

Webpack

common.js实现

```
//webpack commonjs 执行完的时候
(function (modules) {
  // 对webpack文件进行缓存
  var installedModules = {};
  function __webpack_require__(moduleId) {
    //判断缓存情况
    if (installedModules[moduleId]) {
      return installedModules[moduleId].exports;
    }
    var module = installedModules[moduleId] = {
      exports: {}
    };
    //执行key对应的匿名函数
    modules[moduleId].call(module.exports, module, module.exports,
    __webpack_require__);
    // Return the exports of the module
    return module.exports;
  }
  //执行我们的入口函数 把export导出
  return __webpack_require__("./src/index.js");
})({
  //__webpack_exports__ => module.exports = {}
  "./src/data.js": (function (module, __webpack_exports__,
  __webpack_require__) {
    const data = "webpack源码分析课";
    //module.exports = {
    //    default:data
    // }
    __webpack_exports__["default"] = (data);
  }),
  "./src/index.js": (function (module, __webpack_exports__,
  __webpack_require__) {
    //_data_js__WEBPACK_IMPORTED_MODULE_0__ == module.exports;
    var _data_js__WEBPACK_IMPORTED_MODULE_0__ =
    __webpack_require__("./src/data.js");
    //_data_js__WEBPACK_IMPORTED_MODULE_0__["default"]
    //=====data.js=====
    //module.exports = {
    //    default:data
    // }
    console.log(_data_js__WEBPACK_IMPORTED_MODULE_0__["default"]);
    console.log("🌶️");
  })
});
```

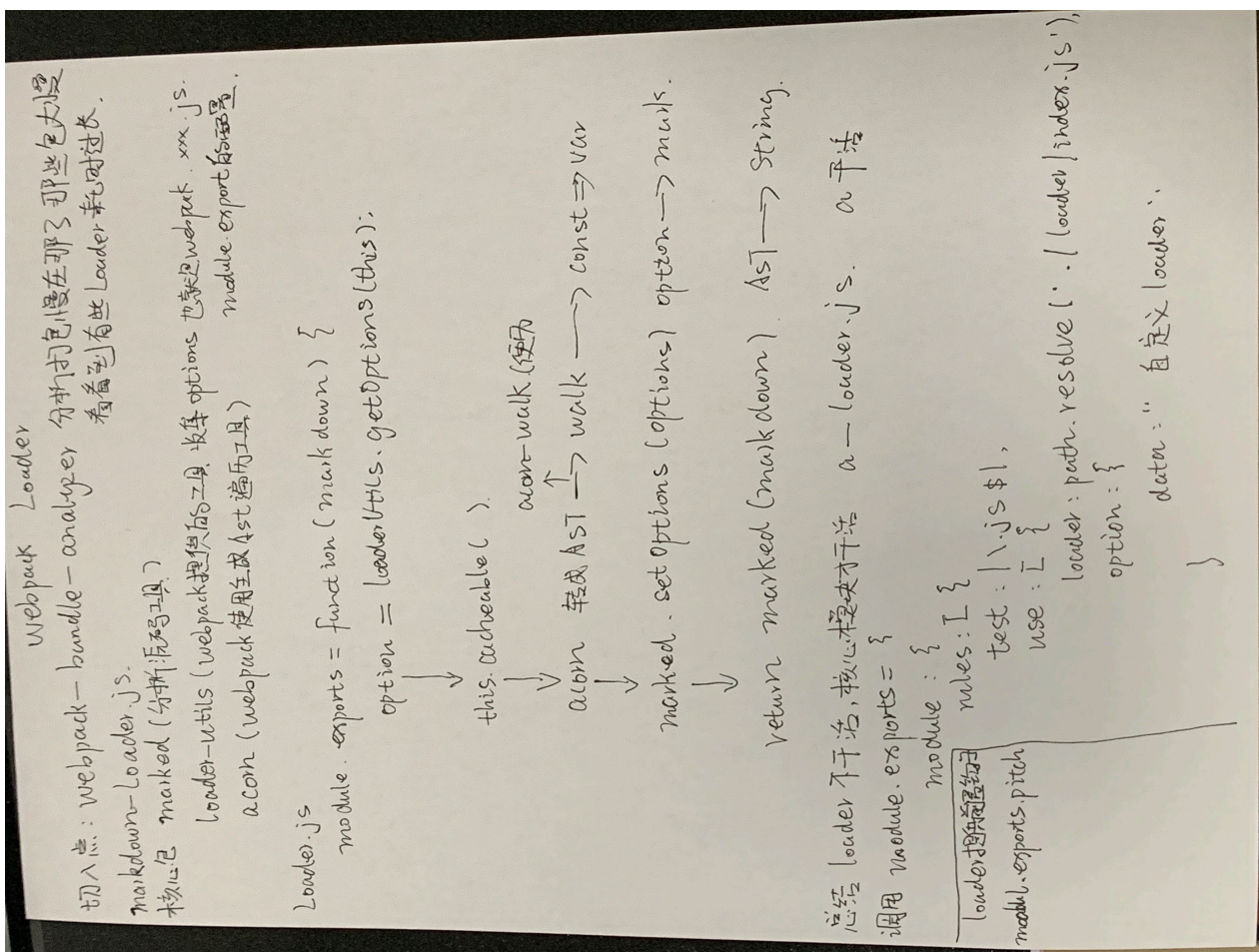
```
    })  
  });  
  //如果有异步的资源webpackJsonpCallback
```

实现webpack

```
const fs = require("fs");  
let input = "./index.js";  
let output = "./dist/index.js";  
const ejs = require("ejs");  
const getEntry = fs.readFileSync(input, "utf-8");  
let template = `  
  (function (modules) {  
    // 对webpack文件进行缓存  
    var installedModules = {};  
    function __webpack_require__(moduleId) {  
      //判断缓存情况  
      if (installedModules[moduleId]) {  
        return installedModules[moduleId].exports;  
      }  
      var module = installedModules[moduleId] = {  
        exports: {}  
      };  
      //执行key对应的匿名函数  
      modules[moduleId].call(module.exports, module, module.exports,  
__webpack_require__);  
      // Return the exports of the module  
      return module.exports;  
    }  
    //执行我们的入口函数 把export导出  
    return __webpack_require__(0);  
  })([  
    (function (module, exports) {  
      <%- getEntry %>  
    })  
  ]);`  
  
let result = ejs.render(template, {  
  getEntry  
})  
fs.writeFileSync(output, result);
```

loader





自己编写loader

markdown-loader.js

```

//分析mark源码工具
const marked = require("marked");

//webpack提供的工具集 收集用户options等 也就是webpack.**.js里module.export的配置等
const loaderUtils = require("loader-utils");

//提供一个对外的函数
/*
  webpack 为什么慢 机制分析
  loader1 string(源代码)--->ast(eg1 遍历这颗 🌲 替换const->var)--->string
  loader2 接受loader2的string/buffer--->ast--->string
*/
module.exports = function (markdown) {
  // merge params and default config
  const options = loaderUtils.getOptions(this);
  // 当前loader开启缓存
  this.cacheable();
  // 用户option--->mark
  marked.setOptions(options);
  //string
  return marked(markdown);
};

```

```
//总结出来loader不敢活，核心模块才干活
//eg: a-loader 干活的是a
```

index.js

```
const loaderUtils = require("loader-utils");
//this 代表当前loader类
module.exports=function(content,map,meta) {
  console.log('前置沟盖---->',this.data.value)
  //拿到options
  const options = loaderUtils.getOptions(this);
  //为了避免使用正则过于复杂 ---->ast 🌲
  //content.replace('const','var')
  return content+'console.log(1)';
}

module.exports.pitch = function(r,p,data) {
  data.value = '😱😱😱前置钩子'
}
```

Ast.js

```
//1、postcss cssnext -> css
//2、webpack loader es6 -> ast -> es5
//3、vue template -> html ast -> vdom
//4、v8 写的js代码 词法分析 语法分析 ast->执行
//5、ast + 设计模式 发布订阅
//webpack使用生成ast的包
const acorn = require('acorn');
//webpack使用生成ast遍历工具
const walk = require('acorn-walk');
const MagicString = require('magic-string');
const source = 'const a = 100'
const result = acorn.parse(source);
const code = new MagicString(source);

console.log(result);

walk.simple(result, {
  Literal(node) {
    console.log(`Found a literal: ${node.value}`)
  },
  VariableDeclaration(node) {
    console.log('👉', node);
    const { start } = node;
    console.log('🍏', start);
    code.overwrite(start, start + 5, "var")
  }
})
```

```
    }  
  })  
  
  console.log('结果🐼',code.toString())
```

调用webpack.config.js

```
const path = require('path');  
const ConsoleLogOnBuildWebpackPlugin =  
require('./plugin/ConsoleLogOnBuildWebpackPlugin')  
  
module.exports = {  
  module:{  
    rules: [{  
      test: /\.js$/,  
      use: [  
        {  
          loader: path.resolve('./loader/index.js'),  
          options: {  
            data:"🍌🍌🍌:自定义loader"  
          }  
        }  
      ]  
    }]  
  },  
  plugins:[  
    new ConsoleLogOnBuildWebpackPlugin()  
  ]  
}
```

性能优化

分析

webpack-bundle-analyzer

performance.timing

DNS查询耗时、TCP链接耗时、request请求耗时、解析dom树耗时、白屏时间、domready时间、onload时间

方案

webpack

ContextReplacementPlugin 正则匹配过滤掉不需要的包 /moment[/\]locale/,/zh-cn/

externals: {jQuery: 'jQuery'} 提取出公共的包

cdn

orm库

1.ORM封装localStorage + basket.js 解决问题

<https://www.cnblogs.com/oadaM92/p/5348793.html>

2.使用PWA完成所有缓存的控制（请开始你的表演）

```
if ('serviceWorker' in navigator) {
  console.log('当前控制权', navigator.serviceWorker.controller);
  navigator.serviceWorker.register('/service-worker.js').then((registration)
=> {
    console.log('serviceWorker注册成功, 范围为: ', registration.scope)
  }).catch((err) => {
    console.log('serviceWorker登记失败', err)
  })
}
```

架构



	CSR	预渲染	SSR	同构
优点	<ul style="list-style-type: none">不依赖数据FP 时间最快客户端用户体验好内存数据共享	<ul style="list-style-type: none">不依赖数据FCP 时间比 CSR 快客户端用户体验好内存数据共享	<ul style="list-style-type: none">SEO 友好首屏性能高, FMP 比 CSR 和预渲染快	<ul style="list-style-type: none">SEO 友好首屏性能高, FMP 比 CSR 和预渲染快客户端用户体验好内存数据共享客户端与服务端代码公用, 开发效率高
缺点	<ul style="list-style-type: none">SEO 不友好FCP、FMP 慢	<ul style="list-style-type: none">SEO 不友好FMP 慢	<ul style="list-style-type: none">客户端数据共享成本高模板维护成本高	<ul style="list-style-type: none">Node 容易形成性能瓶颈

用pjax来处理服务端渲染和客户端渲染的问题

quicklink是一个js库，可以预加载出现在视口的网页链接，提高用户体验。它的加载过程如下：1.检测网页中的链接是否出现在视口中，等待链接出现在视口，执行步骤2。2.等待浏览器空闲后执行3。3.判断当前的网络连接是否是2G，如果是则停止执行，如果不是2G网络，执行步骤4。4.预加载链接指向资源。

请求

10个请求封装成一个问题

- 1.首先同一个CDN的请求数量不能超过5个因为很多浏览器有并发限制，其次每一个文件体积已GZIP压缩过后32k最大为一个衡量标准（这个32是一个经验数字，因为原文件就100多很大了）
- 2.启动类库HTTP强缓存，然后业务JS缓存。同时开启业务JS增量更新方案，也可以控制更细粒度的JS增量更新方案如mtjs。

HTTP

常见状态码

100	Continue	继续。 客户端 应继续其请求
200	OK	请求成功。一般用于GET与POST请求
201	Created	已创建。成功请求并创建了新的资源
202	Accepted	已接受。已经接受请求，但未处理完成
203	Non-Authoritative Information	非授权信息。请求成功。但返回的meta信息不在原始的服务器，而是一个副本
204	No Content	无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档
300	Multiple Choices	多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择
301	Moved Permanently	永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替
302	Found	临时移动。与301类似。但资源只是临时被移动。客户端应继续使用原有URI
303	See Other	查看其它地址。与301类似。使用GET和POST请求查看
304	Not Modified	未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源
400	Bad Request	客户端请求的语法错误，服务器无法理解
401	Unauthorized	请求要求用户的身份认证
	Payment	

402	Required	保留，将来使用
403	Forbidden	服务器理解请求客户端的请求，但是拒绝执行此请求
404	Not Found	服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面
405	Method Not Allowed	客户端请求中的方法被禁止
500	Internal Server Error	服务器内部错误，无法完成请求
501	Not Implemented	服务器不支持请求的功能，无法完成请求
502	Bad Gateway	作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应
503	Service Unavailable	由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中
504	Gateway Time-out	充当网关或代理的服务器，未及时从远端服务器获取请求
505	HTTP Version not supported	服务器不支持请求的HTTP协议的版本，无法完成处理

TCP(握手挥手)

<https://juejin.im/post/5ccd0dfc6fb9a0324a08bb73>

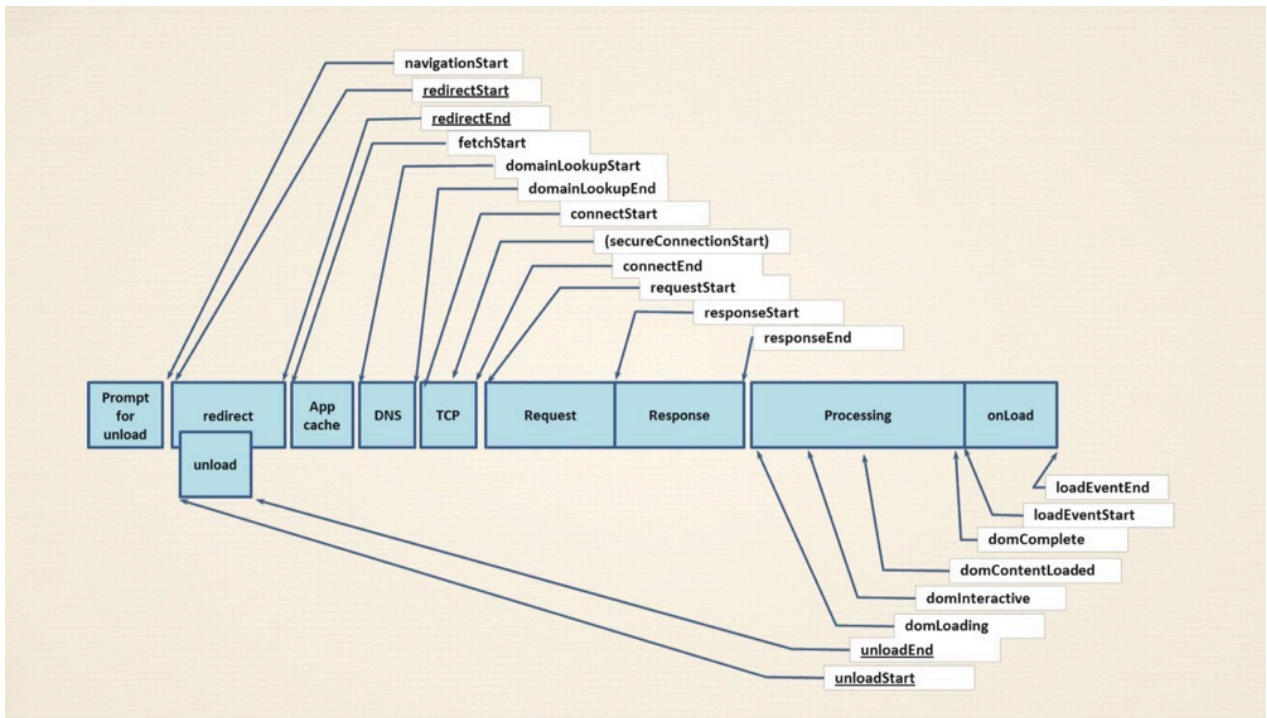
浏览器输入url到页面渲染

http协议工作过程其实是一个事物（当某一项工作需要严格按照若干的步骤去完成，如果不符合就会失败，符合这种特征的就是事物）

navigation-timing

w3c给浏览器制定的标准





1、prompt for unload

页面卸载，从内存中释放掉

2、redirect

本地重定向，与服务器无关，先从本地缓存中寻找

根据索引查找，查看文件是否支持脱机浏览，如果有并支持进行本地跳转

3、App cache

本地缓存

总结 先有一个卸载的提示 然后去浏览器缓存、服务器缓存、反向代理服务器中寻找，如果找到了发生重定向，如果没进入网络层，页面进行卸载

4、DNS

自己的pc、家用路由器、互联网

当自己家用路由器连接到互联网的时候，需要向拨号服务器进行一个拨号请求，拨完号后DNS的地址就保存到你的家用路由器上，当pc启动的时候要进行网络设置，然后DHCP(局域网动态获取ip地址的协议)，路由器给pc分配ip地址的时候把DNS一块写进来

(优化的点 利用cdn)

5、TCP

拿到服务器地址后进行连接

三次握手四次挥手 如果是https的话中间还有一个加密的过程

6、request

7、response

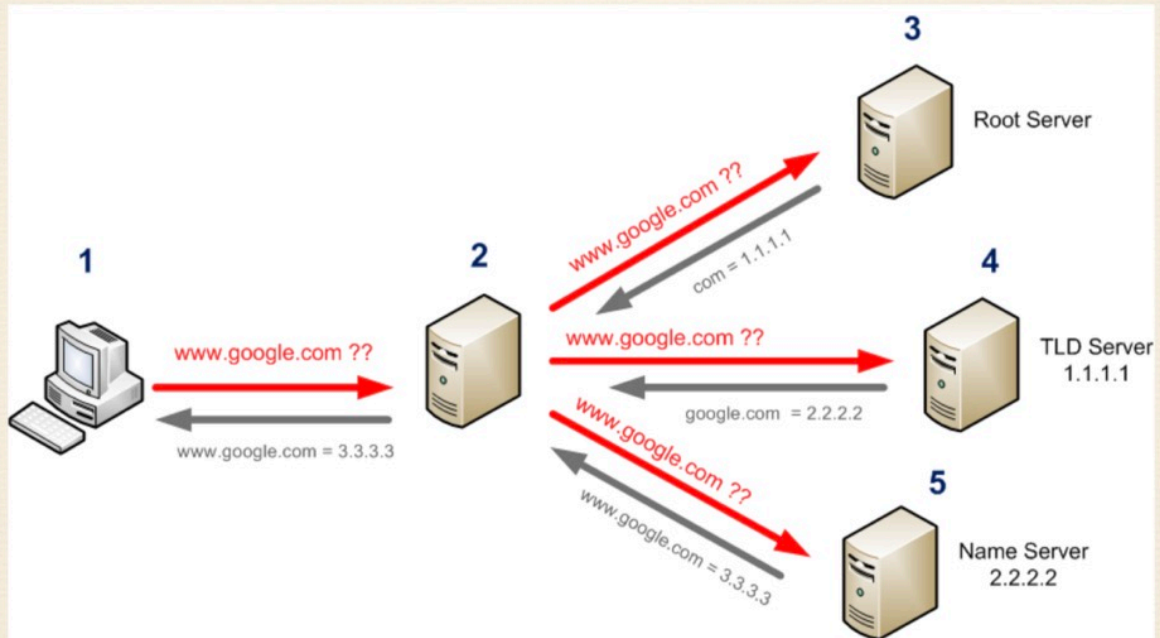
?

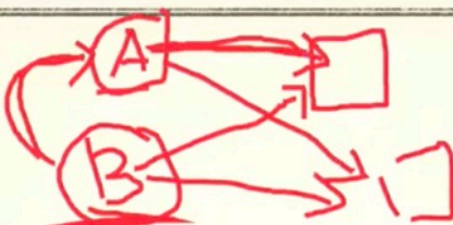
?

?

DNS详解

- ❖ DNS 是Domain Name System, 域名系统，用于将域名转换为IP。
- ❖ 顶级域名
- ❖ 域名资源记录
- ❖ 域名服务器
- ❖ 域名解析





记录类型	含义
SOA: (StartOf Authority, 起始授权记录)	一个区域解析库有且只能有一个SOA记录, 而且必须放在第一条
A记录 (主机记录) IPv4	用于名称解析的重要记录, 将特定的主机名映射到对应主机的IP地址上
CNAME记录 (别名记录)	用于 返回另一个域名, 即当前查询的域名是另一个域名的跳转, 主要用于域名的内部跳转, 为服务器配置提供灵活性
NS记录 (域名服务器记录)	用于返回保存下一级域名信息的服务器地址。该记录只能设置为域名, 不能设置为IP地址。
MX (邮件记录)	用于返回接收电子邮件的服务器地址
IPv6主机记录 (AAAA记录)	与A记录对应, 用于将特定的主机名映射到一个主机的IPv6地址。

大厂会使用镜像（但是在换了域名的时候比较费劲，所以我们在阿里云配置的时候需要等10分钟，数据同步）

http无状态协议

<https://www.cnblogs.com/bellkosmos/p/5237146.html>

http协议是无状态的，无连接的

1. 标准的http协议指的是不包括cookies, session, application的http协议，他们都不属于标准协议，虽然各种网络应用提供商，实现语言、web容器等，都默认支持它
2. 无连接指的是什么

每一个访问都是无连接，服务器挨个处理访问队列里的访问，处理完一个就关闭连接，这事儿就完了，然后处理下一个新的

无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接

http的基本优化

https://mp.weixin.qq.com/s/GlCbijpINrHZ41u_4zT-A?

延迟和带宽（带宽就自己充钱吧）

延迟

浏览器阻塞(并发数量)、DNS查询、建立连接(三次握手四次挥手)

HTTPS与HTTP的一些区别

HTTPS协议需要到CA申请证书，一般免费证书很少，需要交费。

HTTP协议运行在TCP之上，所有传输的内容都是明文，HTTPS运行在SSL/TLS之上，SSL/TLS运行在TCP之上，所有传输的内容都经过加密的。

HTTP和HTTPS使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

HTTPS可以有效的防止运营商劫持，解决了防劫持的一个大问题。

http2和http的区别

新的二进制格式

HTTP1.x的解析是基于文本

多路复用

一个request对应一个id，这样一个连接上可以有多个request，每个连接的request可以随机的混杂在一起

header压缩

HTTP1.x的header带有大量信息，而且每次都要重复发送，HTTP2.0使用encoder来减少需要传输的header大小，通讯双方各自cache一份header fields表，既避免了重复header的传输，又减小了需要传输的大小。

服务端推送

server push，同SPDY一样，HTTP2.0也具有server push功能

HTTP2.0的多路复用和HTTP1.X中的长连接复用有什么区别？

- HTTP/1.* 一次请求-响应，建立一个连接，用完关闭；每一个请求都要建立一个连接；
- HTTP/1.1 Pipeling解决方式为，若干个请求排队串行化单线程处理，后面的请求等待前面请求的返回才能获得执行机会，一旦有某请求超时等，后续请求只能被阻塞，毫无办法，也就是人们常说的线头阻塞；
- HTTP/2多个请求可同时在一个连接上并行执行。某个请求任务耗时严重，不会影响到其它连接的正常执行；

HTTP2.0多路复用有多好？

HTTP 性能优化的关键并不在于高带宽，而是低延迟。TCP 连接会随着时间进行自我「调谐」，起初会限制连接的最大速度，如果数据成功传输，会随着时间的推移提高传输的速度。这种调谐则被称为 TCP 慢启动。由于这种原因，让原本就具有突发性和短时性的 HTTP 连接变的十分低效。HTTP/2 通过让所有数据流共用同一个连接，可以更有效地使用 TCP 连接，让高带宽也能真正的服务于 HTTP 的性能提升。

HTTP/2对同一域名下所有请求都是基于流，也就是说同一域名不管访问多少文文件，也只建立一路路连接。同样Apache的最大大连接数为300，因为有了这个新特性，最大的并发就可以提升到300，比比原来提升了6倍！

Node

安全

<https://github.com/dwqs/blog/issues/68>

<https://www.cnblogs.com/menggirl23/p/10184607.html>

CSRF

跨站请求伪造，是通过伪装成受信任用户的请求来利用受信任的网站进行攻击

(饿了么外卖) 主要是劫持token

防御措施

token认证 referer验证

XSS

跨站脚本攻击

恶意web用户将代码植入到提供给其它用户使用的页面中

反射型

反射型XSS也叫非持久型XSS，是指发生请求时，XSS代码出现在请求 URL 中，作为参数提交给服务器，服务器解析并响应，响应结果中包含XSS代码，最后浏览器解析并执行。

```
 alert(document.cookie) />
```

存储型

存储型XSS，也叫做持久型XSS，主要是将XSS代码发送到服务端(无论是数据库、内存还是文件系统)，然后下次请求页面的时候，XSS代码会随着浏览器渲染出来，就不需要手动再次带上XSS代码。

这个典型的例子就是留言板，用户提交一条包含XSS代码的留言到数据库，当目标用户查看留言的时候就会从服务器解析之后加载出来，浏览器发现有可执行的代码，就当做正常的HTML和CSS或者JS进行解析，XSS攻击就产生了。

DOM

DOM XSS代码不需要服务器端的解析响应的直接参与，而是通过浏览器端的DOM解析。这完全是客户端的事情

1. 通过 `document.cookie` 可以盗取用户的 `cookie`。如果是黑客的话，可能还会做一些其他的手段。
2. 使用JavaScript或者CSS破坏页面正常的结构和样式。黑客可以通过在用户端获取到的用户信息发送到黑客的服务器上。
3. 流量劫持(通过询问某段具有 `window.location.href` 定位到其他页面，进行流量劫持)。

4. Dos攻击：利用合理的客户端请求来占用过多的服务端资源，从而使合法用户无法得到服务端响应。
5. 利用 `iframe`、`frame`、`XMLHttpRequest` 或 `Flash` 方式，以被攻击用户的身份执行一些管理动作，或执行一些如：发微博、加好友、发私信等操作。
6. 利用可被攻击的域收到其他域信任的特点，以受信任来源的身份请求一些平时不允许的操作，如进行不正当的投票活动等等。

防御措施

cookie保护： `httpOnly`

用户输入：用户输入内容编码解码 (he.js) 不合法的东西过滤(HTMLParser.js)