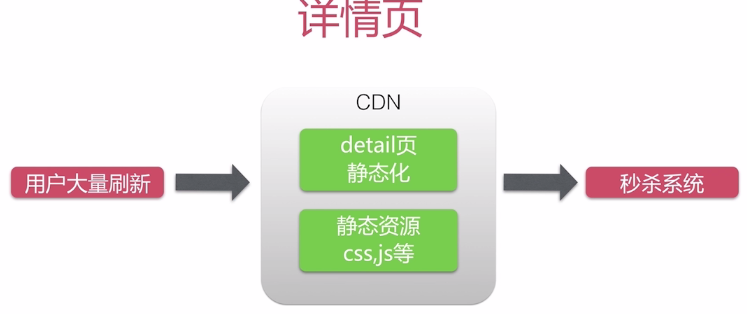
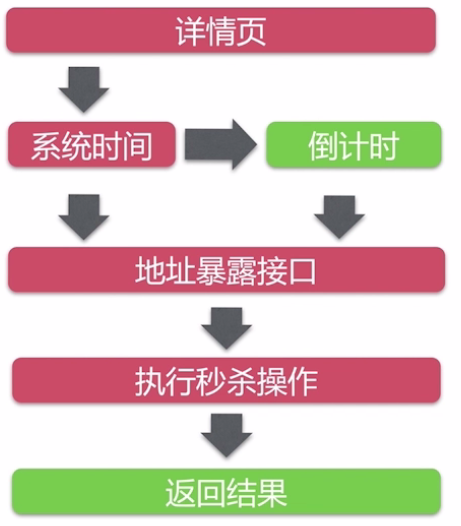
**Java高并发秒杀API之高并发优化**

# 第1章 秒杀系统高并发优化分析

## 最近学习

## 1-1 高并发优化分析（上）



**CND:（内容分发网络）加速用户获取数据的系统**

部署在离用户最近的网络节点上

命中CDN不需要访问后端服务器

互联网公司自己搭建或租用

**获取系统时间不用优化**

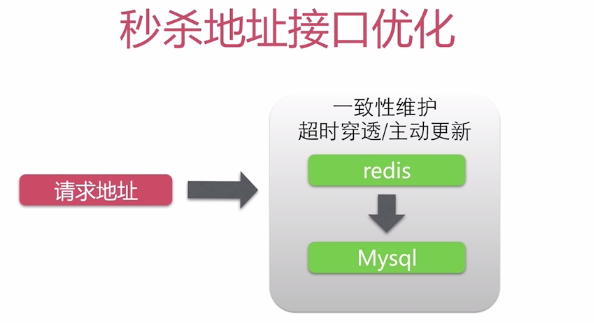
访问一次内存（Cachline）大约10ns

**秒杀地址接口分析**

无法使用CDN缓存

适合服务端缓存：redis等

一致性维护成本低



**秒杀操作优化分析**

无法使用CDN缓存

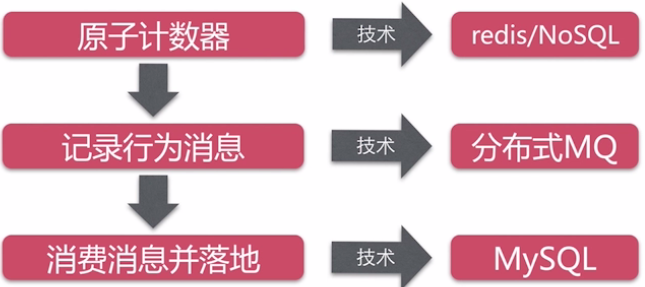
后端缓存困难：库存问题

一行数据竞争：热点商品

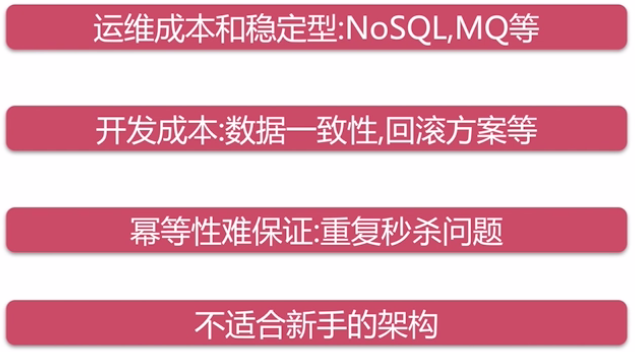
## 1-2 高并发优化分析（下）

其他方案分析

执行秒杀



**成本分析**



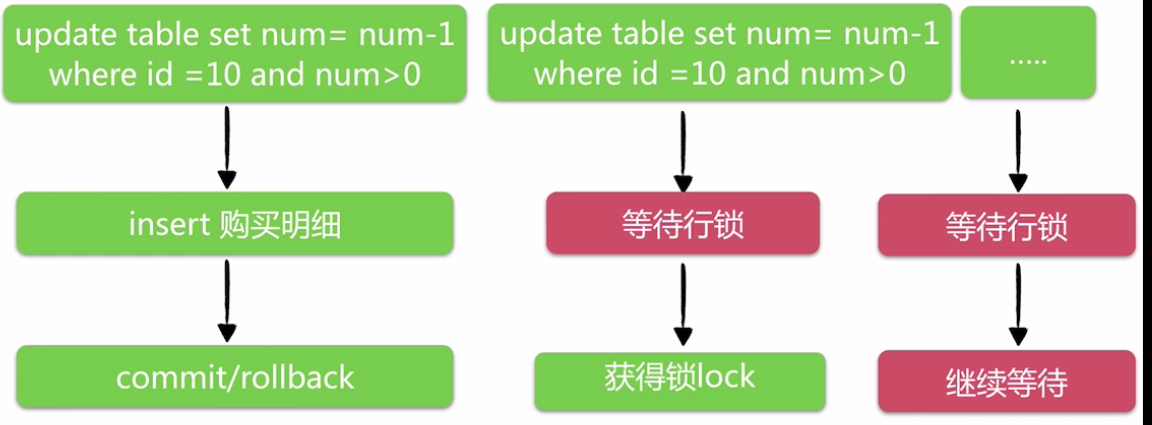
**为什么不用Mysql解决呢？**

认为Mysql低效

**MySql真的低效？**

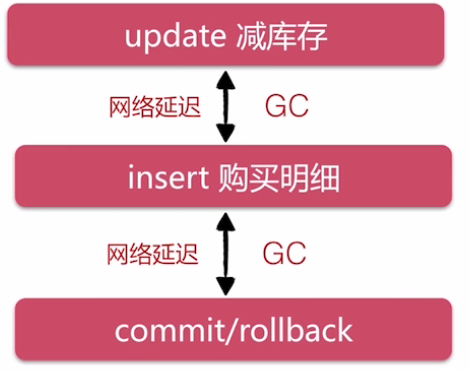
做个测试：同一个id执行update减库存 **每秒查询率QPS**

**Java控制事务行为分析**



**瓶颈分析**

**备注：java GC（Garbage Collection，垃圾收集，垃圾回收）机制。**



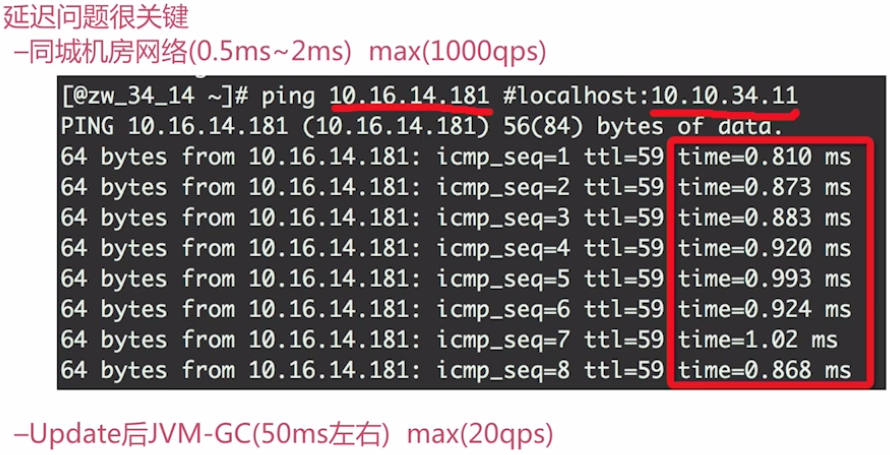
**优化分析**

行级锁在Commit之后释放

优化方向减少行级锁持有时间

**1）网络延迟分析**

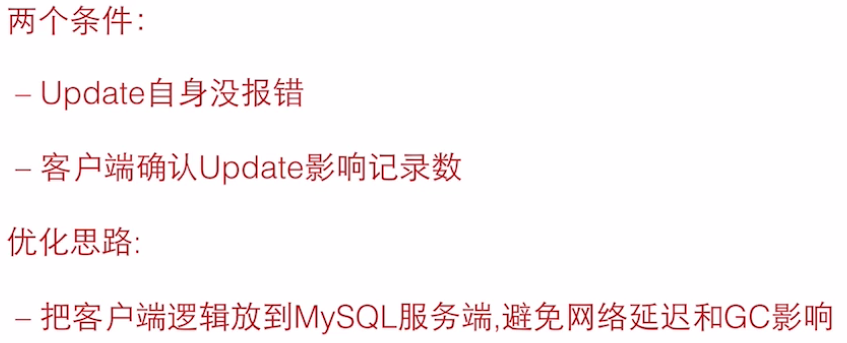
**(同城机房)**



**异地机房**



**如何判断Update更新库存成功？**



**如何放到MySql服务器端？**



**优化总结**

前端控制：暴露接口，按钮防重复

动静态数据分离：CDN缓存，后端缓存

事务竞争优化：减少事务锁时间

# 第2章 redis后端缓存优化编码

## 2-1 redis后端缓存优化编码

**Redis 缓存服务器**

Redis是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。

redis是一个**key-value存储系统**。

Redis 是一个高性能的key-value**数据库**。

Redis支持**主从同步**。

# 第3章 并发优化

## 3-1 秒杀操作-并发优化-1

**回顾事务执行**



**简单优化**（减少rowLock的持有时间）



**深度优化**

事务SQL在Mysql端执行（**存储过程**）

## 3-2 秒杀操作-并发优化-2

--秒杀执行存储过程

DELIMITER $$ -- console; 转化为 $$

--定义存储过程

--参数：in输入参数，out 输出参数

--row\_count():返回上一条修改类型sql（delete,insert,update）的影响行数

--row\_count(): 0:未修改数据； >0：表示修改的行数； <0 :sql错误/未执行修改sql

CREATE PROCEDURE `seckill`.`execute\_seckill`(in v\_seckill\_id bigint,in v\_phone bigint,

in v\_kill\_time timestamp,out r\_result int)

BEGIN

DECLARE insert\_count int DEFAULT 0;

START TRANSACTION;

insert ignore into success\_killed

(seckill\_id,user\_phone,create\_time)

values (v\_seckill\_id,v\_phone,v\_kill\_time);

select row\_count() into insert\_count;

IF (insert\_count = 0) THEN

ROLLBACK;

set r\_result = -1;

ELSEIF(insert\_count<0) THEN

ROLLBACK;

SET r\_result = -2;

ELSE

update seckill

set number = number - 1

where seckill\_id = v\_seckill\_id

and end\_time > v\_kill\_time

and start\_time < v\_kill\_time

and number > 0;

select row\_count() into insert\_count;

IF (insert\_count = 0) THEN

ROLLBACK;

SET r\_result = 0;

ELSEIF(insert\_count<0) THEN

ROLLBACK;

SET r\_result = -2;

ELSE

COMMIT;

set r\_result = 1;

END IF ;

END IF ;

END;

$$

-- 存储过程定义结束

DELIMITER ;

--定义变量

SET @r\_result = -3;

-- 执行存储过程

CALL execute\_seckill(1003,18862141550,now(),@r\_result);

-- 获取结果

SELECT @r\_result;

-- 存储过程：

-- 1.存储过程优化：优化事务行级锁的持有时间

-- 2.不要过度依赖存储过程

-- 3.简单的逻辑可以应用存储过程

-- 4.QPS：一个秒杀单接近6000/qps

## 3-3 秒杀操作-并发优化-3

**Mybatis配置存储过程**

# 第4章 系统部署架构

## 4-1 系统部署架构

### 系统可能用到哪些服务？

CDN:

Jquery和bootstrap一些依赖直接使用的是公网的CDN,我们开发的静态js,甚至我们的详情页都会把它做静态处理，然后推送到cdn（内容分发网络，用来加速用户获取内容的方式，其中我们理解**CDN,**主要一个关键概念就是，用户在cdn获取的数据，不需要再访问我们的服务器，像一些静态化的数据，还有我们把它强制作为静态化像秒杀详情页，把动态数据分离，把静态数据组织成一个html，推送到sdn上去，来达到降低我们服务器请求量的目的）

WebServer.Nginx+Jetty

Nginx（**HTTP和反向代理服务器**）:集群化的部署在多个服务器上，用来做我们的http服务器，同时它还会帮我们后端的Jetty/Tomcat这样的servlet容器做反向代理

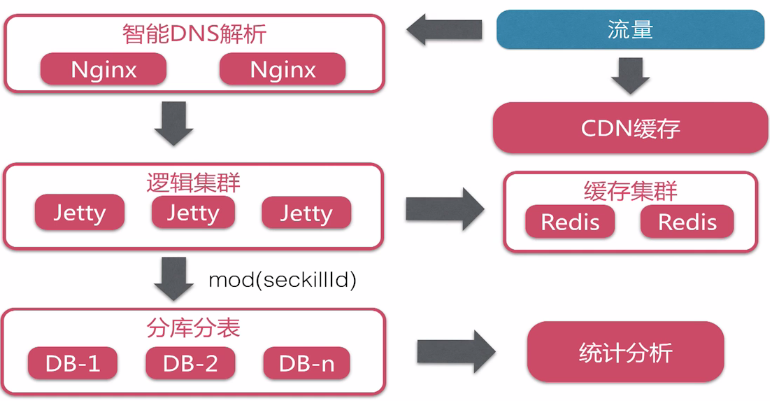
Redis

做服务器缓存，用于对热点数据的快速存取

MySql

借助MySql的事务，来达到秒杀数据的一致性和完整性

### 大型系统部署架构是什么样的？



### 可能参与的角色？

开发：前端+后端

测试

DBA

运维

**集群化部署的感性认识**

# 第5章 课程总结

## 5-1 8-课程总结

### 1）数据层技术回顾

数据库设计和实现

MyBatis理解和使用技巧

MyBatis整合Spring技巧

### 2）业务层技术回顾

业务接口设计和封装

SpringIOC配置技巧

Spring声明式事务使用和理解

### 3）WEB 技术回顾

Restful接口运用

SpringMVC使用技巧

前端交互分析过程

Bootstrap和JS使用

### 4）并发优化

系统瓶颈点分析

事务、锁、网络延迟理解

前端、CDN、缓存等理解使用

集群化部署