前端面经 - 看这篇就够了（帮你拿到大厂offer）

面试题梳理

> 梳理前端常见面试题及答案。

一、web

前端性能优化

性能评级工具（PageSpeed 或 YSlow）

css

- CSS优化、提高性能的方法有哪些
- 多个css合并，尽量减少HTTP请求
- 将css文件放在页面最上面
- 移除空的css规则
- 避免使用CSS表达式
- 选择器优化嵌套，尽量避免层级过深
- 充分利用css继承属性，减少代码量
- 抽象提取公共样式，减少代码量
- 属性值为0时，不加单位
- 属性值为小于1的小数时，省略小数点前面的0
- 使用CSS Sprites将多张图片拼接成一张图片，通过CSS background 属性来访问图片内容

js

- 节流、防抖
- 长列表滚动到可视区域动态加载（大数据渲染）
- 图片懒加载（data-src）
- 使用闭包时，在函数结尾手动删除不需要的局部变量，尤其在缓存dom节点的情况下

- DOM 操作优化
  - 批量添加dom可先 `createElement` 创建并添加节点，最后一次性加入dom
  - 批量绑定事件，使用 `事件委托` 绑定父节点实现，利用了事件冒泡的特性
  - 如果可以使用 `innerHTML` 代替 `appendChild`
  - 在 DOM 操作时添加样式时尽量增加 class 属性，而不是通过 style 操作样式，以减少重排（ `Reflow` ）

网络

- 减少 HTTP 请求数量
- 利用浏览器缓存，公用依赖包（如vue、Jquery、ui组件等）单独打包/单文件在一起，避免重复请求
- 减小 `cookie` 大小，尽量用 `localStorage` 代替
- CDN托管静态文件
- 开启 Gzip 压缩

浏览器内核

- 主要分成两部分：渲染引擎(`layout engineer` 或 `Rendering Engine`)和 `JS` 引擎

- 渲染引擎：负责取得网页的内容（ `HTML` 、 `XML` 、图像等等）、整理讯息（例如加入 `CSS` 等），以及计算网页的显示方式，然后会输出至显示器或打印机。浏览器的内核的不同对于网页的语法解释会有不同，所以渲染的效果也不相同。所有网页浏览器、电子邮件客户端以及其它需要编辑、显示网络内容的应用程序都需要内核

- `JS` 引擎则：解析和执行 `javascript` 来实现网页的动态效果

- 最开始渲染引擎和 `JS` 引擎并没有区分的很明确，后来JS引擎越来越独立，内核就倾向于只指渲染引擎

常见的浏览器内核有哪些

- `Trident` 内核： `IE,MaxThon,TT,The World,360` ,搜狗浏览器等。[又称MSHTML]
- `Gecko` 内核： `Netscape6` 及以上版本， `FF,MozillaSuite/SeaMonkey` 等
- `Presto` 内核： `Opera7` 及以上。 [`Opera` 内核原为：Presto，现为： `Blink` ;]
- `Webkit` 内核： `Safari,Chrome` 等。 [`Chrome` 的 `Blink` （ `WebKit` 的分支）]

cookies、sessionStorage、localStorage 和 indexDB 的区别

- cookie是网站为了标示用户身份而储存在用户本地的数据
- 是否在http请求只能够携带
  - cookie数据始终在**同源**的http请求中携带，跨域需要设置 `withCredentials = true`
  - sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存
- 存储大小：
  - cookie数据大小不能超过 `4k` ；
  - sessionStorage和localStorage虽然也有存储大小的限制，但比cookie大得多，可以达到 `5M` 或更大，因不同浏览器大小不同；
- 有效时间：
  - cookie 设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭
  - localStorage 硬盘存储持久数据，浏览器关闭后数据不丢失除非主动删除数据
  - sessionStorage 存在内存中，数据在当前浏览器窗口关闭后自动删除

画表对比：

特性	**cookie**	**localStorage**	**sessionStorage**	**indexDB**
数据生命周期	一般由服务器生成，可以设置过期时间	除非被清理，否则一直存在	页面关闭就清理	除非被清理，否则一直存在
数据存储大小	**4K**	**5M**	**5M**	无限
与服务端通信	每次都会携带在 header 中，对于请求性能影响	不参与	不参与	不参与

对于 `cookie` ，还需要注意安全性：

属性	作用
value	如果用于保存用户登录态，应该将该值加密，不能使用明文的用户标识
http-only	不能通过 JS 访问 Cookie，减少 XSS 攻击
secure	只能在协议为 HTTPS 的请求中携带
same-site	规定浏览器不能在跨域请求中携带 Cookie，减少 CSRF 攻击

从输入URL到浏览器显示页面过程中都发生了什么

- 参考：从输入URL到浏览器显示页面过程中都发生了什么

对AMD、CMD的理解

- CommonJS 是服务器端模块的规范，Node.js 采用了这个规范。CommonJS 规范加载模块是同步的，也就是说，只有加载完成，才能执行后面的操作。AMD 规范则是非同步加载模块，允许指定回调函数
- AMD 推荐的风格通过返回一个对象做为模块对象，CommonJS 的风格通过对 module.exports 或 exports 的属性赋值来达到暴露模块对象的目的
- AMD

es6模块 CommonJS、AMD、CMD

- CommonJS 的规范中，每个 JavaScript 文件就是一个独立的模块上下文（module context），在这个上下文中默认创建的属性都是私有的。也就是说，在一个文件定义的变量（还包括函数和类），都是私有的，对其他文件是不可见的。
- CommonJS 是同步加载模块,在浏览器中会出现堵塞情况，所以不适用
- AMD 异步，需要定义回调 define 方式
- es6 一个模块就是一个独立的文件，该文件内部的所有变量，外部无法获取。如果你希望外部能够读取模块内部的某个变量，就必须使用 export 关键字输出该变量 es6 还可以导出类、方法，自动适用严格模式

模块化（AMD、CMD、CommonJS、ES6）

模块化的演进过程

- 1.文件划分的方式 污染全局作用域 命名冲突 无法管理模块依赖关系
- 2.命名空间方式 在第一个阶段的基础上 将每个模块只暴露一个全局对象 所有的变量都挂载到这个全局对象上
- 3.IIFE 立即执行函数 为模块提供私有空间
- 以上是早起在没有工具和规范的情况下对模块化的落地方式

模块化规范的出现 模块化规范+模块加载器

1. AMD 异步加载

- require.js的实现 define('modle', [加载资源]， ()=>{})
- 使用起来比较复杂
- 模块js文件请求频繁
- 先加载依赖

```
// require.js 就是使用的这种风格

define(['a.js', 'b.js'], function(A, B) {
    // do something
})


// 实现思路：建一个node节点, script标签
var node = document.createElement('script')
node.type = 'text/javascript'
node.src = '1.js'

// 1.js 加载完后onload的事件
node.addEventListener('load', function(evt) {
    // 开始加载 2.js
    var node2 = document.createElement('script')
    node2.type = 'text/javascript'
    node2.src = '2.js'
    // 插入 2.js script 节点
    document.body.appendChild(node2)
})
// 将script节点插入dom中
document.body.appendChild(node);
```

2. CMD sea.js

- sea.js
- 按需加载
- 碰到require('2.js')就立即执行2.js

```js
define(function() {
    var a = require('2.js')
    console.log(33333)
})
```

3. commonjs 服务端规范

- 一个文件就是一个模块
- 每个模块都有单独的作用域
- 通过module.exports导出成员
- 通过require函数载入模块
- commonjs是以 同步 的方式加载模块 node的执行机制是在启动时去加载模块 在执行阶段不需要加载模块
- CommonJS 模块输出的是一个值的拷贝，一旦输出一个值，模块内部的变化就影响不到这个值
- CommonJS 模块加载的顺序，按照其在代码中出现的顺序
- 由于 CommonJS 是同步加载模块的，在服务器端，文件都是保存在硬盘上，所以同步加载没有问题，但是对于浏览器端，需要将文件从服务器端请求过来，那么同步加载就不适用了，所以，CommonJS 是不适用于浏览器端的。
- CommonJS 模块可以多次加载，但是只会在第一次加载时运行一次，然后运行结果就被缓存了，以后再加载，就直接读取缓存结果。要想让模块再次运行，必须清除缓存

4. ESmodules 浏览器模块化规范

- 在语言层面实现了模块化 通过给script的标签 将type设置成module 就可以使用这个规范了
- 基本特性
  - 自动采用严格模式，忽略use strict
  - 每个ESM模块都是单独的私有作用域
  - ESM是通过CORS去请求外部JS模块的
  - ESM中的script标签会延迟执行脚本
  - ES6 模块是动态引用，引用类型属性被改变会相互影响
- export import 进行导入导出

- 导出的并不是成员的值 而是内存地址 内部发生改变外部也会改变，外部导入的是只读成员不能修改
- ES module中可以导入CommonJS模块
- CommonJS中不能导入ES module模块
- CommonJS始终只会导出一个默认成员
- 注意import不是解构导出对象

浏览器缓存

> 浏览器缓存分为强缓存和协商缓存。当客户端请求某个资源时，获取缓存的流程如下

- 先根据这个资源的一些 `http header` 判断它是否命中强缓存，如果命中，则直接从本地获取缓存资源，不会发请求到服务器；
- 当强缓存没有命中时，客户端会发送请求到服务器，服务器通过另一些 `request header` 验证这个资源是否命中协商缓存，称为 `http` 再验证，如果命中，服务器将请求返回，但不返回资源，而是告诉客户端直接从缓存中获取，客户端收到返回后就会从缓存中获取资源；
- 强缓存和协商缓存共同之处在于，如果命中缓存，服务器都不会返回资源； 区别是，强缓存不对发送请求到服务器，但协商缓存会。
- 当协商缓存也没命中时，服务器就会将资源发送回客户端。
- 当 `ctrl+f5` 强制刷新网页时，直接从服务器加载，跳过强缓存和协商缓存；
- 当 `f5` 刷新网页时，跳过强缓存，但是会检查协商缓存；

强缓存

- `Expires` （该字段是 `http1.0` 时的规范，值为一个绝对时间的 `GMT` 格式的时间字符串，代表缓存资源的过期时间）
- `Cache-Control:max-age` （该字段是 `http1.1` 的规范，强缓存利用其 `max-age` 值来判断缓存资源的最大生命周期，它的值单位为秒）

协商缓存

- `Last-Modified` （值为资源最后更新时间，随服务器response返回，即使文件改回去，日期也会变化）
- `If-Modified-Since` （通过比较两个时间来判断资源在两次请求期间是否有过修改，如果没有修改，则命中协商缓存）
- `ETag` （表示资源内容的唯一标识，随服务器 `response` 返回，仅根据文件内容是否变化判断）

- `If-None-Match`（服务器通过比较请求头部的 `If-None-Match` 与当前资源的 `ETag` 是否一致来判断资源是否在两次请求之间有过修改，如果没有修改，则命中协商缓存）

浏览器是如何渲染网页的

- 参考：浏览器渲染页面过程

重绘（Repaint）和回流（Reflow）

> 重绘和回流会在我们设置节点样式时频繁出现，同时也会很大程度上影响性能。

- **重绘**：当节点需要更改外观而不会影响布局的，比如改变 `color` 就叫称为重绘
- **回流**：布局或者几何属性需要改变就称为回流。
- 回流必定会发生重绘，重绘不一定会引发回流。回流所需的成本比重绘高的多，改变父节点里的子节点很可能会导致父节点的一系列回流。

以下几个动作可能会导致性能问题：

- 改变 `window` 大小
- 改变字体
- 添加或删除样式
- 文字改变
- 定位或者浮动
- 盒模型

重绘和回流其实也和 `Eventloop` 有关。

- 当 `Eventloop` 执行完 `Microtasks` 后，会判断 `document` 是否需要更新，因为浏览器是 `60Hz` 的刷新率，每 `16.6ms` 才会更新一次。
- 然后判断是否有 `resize` 或者 `scroll` 事件，有的话会去触发事件，所以 `resize` 和 `scroll` 事件也是至少 `16ms` 才会触发一次，并且自带节流功能。
- 判断是否触发了 `media query`
- 更新动画并且发送事件
- 判断是否有全屏操作事件
- 执行 `requestAnimationFrame` 回调
- 执行 `IntersectionObserver` 回调，该方法用于判断元素是否可见，可以用于懒加载上，但是兼容性不好 更新界面

- 以上就是一帧中可能会做的事情。如果在一帧中有空闲时间，就会去执行 `requestIdleCallback` 回调

如何减少重绘和回流:

1. 使用 `transform` 替代 `top`
2. 使用 `visibility` 替换 `display: none` ，因为前者只会引起重绘，后者会引发回流（改变了布局）
3. 不要把节点的属性值放在一个循环里当成循环里的变量
4. 不要使用 `table` 布局，可能很小的一个小改动会造成整个 `table` 的重新布局
5. 动画实现的速度的选择，动画速度越快，回流次数越多，也可以选择使用 `requestAnimationFrame`
6. `CSS` 选择符从右往左匹配查找，避免节点层级过多
7. 将频繁重绘或者回流的节点设置为图层，图层能够阻止该节点的渲染行为影响别的节点。比如对于 `video` 标签来说，浏览器会自动将该节点变为图层。
8. 避免使用 `css` 表达式(`expression`)，因为每次调用都会重新计算值（包括加载页面）
9. 尽量使用 `css` 属性简写，如：用 `border` 代替 `border-width` , `border-style` , `border-color`
10. 批量修改元素样式： `elem.className` 和 `elem.style.cssText` 代替 `elem.style.xxx`
11. 需要要对元素进行复杂的操作时，可以先隐藏(`display:"none"`)，操作完成后再显示
12. 需要创建多个 `DOM` 节点时，使用 `DocumentFragment` 创建完后一次性的加入 `document`
13. 缓存 `Layout` 属性值，如： `var left = elem.offsetLeft;` 这样，多次使用 `left` 只产生一次回流

> 设置节点为图层的方式有很多，我们可以通过以下几个常用属性可以生成新图层

- `will-change`
- `video` 、 `iframe` 标签

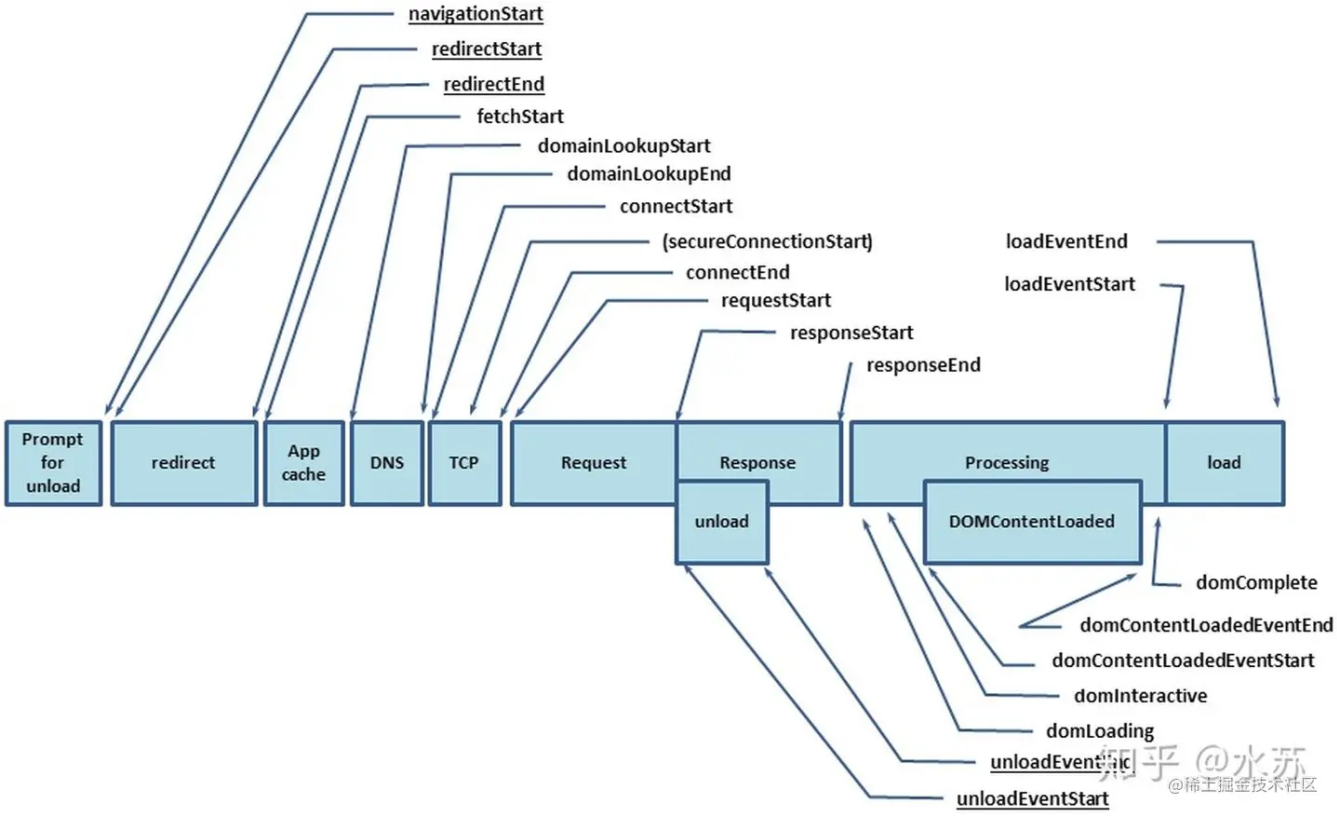
首屏加载优化方案

1. Vue-Router路由懒加载（利用Webpack的代码切割）
2. 使用CDN加速，将通用的库从vendor进行抽离
3. Nginx的gzip压缩
4. Vue异步组件
5. 服务端渲染SSR
6. 如果使用了一些UI库，采用按需加载

7. Webpack开启gzip压缩
8. 如果首屏为登录页，可以做成多入口
9. 图片懒加载减少占用网络带宽
10. 页面使用骨架屏
11. 利用好script标签的async和defer这两个属性。功能独立且不要求马上执行的js文件，可以加入async属性。如果是优先级低且没有依赖的js，可以加入defer属性。

可利用 `performance.timing` 看各个步骤的耗时： 白屏时间：

```
performance.timing.responseStart - performance.timing.navigationStart
```



二、html

html5有哪些新特性、移除了那些元素?

新增功能：HTML5 现在已经不是 SGML 的子集，主要是关于图像，位置，存储，多任务等功能的增加

- 新增选择器 document.querySelector、document.querySelectorAll
- 拖拽释放(Drag and drop) API
- 媒体播放的 video 和 audio
- 本地存储 localStorage 和 sessionStorage

- 离线应用 manifest
- 桌面通知 Notifications
- 语意化标签 article、footer、header、nav、section
- 增强表单控件 calendar、date、time、email、url、search
- 地理位置 Geolocation
- 多任务 webworker
- 全双工通信协议 websocket
- 历史管理 history
- 跨域资源共享(CORS) Access-Control-Allow-Origin
- 页面可见性改变事件 visibilitychange
- 跨窗口通信 PostMessage
- Form Data 对象
- 绘画 canvas

移除的元素:

- 纯表现的元素：basefont、big、center、font、 s、strike、tt、u
- 对可用性产生负面影响的元素：frame、frameset、noframes

viewport

```html
<meta name="viewport" content="width=device-width,initial-scale=1.0,minimum-scale=1.0,maximum-scale=
<!--
    width     设置viewport宽度，为一个正整数，或字符串‘device-width’
    device-width   设备宽度
    height    设置viewport高度，一般设置了宽度，会自动解析出高度，可以不用设置
    initial-scale     默认缩放比例（初始缩放比例），为一个数字，可以带小数
    minimum-scale     允许用户最小缩放比例，为一个数字，可以带小数
    maximum-scale     允许用户最大缩放比例，为一个数字，可以带小数
    user-scalable     是否允许手动缩放
-->
```

延伸提问：怎样处理 移动端 1px 被 渲染成 2px问题?

- 局部处理
  - meta标签中的 viewport属性 ， initial-scale 设置为 1
  - rem按照设计稿标准走，外加利用transfrome 的scale(0.5) 缩小一倍即可；
- 全局处理

- mate标签中的 viewport属性， initial-scale 设置为 0.5
- rem 按照设计稿标准走即可

html文档解析

- 参考：html文档解析为AST语法树

三、css

1. css选择器优先级

```
!important > inline > id > class > tag > * > inherit > default
```

- !important：大于其他
- 行内：1000
- id选择器：100
- 类，伪类和属性选择器，如.content：10
- 类型选择器和伪元素选择器：1
- 通配符、子选择器、相邻选择器：0

同级别的后写的优先级高。

相同class样式，css中后写的优先级高，和html中的class名字前后无关

2. 水平垂直居中

- 文本水平居中： `text-algin: center`
- 文本垂直居中： `line-height` 等于容器 `height` ； `display: flex; algin-items: center;`
- div水平居中：
  1. margin: 0 auto;
  2. 已知父元素宽度：margin-left: width / 2; transform: tranlateX(-50%)
  3. 未知父元素宽度：position: absolute: top: 50%; transform: tranlateY(-50%)
  4. display: flex; justify-content: center;
- div垂直居中：
  1. 已知父元素高度：margin-top: height / 2; transform: tranlateY(-50%)
  2. 未知父元素高度：position: absolute: top: 50%; transform: tranlateY(-50%)
  3. display: flex; algin-items: center;

3. 移除inline-block间隙

- 移除空格
- 使用margin负值
- 使用font-size:0
- letter-spacing
- word-spacing

4. 清除浮动

浮动的影响： （1）由于浮动元素脱离了文档流，所以父元素的高度无法被撑开，影响了与父元素同级的元素 （2）与浮动元素同级的非浮动元素会跟随其后，因为浮动元素脱离文档流不占据原来的位置 （3）如果该浮动元素不是第一个浮动元素，则该元素之前的元素也需要浮动，否则容易影响页面的结构显示 **清除浮动的3个方法**：

1. 使用 `clear: both` 清除浮动 在**浮动元素后面**放一个空的div标签，div设置样式clear:both来清除浮动。它的优点是简单，方便兼容性好，但是一般情况下不建议使用该方法，因为会造成结构混乱，不利于后期维护。

2. 利用伪元素 `after` 来清除浮动 给**父元素**添加了:after伪元素，通过清除伪元素的浮动，达到撑起父元素高度的目的

```css
.clearfix:after{
    content: "";
    display: block;
    visibility: hidden;
    clear: both;
}
```

3. 使用CSS的 `overflow` 属性 当给**父元素**设置了overflow样式，不管是overflow:hidden或overflow:auto都可以清除浮动只要它的值不为visible就可以了，它的本质就是建构了一个BFC，这样使得达到撑起父元素高度的效果

- 参考：css清除浮动

5. （外）边距重叠

布局垂直方向上两个元素的间距不等于margin的和，而是取较大的一个

1. 同级相邻元素 **现象**：上方元素设 置`margin-bottom: 20px` ，下方元素设置 `margin-top: 10px` ，实际的间隔是 `20px` **避免办法**：同级元素不要同时设置，可都设置margin-bottom或margin-top的一个，或者设置padding

2. 父子元素 **现象**：父元素设 置`margin-top: 20px` ，下方元素设置 `margin-top: 10px` ，实际的间隔是 `20px` **避免办法**：父元素有padding-top，或者border-top。或者触发 `BFC`

6. 三栏布局

要求：左边右边固定宽度，中间自适应

1. float

```html 复制代码
<style>
.left{
    float: left;
    width: 300px;
    height: 100px;
    background: #631D9F;
}
.right{
    float: right;
    width: 300px;
    height: 100px;
    background: red;
}
.center{
    margin-left: 300px;
    margin-right: 300px;
    background-color: #4990E2;
}
</style>
<body>
    <div class="left">左</div>
    <div class="right">右</div>
    <div class="center">中</div>
    <!-- center如果在中间，则right会被中间block的元素挤到下一行 -->
</body>
```

2. position

```html 复制代码
<style>
.left{
    position: absolute;
```

```
    left: 0;
    width: 300px;
    background-color: red;
}
.center{
    position: absolute;
    left: 300px;
    right: 300px;
    background-color: blue;
}
.right{
    position: absolute;
    right: 0;
    width: 300px;
    background-color: #3A2CAC;
}
</style>
<body>
    <div class="left">左</div>
    <div class="center">中</div>
    <div class="right">右</div>
</body>
```

3. flex

html 复制代码

```
<style>
.main {
    display: flex;
}
.left{
    width: 300px;
}
.center{
    flex-grow: 1;
    flex-shrink: 1;
}
.right{
    width: 300px;
}
</style>
<body class="main">
    <div class="left">左</div>
    <div class="center">中</div>
    <div class="right">右</div>
</body>
```

7. BFC

> BFC 是 Block Formatting Context ，也就是 块级格式化上下文 ，是用于布局块级盒子的一块渲染区域。

简单来说，BFC 实际上是一块区域，在这块区域中遵循一定的规则，有一套独特的渲染规则。

文档流其实分为 普通流 、 定位流 和 浮动流 和三种，普通流其实就是指BFC中的FC，也即 格式化上下文 。

普通流：元素按照其在 HTML 中的先后位置**从上到下、从左到右**布局，在这个过程中，行内元素水平排列，直到当行被占满然后换行，块级元素则会被渲染为完整的一个新行。

格式化上下文：页面中的一块渲染区域，有一套渲染规则，决定了其子元素如何布局，以及和其他元素之间的关系和作用

§ BFC的几条规则： 1）BFC 区域内的元素外边距会发生重叠。
2）BFC 区域内的元素不会与浮动元素重叠。
3）计算 BFC 区域的高度时，浮动元素也参与计算。
4）BFC 区域就相当于一个容器，内部的元素不会影响到外部，同样外部的元素也不会影响到内部。

§ BFC的应用：

1. 清除浮动：父元素设置overflow: hidden触发BFC实现清除浮动，防止父元素高度塌陷，后面的元素被覆盖，实现文字环绕等等。
2. 消除相邻元素垂直方向的边距重叠：第二个子元素套一层，并设置overflow: hidden，构建BFC使其不影响外部元素。
3. 消除父子元素边距重叠，父元素设置overflow: hidden

§ 触发BFC的方式：

1. float 不为 none，浮动元素所在的区域就是一个 BFC 区域。
2. position 的值不是 static 或 relative 的元素所在的区域就是一个 BFC 区域
3. display为 table-cell 的表格单元格元素所在的区域也是一个 BFC 区域
4. overflow 不为 visible 的元素所在的区域也是一个 BFC 区域

- 参考：CSS 中你应该了解的 BFC

8. flex布局

弹性布局，Flex 布局将成为未来布局的首选方案。 **兼容性：**

1. Webkit 内核的浏览器，必须加上-webkit前缀。
2. ie9不支持

基本概念：

1. 容器&项目：采用 Flex 布局的元素，称为 Flex 容器（flex container），简称"容器"。它的所有子元素自动成为容器成员，称为 Flex 项目（flex item），简称"项目"。
2. 主轴&交叉轴：堆叠的方向，默认主轴是水平方向，交叉轴是垂直方向。可通过 `flex-derection: column` 设置主轴为垂直方向。

容器属性：

- display: flex
- flex-direction：主轴的方向（即项目的排列方向），row | row-reverse | column | column-reverse;
- flex-wrap：是否换行，nowrap | wrap | wrap-reverse;
- flex-flow：direction 和 wrap简写
- justify-content：主轴对齐方式，flex-start | flex-end | center | space-between | space-around;
- align-items：交叉轴对齐方式，flex-start | flex-end | center | baseline | stretch;
- align-content: 多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。flex-start | flex-end | center | space-between | space-around | stretch;

项目的属性:

- order：项目的排列顺序，数值越小，排列越靠前，默认为0。
- flex-grow：放大比例，默认为0，指定元素分到的剩余空间的比例。
- flex-shrink：缩小比例，默认为1，指定元素分到的缩减空间的比例。
- flex-basis：分配多余空间之前，项目占据的主轴空间，默认值为auto
- flex：grow, shrink 和 basis的简写，默认值为0 1 auto

- align-self：单个项目不一样的对齐方式，默认值为auto，auto | flex-start | flex-end | center | baseline | stretch;

- 参考：Flex 布局教程：语法篇 - 阮一峰

9. CSS动画

1. transition过渡 将变化按照设置的时间长度缓慢执行完毕，而不是立即执行。

`delay` 的真正意义在于，它指定了动画发生的顺序，使得多个不同的transition可以连在一起，形成复杂效果。

transition的值是简写，扩展开依次是：

1. transition-property：过渡属性
2. transition-duration：过渡时间长度
3. transition-delay：延迟几秒执行
4. transition-timing-function
    - linear：匀速
    - ease-in：加速
    - ease-out：减速
    - cubic-bezier函数：自定义速度模式，最后那个cubic-bezier，可以使用工具网站来定制。

css 复制代码

```
/* 变化在1s过渡 */
transition: 1s;
/* 指定过渡属性 */
transition: 1s height;
/* 指定多个属性同时发生过渡 */
transition: 1s height, 1s width;
/* 指定delay延时时间 */
transition: 1s height, 1s 1s width;
/* 指定状态变化速度 */
transition: 1s height ease;
/* 指定自定义移状态变化速度 */
transition: 1s height cubic-bezier(.83,.97,.05,1.44);
```

transition的局限 transition的优点在于简单易用，但是它有几个很大的局限。

1. transition需要事件触发，所以没法在网页加载时自动发生。

2. transition是一次性的，不能重复发生，除非一再触发。
3. transition只能定义开始状态和结束状态，不能定义中间状态，也就是说只有两个状态。
4. 一条transition规则，只能定义一个属性的变化，不能涉及多个属性。

CSS Animation就是为了解决这些问题而提出的。

- Transition 强调过渡，Transition + Transform = 两个关键帧的Animation
- Animation 强调流程与控制，Duration + TransformLib + Control = 多个关键帧的Animation

2. animation动画

```css
.element:hover {
  animation: 1s rainbow;
  /*
  animation: 1s rainbow infinite; 关键字infinite让动画无限次播放
  animation: 1s rainbow 3; 指定动画播放次数
  */
}

@keyframes rainbow {
  0% { background: #c00; }
  50% { background: orange; }
  100% { background: yellowgreen; }
}
```

其中animation的值是简写，扩展开依次是：

1. animation-name: 指定一个 @keyframes 的名称，动画将要使用这个@keyframes定义。
2. animation-duration: 整个动画需要的时长。
3. animation-timing-function: 动画进行中的时速控制，比如 ease 或 linear.
4. animation-delay: 动画延迟时间。
5. animation-direction: 动画重复执行时运动的方向。
6. animation-iteration-count: 动画循环执行的次数。
7. animation-fill-mode: 设置动画执行完成后/开始执行前的状态，比如，你可以让动画执行完成后停留在最后一幕，或恢复到初始状态。
8. animation-play-state: 暂停/启动动画。

- 参考：CSS动画简介

10. CSS优化、提高性能的方法有哪些

- 多个css合并，尽量减少HTTP请求
- 将css文件放在页面最上面
- 移除空的css规则
- 避免使用CSS表达式
- 选择器优化嵌套，尽量避免层级过深
- 充分利用css继承属性，减少代码量
- 抽象提取公共样式，减少代码量
- 属性值为0时，不加单位
- 属性值为小于1的小数时，省略小数点前面的0
- css雪碧图

四、javascript

- 参考：javascript面试题整理

五、vue

- 参考：vue面试题整理

六、webpack

- `webpack` 是一个模块打包工具，你可以使用webpack管理你的**模块依赖**，并**编绎输出**模块们所需的静态文件。
- 它能够很好地**管理、打包**Web开发中所用到的HTML、Javascript、CSS以及各种静态文件（图片、字体等），让开发过程更加高效。
- 对于不同类型的资源，webpack有对应的模块加载器 `loader` 。
- webpack模块打包器会分析模块间的**依赖关系**，最后生成优化且合并后的静态资源。
- 其**插件**功能提供了处理各种文件过程中的各个生命周期钩子，使开发者能够利用插件功能开发很多自定义的功能。

1. 我写过的webpack相关的文章

- webpack打包原理&手写webpack核心打包过程
- 在webpack-dev-server中添加mock中间件实现前端模拟数据功能
- webpack常用配置详解

- webpack配置优化

2. 核心打包原理简述

3. Loader

编写一个loader:

`loader` 就是一个 `node` 模块，它输出了一个函数。当某种资源需要用这个 `loader` 转换时，这个函数会被调用。并且，这个函数可以通过提供给它的 `this` 上下文访问 `Loader API` 。 `reverse-txt-loader` 。

解析顺序： 从下向上，从右向左

js 复制代码

```
// 定义
module.exports = function(src) {
    //src是原文件内容（abcde），下面对内容进行处理，这里是反转
    var result = src.split('').reverse().join('');
    //返回JavaScript源码，必须是String或者Buffer
    return `module.exports = '${result}'`;
}
//使用
{
    test: /\.txt$/,
    loader: 'reverse-txt-loader'
}
```

4. 配置举例

§ base

js 复制代码

```
module.exports = {
    entry: {
        app: './src/main.js'
    },
    output: {
        path: path.resolve(__dirname, '../dist'),
        filename: '[name].js',
        publicPath: ''
    },
    resolve: {
```

```
        extensions: ['.js', '.vue', '.json', '.css', '.less'],
        alias: {
            '@': resolve('src'),
            src: resolve('src'),
            static: resolve('static')
        }
    },
    module: {
        rules: [
            {
                test: /\.vue$/,
                loader: 'vue-loader',
                options: vueLoaderConfig
            },
            {
                test: /\.js$/,
                loader: 'babel-loader',
                include: [resolve('src'), resolve('test')]
            },
            {
                test: /\.(png|jpe?g|gif|svg)(\?.*)?$/,
                loader: 'url-loader',
                query: {
                    limit: 1,
                    name: utils.assetsPath(`img/[name].[ext]`)
                },
                include: /nobase64/
            },
            {
                test: /\.svg(\?\S*)?$/,
                loader: 'svg-sprite-loader',
                query: {
                    prefixize: true,
                    name: '[name]-[hash]'
                },
                include: [resolve('src')],
                exclude: /node_modules|bower_components/
            }
        ]
    },
    plugins: [
        new VueLoaderPlugin(),
        new ProgressBarPlugin({
            format: 'build [:bar] ' + chalk.green.bold(':percent') + ' (:elapsed seconds) : (:msg)',
            clear: false,
            width: 60
        })
    ]
};
```

s dev

js 复制代码

```
var path = require('path');
var utils = require('./utils');
var webpack = require('webpack');
var config = require('../config');
var merge = require('webpack-merge');
var baseWebpackConfig = require('./webpack.base.conf');
var CopyWebpackPlugin = require('copy-webpack-plugin');
var MyPlugin = require('./htmlPlugin');
var HtmlWebpackPlugin = require('html-webpack-plugin');
var MiniCssExtractPlugin = require('mini-css-extract-plugin');
var OptimizeCSSPlugin = require('optimize-css-assets-webpack-plugin');
const TerserPlugin = require('terser-webpack-plugin');

module.exports = merge(baseWebpackConfig, {
    mode: 'development',
    externals: {
        vue: 'Vue'
    },
    devtool: '#source-map',
    output: {
        path: config.build.assetsRoot,
        filename: utils.assetsPath(`js/[name].js`),
        chunkFilename: utils.assetsPath(`js/[name].js`)
    },
    plugins: [
        new webpack.HotModuleReplacementPlugin(),
        new FriendlyErrorsPlugin(),
        new CopyWebpackPlugin([
            {
                from: path.resolve(__dirname, `../static`),
                to: path.resolve(config.dev.assetsPublicPath, `/dist/static`)
            }
        ]),
        new HtmlWebpackPlugin({
            alwaysWriteToDisk: true,
            filename: 'index.html',
            template: path.resolve(__dirname, `../src/index.html`), // 模板路径
            inject: false,
            excludeChunks: [],
            baseUrl: '',
            currentEnv: 'development',
            envName: 'local',
            curentBranch: ''
```

```
    }),
    new HtmlHardDisk()
],
optimization: {
    /*
        作用域提升插件
        [注意] 这个插件在 mode: production 时时默认开启的
        这样配置时为了在 development 时也开启
https://webpack.js.org/configuration/optimization/#optimizationconcatenatemodules
  */
    concatenateModules: true,
    splitChunks: {
        cacheGroups: {
            commons: {
                test: /[\\/]node_modules[\\/]/,
                name: 'vendors',
                chunks: 'all'
            },
            styles: {
                name: 'styles',
                test: /\.css$/,
                chunks: 'all',
                enforce: true
            }
        }
    },
    runtimeChunk: {
        name: 'manifest'
    },
    minimizer: [
        // 代码压缩UglifyJsPlugin的升级版
        new TerserPlugin({
            cache: true,
            parallel: true,
            terserOptions: {
                sourceMap: true,
                warnings: false,
                compress: {
                    // warnings: false
                },
                ecma: 6,
                mangle: true
            },
            sourceMap: true
        }),
        new OptimizeCSSPlugin({
            cssProcessorOptions: {
                autoprefixer: {
                    browsers: 'last 2 version, IE > 8'
```

```
                    }
                }
            })
        ]
    }

}
```

§ prod

```
js 复制代码
```

```
var path = require('path');
var utils = require('./utils');
var webpack = require('webpack');
var merge = require('webpack-merge');
var baseWebpackConfig = require('./webpack.base.conf');
var HtmlWebpackPlugin = require('html-webpack-plugin');
var FriendlyErrorsPlugin = require('friendly-errors-webpack-plugin');
var CopyWebpackPlugin = require('copy-webpack-plugin');
var HtmlHardDisk = require('html-webpack-harddisk-plugin');
var MiniCssExtractPlugin = require('mini-css-extract-plugin');
var OptimizeCSSPlugin = require('optimize-css-assets-webpack-plugin');
const TerserPlugin = require('terser-webpack-plugin');

module.exports = merge(baseWebpackConfig, {
    mode: 'production',
    externals: {
        vue: 'Vue'
    },
    output: {
        path: config.build.assetsRoot,
        filename: utils.assetsPath(`js/[name].js`),
        chunkFilename: utils.assetsPath(`js/[name]-[chunkhash].js`)
    },
    plugins: [
        // http://vuejs.github.io/vue-loader/en/workflow/production.html
        new webpack.DefinePlugin({
            'process.env': env
        }),
        // extract css into its own file
        new MiniCssExtractPlugin({
            filename: utils.assetsPath('css/[name].css'),
            allChunks: true
        }),
        // copy custom static assets
        new CopyWebpackPlugin([
            {
                from: path.resolve(__dirname, `../static`),
```

```
            to: path.resolve(__dirname, `../dist/static`)
        }
    ]),
    new HtmlWebpackPlugin({
        alwaysWriteToDisk: true,
        filename: 'index.html',
        template: path.resolve(__dirname, `../src/index.html`), // 模板路径
        inject: false,
        baseUrl: '',
        currentEnv: 'production',
        envName: 'prod',
        curentBranch: ''
    }),
    new BundleAnalyzerPlugin()
],
optimization: {
    minimizer: [
        // 代码压缩UglifyJsPlugin的升级版
        new TerserPlugin({
            //  cache: true,
            parallel: true,
            terserOptions: {
                warnings: false,
                compress: {
                },
                //  ecma: 6,
                mangle: true
            },
            sourceMap: true
        }),
        new OptimizeCSSPlugin({
            cssProcessorOptions: {
                autoprefixer: {
                    browsers: 'last 2 version, IE > 8'
                }
            }
        })
    ],
    splitChunks: {
        cacheGroups: {
            commons: {
                test: /[\\/]node_modules[\\/]/,
                name: 'vendors',
                chunks: 'all'
            }
        }
    },
    runtimeChunk: {
        name: 'manifest'
```

```
        }
    }
};
```

5. 手动chungk分包

- optimization.splitChunks（4之前老版本用CommonsChunkPlugin）
- vue路由懒加载

6. 打包加速的方法

- devtool 的 sourceMap较为耗时
- 开发环境不做无意义的操作：代码压缩、目录内容清理、计算文件hash、提取CSS文件等
- 第三方依赖外链script引入：vue、ui组件、JQuery等
- HotModuleReplacementPlugin：热更新增量构建
- DllPlugin& DllReferencePlugin：动态链接库，提高打包效率，仅打包一次第三方模块，每次构建只重新打包业务代码。
- thread-loader,happypack：多线程编译，加快编译速度
- noParse：不需要解析某些模块的依赖
- babel-loader开启缓存cache
- splitChunks（老版本用CommonsChunkPlugin）：提取公共模块，将符合引用次数(minChunks)的模块打包到一起，利用浏览器缓存
- Tree Shaking 摇树：基于ES6提供的模块系统对代码进行静态分析, 并在压缩阶段将代码中的死代码（dead code)移除，减少代码体积。

7. 打包体积 优化思路

- webpack-bundle-analyzer插件可以可视化的查看webpack打包出来的各个文件体积大小，以便我们定位大文件，进行体积优化
- 提取第三方库或通过引用外部文件的方式引入第三方库
- 代码压缩插件 `UglifyJsPlugin`
- 服务器启用gzip压缩
- 按需加载资源文件 `require.ensure` =
- 剥离 `css` 文件，单独打包
- 去除不必要插件，开发环境与生产环境用不同配置文件

- SpritesmithPlugin雪碧图，将多个小图片打包成一张，用background-image，backgroud-pisition，width，height控制显示部分
- url-loader 文件大小小于设置的尺寸变成base-64编码文本，大与尺寸由file-loader拷贝到目标目录

8. Tree Shaking 摇树

背景： 项目中，有一个入口文件，相当于一棵树的主干，入口文件有很多依赖的模块，相当于树枝。实际情况中，虽然依赖了某个模块，但其实只使用其中的某些功能。通过 tree-shaking，将没有使用的模块摇掉，这样来达到删除无用代码的目的。

思路： 基于ES6提供的模块系统对代码进行静态分析,并将代码中的死代码（dead code)移除的一种技术。因此，利用Tree Shaking技术可以很方便地实现我们代码上的优化，减少代码体积。

Tree Shaking 摇树 是借鉴了 rollup 的实现。

摇树删除代码的原理： webpack基于ES6提供的模块系统，对代码的依赖树进行静态分析，把import & export标记为3类：

- 所有import标记为/* harmony import */
- 被使用过的export标记为/harmony export([type])/，其中[type]和webpack内部有关，可能是binding，immutable等；
- 没有被使用的export标记为/* unused harmony export [FuncName] */，其中[FuncName]为export的方法名，之后使用Uglifyjs（或者其他类似的工具）进行代码精简，把没用的都删除。

为何基于es6模块实现（ES6 module 特点：）：

- 只能作为模块顶层的语句出现
- import的模块名只能是字符串常量
- import binding是immutable的

条件：

1. 首先源码必须遵循 ES6 的模块规范 (import & export)，如果是 CommonJS 规范 (require) 则无法使用。

2. 编写的模块代码不能有副作用，如果在代码内部改变了外部的变量则不会被移除。

配置方法：

1. 在package.json里添加一个属性：

```
{
    // sideEffects如果设为false，webpack就会认为所有没用到的函数都是没副作用的，即删了也没关系。
    "sideEffects": false,
    // 设置黑名单，用于防止误删代码
    "sideEffects": [
        // 数组里列出黑名单，禁止shaking下列代码
        "@babel/polly-fill",
        "*.less",
        // 其它有副作用的模块
        "./src/some-side-effectful-file.js"
    ],
}
```

js 复制代码

tree-shaking 摇掉代码中未使用的代码 在生产模式下自动开启

tree-shaking并不是webpack中的某一个配置选项，是一组功能搭配使用后的优化效果，会在生产模式下自动启动

```
// 在开发模式下，设置 usedExports: true ，打包时只会标记出哪些模块没有被使用，不会删除，因为可能会影响 sour
{
    mode: 'develpoment',
    optimization: {
        // 优化导出的模块
        usedExports: true
    },
}
// 在生产模式下默认开启 usedExports: true ，打包压缩时就会将没用到的代码移除
{
    mode: 'production',
    //  这个属性的作用就是集中配置webpack内部的优化功能
    optimizition: {
        // 只导出外部使用的模块成员  负责标记枯树叶
        usedExports: true,
        minimize: true, // 自动压缩代码  负责摇掉枯树叶
        /**
         * webpack打包默认会将一个模块单独打包到一个闭包中
         * webpack3中新增的API 将所有模块都放在一个函数中 ，尽可能将所有模块合并在一起，
         * 提升效率，减少体积  达到作用域提升的效果
         */
```

js 复制代码

```
        concatenateModules: true,
    },

}
```

使用摇树的注意事项：

1. 使用 ES6 模块语法编写代码
2. 工具类函数尽量以单独的形式输出，不要集中成一个对象或者类
3. 声明 sideEffects
4. 自己在重构代码时也要注意副作用

tree-shaking & babel 使用babel-loader处理js代码会导致tree-shaking失效的原因：

- treeshaking 使用的前提必须是ES module组织的代码，也就是说交给ESMOdule处理的代码必须是ESM。当我们使用babel-loader处理js代码之后就有可能将ESM 转换 成commonjs规范（preset-env插件工作的时候就会将esm => coommonjs）

解决办法：
收到配置preset-env的modules： false,确保不会开启自动转换的插件(在最新版本的babel-loader中自动帮我们关闭了转换成commonjs规范的功能)

```
presets: [
    ['@babel/preset-env', {module: 'commonjs'}]
]
```

9. 常用插件简述

- webpack-dev-server
- clean-webpack-plugin：编译前清理输出目录
- CopyWebpackPlugin：复制文件
- HotModuleReplacementPlugin：热更新
- ProvidePlugin：全局变量设置
- DefinePlugin：定义全局常量
- splitChunks（老版本用CommonsChunkPlugin）：提取公共模块，将符合引用次数的模块打包到一起
- mini-css-extract-plugin（老版本用ExtractTextWebpackPlugin）：css单独打包
- TerserPlugin（老版本用UglifyJsPlugin）：压缩代码

- progress-bar-webpack-plugin：编译进度条
- DllPlugin& DllReferencePlugin：提高打包效率，仅打包一次第三方模块
- webpack-bundle-analyzer：可视化的查看webpack打包出来的各个文件体积大小
- thread-loader,happypack：多进程编译，加快编译速度

nodejs

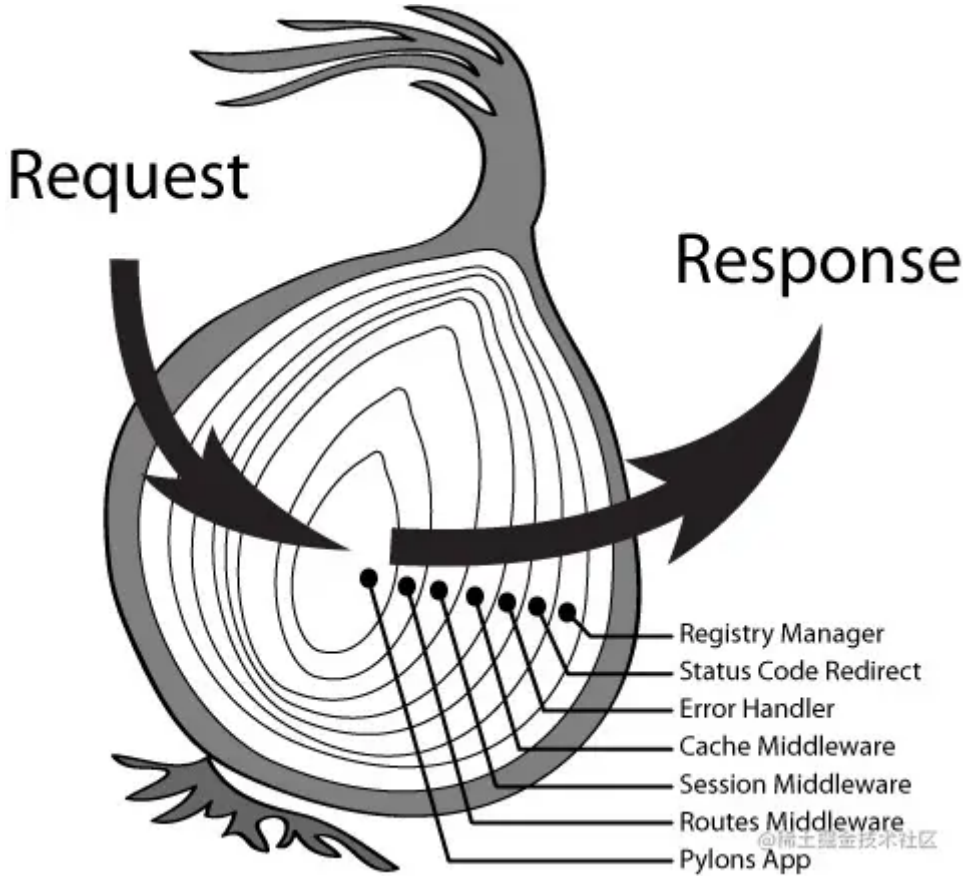
1. 我写的nodejs相关文章

- koa+mongondb实现用户登录注册模块
- nodejs的fs模块回调函数风格的异步操作转化为promise和await风格

2. nodejs常用模块

- path
- fs
- http，url
- express
- koa
- mongoose
- process
- cookie, session
- crypto加密相关
- os

3. koa 中间件思想，洋葱模型




```
class Middleware {
    constructor() {
        this.middlewares = [];
    }
    use(fn) {
        if(typeof fn !== 'function') {
            throw new Error('Middleware must be function, but get ' + typeof fn);
        }
        this.middlewares.push(fn);
        return this;
    }
    compose() {
        const middlewares = this.middlewares;
        return dispatch(0);
        function dispatch(index) {
            const middleware = middlewares[index];
            if (!middleware) {return;}
            try{
                const ctx = {};
                const result = middleware(ctx, dispatch.bind(null, index + 1));
                return Promise.resolve(result);
            } catch(err) {
                return Promise.reject(err);
            }
        }
    }
```

```
}

// 使用
const middleware = new Middleware();
middleware.use(async (ctx, next) => {
    console.log(1);
    await next();
    console.log(2);
});
middleware.use(async (ctx, next) => {
    console.log(3);
    await next();
    console.log(4);
});
middleware.compose();// 1 3 4 2
```

4. express和koa的区别

1. 编码风格：express采用回调函数风格，koa1 采用 generator，koa2(默认)使用await，风格上更加优雅，koa与es6,7的结合更加紧密;
2. 错误处理：express采用在回调中错误优先的处理方式，深层次的异常捕获不了，使得必须在每一层回调里面处理错误，koa的使用try catch捕获错误，将错误上传可以统一处理错误;
3. Koa 把 Express 中内置的 router、view 等功能都移除了，使得框架本身更轻量;
4. express社区较大，文档也相对较多，koa社区相对较小;

网络

1. 网络七层协议（OSI模型）

OSI是一个定义良好的协议规范集，并有许多可选部分完成类似的任务。它定义了开放系统的层次结构、层次之间的相互关系以及各层所包括的可能的任务，作为一个框架来协调和组织各层所提供的服务。

OSI参考模型并没有提供一个可以实现的方法，而是描述了一些概念，用来协调进程间通信标准的制定。即OSI参考模型并不是一个标准，而是一个在制定标准时所使用的概念性框架。

- 第7层 **应用层** 应用层（Application Layer）提供为应用软件而设计的接口，以设置与另一应用软件之间的通信。例如：HTTP、HTTPS、FTP、Telnet、SSH、SMTP、POP3等。

- 第6层 **表示层** 表示层（Presentation Layer）把数据转换为能与接收者的系统格式兼容并适合传输的格式。

- 第5层 **会话层** 会话层（Session Layer）负责在数据传输中设置和维护计算机网络中两台计算机之间的通信连接。

- 第4层 **传输层** 传输层（Transport Layer）把传输表头（TH）加至数据以形成数据包。传输表头包含了所使用的协议等发送信息。例如:传输控制协议（TCP）等。

- 第3层 **网络层** 网络层（Network Layer）决定数据的路径选择和转寄，将网络表头（NH）加至数据包，以形成分组。网络表头包含了网络资料。例如:互联网协议（IP）等。

- 第2层 **数据链路层** 数据链路层（Data Link Layer）负责网络寻址、错误侦测和改错。当表头和表尾被加至数据包时，会形成信息框（Data Frame）。数据链表头（DLH）是包含了物理地址和错误侦测及改错的方法。数据链表尾（DLT）是一串指示数据包末端的字符串。例如以太网、无线局域网（Wi-Fi）和通用分组无线服务（GPRS）等。

分为两个子层：逻辑链路控制（logical link control，LLC）子层和介质访问控制（Media access control，MAC）子层。

- 第1层 **物理层** 物理层（Physical Layer）在局部局域网上发送数据帧（Data Frame），它负责管理电脑通信设备和网络媒体之间的互通。包括了针脚、电压、线缆规范、集线器、中继器、网卡、主机接口卡等。

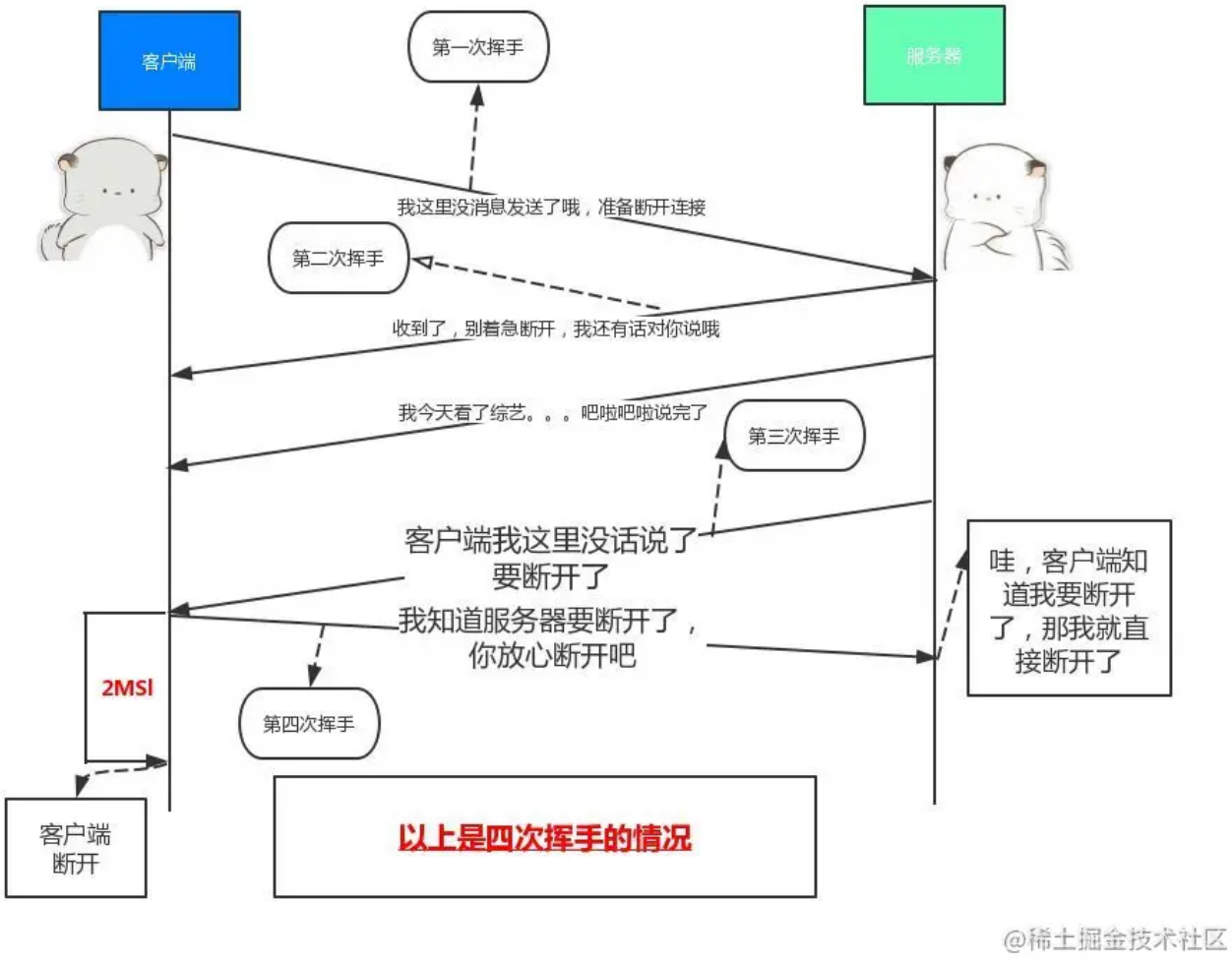
2. TCP 协议三次握手

- 客户端通过 `SYN` 报文段发送连接请求，确定服务端是否开启端口准备连接。状态设置为 `SYN_SEND`；
- 服务器如果有开着的端口并且决定接受连接，就会返回一个 `SYN+ACK` 报文段给客户端，状态设置为 `SYN_RECV`；
- 客户端收到服务器的 `SYN+ACK` 报文段，向服务器发送 `ACK` 报文段表示确认。此时客户端和服务器都设置为 `ESTABLISHED` 状态。连接建立，可以开始数据传输了。

翻译成大白话就是： **客户端**：你能接收到我的消息吗？ **服务端**：可以的，那你能接收到我的回复吗？ **客户端**：可以，那我们开始聊正事吧。

为什么是3次? ：避免历史连接，确认客户端发来的请求是这次通信的人 **为什么不是4次?** ：3次够了第四次浪费

3. TCP 协议四次挥手



1. 为什么不是两次?
    - 两次情况客户端说完结束就立马断开不再接收，无法确认服务端是否接收到断开消息，但且服务端可能还有消息未发送完。
2. 为什么不是三次?
    - 3次情况服务端接收到断开消息，向客户端发送确认接受消息，客户端未给最后确认断开的回复。

http

1. HTTP 请求报文结构

1. 首行是**Request-Line**包括：**请求方法**，**请求URI**，**协议版本**，**CRLF**
2. 首行之后是若干行**请求头**，包括**general-header**，**request-header**或者**entity-header**，每个一行以CRLF结束
3. 请求头和消息实体之间有一个**CRLF分隔**
4. 根据实际请求需要可能包含一个**消息实体** 一个请求报文例子如下：

```sh 复制代码
GET /Protocols/rfc2616/rfc2616-sec5.html HTTP/1.1
Host: www.w3.org
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1
Referer: https://www.google.com.hk/
Accept-Encoding: gzip,deflate,sdch
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cookie: authorstyle=yes
If-None-Match: "2cc8-3e3073913b100"
If-Modified-Since: Wed, 01 Sep 2004 13:24:52 GMT

name=qiu&age=25
```

2. HTTP 响应报文结构

- 首行是状态行包括：**HTTP版本，状态码，状态描述**，后面跟一个CRLF
- 首行之后是**若干行响应头**，包括：**通用头部，响应头部，实体头部**
- 响应头部和响应实体之间用**一个CRLF空行**分隔
- 最后是一个可能的**消息实体** 响应报文例子如下：

```sh 复制代码
HTTP/1.1 200 OK
Date: Tue, 08 Jul 2014 05:28:43 GMT
Server: Apache/2
Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT # 最后修改时间，用于协商缓存
ETag: "40d7-3e3073913b100" # 文件hash，用于协商缓存
Accept-Ranges: bytes
Content-Length: 16599
Cache-Control: max-age=21600 # 强缓存（浏览器端）最大过期时间
Expires: Tue, 08 Jul 2014 11:28:43 GMT # 强缓存（浏览器端）过期时间
P3P: policyref="http://www.w3.org/2001/05/P3P/p3p.xml"
Content-Type: text/html; charset=iso-8859-1

{"name": "qiu", "age": 25}
```

3. HTTP常见状态码及其含义

- 1XX：信息状态码
  - 100 Continue 继续，一般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息
- 2XX：成功状态码

    - 200 OK 正常返回信息
    - 201 Created 请求成功并且服务器创建了新的资源
    - 202 Accepted 服务器已接受请求，但尚未处理
- 3XX：重定向
    - 301 Moved Permanently 请求的网页已永久移动到新位置。
    - 302 Found 临时性重定向。
    - 303 See Other 临时性重定向，且总是使用 GET 请求新的 URI。
    - 304 Not Modified 自从上次请求后，请求的网页未修改过。
- 4XX：客户端错误
    - 400 Bad Request 服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。
    - 401 Unauthorized 请求未授权。
    - 403 Forbidden 禁止访问。
    - 404 Not Found 找不到如何与 URI 相匹配的资源。
- 5XX: 服务器错误
    - 500 Internal Server Error 最常见的服务器端错误。
    - 503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）。

4. 常见web安全及防护

- 参考：常见web攻击及防护方法

5. http1.0、http1.1、http2.0区别

§ http1.0 vs http1.1

1. **长连接**，Connection请求头的值为Keep-Alive时，客户端通知服务器返回本次请求结果后保持连接，可有效减少TCP的三次握手开销；
2. **缓存处理**，增加了 `Cache-Control` 等缓存相关头（看下面详细介绍）；
3. **请求头增加Host**，在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机（Multi-homed Web Servers），并且它们共享一个IP地址。HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误（400 Bad Request）。
4. **请求头增加 `range` 可用于** 断点续传 ，它支持只请求资源的某个部分，可用于 断点续传 。
5. **增加请求方法**：（OPTIONS,PUT, DELETE, TRACE, CONNECT）

6. **新增了24个状态码**，（如100Continue，发请求体之前先用 请求头 试探一下服务器，再决定要不要发 请求体 ）

** 6. http1.x vs http2.0**

1. **服务端推送**
2. **多路复用**，多个请求都在同一个TCP连接上完成
3. **报文头部压缩**，HTTP2.0可以维护一个字典，差量更新HTTP头部，大大降低因头部传输产生的流量

- 参考：HTTP1.0、HTTP1.1 和 HTTP2.0 的区别
-

7. Cache-Control

用于判断 强缓存 ，也就是是否直接取在客户端缓存文件，不请求后端。

请求头和响应头都可以使用。

请求时的缓存指令包括：no-cache、no-store、max-age、max-stale、min-fresh、only-if-cached， **响应消息中的指令**包括：public、private、no-cache、no-store、no-transform、must- revalidate、proxy-revalidate、max-age。

常用指令含义如下：

- `no-cache` ： 不使用本地缓存。需要使用缓存协商，先与服务器确认返回的响应是否被更改，如果之前的响应中存在ETag，那么请求的时候会与服务端验证，如果资源未被更改，则可以避免重新下载。
- `no-store` ： 直接禁止游览器缓存数据，每次用户请求该资源，都会向服务器发送一个请求，每次都会下载完整的资源。
- `public` ： 可以被所有的用户缓存，包括终端用户和CDN等中间代理服务器。
- `private` ： 只能被终端用户的浏览器缓存，不允许CDN等中继缓存服务器对其缓存。
- `max-age` ： 指示客户机可以接收生存期不大于指定时间（以秒为单位）的响应。

8. https

> HTTPS = HTTP + 加密 + 认证 + 完整性保护

`http的不足` ：

- 通信使用未加密的明文，内容容易被窃取
- 不验证通信方的身份，容易遭遇伪装
- 无法验证报文的完整性，容易被篡改

`https` 就是为了解决上述http协议的安全性问题诞生的。https并非是应用层的新协议，是基于http协议的，将http和tcp协议接口部分用 `SSL` 和 `TLS` 协议代替而已。

http: IP → TCP → HTTP（应用层）

https: IP → TCP → SSL → HTTP（应用层）

§ 加密

- SSL（Secure Sockets Layer安全套接字层协议）
- TLS（Transport Layer Security传输层安全协议）

对称加密：用同一个密钥加密、解密。常用对称加密算法：AES，RC4，3DES **非对称加密**：用一个密钥加密的数据，必须使用另一个密钥才能解密。常用非对称加密算法：RSA，DSA/DSS

常用HASH算法：MD5，SHA1，SHA256

其中 `非对称加密` 算法用于在握手过程中加密生成的密码， `对称加密` 算法用于对真正传输的数据进行加密，而 `HASH` 算法用于验证数据的完整性。

由于浏览器生成的密码是整个数据加密的关键，因此在传输的时候使用了非对称加密算法对其加密。

非对称加密算法会生成公钥和私钥，公钥只能用于加密数据，因此可以随意传输，而网站的私钥用于对数据进行解密，所以网站都会非常小心的保管自己的私钥，防止泄漏。

§ CA 证书包含的信息：：

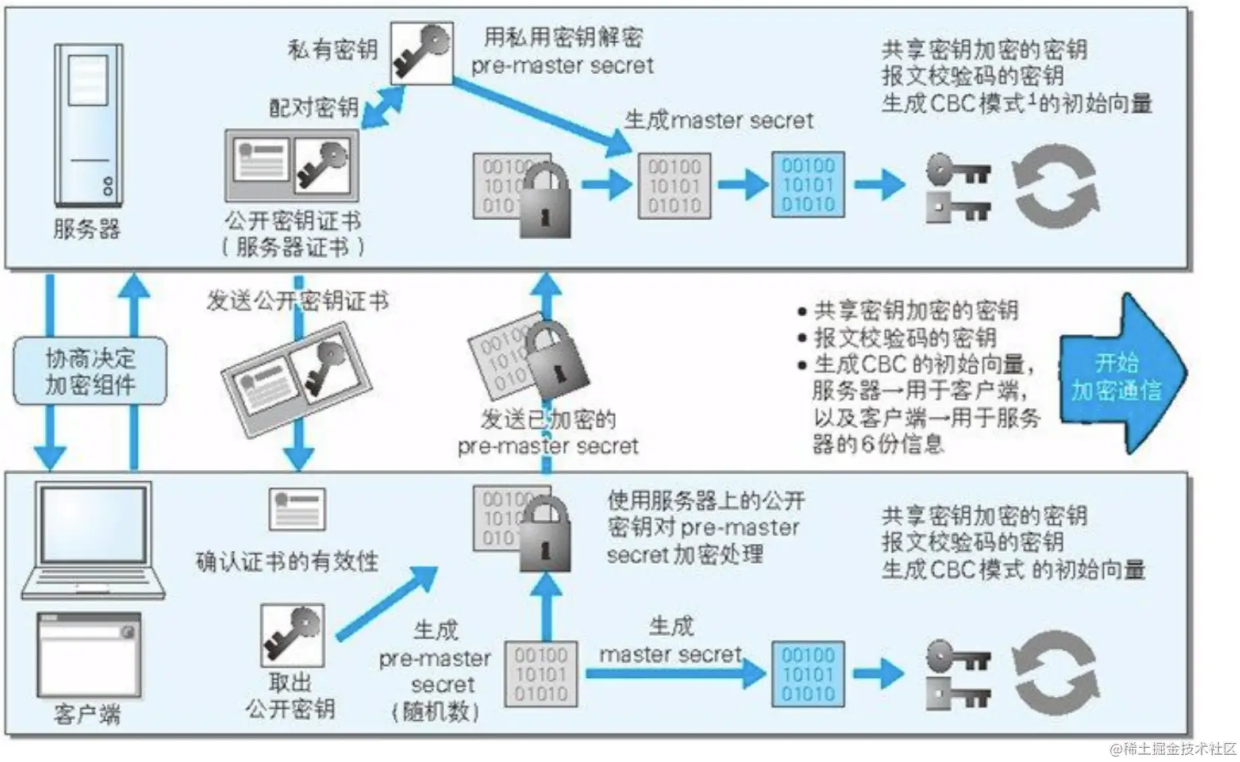
CA（Certification Authority证书颁发机构）

- 公钥
- 网站地址
- 证书的颁发机构
- 过期时间

§ https加密通信的流程：

1. 浏览器输入url请求网站，并附带自己支持的一套加密规则。
2. 网站制作证书，这个应该是网站建站的时候就已经弄好了。证书分为服务器证书和客户端证书，我们所说的证书一般都是指服务器证书（ssl证书详细解读）。制作过程如下：
    1. 制作CSR文件。就是制作Certificate Secure Request证书请求文件，制作过程中，系统会产生2个密钥，一个是公钥就是这个CSR文件，另一个是私钥，存在服务器上。
    2. CA认证。将CSR文件提交给CA，一般有两种认证方式：域名认证 和 企业文档认证。
    3. 证书安装。收到CA证书后，将证书部署在服务器上。
3. 网站从浏览器发过来的加密规则中选一组自身也支持的加密算法和hash算法，并向浏览器发送带有公钥的证书，当然证书还包含了很多信息，如网站地址、证书的颁发机构、过期时间等。
4. 浏览器解析证书。
    1. 验证证书的合法性。如颁发机构是否合法、证书中的网站地址是否与访问的地址一致，若不合法，则浏览器提示证书不受信任，若合法，浏览器会显示一个小锁头。
    2. 若合法，或用户接受了不合法的证书，浏览器会生成一串随机数的密码（即密钥），并用证书中提供的公钥加密。
    3. 使用约定好的hash计算握手消息，并使用生成的随机数（即密钥）对消息进行加密，最后将之前生成的所有消息一并发送给网站服务器。
5. 网站服务器解析消息。用已有的私钥将密钥解密出来，然后用密钥解密发过来的握手消息，并验证是否跟浏览器传过来的一致。然后再用密钥加密一段握手消息，发送给浏览器。
6. 浏览器解密并计算握手消息的HASH，如果与服务端发来的HASH一致，此时握手过程结束，之后所有的通信数据将由之前浏览器生成的随机密码并利用对称加密算法进行加密。这里浏览器与网站互相发送加密的握手消息并验证，目的是为了保证双方都获得了一致的密码，并且可以正常的加密解密数据，为后续真正数据的传输做一次测试。

下图表示https加密通信的过程：



设计模式

- 常用设计模式分类
- 单例模式（Singleton Pattern）
- 发布订阅模式（又叫观察者模式Observer Pattern）
- 中介者模式（Mediator Pattern）
- 策略模式（Strategy Pattern）
- 代理模式（Proxy Pattern）
- 迭代器模式（Iterator Pattern）
- 适配器模式（Adapter Pattern）
- 命令模式（Command Pattern）
- 组合模式（Composite Pattern）
- 模板方法模式（Template Method）
- 享元模式（Flyweight Pattern）
- 模职责链模式 （Chain of Responsibility Pattern）
- 状态模式(State Pattern)

git 常用操作

- 参考：Git常用操作指南

命令	作用
git status	
git diff	
git add	
git commit	
git push	
git branch	
git checkout	
git pull/fetch	git pull = git fetch + git merge
git merge	

§ git rebase 作用：将一个分支的更改合并入另一个分支。

目的：

1. 让多个人在同一个分支开发的提交节点形成一条线，而不是多条线
2. 让你提交的commit在该分支的最前面

使用场景：

1. 开发分支合并主分支的更新；
2. 不能在主分支上使用rebase，因为会破坏历史记录

§ merge 和 rebase 的区别 git rebase和git merge做的事其实是一样的。它们都被设计来将一个分支的更改并入另一个分支，只不过方式有些不同。

- git merge 将两个分支的提交记录按照顺序排序
- git rebase 把被合并的旁分支的修改直接追加在主分支提交记录后面。

一些其他常见问题

- 自我介绍
- 你有什么爱好?
- 你最大的优点和缺点是什么?
- 你为什么会选择这个行业，职位?
- 你觉得你适合从事这个岗位吗?
- 你有什么职业规划?
- 你对工资有什么要求?
- 如何看待前端开发?
- 未来三到五年的规划是怎样的?
- 你的项目中技术难点是什么？遇到了什么问题？你是怎么解决的?
- 你们部门的开发流程是怎样的
- 你认为哪个项目做得最好?
- 说下工作中你做过的一些性能优化处理?
- 最近在看哪些前端方面的书?
- 平时是如何学习前端开发的?
- 你最有成就感的一件事?
- 你为什么要离开前一家公司?
- 你对加班的看法?
- 你希望通过这份工作获得什么?
- 面试完你还有什么问题要问的吗?