

前端全链路性能优化实战



扫码试看/订阅

《前端全链路性能优化实战》视频课程

第五章：服务端和网络优化

5.1 CDN：如何合理配置 CDN 缓存？

5.2 DNS：主流的 DNS 优化方法有哪些？

5.3 HTTP：如何减少 HTTP 请求数？

5.4 Cookie：减少 Cookie 大小的策略和益处

5.5 服务器：缓存配置和优化方案

第五章：服务端和网络优化

5.6 服务器：如何开启和配置 gzip 压缩

5.7 HTTPS：如何开启全站 HTTPS？

5.8 HTTP/2：升级 HTTP/2 的好处有哪些？如何升级？

5.1 CDN：如何合理配置 CDN 缓存？

CDN 定义

- 内容分发网络（Content Delivery Network，简称 CDN）是利用最靠近每一位用户的服务器，更快、更可靠地将文件发送给用户分发网络。

CDN 优点

- 提速：会给用户指派较近、较顺畅的服务器节点，将数据传输给用户
- 低成本：服务器被放到不同地点，减少了互连的流量，也降低了带宽成本
- 高可用度：当某个服务器故障时，自动调用邻近地区的服务器

CDN 回源

- 回源是指浏览器访问 CDN 集群上静态文件时，文件缓存过期，直接穿透 CDN 集群而访问源站机器的形为。

CDN 缓存

- 三级缓存：浏览器本地缓存、CDN 边缘节点缓存、CDN 源站缓存
- 缓存设置：缓存时间设置的过短，CDN 边缘节点缓存经常失效，导致频繁回源，增大了源站负载，访问也慢；缓存时间设置的过长，文件更新慢，用户本地缓存不能及时更新；所以结合业务情况而定
- 不同静态资源类型缓存时间：
 - HTML：3分钟
 - JS、CSS：10分钟、1天、30天

CDN 缓存设置

```
http{
    ...
    server {
        listen 80;
        server_name 123.com
        location ~* \.(jpg|jpeg|gif|bmp|png){
            expires 30d;
        }
    }
}
```

CDN 灰度发布

- 原理：在部分地区、部分地区的部分运营商优先发布静态资源，验证通过后，再进行全量发布。
- 实施：域名方面，设置特殊 VIP 解析至要灰度的城市、运营商；源站机器方面，给灰度的城市、运营商配置单独源站机器；灰度的城市、运营商解析至这些特有机器上。

CDN 大促备战

- 增加机房带宽
- 增加运营商流量
- 灾备：CDN 应用缓存时间由10分钟设置成1个小时，大促后恢复。

5.2 DNS：主流的 DNS 优化方法有哪些？

什么是 DNS

- 域名系统（Domain Name System）是将网站域名和IP地址相互映射的一个分布式数据库，能够更方便的访问互联网。

客户端处理

- Android DNS 模块 (OkHttp)
 - 支持 HTTP/2, HTTP/2 通过使用多路复用技术在一个单独的 TCP 连接上支持并发, 通过在一个连接上一次性发送多个请求来发送或接收数据。
 - 如果 HTTP/2 不可用, 连接池复用技术也可以极大减少延时
 - 支持 GZIP, 可以压缩下载体积
 - 响应缓存可以完全避免网络重复请求。
 - 如果服务器配置了多个 IP 地址, 当第一个IP连接失败的时候, OkHttp 会自动尝试下一个 IP。

客户端处理

- iOS DNS 模块（自研）
 - App 启动时，缓存所有可能要用到的域名 IP，同时异步处理，客户端无需得到缓存结果
 - 如果 Cache 中有此域名的缓存，直接返回缓存的 IP
 - 如果缓存中没有此域名，则重新向 HTTPDNS SERVER 进行申请，结果会在此回调中返回

前端处理

- 浏览器并发数限制，分布设置成多个域名
 - 用户访问：Java、PHP 等 API 接口
 - 页面和样式：HTML/JS/CSS
 - 图片：jpg、png、gif 等

5.3 HTTP：如何减少 HTTP 请求数？

如何减少 HTTP 请求数

- CSS Sprites
- 图片使用 DataURI、Web Font
- JS/CSS 文件合并
- JS/CSS请求 Combo
- 接口合并
- 接口存储 LocalStorage
- 静态资源存储 LocalStorage

5.4 Cookie：减少 Cookie 大小的策略和益处

减少 Cookie

- 策略：
 - 主站首页设置白名单
 - 定期删除非白名单 Cookie
- 好处：
 - 减少页面间传输大小
 - 对 Cookie 进行有效管理

5.5 服务器：缓存配置和优化方案

Expires

- 定义
 - 响应头包含日期/时间，即在此时候之后，响应过期
 - 无效的日期，比如0, 代表着过去的日期，即该资源已经过期
 - 如果在 Cache-Control 响应头设置了 "max-age" 或者 "s-max-age" 指令，那么 Expires 头会被忽略
- 语法：Expires: Tue, 17 Dec 2019 07:01:44 GMT
- 实例：Expires: Tue, 17 Dec 2019 07:14:29 GMT

Cache-Control

- 定义
 - 通用消息头字段，通过指定指令来实现缓存机制。缓存指令是单向的，这意味着在请求中设置的指令，不一定被包含在响应中。
- 语法：
 - Cache-Control: max-age=<seconds>【设置缓存存储的最大周期，超过这个时间缓存被认为过期(单位秒)。与Expires相反，时间是相对于请求的时间。】
- 实例：
 - Cache-Control: max-age=600

ETag

- 定义
 - HTTP 响应头是资源的特定版本的标识符。这可以让缓存更高效，并节省带宽，因为如果内容没有改变，Web 服务器不需要发送完整的响应。而如果内容发生了变化，使用 ETag 有助于防止资源的同时更新相互覆盖。
 - 如果给定 URL 中的资源更改，则一定要生成新的 ETag 值。因此 ETags 类似于指纹，也可能被某些服务器用于跟踪。比较 ETags 能快速确定此资源是否变化，但也可能被跟踪服务器永久存留。
- 语法：ETag: "<etag_value>"
- 实例：ETag: "5c6ccc12-1d45"

Last-Modified

- The Last-Modified 是一个响应首部，其中包含源头服务器认定的资源做出修改的日期及时间。它通常被用作一个验证器来判断接收到的或者存储的资源是否彼此一致。由于精确度比 ETag 要低，所以这是一个备用机制。包含有 If-Modified-Since 或 If-Unmodified-Since 首部的条件请求会使用这个字段。
- 语法：Last-Modified: <day-name>, <day> <month> <year>
<hour>:<minute>:<second> GMT
- 实例：Last-Modified: Wed, 20 Feb 2019 03:40:02 GMT

Date

- 定义
 - Date 是一个通用首部，其中包含了报文创建的日期和时间。
- 语法
 - Date: <day-name>, <day> <month> <year> <hour>:<minute>:<second> GMT
- 实例
 - Date: Tue, 17 Dec 2019 07:08:41 GMT

Status

- 定义
 - HTTP 响应状态代码指示特定 HTTP 请求是否已成功完成。响应分为五类：信息响应(100-199)，成功响应(200-299)，重定向(300-399)，客户端错误(400-499)和服务器错误(500-599)。
- 实例
 - Status: 200

实例

Request URL:

<https://mycdn.com/bridge.js>

Response Headers:

Access-Control-Allow-Origin: *

Age: 346

Cache-Control: max-age=600

Connection: keep-alive

Content-Encoding: gzip

Content-Length: 2477

Content-Type: application/javascript

Date: Tue, 17 Dec 2019 07:08:41 GMT

ETag: "5c6ccc12-1d45"

Expires: Tue, 17 Dec 2019 07:14:29 GMT

Last-Modified: Wed, 20 Feb 2019 03:40:02 GMT

Server: nginx

Strict-Transport-Security: max-age=360

5.6 服务器：如何开启和配置 gzip 压缩

好处

- 对文本进行压缩（HTML/CSS/JS）
- 而对非文本不压缩（jpg/gif/png）
- 压缩比约50% - 70%

配置方法

- Nginx 配置: nginx.conf 文件增加 gzip on
- Apache 配置: AddOutputFilterByType 和 AddOutputFilter

生效检测

- Response header 查看是否有 Content-Encoding: gzip, 代表服务端已开启 gzip。

5.7 HTTPS：如何开启全站 HTTPS？

基本概念

- HTTPS，超文本传输安全协议（HyperText Transfer Protocol Secure）是一种通过计算机网络进行安全通信的传输协议。
- HTTPS 是经 HTTP 进行通信，但利用 SSL/TLS 进行数据加密。
- HTTPS 主要目的是提供对服务器身份认证，保护数据隐私和完整性。

工作原理

- 浏览器发起 HTTPS 请求
- 传输证书
- 浏览器解析证书
- 传送加密信息
- 服务器解密信息
- 传输加密后的信息
- 浏览器解密信息

优点

- SEO
- 安全

实施

1. 经销商购买证书

- GoGetSSL
- SSLs.com
- SSLmate.com

实施

2. 本地测试证书

1) 本地 HomeBrew 安装

- `brew install mkcert`

2) 本地安装根证书

- `$ mkcert ---install`

3) 本地生成签名

- `$ mkcert 123.com //生成123.com的证书`

实施

2. 本地测试证书

4) 本地 nginx 配置

```
server {  
    listen      443 ssl; # 启用HTTPS  
    server_name 123.com; # 这里是刚才的域名  
  
    ssl_certificate      123+3.pem;  
    ssl_certificate_key  123+3-key.pem;  
  
    ...  
}
```


5.8 HTTP/2：升级 HTTP/2 的好处和方法

HTTP/2 概念

- HTTP/2（超文本传输协议第2版，最初命名为 HTTP 2.0），简称为 h2（基于 TLS/1.2或以上版本的加密连接）或 h2c（非加密连接），是 HTTP 协议的第二个主要版本。

HTTP/2 优点

- 采用二进制格式传输数据
- 多路复用，允许通过一个 HTTP/2 连接发起多个请求
- 对 Header 头压缩（Header Compression），传输体积小
- 服务端推送（Server Push），服务端能够更快的把资源推送给客户端

HTTP/2 站点的优势

- 可以降低服务器压力
- 提升网站访问速度
- 保护网站安全

在 Nginx 上启用 HTTP/2

1. 升级 OpenSSL

- `$ openssl version`

2. 重新编译


- `$ cd nginx-xxxx`
- `$./configure --with-http_ssl_module --with-http_v2_module`
- `$ make && make install`

在 Nginx 上启用 HTTP/2

3. 验证 HTTP/2

- 浏览器下查看有没有小绿锁

4. 浏览器请求截图

Name	Method	Status	Protocol	Scheme
 unionlog.js	GET	200	h2	https



扫码试看/订阅

《前端全链路性能优化实战》视频课程