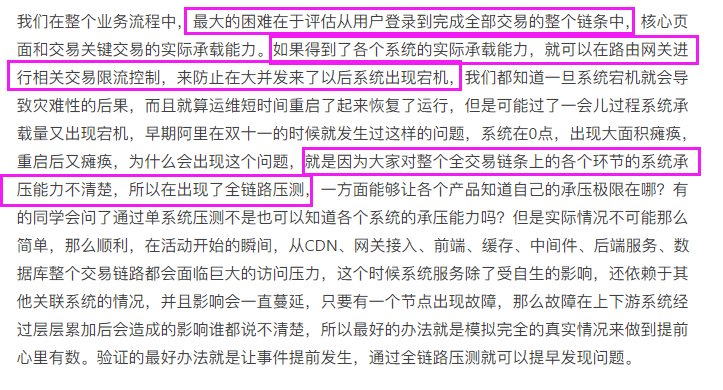
### TDD的好处



## 全链路压测经验



### 为什么需要全链路测试



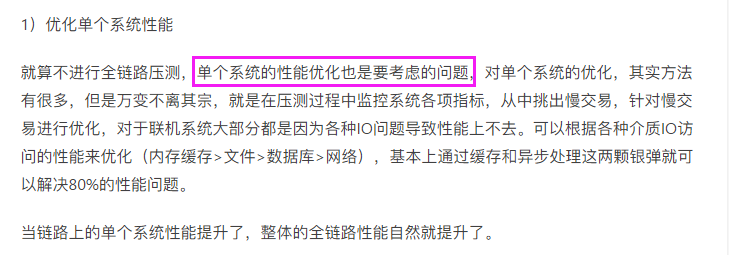
### 全链路压测常常遇到的问题

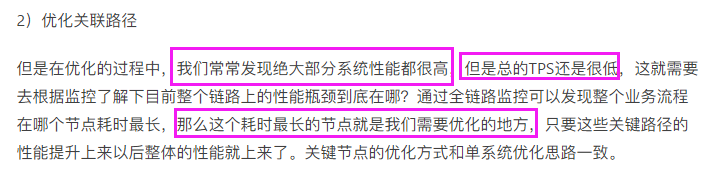
1. 涉及的系统太多，牵扯的开发人员太多；
2. 模拟的测试数据和访问流量不真实；
3. 压测生产数据未隔离，影响生产环境；（举个例子，电商系统在生产环境进行全链路压测的时候可能会有很多压测模拟用户去下单，如果不做处理，直接下单的话会导致系统一下子会产生很多废订单，从而影响到库存和生产订单数据，影响到日常的正常运营。）

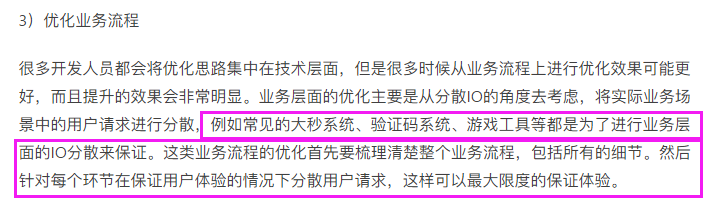
### 如何开展全链路压测

1. 分析需压测业务场景涉及系统（分析清楚了业务场景才能梳理出来涉及的相关系统，分析清楚后也可以更快的找到性能瓶颈进行系统优化-架构师）
2. 协调各个压测系统资源（最难的工作其实不是系统优化、压测环境搭建等技术工作，最难的是压测资源的协调工作。）
3. 压测环境（压测环境也是个比较头疼的问题，很多系统可能压根就没有压测环境，所以全链路压测有个和传统压测比较大的区别就是，全链路压测是在生产环境）
4. 压测数据
5. 压测数据隔离（我们当时的方案是对于压测的这些脏数据都做了特定标示，对于虚拟用户、虚拟商品、虚拟订单、虚拟库存都是有特殊标示的，这样这类数据在统计的时候都不会进行统计，在压测后也会对这些数据进行清理，防止污染正常业务数据。）
6. 压测数据实时监控（我们需要对压测过程中各个系统的cpu、内存、磁盘io都进行系统层面的监控，同时也需要对各个业务节点的耗时进行监控，一方面从业务层面去监控压测事务性能，另一方面从系统层面监控，这样我们可以先从业务层面找到性能瓶颈，再单独分析各个系统的系统层面的瓶颈，最终找到优化方案。）

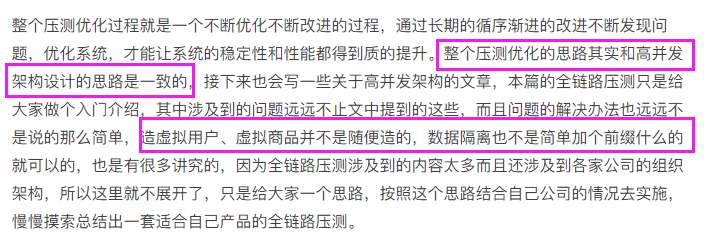
### 全链路压测优化思路







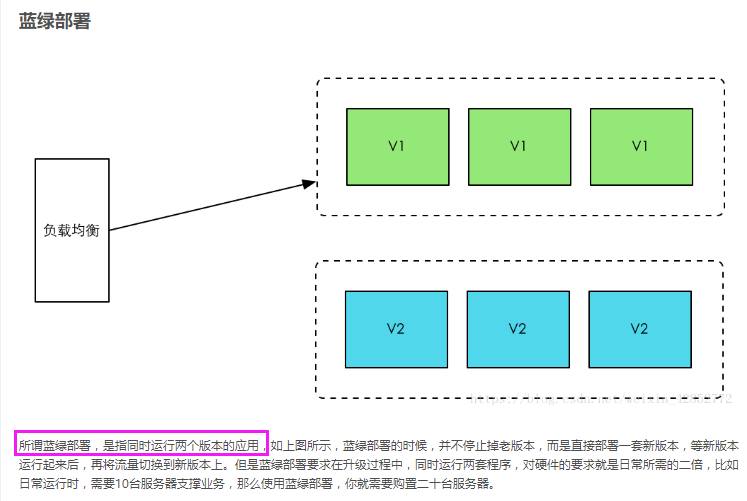
### 总结

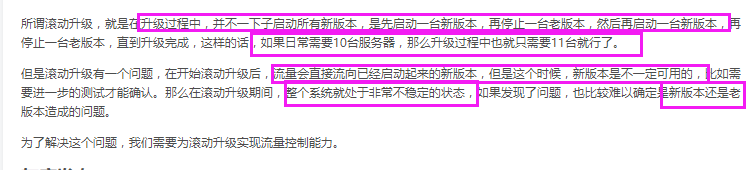


## Nginx根据ip进行灰度发布A/BTest

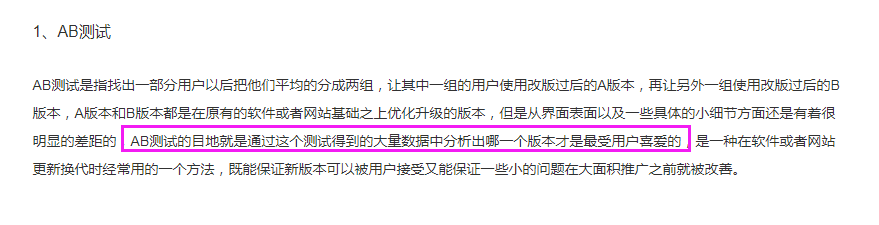
在一般情况下，升级服务器端应用，需要将应用源码或程序包上传到服务器，然后停止掉老版本服务，再启动新版本。但是这种简单的发布方式存在两个问题，一方面，在新版本升级过程中，服务是暂时中断的，另一方面，如果新版本有BUG，升级失败，回滚起来也非常麻烦，容易造成更长时间的服务不可用。

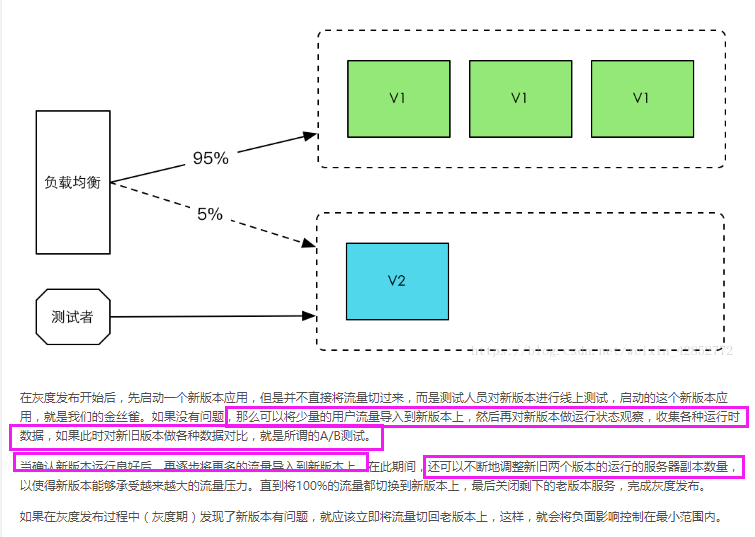
### 蓝色部署





### 灰度部署

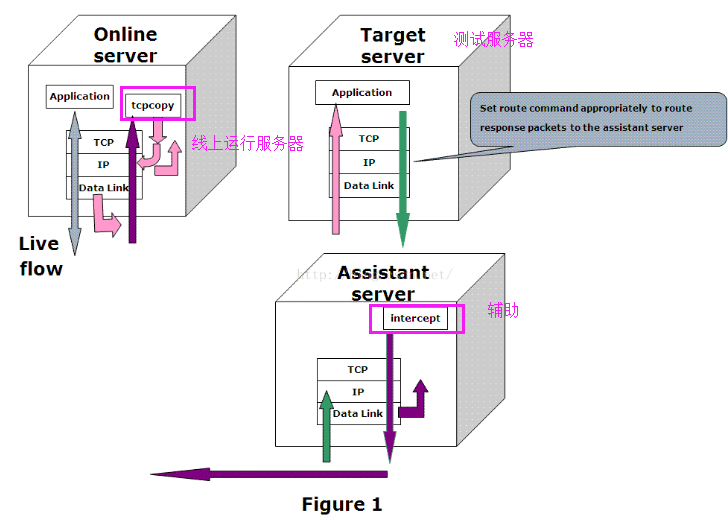


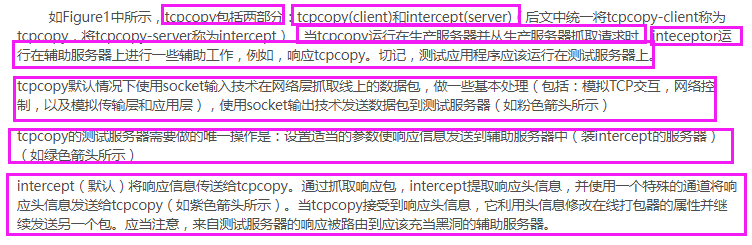


## 真实流量监测工具tcpcopy

虽然真实的实时流量对于Internet服务器应用程序的测试很重要，但是由于生产环境中的情况很负责，测试环境很难完全模拟线上环境。为了能够更真实的测试，我们开发了一款线上流量复制工具-TCPCopy，它可以使用线上真实的流量来对测试环境中的服务器进行测试。目前，TcpCopy技术已经在中国很多公司大量使用。

### 框架





## 单元测试minUnit（这个热门度和活跃度都不行了）

# linux调优（linx没有直接去学）

内核代码组织架构

系统IO参数调优配置

网络协议栈参数配置

内核代码编译与内核更换

# GCC调优

## 快速编译工程Makefile编写



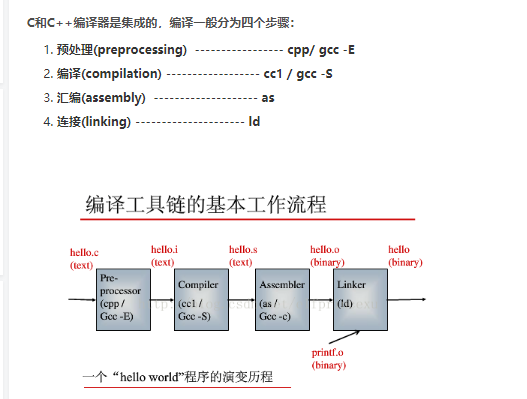
## 编译机制，编译原理

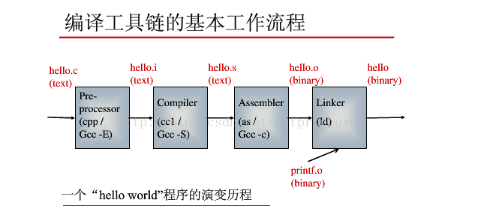
hello.c经过预处理变成预处理后的c文件 hello.i

hello.i经过编译变成汇编语言源程序 hello.s

hello.s经过汇编变成二进制文件hello.o

hello.o经过连接变成hello





### 1预处理阶段

在该阶段，编译器将上述代码中的stdio.h编译进来，并且用户可以使用Gcc的选项”-E”进行查看，该选项的作用是让Gcc在预处理结束后停止编译过程。预处理阶段主要处理#include和#define，它把#include包含进来的.h 文件插入到#include所在的位置，把源程序中使用到的用#define定义的宏用实际的字符串代替，我们可以用-E选项要求gcc只进行预处理而不进行后面的三个阶段，

### 2编译阶段

接下来进行的是编译阶段，在这个阶段中，Gcc首先要检查代码的规范性、是否有语法错误等，以确定代码的实际要做的工作，在检查无误后，Gcc把代码翻译成汇编语言。用户可以使用”-S”选项来进行查看，该选项只进行编译而不进行汇编，生成汇编代码。

### 3汇编阶段

汇编阶段把\*.s文件翻译成二进制机器指令文件\*.o，如命令gcc -c hello.s -o hello.o，其中-c告诉gcc进行汇编处理。这步生成的文件是二进制文件，直接用文本工具打开看到的将是乱码，我们需要反汇编工具如GDB的帮助才能读懂它；

这个阶段接收.c, .i, .s的文件都没有问题。比如gcc -c hello.i -o hello.o等

### 4链接阶段

读者可以重新查看这个小程序，在这个程序中并没有定义”printf”的函数实现，且在预编译中包含进的”stdio.h”中也只有该函数的声明，而没有定义函数的实现，那么，是在哪里实现”printf”函数的呢？最后的答案是：系统把这些函数实现都被做到名为libc.so.6的库文件中去了，在没有特别指定时，Gcc会到系统默认的搜索路径”/usr/lib”下进行查找，也就是链接到libc.so.6库函数中去，这样就能实现函数”printf”了，而这也就是链接的作用。