**NIO**

1.非阻塞: 避免创建大量线程，相对于多线程开销较小。

2.基于事件驱动: 线程上下文的切换都是有意义的。

3.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。

**Spring**

**MVC**

**IOC:**

1.资源集中管理，实现资源的可配置和易管理。**实现面向接口编程,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

**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

**性能**

**性能优化指标**

1. 响应时间：发出请求到收到相应数据的时间
2. 并发数：能同时处理的用户数。
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

**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

**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

**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

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

根据不同业务场景有不同的侧重，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？对于一些典型的场景可能会有成熟的解决方案。

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-> 选择不同的存储引擎

**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

Cobar：分布式数据库访问代理

3)NoSql

HBase: MongoDB:

**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

**消息队列**

通过消息分解系统解耦性，不同子系统处理同一消息

**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

**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：AB，A宕机、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

**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。