

页面性能： 优化 HTTP 缓存

Web 缓存基础：术语、HTTP 报头和缓存策略

<https://juejin.cn/post/6844903472404365320>

HTTP 缓存机制

<https://juejin.cn/post/6844904116972421128>

- 强缓存：也称为本地缓存，不向服务器发送请求，直接使用客户端本地缓存数据
- 协商缓存：也称 304 缓存，向服务器发送请求，由服务器判断请求文件是否发生改变。如果未发生改变，则返回 304 状态码，通知客户端直接使用本地缓存；如果发生改变，则直接返回请求文件。

缓存策略

- 访问入口永远不缓存
- 资源文件
 - 设置强缓存并设置超长过期时间 Cache-Control: max-age=31536000
 - 资源文件更新时使用新的文件指纹

强缓存失效的方法

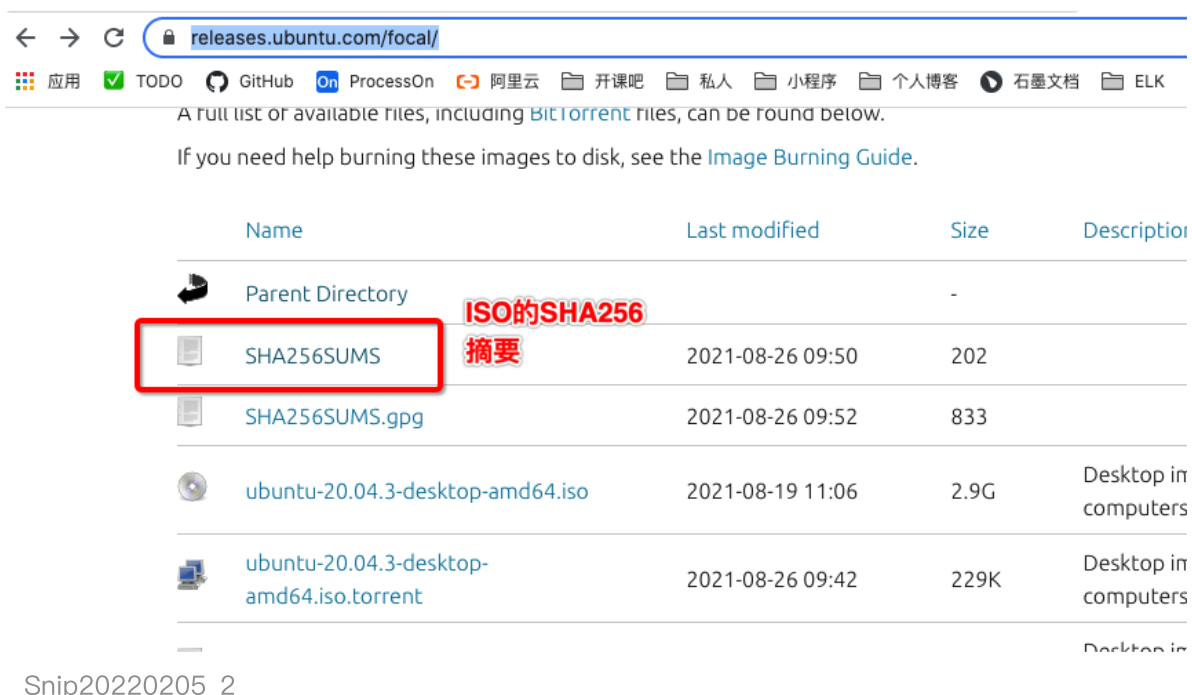
- 文件名加版本号
 - index.js?version=1
 - index.css?version=1

- 文件名哈希值 – 文件指纹

哈希值与文件指纹

如何判断 Ubuntu ISO 镜像的真伪

Ubuntu <https://releases.ubuntu.com/focal/>



The screenshot shows the Ubuntu releases page for the focal series. The URL bar displays 'releases.ubuntu.com/focal/'. Below the navigation bar, there is a table of available files. The file 'SHA256SUMS' is highlighted with a red box, and a red label 'ISO的SHA256摘要' is placed next to it. The table lists the following files:

Name	Last modified	Size	Description
Parent Directory		-	
SHA256SUMS	2021-08-26 09:50	202	
SHA256SUMS.gpg	2021-08-26 09:52	833	
ubuntu-20.04.3-desktop-amd64.iso	2021-08-19 11:06	2.9G	Desktop in computers
ubuntu-20.04.3-desktop-amd64.iso.torrent	2021-08-26 09:42	229K	Desktop in computers

摘要与哈希算法

摘要算法又称哈希算法、散列算法。摘要也称哈希值，表示输入任意长度的数据，都会输出固定长度的数据。通过摘要算法（比如 MD5 和 SHA-1）就可以得到该哈希值。

什么是哈希算法

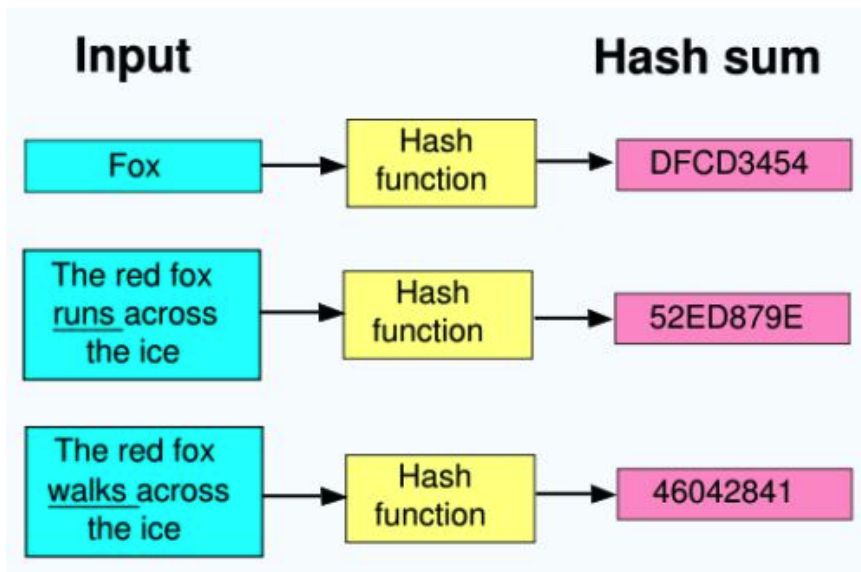


image-20220205133040516

- 不定长输入转定长摘要
- 满足雪崩效应
- 单项不可逆

常见的哈希算法

- MD5：输出 128bit 长度的二进制串
- SHA-1：输出 160bit 长度的二进制串
- SHA-256：输出 256bit 长度的二进制串
- SHA-512：输出 512bit 长度的二进制串

Webpack 与文件指纹

- **版本管理**：在发布版本时，通过文件指纹来区分 修改的文件 和 未修改的文件。
- **使用缓存**：未修改的文件，文件指纹保持不变，浏览器继续使用缓存访问。

文件指纹设置

我们在配置文件（`webpack.config.js`）中，通过占位符设置文件指纹。

占位符

名称	含义
[ext]	资源后缀名
[id]	文件标识符
[name]	文件名称
[path]	文件的相对路径
[folder]	文件所在的文件夹
[hash]	模块标识符的 hash
[chunkhash]	chunk 内容的 hash
[contenthash]	文件内容的 hash
[query]	文件的 query，例如，文件名？后面的字符串
[emoji]	一个随机的指代文件内容的 emoji

设置方法

```
// webpack.config.js

module.exports = {
```

JavaScript

```
// ...
entry: {
  app: "./src/app.js",
  index: "./src/index.js",
},
output: {
  filename: "[name][chunkhash:8].js",
  // ...path
},
};
```

利用 Express 中间件实现缓存

JavaScript

```
const http = require('http');
const express = require('express');
const ecstatic = require('ecstatic');
const history = require('connect-history-api-fallback');
const path = require('path')
const app = express();

app.use(history());
app.use(function (req, res, next) {
  console.log('url:', req.url)
  if (req.url === '/index.html') {
    res.set({
      'Cache-Control': 'public, max-age=0',
      'Expires': new Date(Date.now() + 0).toUTCString()
    });
  } else {
    res.set({
      'Cache-Control': 'public, max-age=31536000',
      'Expires': new Date(Date.now() + 31536000000).toUTCString()
    });
  }
  next();
});
app.use(
  express.static(path.resolve(__dirname, '../dist'),)
```

```
http.createServer(app).listen(6565, () => {  
  console.log('调试服务器启动 at: ', 6565)  
});
```