高并发解决方案

从多个维度谈前端→网关→后端→运维→监控→硬件
前端：
1. 动静分离区别　将静态资源和动态资源完全分开部署，
需要将静态资源存入到第三方 cdn 产商中 例如
阿里云 oss、七牛云 等 价格大概在每 GB/0.5 元左右
CDN 会在全国各地都有节点，遵循用户就近原则访问，
从而提高用户访问体验。
2. 需要对引入的静态资源做压缩 例如 生成.min 文件
减少占用宽带资源
3. 对静态资源做二次压缩
CDN 缺陷：刷新 cdn 缓存缺陷

在外网中传输我们的数据 有带宽限制。
基本上我们的网站 一个网页 静态资源是占用带宽 80%
前后端分离区别　前端开发 和 后端分开开发模式 开发模式
动静分离区别　将静态资源和动态资源完全分开部署

后端：
从 JVM、数据库、缓存、MQ、网关层面、硬件层面等多个维度来进行回答
1. JVM 层面 需要对 JVM 实现参数调优 减少 gc 回收频率 减少 stw 问题
根据服务器配置和业务需求，选择合适的 GC 垃圾收集器 例如 ZGC、G1 等
2. 使用 redis 缓存减少对数据库的访问压力（mysql 与 redis 数据一致性问题、雪崩、穿透、击穿）
3.数据库层面 定位慢查询 sql 语句优化 索引优化 索引遵循最左匹配原则 分表分库 使用范围、一致性 hash 分片
4.利用 MQ 实现流量削峰问题，MQ 消费者根据自身合适能力来进行消费，为了避免消息堆积 建议 MQ 消费者集群和批量消费消息，整合 k8s 当流量突然大的时候快速扩容与缩容。
5.利用网关层面对服务器接口实现保护，通过限流配置预防突发大流量对系统的冲击。限流规则：频率 底层算法采用 令牌桶、漏桶、滑动窗口算法
限流框架：谷歌 guava、阿里巴巴 sentinel、基于 redis 等
整合 java 中 getaway 或者 nginx+lua 第十期--手写限流算法
6. 当系统出现过多的异常或慢请求，上游服务则需开启熔断 当下游的服务因为某种原因导致服务不可用或响应过慢时，上游服务为了保证自己整体服务的可用性，不再继续调用目标服务，直接返回。当下游服务恢复后，上游服务会恢复调用　底层采用滑动窗口算法实现
服务降级、限流、熔断 隔离机制
7. 使用 openresty/nginx 对服务器实现集群 保证服务器的高可用
Jvm、mysql、tomcat、redis 优化

运维层面：

1.使用 k8s 部署微服务项目，在高并发的情况下实现弹性扩容与缩容
1. 通过整合 SkyWalking 分布式追踪系统 实现服务监控 当服务器发生了 cpu 使用率飙高、内存溢出 等问题 能够提前预防告警,底层是基于 agent 代理实现。
2. 利用分布式追踪系统排查 RPC 远程调用过程中某链发生错误问题，底层采用 spanid、全局 id 记录。
3. 构建分布式主动告警系统，当系统发生错误采用公众号消息模板主动将错误推送给开发者，开发者无需登录服务器端查看错误日志。
4. 构建分布式日志采集系统 elk+kafka，采集分布式系统的日志

访问层面：
1. 高并发的情况下需要接入第三方 DDOS 防御系统 不能暴露用户真实的 IP 地址。
2. 通过 ddos 系统可以配置防御 （图形验证码）防止机器模拟请求攻击等

硬件层面：
1. 硬盘选择（固态硬盘）
2. Cpu 核心数 物理机器服务器 64 核 128 线程
3. 内存
我们项目的架构采用云原生架构
第十一期重点-*-

参考资料：
http://static.mayikt.com/jquery-1.11.1.min.js?t=2118bbe3-b1d9-44f7-9eb2-eb629674f7de

云原生架构
云原生架构--k8s
云原生架构--springcloud

断路原理
Raft、zab 区别