# 第一篇 J2SE

## 1.1 String

String类为什么是final

主要是为了“效率” 和 “安全性” 的缘故

String str = new String("xyz");创建了几个字符串对象(2)？   
答：两个对象，一个是静态区的"xyz"，一个是用new创建在堆上的对象。

重写equals为何要重写hashCode

## 1.2 Collection

### ConcurrentHashMap

同步的数据结构，例如concurrentHashMap的源码理解以及内部实现原理，为什么他是同步的且效率高

### 存储原理

Map或者HashMap的存储原理

由数组+链表的一个结构组成

## 1.3 Thread

**线程同步**

### ThreadLocal

为每个使用该变量的线程提供独立的变量副本，所以每一个线程都可以独立地改变自己的副本，而不会影响其它线程所对应的副本

### Synchronized

### Atomicinteger

### Volatile

### ReentrantLock

ReentrantLock 和synchronized的区别

常用的线程池模式以及不同线程池的使用场景

3. newFixedThreadPool此种线程池如果线程数达到最大值后会怎么办，底层原理。

ThreadPool的深入考察；

### BlockingQueue的使用

定时线程的使用

## 1.4 NIO

* 非阻塞: 避免创建大量线程，相对于多线程开销较小。
* 基于事件驱动: 线程上下文的切换都是有意义的。
* 复用单一长连接，并使用线程池并发处理请求，减少握手和提高并发效率，性能较好。长连接双向异步推送，比轮询、阻塞（线程无效切换的开销）效率高。

Netty: 异步高性能的通信框架,RPC工具，基础通信组件。

## 1.5 反射

Class.forName和classloader的区别

Class.forName(xxx);的作用是要求JVM查找并加载指定的类。

**类加载机制**

Class的装载分了三个阶段，loading，linking和initializing。

Loading:JVM会在 Java 堆中也创建一个 java.lang.Class 类的对象。

Linking：分配内存

Initializing:初始化类变量和其他资源

## 1.6 JVM

1. Java的内存模型, 每个区域可能造成内存的溢出

**2. jvm性能调优**

* 控制GC的行为. 频繁GC导致性能下降,因此经常会根据系统运行的程序的特性来更改GC行为
* 调整JVM堆栈大小.xmx xxx
* 控制JVM线程的内存分配.xss

4. 介绍GC 和GC Root不正常引用。

6. jvm 如何分配直接内存， new 对象如何不分配在堆而是栈上，常量池解析

7. 数组多大放在 JVM 老年代（不只是设置 PretenureSizeThreshold ，问通常多大，没做过一问便知）

 老年代中数组的访问方式

谁会被 GC ，什么时候 GC,  如果想不被 GC 怎么办

9. GC 算法，永久代对象如何 GC ， GC 有环怎么处理

11. 12. 如果想在 GC 中生存 1 次怎么办

## 1.7 面向对象

**接口和抽象类的区别是什么？**

1.abstract是一种继承关系，一个类只能使用一次继承关系。但是，一个类却可以实现多个interface。

2.在abstract class 可以有自己的数据成员，也可以有非abstarct的成员方法

3.abstract class和interface所反映出的设计理念不同。其实abstract class表示的是"is-a"关系，interface表示的是"like-a"关系。

**什么是java序列化，如何实现java序列化?**

序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。序列化是为了解决在对对象流进行读写操作时所引发的问题

# 第二篇 J2EE

## 2.1 Web

### HTTP协议，GET和POST 的区别

1.根据HTTP规范，GET用于信息获取，而且应该是安全的和幂等的。POST表示可能修改变服务器上的资源的请求

2.GET请求的数据会附在URL之后,如：login.action?name=hyddd。POST把提交的数据则放置在是HTTP包body中

3.GET方式提交的数据最多只能是1024字节，理论上POST没有限制，可传较大量的数据

### MVC分层结构

分层设计1.有利于分工 2.复用3.好扩展，好维护

Listener vs Filter

filter:过滤请求。过滤字符编码，逻辑判断，如是否登录，有没有权限访问页面

listener：它也是随web应用的启动而启动，只初始化一次，随web应用的停止而销毁。主要作用是： 做一些初始化

## 2.2 Spring

spring如何实现事物管理的

### 2.2.1.MVC

springMVC的原理

请求🡪 DispatcherServlet🡪HandlerMapping🡪Controller

**2.IOC:** spring作为第三方管理资源对象

1)资源集中管理，实现资源的可配置和易管理。**(面向接口编程,扩展性高)**。

2)降低了资源双方的耦合度。让开发人员更多的关注业务代码

### 2.2.2 AOP

1)aspectj静态代理

2)CGLIB动态代理/JDK动态代理: JDK针对接口

## 2.3 Hibernate

Hibernate的缓存机制

Hibernate的缓存包括Session的缓存和SessionFactory的缓存。

Session的缓存根据ID来缓存对象，也就是Session的get、load操作时。应用场景太单一，系统中大量的列表式查询缓存起不到作用系统中通过ThreadLocal在线程中重用Session，每个线程可能需要大量处理不用的业务逻辑，缓存命中率很低。

SessionFactory的缓存是应用级。EHCache是Hibernate中的二级缓存插件，使用Hibernate的系统可以直接使用EHCache缓存。

## 2.4 Web service

Web service是什么

WebService，顾名思义就是基于Web的服务。基于HTTP传输协议的程序。接收和响应外部系统的某种请求。从而实现远程调用。

## Zookeeper

Zookeeper是一个高性能，分布式的，开源分布式应用协调服务。分布式应用可以基于它实现同步，配置管理，集群管理，名空间

## Freemarker

在java领域，表现层技术主要有三种：jsp、freemarker、velocity。

优点：1.不能写java代码，实现严格的mvc分离2.可使用表达式语言

# 第三篇 软件架构

**软件架构优劣的衡量要素：**1.性能2.可用性 3.伸缩性4.扩展性5.安全性

## 3.1性能

### 1.1性能优化指标

1. 并发数：能同时处理的用户数。
2. 响应时间：发出请求到收到相应数据的时间
3. 吞吐量：系统单位处理的请求数。QPS（query per s）
4. 性能计数器：包括System Load、对象和线程数、内存、CPU、磁盘、网络IO

### 1.2性能测试方法

1. 性能测试：性能预期
2. 负荷测试：找到性能安全临界线
3. 压力测试：超过安全符合继续施压，获得最大压力承受能力

### 1.3 Web前端性能优化

1.减少Http请求:合并css、js

2.使用浏览器缓存:生成新的js时，只需更新Html中引用

3.启用压缩：减少数据传输

4.CSN加速：缓存静态资源，如图片、文件、CSS、JavaScript脚本、静态网页等。

5.反向代理

### 1.4应用服务器的优化

1. 缓存：加快数据读取，降低DB访问压力。网站遇到性能瓶颈，第一个想到的解决方案
2. 异步操作：加快请求响应及实现削峰。需要修改业务流程进行配合。
3. 使用集群：使用负载均衡，有更好的响应延迟，提高吞吐能力。
4. 代码优化：1）多线程,最大发挥CPU和处理IO阻塞 2）资源复用，比如单例和线程池 3）数据结构 4）垃圾回收

## 3.2伸缩性

**不同功能物理分离实现伸缩**

分割的粒度可以非常小，甚至可以一个关键网页部署一个独立服务，比如对电商非常重要的商品详情页面，商铺页面，搜索列表页面，每个页面都可以独立部署，专门维护

**单一功能通过集群规模实现伸缩:**应用服务器，搜索，缓存，数据库

### 缓存

### 数据存储服务

1）主从分离

2）分表分库

3)NoSql

HBase: MongoDB:

## 3.3扩展性

**纵向拆分**：按功能模块拆分为多个应用系统，新增业务可以设计成独立的web系统。

**横向拆分：**拆分出复用的基础公用业务，独立部署为分布式服务。

分布式系统集成方式：1）分布式消息队列2）分布式服务

### 消息中间件

优势：事件驱动（订阅发布）可一对多推送消息，异步，消息可靠

**消息队列**

通过消息分解系统解耦性，不同子系统处理同一消息

**使用场景**

1.系统整合：同构/异构系统整合，分布式环境中

2.降低模块间耦合：

1)扩展性：模块交互点统一，变化集中在一点，不用逐个模块修改,垂直切分容易2)可用性：AB,A宕机 3)性能：系统差不干扰B

3. 异步:1.推消息 2.流量削峰(据同步:web应用->缓存，搜索，db)

Kafka日志处理

ActiveMQ:基于JMS

### 分布式服务

通过接口分解系统解耦性，不同子系统通过相同的接口描述进行服务调用。

**SOA**

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，

1.扩展性 2提高开发和维护效率 3.伸缩性?

**面临的挑战**

1. 远程调RPC用2.服务管理和配置管理3.负载均衡器单点故障4.实时管理集群5.分布式事务

**Restful**

规范

**Atom**

协议

## 3.4可用性

实现高可用架构的手段是数据冗余备份和服务失效转移。

服务器：由无状态服务器和有状态的Session服务器组成。

# 第四篇 DB

## 4.1 MySQL

**一条sql执行过长时间，如何优化**

1.一般**联表**消耗更多系统资源，还可能使索引失效，涉及多表的联表或者子查询

1）看是否能进行业务拆分2）相关字段冗余或者3）创建汇总表。4)分解关联查询，单表查询之后的结果进行字段整合  
2.需要联表查询

1)那么考虑对相对应的查询条件做索引。加快查询速度  
3、针对数量大的表进行历史表分离（如交易流水表）  
4、读写分离，降低主库压力

**MyISAM**

1. 不支持事务、外键和行级锁,适合小项目

优点:1.读多写少的项目，效率高2.没有磁盘碎片，节省空间

**存储引擎**

影响引擎性能的因素：数据大小，I/O请求量，主键还是二级索引

**InnoDB：**优先选择InnoDB，除非用到InnoDB不具备的特性，没有其他办法替代。

1. 事务2.热备份3.崩溃恢复4.聚簇索引

**聚簇索引**:按物理位置顺序存放，引擎缓存索引和数据

插入慢，排序快。非聚簇索引（二级索引）则没有按序存放，需要额外消耗资源来排序。

**MyISAM：**一般读写性能高，但随着应用压力上升，各种锁争用，崩溃数据丢失问题

1. MyISAM引擎缓存索引，而不缓存数据

## 4.2 MySQL性能优化

索引优化 查询优化 库表结构优化

### 索引优化

**索引**

1）索引大大减少了服务器需要扫描的数据量

2）索引可以帮助服务器避免排序和临时表

3）索引可以将随机I/O变成顺序I/O

一般oracle会判断,使用索引返回的数据量不超过总表的5%时,才走索引..否则就是全表扫描

### 查询优化

1.2.1优化数据访问

查询性能低下最基本原因是访问数据太多。

**1)应用服务**

* 确认应用服务是否检索大量查过需要的数据（访问太多的行或列）

加limit

* 返回所有列或者多表关联返回需要的列
* 重复查询相同的数据：缓存

**2)MySQL服务器**

确认MySQL服务器是否在分析大量超过需要的数据行

**响应时间**

**扫描行数和返回行数**

**扫描行数和访问类型**

ALL-index-range-ref

一般MYSQL能够使用如下三种方式应用where条件记录，从好到坏依次为

1) 在存储索引层，在索引中使用where条件来过滤不匹配的记录。

2) 在mysql服务器层, 扫描覆盖索引过滤并返回记录。无需在回表查询记录(Using index)。

3) 在mysql服务器层，先从数据表读出记录然后过滤(Using Where)

如果发现查询需要扫描大量的数据但返回少数的行，那么通常可以尝试下面的技巧：

1)使用索引覆盖扫描，把所有需要的列都放到索引中，这样存储引擎无须返回表获取对应行就可以返回结果了。

2)改变库表结构，使用单独的汇总表。

3)重写这个复杂的查询

重构查询方式

**1)复杂查询还是简单查询**

MySQL吞吐量：10w/s查询

扫描内存：100+w行/s

**2)分解关联查询**

**单表 vs 联表**

看具体业务场景。没到阀值前，都可以，有的业务也许永远到达这个阀值。

高并发高读环境：单表or冗余，可做缓存。建覆盖索引。

联表后期扩展和维护工作量大，简单说dao层的复用率低。

1.2.3 优化特定类型的查询

**Count（）**

1.MyISAM 2. 索引覆盖扫描3.汇总表 4.增加memcached

**关联查询**

1. ON或using列有索引,一般在连表顺序第二个create index。
2. Group by和Order By之设计一个表的列

**子查询：**尽量 关联子查询替代，当前MySQL版本需要

GROUP BY和DISTINCT

LIMIT：一般ORDER BY …LIMIT

### 库表结构优化

选择合适的数据类型

**更小的通常更好**：占用更少的磁盘、内存和CPU缓存，CUP处理周期

**简单就好**：简单类型CPU处理快。例如整形比字符操作代价低

**尽量避免NULL**：NULL使索引更复杂，需要更多存储空间

范式和反范式

范式需要联表，代价昂贵。还可能使一些索引策略无效，比如覆盖索引。

缓存表和汇总表

冗余是在一张表。

汇总表：group by 或count

缓存表：存储从schema中获取比较慢的数据表。

例如主键+部分列。用缓存或solr(搜索效率更高)

**优点**：避免复杂，昂贵的实时跟新操作

[**MYSQL数据库优化**](http://blog.csdn.net/dongnan591172113/article/details/52174619)

不同业务场景优化的策略不同，比如数据量、访问量（QPS每秒查询率）、并发量、读写比例、数据是否快速增长、数据需不需要强一致

优化SQL和索引-> 缓存-> 主从复制或主主复制-> 分区表-> 垂直拆分-> 水平切分-> 选

**避免扫描大量的数据但只返回少量的行**

1.覆盖索引

2.改变表结构，例如汇总表

3.重写复杂的查询

4.定期备份，删除

**减少索引和数据的碎片**

Optimize table;

## 4.3 NOSQL

现代互联网应用，关系数据库的很多主要特性却往往无用武之地

1.数据库事务一致性需求

2.数据库的写实时性和读实时性需求

3.对复杂的SQL查询，特别是多表关联查询的需求

去掉关系型数据库的两大重要基础：以关系代数为基础的结构化查询语句（SQL）和事务一致性保证（ACID）

1. 高并发读写2)海量数据的高效率存储和访问3)高可用性和可伸缩性

### 4.3.1 Redis

redis是一款开源的高性能key-value数据库，拥有丰富的键值储存类型，并提供多种语言的API。与一般数据库不同，redis是使用内存作为主存，周期性的将数据写到硬盘上。因为redis读写数据都使用内存，所以它的速度是非常快的，很适合我们来存放一些临时性的数据。

Redis vs Memcached

1.使用简单的key-value存储的话，Memcached的内存利用率更高

2. 由于Redis只使用单核，而Memcached可以使用多核， Memcached相对性能更高

3.拥有更多的数据结构和并支持更丰富的数据操作。

4.集群管理的不同

# 第五章 服务器

## Apache和tomcat

apache是web服务器，专门提供HTTP服务。处理静态内容，比如HTML/图片。

tomcat是java应用服务器，servlet容器，是apache的扩展

## Nginx与apache

轻量级，异步非阻塞，高并发环境保持低资源低消耗的高性能。而apache是阻塞的。但比较稳定。

1.反向代理，缓存静态文件 2.负载均衡

**设计原则**

**工厂方法模式的优点**

低耦合、高内聚，符合开放封闭原则

1开闭原则2单一职责 2依赖倒转

# 第六篇 Linux