面试题

CSS

介绍盒模型?☆☆☆

盒子水平垂直居中的实现方案?☆

介绍BFC?如何触发BFC?☆☆

伪元素和伪类的区别?

介绍position定位?☆☆

介绍display?

display:none和visibility:hidden的区别?

选择器优先级和权重值是怎样的?☆（拼多多20选择题）

介绍Flex弹性布局?（拼多多20大题）☆

透明度设置中opacity和rgba的区别?

CSS引入方式有哪些?link和@import的区别?

link引入的CSS为什么要放在head标签中?而js放在最后?（拼多多1面）

async和defer的区别?（拼多多1面）

CSS性能优化的技巧有哪些?

CSS3新特性有哪些?

介绍CSS预处理器?

介绍懒加载和预加载?（拼多多20选择题）

CSS高度重叠问题?（猿辅导20选择题）

CSS动画的性能优化?

使用CSS画一个椭圆形/三角形?☆

CSS画圆的方式有哪些?☆☆

如何画一条0.5px的线?（沃德博创实习一面）

移动端适配方案有哪些?（腾讯第一次1面）

rem和em的区别?（腾讯第一次1面）

CSS可以继承的属性?

HTML

行内元素和块元素的区别?

Doctype的作用是什么?

HTML5新特性有哪些?

对HTML语义化的理解?☆

如何优化SEO?

JS

数据类型/操作符

JavaScript的数据类型有哪些?(字节第一次1面)☆

基本类型和引用类型的区别?(字节第一次1面，字节data1面)☆

Map和WeakMap的区别?

赋值、浅拷贝和深拷贝的区别?手写代码?☆

拥有Symbol.iterator()的数据类型有哪些?（拼多多20选择题）

typeof运算符能判断哪些类型?在具体案例中判断typeof的结果?

介绍instanceof的作用和原理？（快手1面）

使用双等号时发生的常见隐式类型转换?

[]==![]和{}=={}的结果和原因是什么?（腾讯第一次1面）

{}+[]和[]+{}的结果和原因是什么?
使用Number()、Boolean()进行显式类型转换的转换结果?
以下对象占用的堆内存是多少byte? （拼多多20选择题）
ES规定有符号整数的最大值和最小值分别为多少? （拼多多20选择题）
（大意是）使用bufferArray的Unit32Array/Float32Array等存储png有4个颜色通道，每个通道占8字节，64x64像素的图片? （猿辅导20选择题）
‘1’+2，‘1’-2分别打印什么? （拼多多20选择题）
加法运算符和三目运算符的优先级? （拼多多20选择题）
用最简洁的代码建立一个数组，其值依次为0-99?（沃德博创实习1面）

面向对象
new的具体过程是什么? ☆
实现继承的方案有哪些?各自的优缺点是什么? ☆

作用域/this/闭包
什么是闭包?优缺点? ☆☆☆☆(字节data1面、沃德博创实习1面、酷家乐1面)
this的指向? （拼多多20大题）
this代码题
根据作用域的知识，请问以下代码的运行结果?

内存/性能
js的垃圾回收机制?(字节data1面)
ES中垃圾回收时机? （猿辅导20选择题）
如何减少白屏时间? （猿辅导20选择题、拼多多20选择题）
reflow（重排）、repaint（重绘）和 composite （合并）是什么?☆
如何减少重排重绘? ☆
throttle（节流）与debounce（防抖）的区别?手写代码? ☆

执行机制/异步
介绍事件循环(Event Loop)?(字节data1面)☆
介绍Promise? （字节第一次1面）☆
promise.then()、process.nextTick()、同步代码的执行顺序? （拼多多20大题）
异步代码的执行结果?

chrome
浏览器渲染页面的过程?

安全/跨域
xss和csrf的攻防? ☆
CSRF攻击具体是怎么实现的? （拼多多2面）
什么是同源策略?跨域的常见方案有哪些? ☆（拼多多1面、拼多多2面）
同源站点是否可能拥有相同的js引擎上下文? （猿辅导20选择题）
非同源站点是否可以使用window.postMessage() 进行通信? （猿辅导20选择题）
为什么浏览器的请求有两次，第一次是options，第二次才是真正请求?

设计模式
框架/工程化
介绍MVC、MVVM?
介绍双向数据绑定原理? （腾讯第二次1面、字节第一次2面）☆☆☆

- 前端工程化中的依赖反转、控制反转等，是为了实现什么目的？（猿辅导20选择题）
- 以下哪些可以实现webpack的tree shaking？（猿辅导20选择题）
- 介绍webpack？（拼多多1面）
- webpack、grunt、gulp的不同？
- webpack中的bundle、chunk、module是什么？
- webpack中的loader和plugin是什么？
- 有哪些常见的Loader？以及它们的作用？
- 有哪些常见的Plugin？以及它们的作用？
- webpack的构建流程是什么？
- 什么是Tree-shaking？
- 热更新HMR原理是什么？
- 如何提高webpack的构建速度？
- webpack的优缺点？
- 怎么配置单页应用？怎么配置多页应用？
- 介绍react？react的生命周期？（拼多多2面）
- 介绍vue？vue的生命周期？（拼多多2面）

DOM

- 介绍DOM事件流？
- 如何阻止事件冒泡？
- 如何取消默认事件？
- 怎样使用DOM API获取带有xx类的a标签？（拼多多20选择题）

其他技术

- 介绍Web worker？（腾讯1面）
- 介绍WebAssembly？（腾讯1面）

网络

- 前端登录的方式有哪些？各自的优缺点以及登录流程是怎样的？（字节第一次2面）
- Cookie和session的区别？cookie和session之间如何联系？如果禁用cookie可以访问session吗？（腾讯第二次1面）
- 如何判断用户的使用的机型？（腾讯1面，腾讯第二次1面）
- 介绍从输入url到页面展示的过程？☆☆☆（拼多多1面）
- 为什么需要三次握手，两次握手不可以吗？（拼多多1面）
- TCP和UDP的区别？☆☆☆
- 介绍DNS实现原理？（字节第一次2面）
- DNS优化的优化方案有哪些？
- DNS服务器有哪些类型？

HTTP

- 介绍HTTPS的加密？(腾讯第一次1面，字节data1面)☆☆
- HTTP常见状态码？☆☆
- HTTP2.0新特性有哪些？（腾讯第二次1面）☆
- POST和GET的区别是什么？（腾讯第二次1面）☆☆
- HTTP队头阻塞及解决方案？（拼多多20大题）

操作系统

面试题

CSS

介绍盒模型？☆☆☆

盒模型分类：

1. 标准盒模型，大小：width/height = content，设置方式：box-sizing: content-box。

2. IE盒模型，大小：width/height = content + border + padding，设置方式：box-sizing: border-box。

盒子水平垂直居中的实现方案？☆

答：下面介绍四种方案

- 方案一：定位1
  缺点：必须知道盒子的宽高。

```
html,body{
    position: relative;
    height: 100%;
}
#myDiv{
    background-color: #f00;
    position: absolute;
    left: 50%;
    top: 50%;
    margin-left: -150px;
    margin-top: -150px;
    width: 300px;
    height: 300px;
}
```

- 方案二：定位2
  优点：无需知道盒子宽高
  缺点：盒子必须设置width和height

```
html,body{
    position: relative;
    height: 100%;
}
#myDiv{
    background-color: #f00;
    width: 300px;
    height: 300px;
    position: absolute;
    top: 0;
    bottom: 0;
    left: 0;
    right: 0;
    margin: auto;
}
```

- 方案三：定位3
  优点：不设置盒子的width/height也有效；无需知道盒子宽高。

```
html,body{
    position: relative;
    height: 100%;
}
#myDiv{
    background-color: #f00;
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%,-50%);
}
```

- 方案四：flex

```
html,body{
    height: 100%;
}
body{
    display: flex;
    justify-content: center;
    align-items: center;
}
#myDiv{
    width: 300px;
    height: 300px;
    background-color: #f00;
}
```

- 方案四：grid

介绍BFC？如何触发BFC？☆☆

BFC：具有 BFC 特性的元素可以看作是隔离了的独立容器，容器里面的元素不会在布局上影响到外面的元素，并且 BFC 具有一些特性。
触发方式：只要元素满足下面任一条件即可触发 BFC ：

1. body 根元素
2. 浮动元素：float 除 none 以外的值
3. 绝对定位元素：position (absolute、fixed)
4. display 为 inline-block、table-cells、flex
5. overflow 除了 visible 以外的值 (hidden、auto、scroll)

BFC 特性：

1. 同一个 BFC 下外边距会发生折叠。解决方案：将其放在不同的 BFC 容器中。
2. BFC 可以包含浮动的元素。这个特性解决了高度塌陷问题。
3. BFC 可以阻止元素被浮动元素覆盖。

10 分钟理解 BFC 原理，知乎

伪元素和伪类的区别?

伪类用于当已有元素处于的某个状态时，为其添加对应的样式，这个状态是根据用户行为而变化的。伪元素用于创建一些不在文档树中的元素，并为其添加样式。

介绍position定位? ☆☆

除static外，其余三种均开启了定位。开启定位后，可以设置top、right、bottom、left和z-index。

1. static：默认值。
2. relative：未脱离文档流，参照物为原来的位置。
3. absolute：脱离文档流，参照物为最近的已开启定位祖先元素，如果所有祖先元素均没有开启定位，则会相对于浏览器窗口进行定位。
4. fixed：脱离文档流，参照物为浏览器窗口，不受滚动条影响。
5. sticky：有点像relative和fixed的结合
    - sticky必须搭配top、bottom、left、right之一，否则等同于relative定位

介绍display?

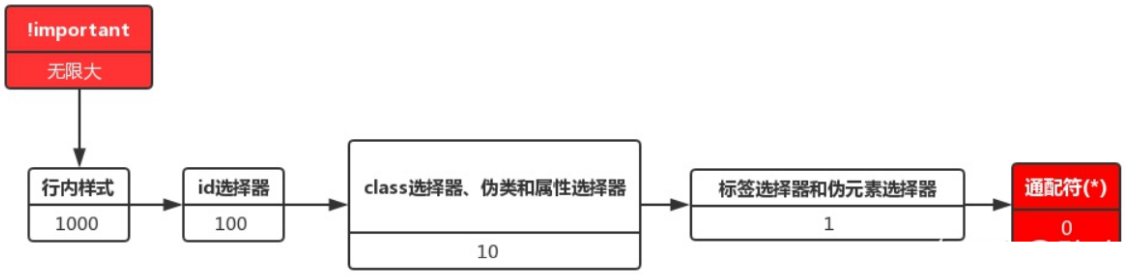
inline：作为内联元素显示。
block：作为块元素显示。
inline-block：作为行内块元素显示，既能设置宽高，又不会独占一行。
none：隐藏元素，且不会在占页面位置。
table：块级表格元素

display:none和visibility:hidden的区别?

区别	display:none	visibility：hidden的
是否占据空间	不占据任何空间，在文档渲染时，该元素如同不存在（但依然存在文档对象模型树中）	该元素空间依旧存在
是否渲染	会触发reflow（回流），进行渲染	只会触发repaint（重绘），因为没有发现位置变化，不进行渲染
是否是继承属性	不是继承属性，元素及其子元素都会消失	是继承属性，若子元素使用了visibility:visible，则不继承，这个子孙元素又会显现出

补充说明：父元素设置为visibility:hidden之后，子元素只要设置为visibility:visible就可以显示出子元素了，但是如果父元素设置为display:none的话，则没办法显示子元素了

选择器优先级和权重值是怎样的？☆（拼多多20选择题）



介绍Flex弹性布局？（拼多多20大题）☆

容器：指的是采用 Flex 布局的元素，它的属性有：

1. flex-direction：决定主轴的方向和起始点。
2. flex-wrap：决定项目换行。
3. flex-flow：1和2的简写。
4. justify-content：决定项目在主轴上的对齐方式。
5. align-items：决定项目在交叉轴上的对齐方式。
6. align-content：决定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。

项目：指的是容器的子孙元素，它的属性有：

1. order：决定项目的排列顺序。数值越小，越靠前。
2. flex-grow：决定项目的放大比例。默认为0，即如果存在剩余空间，也不放大。
3. flex-shrink：决定项目的缩小比例。默认为1，即如果空间不足，该项目将缩小。

4. flex-basis：决定项目在主轴方向上的初始大小。
5. flex：2、3、4的简写。
6. align-self：决定单个项目有与其他项目不一样的对齐方式，可覆盖align-items属性。

透明度设置中opacity和rgba的区别?

rgba仅改变元素自身的透明度，而opacity不仅会影响元素自身还会影响其子孙元素和内部文字的透明度。

CSS引入方式有哪些？link和@import的区别?

CSS引入的四种方式为：

1. 内联方式
2. 嵌入方式style
3. 链接方式link
4. 导入方式import

link和@import的区别：（建议不要使用@import）

1. 归属问题：link 属于 HTML，而 @import 属于 CSS，所以导入语句应写在 CSS 中。
2. 加载问题：当 HTML 被加载时，link 文件会同时被加载，而 @import 引用的文件则会等页面全部下载完毕再被加载。

link引入的CSS为什么要放在head标签中？而js放在最后？（拼多多1面）

最佳实践：用于引入css的link标签放在head标签内，而用于引入js的script标签，放在最后的 `</body>` 之前。
根本原因：

1. js会阻塞html的加载
2. css不会阻塞html的加载

async和defer的区别？（拼多多1面）

补充：

1. 没有 defer 或 async的情况下，浏览器会立即加载并执行指定的脚本。

前提：

1. async和defer都是只针对包含外部文件的script标签。

区别：

1. async和defer都是立即下载。（注意：这其中的下载过程并不会阻塞html parse，而执行过程将会阻塞html parse。）async是下载完立即执行，与DOMContentLoaded事件的顺序无关。defer则是下载完并且在html parse完毕之后，再执行，发生在DOMContentLoaded事件之前。
2. 红宝书中写道：html5规定，多个defer文件将按照书写顺序执行，并且均在DOMContentLoaded事件之前执行，但是在实际过程中，defer的脚本间未必按照顺序执行或者在DOMContentLoaded事件之前执行。
3. 多个async文件的书写顺序与执行顺序无关。

CSS性能优化的技巧有哪些?

实用型：

1. 文件压缩
2. 去除无用CSS
3. 内联首屏关键CSS
4. 异步加载次要CSS。有三种异步加载CSS的实现方式：
    - 将link标签的media属性设置为用户浏览器不匹配的媒体类型。在文件加载完成之后，需要将media的值设回screen或all，从而让浏览器开始解析CSS。
    - 将link标签的rel属性标记为alternate可选样式表。加载完成之后，也需要将rel改回去。
    - 使用js动态创建样式表link元素，并插入到DOM中。
    - 设置preload。这种方式比使用不匹配的media方法能够更早地开始加载CSS。案例：`<link rel="preload" href="style.css" as="style">`

建议型：

1. 不要使用@import：导致浏览器无法并行下载CSS文件，同时，多个@import会发生下载顺序紊乱。
2. 减少使用昂贵的属性，如：如box-shadow、border-radius等。
3. 有选择地使用选择器：CSS选择器的匹配是从右向左进行的。
4. 减少reflow和repaint
    - 重排会导致浏览器重新计算整个文档，重新构建渲染树。应避免：改变font-size和font-family、改变元素的内外边距、通过JS改变CSS类、通过JS获取DOM元素的位置相关属性（如width/height/left等）、CSS伪类激活、滚动滚动条或者改变窗口大小。
    - 当元素的外观（如color，background，visibility等属性）发生改变时，会触发重绘。

CSS性能优化的8个技巧，奇舞周刊

CSS3新特性有哪些?

1. 圆角（border-radius），阴影（box-shadow），
2. 文字特效（text-shadow），线性渐变（gradient），旋转（transform）
3. transform:rotate(9deg) scale(0.85,0.90) translate(0px,-30px) skew(-9deg,0deg);// 旋转,缩放,定位,倾斜
4. 增加了更多的CSS选择器、rgba
5. 在CSS3中唯一引入的伪类是 ::selection
6. 媒体查询@media，多栏布局

介绍CSS预处理器?

CSS预处理器的基本思想： 用一种专门的编程语言, 为CSS增加一些特性, 将CSS作为目标生成文件。
CSS预处理器的作用：提供 CSS 缺失的样式层复用机制、减少冗余代码，提高样式代码的可维护性。
主流的 CSS 预处理器：Less、Sass、Stylus。

介绍懒加载和预加载？（拼多多20选择题）

1. 懒加载（延迟加载）
   图片懒加载：只有当图片出现在浏览器的可视区域内时，才设置图片正真的路径，让图片显示出来。
   懒加载的原理：先在页面中把所有的图片统一使用一张占位图进行占位，把真正的路径存在元素的某个属性里，要用的时候就取出来，再设置。
2. 预加载
   图片预加载：提前加载图片，当用户需要查看时，可直接从本地缓存中渲染。

CSS高度重叠问题？（猿辅导20选择题）

```
<div class='box1' style='margin-bottom:20px'></div>
<div class='box2'></div>

<style>
.box1,.box2{
    width:300px;
    height:300px;
    margin:30px;
}
</style>
```

具体问题：box1和box2垂直距离是多少?

答：打印30px

CSS动画的性能优化?

1. 尽可能多地利用硬件能力，如使用3D变形来开启GPU加速
2. 尽可能少的使用 box-shadow 和 gradients等昂贵的属性
3. 尽可能让动画元素不在文档流中，如：设置为fixed或absolute
4. 避免重排重绘

使用CSS画一个椭圆形/三角形? ☆

椭圆

```
div{
    width: 200px;
    height: 100px;
    background-color: #acc;
    border-radius: 100px/50px;
}
```

三角形

```
div{
    width: 0;
    height: 0;
    border-left: 50px solid transparent;
    border-right: 50px solid transparent;
    border-bottom: 100px solid #acc;
}
```

CSS画圆的方式有哪些? ☆☆

1. Border Radius

```
div{
    width: 100px;
    height: 100px;
    background-color: #acc;
    border-radius: 50px;
}
```

2. SVG

```
<svg width="80" height="80">
    <circle class="circle" cx="40" cy="40" r="40" />
</svg>
```

3. Clip Path

```
div {
    background: #456BD9;
    clip-path: circle(50%);
    height: 100px;
    width: 100px;
}
```

4. Radial Gradient

```
div {
    background: radial-gradient(black 70%, transparent 70%);
    height: 100px;
    width: 100px;
}
```

如何画一条0.5px的线？（沃德博创实习一面）

进行缩小处理

1. 采用meta viewport的方式：这种方式仅针对移动端
2. `transform: scale(0.5,0.5);`

移动端适配方案有哪些？（腾讯第一次1面）

1. `@media`媒体查询
2. rem
3. px + viewport：通过动态设置meta标签的viewport让css中的1px等于设备的1px
4. vh/vw：1vw相当于占整个视口宽度的1%

rem和em的区别？（腾讯第一次1面）

- em就是相对于元素自身font-size的大小，若没有对元素显式地使用绝对单位来声明font-size时，该元素font-size属性自动继承自父元素。如果你在网页中任何地方都没有设置文字大小的话，那它将等于浏览器默认文字大小，通常是16px。
- rem：相对于根元素的font-size

CSS可以继承的属性?

资料

HTML

行内元素和块元素的区别?

答:
常见块元素：div、p、h1~h6、ul、ol、dl、li、table、nav等
常见行内元素：span、img、a、lable、input等

二者的区别:

- 块级:
  元素会独占一行，宽度默认填满其父元素宽度
  元素可以设置宽高
  display:block;
- 行内:
  元素不会独占一行。其宽度随内容的变化而变化
  元素不可以设置宽高
  display:inline;

Doctype的作用是什么?

答:
doctype：用于告知浏览器解析器应该用哪个规范来解析文档。
文档类型分为:

1. 严格模式：又称标准模式，是指浏览器按照 W3C 标准解析代码。
2. 混杂模式：又称怪异模式或兼容模式，是指浏览器用自己的方式解析代码。

HTML5新特性有哪些?

答:

1. 语义化标签
2. 视频和音频：video、audio
3. canvas和svg绘图
4. 增强型表单：type=number/email/tel等
5. web worker：是运行在后台的js，独立于其他脚本，不影响页面性能
6. web storage：localStorage、sessionStorage
7. webSocket

对HTML语义化的理解? ☆

HTML语义化：让页面的内容结构化，便于对浏览器、搜索引擎解析。
优点:

1. 有利于SEO
2. 方便设备解析（如屏幕阅读器、盲人阅读器、移动设备）
3. 提高代码可读性，便于开发和维护

如何优化SEO?

答:

1. 优化布局结构，提倡扁平化结构
2. 将重要内容放在代码最前位置
3. 使用语义化标签
4. 把握重要标签：如 title、description、keywords
5. 设置图片的alt属性
6. 提高网站速度：网站速度是搜索引擎排序的一个重要指标
7. 少用 iframe：搜索引擎不会抓取 iframe 中的内容

JS

数据类型/操作符

JavaScript的数据类型有哪些？(字节第一次1面)☆

引用类型：Object。
基本类型：Number、String、Boolean、Null、Undefined、BigInt(可以表示任意长度的整数，用末尾加n的方式创建)、Symbol(其值表示唯一的标识符)。

基本类型和引用类型的区别？(字节第一次1面，字节data1面)☆

存储：基本类型存储在栈，引用类型存储在堆。
访问：基本类型按值访问，引用类型按地址访问。

追问：为什么一定要分“堆”和“栈”两个存储空间呢？所有数据直接存放在“栈”中不就可以了吗？

这是因为 JavaScript 引擎需要用栈来维护程序执行期间上下文的状态，如果栈空间大了话，所有的数据都存放在栈空间里面，那么会影响到上下文切换的效率，进而又影响到整个程序的执行效率。

Map和WeakMap的区别？

1. WeakMap只接受对象作为key，而Map均可。
2. WeakMap的key所引用的对象都是弱引用，只要对象的其他引用被删除，垃圾回收机制就会释放该对象占用的内存，从而避免内存泄漏。
3. WeakMap的成员随时可能被垃圾回收，导致成员数量不稳定，所以没有size属性。
4. WeakMap没有clear()方法
5. WeakMap不能遍历

赋值、浅拷贝和深拷贝的区别？手写代码？☆

区别：

--	和原数据是否指向同一对象	第一层数据为基本数据类型	原数据中包含子对象
赋值	是	改变会使原数据一同改变	改变会使原数据一同改变
浅拷贝	否	改变**不**会使原数据一同改变	改变会使原数据一同改变
深拷贝	否	改变**不**会使原数据一同改变	改变**不**会使原数据一同改变

1. 浅拷贝：
   手写遍历法

```
function isArray(val) {
    return Object.prototype.toString.call(val).includes("Array");
}

function shallowClone(src) {
    const target = isArray(src) ? [] : {};
```

```
    for (let key in src) { //遍历所有可枚举属性，包括继承属性
        if (src.hasOwnProperty(key)) { //排除继承的属性
            target[key] = src[key];
        }
    }
    return target;
}
```

Object.assign()

```
let obj2 = Object.assign({}, obj1);
```

展开运算符...

```
let obj2= {... obj1}
```

Array.prototype.concat()

```
let arr2 = arr.concat();
```

Array.prototype.slice()

```
let arr3 = arr.slice();
```

2. 深拷贝：
   JSON.stringify和JSON.parse：不能处理函数和正则。存在循环检测，破解了递归爆栈。

```
function cloneJSON(source) {
    return JSON.parse(JSON.stringify(source));
}
```

递归：可以判断类型且存在循环检测

```
function deepClone(obj, hash = new WeakMap()) {
    if (obj === null) return obj;
    if (obj instanceof Date) return new Date(obj);
    if (obj instanceof RegExp) return new RegExp(obj);
    if (typeof obj !== "object") return obj;
    if (hash.get(obj)) return hash.get(obj);
    let cloneObj = new obj.constructor();
    hash.set(obj, cloneObj);
    for (let key in obj) {
```

```
            if (obj.hasOwnProperty(key)) {
                cloneObj[key] = deepClone(obj[key], hash);
            }
        }
        return cloneObj;
    }
```

拥有Symbol.iterator()的数据类型有哪些？（拼多多20选择题）

答：

1. Strings
2. Arrays
3. Maps
4. Sets
5. The arguments object
6. Some DOM collection types like NodeList

typeof运算符能判断哪些类型？在具体案例中判断typeof的结果？

Number、String、Boolean、Undefined、BigInt、Symbol、Object、Function。

```
typeof null;typeof [];typeof {};//"object"
typeof NaN;//"number"
typeof Symbol();//"symbol"
typeof 123n;//"bigint"
```

引申出instanceof问题....

介绍instanceof的作用和原理？（快手1面）

语法：`object instanceof constructor`
原理：用来检测 constructor.prototype 是否存在于参数 object 的原型链上。

问：`==`和`===`的区别以及各自的使用场景是什么？(腾讯第一次1面)
区别：
`===` 严格相等，会比较类型和值。
`==` 抽象相等，会先进行类型转换，然后再比较值。*引申出隐式类型转换的问题....*
适用场景：
绝大多数场合应该使用 `===` 。

引申出隐式类型转换问题....

使用双等号时发生的常见隐式类型转换？

```
undefined==null;//true（2020拼多多笔试）
false==undefined;//false（2020拼多多笔试）
false==null;//false（2020拼多多笔试）
NaN==NaN;//false
[]==[];//false
[]==![];//true
{}=={};//false
{}==!{};//false
{}+[];//0
[]+{};//"[object Object]"
[]+[];//0
{}+{};//"[object Object][object Object]"
```

`[]==![]`和`{}==!{}`的结果和原因是什么？（腾讯第一次1面）

`[]==![]`→`[]=false`→`""==0`→`0==0`→true

`{}==!{}`→`{}==false`→`"[object Object]"==0`→`NaN==0`→false

`{}+[]`和`[]+{}`的结果和原因是什么？

`{}+[]`→`+[]`→`0`
`[]+{}`→`""+{}`→`"[object Object]"`

使用Number()、Boolean()进行显式类型转换的转换结果？

Number()

原始类型	参数	输出
Number	*	不变
Boolean	true/false	1/0
Null	null	0
Undefined	undefined	NaN
String	`"空格000整数.小数"`	整数.小数
String	`"0x十六进制数"`	十进制数
String	`""`或`" "`	0
String	除上述三种情况外	NaN

Boolean()

数据类型	转true的值	转false的值
`String`	非空字符串（包括空格）	`""`
`Number`	非0值（包括Infinity）	0和**NaN**
`Object`	对象(包括空对象)	
`Undefined`	n/a	undefined
`Null`		null

以下对象占用的堆内存是多少byte？（拼多多20选择题）

```
// 类似的代码如下
let obj={
    str:'hehk'
    num:1324254222
}
```

ES规定有符号整数的最大值和最小值分别为多少？（拼多多20选择题）

选项：Math.pow(2,53)-1等

（大意是）使用bufferArray的Unit32Array/Float32Array等存储png有4个颜色通道，每个通道占8字节，64x64像素的图片？（猿辅导20选择题）

‘1’+2，‘1’-2分别打印什么？（拼多多20选择题）

答：”3”、-1

加法运算符和三目运算符的优先级？（拼多多20选择题）

```
let val='hhh';
console.log('valus is ' + (val==='hhh')? 'a':'b' );
```

具体问题：打印结果是什么?
答：打印”a”

用最简洁的代码建立一个数组，其值依次为0-99？（沃德博创实习1面）

```
let arr = [...Array(100).keys()];//keys() 方法用于从数组创建一个包含数组键的可迭代对象。
```

面向对象

new的具体过程是什么？☆

1. 创建一个新对象
2. this指向新对象
3. 执行构造函数中的代码
4. 返回新对象

实现继承的方案有哪些？各自的优缺点是什么？☆

1. 原型链继承

本质：将子类的原型赋值成父类的实例。
优点：

- 可以复用方法

缺点：

- 子类无法向父类的构造函数传参。

```
function Sup(){
    this.property=true;
}
Sup.prototype.getProperty=function(){
    console.log(this.property);
}
function Sub(){
    this.property=false;
}
Sub.prototype=new Sup();//关键代码。需要注意:对Sub.prototype添加属性和方法时，需要放到这行代码之后，如果放在之前，则添加无效。
let ins=new Sub();
ins.getProperty();//false
```

2. 构造函数继承
   本质：在子类构造函数中使用 apply 或 call 调用父类构造函数。

优点：

- 可以传参

缺点：

- 无法复用方法：每个子类都有父类实例函数的副本，影响性能。
- 无法继承父类的原型。

```
function Sup(aColor){
    this.color=["yellow"];
    this.color.push(aColor);
}
function Sub(){
    Sup.call(this,"red");//修改this，同时向父类传参
}
let ins1=new Sub();
let ins2=new Sub();
ins1.color.push("blue");
console.log(ins1.color);// ["yellow", "red", "blue"]
console.log(ins2.color);// ["yellow", "red"]
```

3. 原型链+构造函数组合继承(常用)

本质：属性用构造函数继承，方法用原型链继承。

优点：

- 化解了上面两种方法的缺点。

```
function Sup(){
    this.color=["yellow"];
}
Sup.prototype.say=function(){
    return this;
}
function Sub(){
    Sup.call(this);
}
Sub.prototype=new Sup();
Sub.prototype.constructor=Sub;
let ins1=new Sub();
let ins2=new Sub();
console.log(ins1===ins1.say());//true,得益于原型链
ins1.color.push("blue");
console.log(ins1.color);//["yellow", "blue"]，得益于构造函数模式
console.log(ins2.color);//["yellow"]
```

4. 原型式继承

本质：借助工具函数，跟原型链基本一样

```
/* 1. 自己写 */
function obj(o){
    function F(){}
    F.prototype=o;
    return new F();
}
let person={
    color:["yellow"]
};
let people1=obj(person);//person是people的原型
let people2=obj(person);
people1.color.push("blue");
console.log(people1.color);// ["yellow", "blue"]
console.log(people2.color);// ["yellow", "blue"]
console.log(person.color);// ["yellow", "blue"]

/* 2. 用api */
let person={
    color:["yellow"]
};
let people1=Object.create(person);
let people2=Object.create(person);
people1.color.push("blue");
console.log(people1.color);// ["yellow", "blue"]
console.log(people2.color);// ["yellow", "blue"]
console.log(person.color);// ["yellow", "blue"]
```

5. 寄生式继承

```
/* 5.寄生式继承：增强原型式继承 */
let person={
    name:"boom"
};
function heighten(o){
    let people=Object.create(o);
    people.sayHi=function(){
        console.log("hi");
    }
    return people;
}
let ins=heighten(person);
ins.sayHi();//hi
person.sayHi();//报错
```

6. 寄生+组合继承
   本质：把组合继承里面 `Sub.prototype=new Sup();` 这步用工具函数来完成
   优点：
   - 在组合继承的优点上，还少调用了一个父类构造函数

```
function Sup(){
    this.color=["yellow"];
}
Sup.prototype.say=function(){
    return this;
}
function Sub(){
    Sup.call(this);
}
function inheritPrototype(sub,sup){
    let prot=Object.create(sup.prototype);
    prot.constructor=sub;
    sub.prototype=prot;
}
inheritPrototype(Sub,Sup);
Sub.prototype.constructor=Sub;
let ins1=new Sub();
let ins2=new Sub();
console.log(ins1===ins1.say());//true,得益于原型链
ins1.color.push("blue");
console.log(ins1.color);//["yellow", "blue"]，得益于构造函数模式
console.log(ins2.color);//["yellow"]
```

作用域/this/闭包

什么是闭包？优缺点？☆☆☆☆(字节data1面、沃德博创实习1面、酷家乐1面)

闭包：当函数可以记住并访问所在的词法作用域，即使函数是在当前词法作用域之外执行，这就产生了闭包。
优点：

- 避免变量污染全局
- 模块化
- 防止内存泄漏

this的指向？（拼多多20大题）

具体问题：给代码，问执行结果和原因。

this代码题

(快手)

```
function A() {
    this.a = 1
    return {//注意这一句哈
        a: 2
    }
}
A.prototype.a = 3
const a = new A()

console.log(a.a)//2
console.log(a.constructor)//是Object
console.log(a.__proto__)//是Object.prototype
```

根据作用域的知识，请问以下代码的运行结果?

```
function a(){
    console.log(myName);
}
function b(){
    let myName='bbb';
    a();
}
let myName='hhh';
b();
```

上述代码，打印”hhh”

内存/性能

js的垃圾回收机制？(字节data1面)

垃圾数据回收的策略分为

1. 手动回收
2. 自动回收，js采用自动回收策略，产生的垃圾是由垃圾回收器来释放的。

不同内存处理方式不同：

1. 针对栈内存：js引擎会通过移动记录当前执行状态的指针（ESP） 来销毁内存。

2. 针对堆内存
   V8 中会把堆分为新生代和老生代，新生代存放的是生存时间短的对象，老生代中存放生存时间久的对象。副垃圾回收器回收新生代，主垃圾回收器回收老生代。
   - 新生代中用 Scavenge 算法。把新生代空间划分为：对象区域和空闲区域。首先，标记对象区域中的垃圾，然后，把存活对象复制到空闲区域中，并且是边复制边整理。完成后，对象区域与空闲区域进行角色翻转，之后无限重复。js引擎采用了对象晋升策略，经过两次垃圾回收依然还存活的对象，会被移动到老生区中。
   - 由于老生区的对象比较大，若要在老生区中使用 Scavenge 算法进行垃圾回收，复制这些大的对象将会花费比较多的时间。老生区采用的策略是：先是标记阶段，从根元素开始遍历，能到达的元素称为活动对象，没有到达的元素就可以判断为垃圾数据。为了避免内存碎片，会将所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存。

垃圾回收时机：为了避免造成全停顿，V8 采用增量标记算法：将标记过程分为一个个的子标记过程，同时让垃圾回收标记和 JavaScript 应用逻辑交替进行。

垃圾回收器的总体流程：

1. 标记
2. 清除
3. 整理

ES中垃圾回收时机？（猿辅导20选择题）

具体问题：给了一段包含异步函数和普通函数代码。

如何减少白屏时间？（猿辅导20选择题、拼多多20选择题）

具体问题：以下哪些操作可以减少白屏时间?

- 使用服务器渲染?
- 使用HTTP代替HTTPS?
- 对未显示在屏幕上的资源进行缓加载?
- 尽可能使用多个不同域名来请求静态资源?

reflow（重排）、repaint（重绘）和 composite （合并）是什么?☆

- reflow：更新元素的几何属性（如：改变元素的宽度、高度等）时，将发生reflow。重排需要更新完整的渲染流水线，所以开销也是最大的。
- repaint：更新元素的绘制属性（如：更改元素的背景颜色）时，将发生repaint。重绘省去了布局和分层阶段，所以执行效率会比重排操作要高一些。
- composite：更改一个既不要布局也不要绘制的属性（如：transform）时，将发生composite。相对于重绘和重排，合成能大大提升绘制效率。

如何减少重排重绘？☆

1. 将经常改变的元素，设置定位为fix或absolute，以避免影响其周围元素
2. 避免不必要的CSS选择器
3. 对移除或者隐藏的元素进行修改

4. 通过临时存储值，减少测量次数
5. 降低DOM深度和分支中的元素数量，这样重绘重排的速度更快
6. 批量修改CSS属性

throttle（节流）与debounce（防抖）的区别？手写代码？☆

演示案例
throttle（节流）：定期执行任务；
debounce（防抖）：当任务触发的间隔超过指定间隔的时候，任务才会执行。

debounce（防抖）的代码

```
//html:
<button id="myid"></button>

//js:
var timeoutID
const debounce = (fn, delay) => {
    let timeoutID;
    return function (...args) {
        if (timeoutID) {
            clearTimeout(timeoutID);
        }
        timeoutID = setTimeout(() => {
            fn(...args);
        }, delay)
    }
}
document.getElementById('myid').addEventListener('click',
debounce(e => {
    console.log('clicked');
}, 2000));
```

执行机制/异步

介绍事件循环(Event Loop)？(字节data1面)☆

JavaScript是一个单线程的脚本语言，Event Loop是javascript的执行机制。
先了解如下几个概念：
同步任务：指的是在主线程上排队执行的任务。
异步任务：指的是进入任务队列的任务。异步任务分为宏任务(macro task)和微任务(micro task)：

- 常见宏任务： setTimeout、setInterval、 setImmediate、script（整体代码）、 I/O 操作、UI 渲染等。
- 常见微任务： process.nextTick、Promise、MutationObserver 等。

事件循环机制：

1. 同步任务在主线程上执行，形成执行栈。
2. 异步任务有了运行结果之后，就在任务队列之中放置一个事件。
3. 当所有同步任务执行完毕，则读取任务队列里面的事件。先读取任务队列里面的全部微任务，再去读取宏任务，且每读取完一个宏任务，就检查有没有微任务，有的话把微任务都执行了。
4. 不断重复第三步。

介绍Promise？（字节第一次1面）☆

Promise是ES6的新特性。它是一个用来传递异步消息的对象。它有两个特点:

1. 对象状态不受外界影响。Promise有三个状态：pending（进行中）、resolved（已完成）、rejected（已失败）。它的状态仅由异步操作的结果决定。
2. 一旦状态改变，就不会再改变了。Promise由两种状态改变：pending到resolved，pending到rejected，只要这两种情况发生，状态就不会再改变了。

promise.then()、process.nextTick()、同步代码的执行顺序？（拼多多20大题）

具体问题：给代码，问执行结果和原因。

异步代码的执行结果?

（拼多多2面）

```
async function async1() {
    console.log('async1 start');
    await async2();
    console.log('async1 end');
}

async function async2() {
    console.log('async2');
}

console.log('script start');

setTimeout(function() {
    console.log('setTimeout');
}, 0)

async1();

new Promise(function(resolve) {
    console.log('promise1');
```

```
    resolve();
}).then(function() {
    console.log('promise2');
    return new Promise(function(resolve) {
        resolve()
    })
}).then((result) => {console.log("promise 3")});

console.log("script end")
```

(快手)

```
setTimeout(() => {
  console.log(1)
}, 200);
console.log(2);
const promise = new Promise((resolve, reject) => {
  console.log(3);
  setTimeout(() => {
    console.log(4)
  }, 100);
  resolve();
})
promise.then(res => {
  console.log(5);
})
console.log(6)
//打印 2 3 6 5 4 1
```

chrome

浏览器渲染页面的过程?

1. HTML 转换为 **DOM树**
2. CSS 转换为 **CSSOM**
3. 合并DOM和CSSOM，生成**渲染树**（这棵树中不包含不可见节点）
4. 根据渲染树来**布局**，以计算每个节点的几何信息。
5. 对布局树进行分层，生成**分层树**
6. 为每个图层生成**绘制列表**
7. 合成线程将图层分成**图块（tile）**，并在光栅化（raster）线程池中将图块转换成**位图**
8. 合成线程发送绘制图块命令 DrawQuad 给浏览器进程
9. 浏览器进程生成页面，并显示到显示器上

安全/跨域

xss和csrf的攻防？☆

XSS（跨站脚本攻击）：恶意注入代码。

XSS的分类：1. 存储型：恶意代码存在数据库。2. 反射型：恶意代码存在URL。3. DOM型：属于前端安全漏洞

攻防：

1. 在输入中注入js代码
   解决方法：
   - 字符转义
   - 限制输入类型和长度
2. 在URL中注入代码
   解决方法：后端获取，前端转义后输出
3. 预防方案：
   - 后端为cookie设置httpOnly属性，让cookie无法通过js读写
   - 设置严格的CSP（Content Security Policy），CSP 的实质就是白名单制度，开发者明确告诉客户端，哪些外部资源可以加载和执行。CSP可以通过meta标签或者http的header来启用和配置。

CSRF（跨站请求伪造）：执行非用户本意的操作。（拼多多2面）

典型流程：

1. 受害者登录 a.com，并保留了登录凭证（Cookie）
2. 攻击者引诱受害者访问了b.com
3. b.com 向 a.com 发送了一个请求：a.com/act=xx，浏览器会默认携带a.com的Cookie
4. a.com接收到请求后，对请求进行验证，并确认是受害者的凭证，误以为是受害者自己发送的请求
5. a.com以受害者的名义执行了act=xx

解决和预防：

1. 使用post请求处理关键业务而非get
2. 加token
3. 加验证码
4. 同源检测，如：使用Referer Header确定来源域名

CSRF攻击具体是怎么实现的？（拼多多2面）

什么是同源策略？跨域的常见方案有哪些？☆（拼多多1面、拼多多2面）

同源：协议、域名、端口三者相同。（猿辅导2020笔试选择题）
跨域：非同源即跨域。
同源限制：缓存、DOM、ajax请求。
跨域方案：

1. jsonp：只支持get
2. CORS：后端通过access-control-allow-origin设置允许访问的域名

同源站点是否可能拥有相同的js引擎上下文？（猿辅导20选择题）

非同源站点是否可以使用window.postMessage() 进行通信？（猿辅导20选择题）

为什么浏览器的请求有两次，第一次是options，第二次才是真正请求?

发送2次请求需要满足以下2个条件：

1. 必须要在跨域的情况下
2. 预检(Preflighted)的跨域请求：除GET、HEAD和POST(only with application/x-www-form-urlencoded, multipart/form-data, text/plain Content-Type)以外的跨域请求。

原因：之所以会发送2次请求，是因为我们使用了带预检的跨域请求。该请求会先发送一个类型为OPTIONS的预检请求，以检测实际请求是否可以被服务器所接受。比如我们在请求头部增加了authorization授权项，那么在服务器响应头中需要放入Access-Control-Allow-Headers，并且其值中必须要包含authorization，否则OPTIONS预检会失败，从而导致不会发送真实的请求。

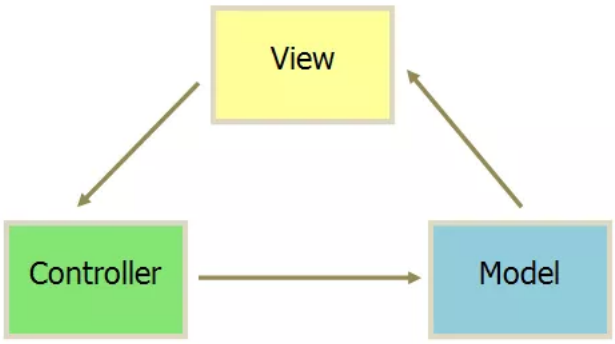
设计模式

框架/工程化

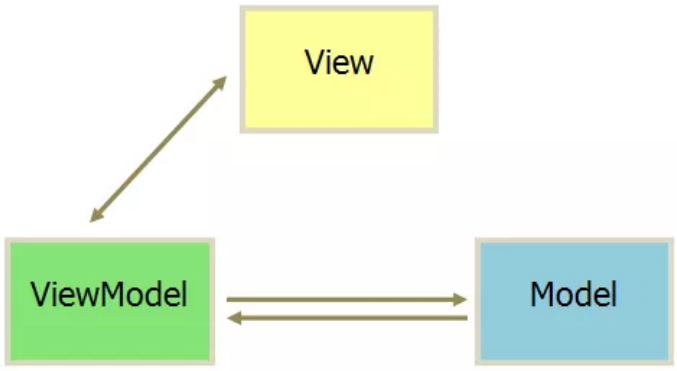
介绍MVC、MVVM?

MVC特点：
各部分之间单向通信

1. View 传送指令到 Controller；
2. Controller 完成业务逻辑后，要求 Model 改变状态；
3. Model 将新的数据发送到 View，用户得到反馈。

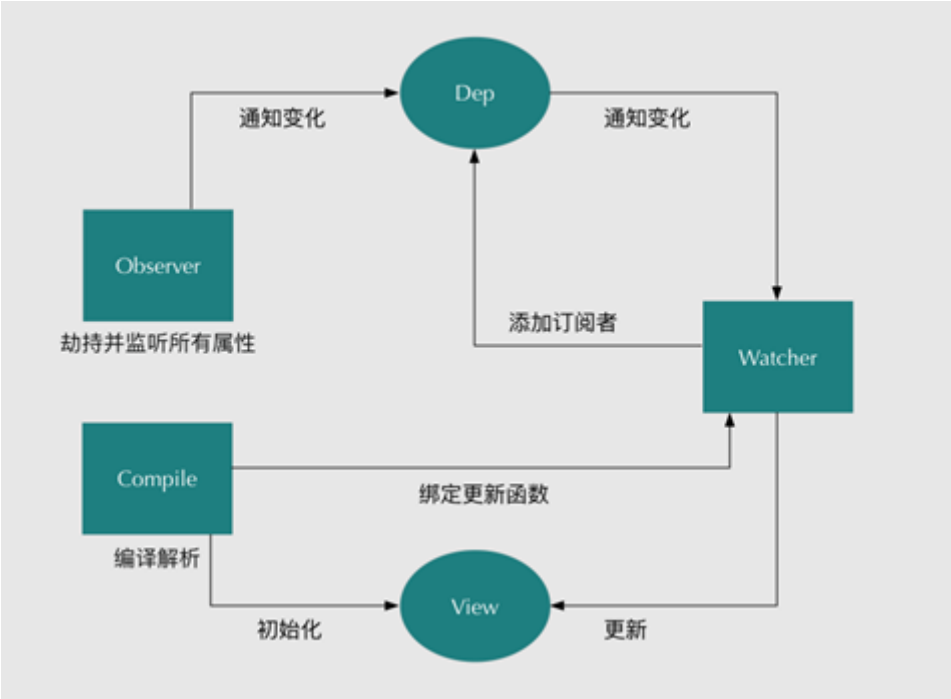


MVVM特点：各部分之间双向通信，并且采用双向绑定。



介绍双向数据绑定原理？（腾讯第二次1面、字节第一次2面）

☆☆☆



首先要对数据进行劫持监听，所以我们需要设置一个监听器Observer，用来监听所有属性。如果属性发上变化了，就需要告诉订阅者Watcher看是否需要更新。因为订阅者是有很多个，所以我们需要有一个消息订阅器Dep来专门收集这些订阅者，然后在监听器Observer和订阅者Watcher之间进行统一管理的。接着，我们还需要有一个指令解析器Compile，对每个节点元素进行扫描和解析，将相关指令对应初始化成一个订阅者Watcher，并替换模板数据或者绑定相应的函数，此时当订阅者Watcher接收到相应属性的变化，就会执行对应的更新函数，从而更新视图。

前端工程化中的依赖反转、控制反转等，是为了实现什么目的？（猿辅导20选择题）

选项：A. 提升性能 B. 松散耦合 C.方便单元测试

以下哪些可以实现webpack的tree shaking？（猿辅导20选择题）

选项：A. CommonJS B.ES Module C.UMD D.AMD

介绍webpack？（拼多多1面）

webpack 是一个现代js应用程序的静态模块打包器，将项目当作一个整体，通过一个给定的的主文件，webpack将从这个文件开始找到你的项目的所有依赖文件，使用loaders处理它们，最后打包成一个或多个浏览器可识别的js文件。

webpack、grunt、gulp的不同？

三者都是前端构建工具。
grunt 和 gulp 是基于任务和流的。找到一个（或一类）文件，对其做一系列链式操作，更新流上的数据， 整条链式操作构成了一个任务，多个任务就构成了整个web的构建流程。

webpack 是基于入口的。webpack 会自动地递归解析入口所需要加载的所有资源文件，然后用不同的 Loader 来处理不同的文件，用 Plugin 来扩展 webpack 功能

webpack 与两者最大的不同就是支持代码分割，模块化（AMD,CommonJ,ES2015），全局分析。

webpack中的bundle、chunk、module是什么?

1. bundle是由webpack打包出来的文件。
2. chunk是指webpack在进行模块的依赖分析的时候，代码分割出来的代码块。
3. module是开发中的单个模块。

webpack中的loader和plugin是什么?

- loader：模块转换器，用于将模块的原内容按照需要转成你想要的内容
- plugin 插件：在webpack构建流程中的特定时机注入扩展逻辑，来改变构建结果，是用来自定义webpack打包过程的方式，一个插件是含有apply方法的一个对象，通过这个方法可以参与到整个webpack打包的各个流程(生命周期)。

有哪些常见的Loader？以及它们的作用?

css-loader：加载 CSS，支持模块化、压缩、文件导入等特性
style-loader：把 CSS 代码注入到 JavaScript 中，通过 DOM 操作去加载 CSS
slint-loader：通过 SLint 检查 JavaScript 代码
babel-loader：把 ES6 转换成 ES5
file-loader：把文件输出到一个文件夹中，在代码中通过相对 URL 去引用输出的文件
url-loader：和 file-loader 类似，但是能在文件很小的情况下以 base64 的方式把文件内容注入到代码中去

有哪些常见的Plugin？以及它们的作用?

define-plugin：定义环境变量
terser-webpack-plugin：通过TerserPlugin压缩ES6代码
html-webpack-plugin 为html文件中引入的外部资源，可以生成创建html入口文件
mini-css-extract-plugin：分离css文件
clean-webpack-plugin：删除打包文件
happypack：实现多线程加速编译

webpack的构建流程是什么?

1. 初始化参数：从配置文件和 Shell 语句中读取与合并参数，得出最终的参数
   开始编译：用上一步得到的参数初始化 Compiler 对象，加载所有配置的插件，执行对象的 run 方法开始执行编译
2. 确定入口：根据配置中的 entry 找出所有的入口文件
3. 编译模块：从入口文件出发，调用所有配置的 Loader 对模块进行翻译，再找出该模块依赖的模块，再递归本步骤直到所有入口依赖的文件都经过了本步骤的处理
4. 完成模块编译：在经过第4步使用 Loader 翻译完所有模块后，得到了每个模块被翻译后的最终内容以及它们之间的依赖关系
5. 输出资源：根据入口和模块之间的依赖关系，组装成一个个包含多个模块的 Chunk，再把每个 Chunk 转换成一个单独的文件加入到输出列表，这步是可以修改输出内容的最后机会

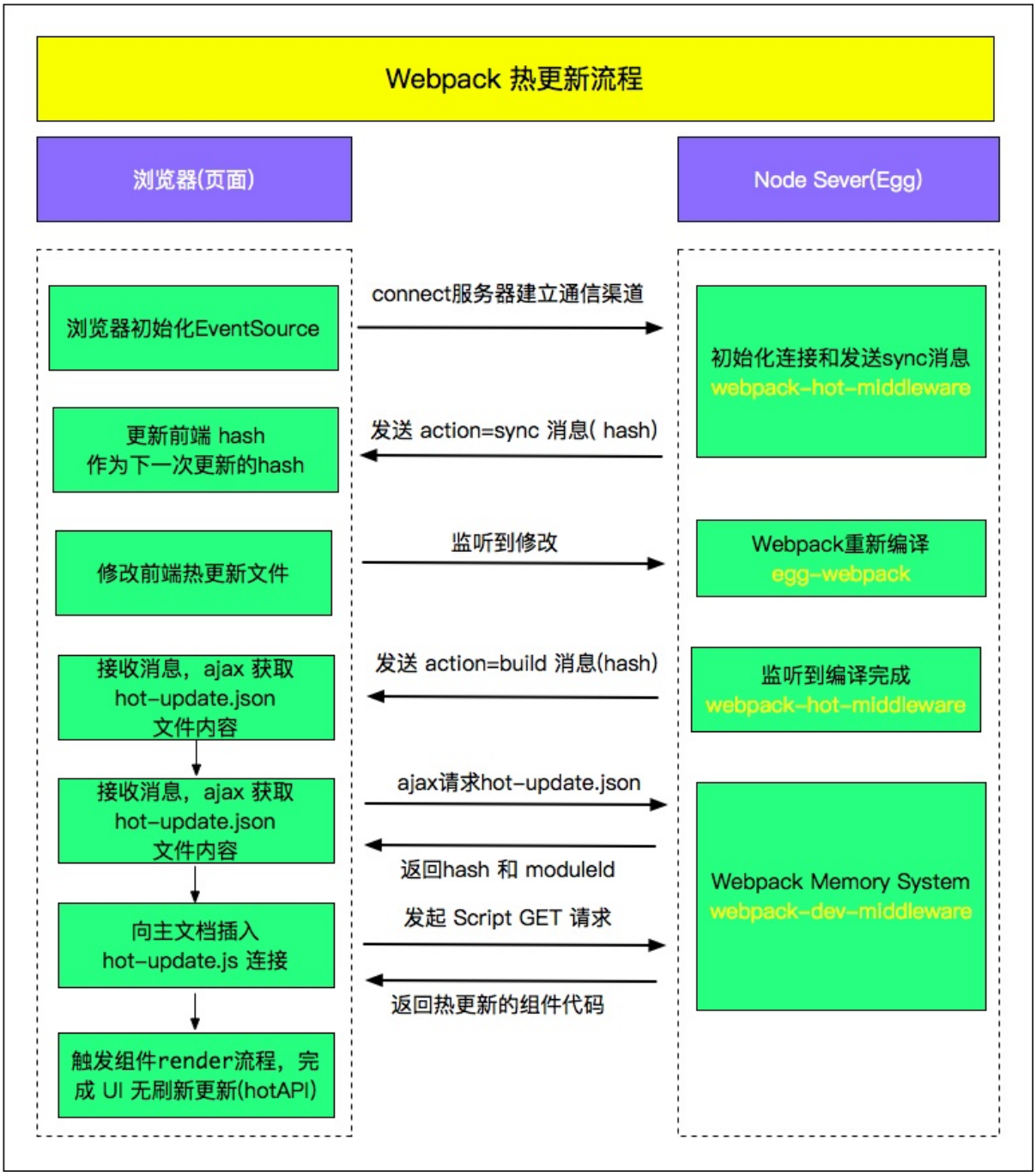
6. 输出完成：在确定好输出内容后，根据配置确定输出的路径和文件名，把文件内容写入到文件系统

什么是Tree-shaking?

通过tree shaking的分析，可以使你代码里没有使用的代码全部删除。

热更新HMR原理是什么?

热更新HMR即Hot Module Replacement是指当对代码修改并保存后，webpack将会对代码进行重新打包，并将改动的模块发送到浏览器端，浏览器用新的模块替换掉旧的模块，去实现局部更新页面而非整体刷新页面。



热更新流程：

1. Webpack编译期，为需要热更新的 entry 注入热更新代码(EventSource通信)

2. 页面首次打开后，服务端与客户端通过 EventSource 建立通信渠道，把下一次的 hash 返回前端
3. 客户端获取到hash，这个hash将作为下一次请求服务端 hot-update.js 和 hot-update.json的hash
4. 修改页面代码后，Webpack 监听到文件修改后，开始编译，编译完成后，发送 build 消息给客户端
5. 客户端获取到hash，成功后客户端构造hot-update.js script链接，然后插入主文档
6. hot-update.js 插入成功后，执行hotAPI 的 createRecord 和 reload方法，获取到 Vue 组件的 render方法，重新 render 组件， 继而实现 UI 无刷新更新。

如何提高webpack的构建速度?

1. 通过externals配置来提取常用库
2. 利用DllPlugin和DllReferencePlugin预编译资源模块 通过DllPlugin来对那些我们引用但是绝对不会修改的npm包来进行预编译，再通过DllReferencePlugin将预编译的模块加载进来。
3. 使用Happypack 实现多线程加速编译
4. 使用Tree-shaking和Scope Hoisting来剔除多余代码
5. 使用fast-sass-loader代替sass-loader
6. babel-loader开启缓存
7. 不需要打包编译的插件库换成全局”script”标签引入的方式
8. 优化构建时的搜索路径

webpack的优缺点?

优点：

1. 专注于处理模块化的项目，能做到开箱即用，一步到位
2. 可通过plugin扩展，完整好用又不失灵活
3. 使用场景不局限于web开发
4. 社区庞大活跃，经常引入紧跟时代发展的新特性，能为大多数场景找到已有的开源扩展
5. 良好的开发体验
   缺点：
6. 只能用于采用模块化开发的项目

怎么配置单页应用？怎么配置多页应用?

单页应用：可以理解为webpack的标准模式，直接在entry中指定单页应用的入口即可。
多页应用：可以使用webpack的 AutoWebPlugin来完成简单自动化的构建，但是前提是项目的目录结构必须遵守他预设的规范。 多页应用中要注意的是：

1. 每个页面都有公共的代码，可以将这些代码抽离出来，避免重复的加载。比如，每个页面都引用了同一套css样式表
2. 随着业务的不断扩展，页面可能会不断的追加，所以一定要让入口的配置足够灵活，避免每次添加新页面还需要修改构建配置

介绍react？react的生命周期？ （拼多多2面）

介绍vue？vue的生命周期？ （拼多多2面）

DOM

介绍DOM事件流?

事件：指的是用户或浏览器自身执行的某种动作。如：click、load、mouseover。
事件流：描述了事件在页面中传播的顺序。
DOM事件流包括三个阶段：

1. 捕获阶段
2. 目标阶段
3. 冒泡阶段

如何阻止事件冒泡?

w3c的方法是event.stopPropagation()，IE则是使用event.cancelBubble = true

如何取消默认事件?

w3c的方法是event.preventDefault()，IE则是使用event.returnValue = false

怎样使用DOM API获取带有xx类的a标签？（拼多多20选择题）

选项：包含querySelector和querySelectorAll、filter()等方法的具体代码

其他技术

介绍Web worker？（腾讯1面）

Web worker是HTML5的新特性。JavaScript是运行在单线程环境中的，也就是说他没有办法同时运行多个脚本。而web worker 是运行在后台的 JavaScript，独立于其他脚本，不会影响页面的性能。您可以继续做任何愿意做的事情：点击、选取内容等等，而此时 web worker 在后台运行。

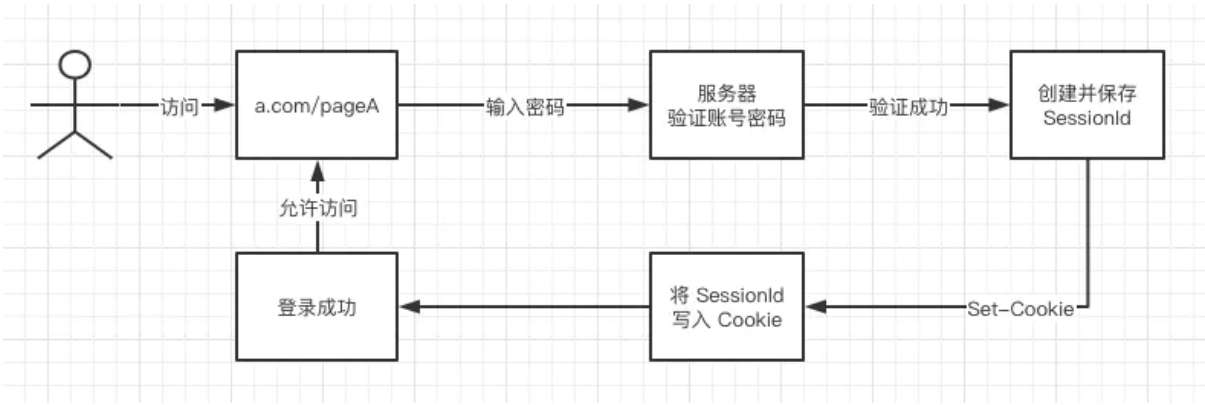
介绍WebAssembly？（腾讯1面）

是一种类汇编语言，它可以使非JavaScript编写的代码在浏览器上运行。相比JavaScript，WebAssembly文件体积更小，在编译、优化、执行方面性能高效，同时进行手动GC。

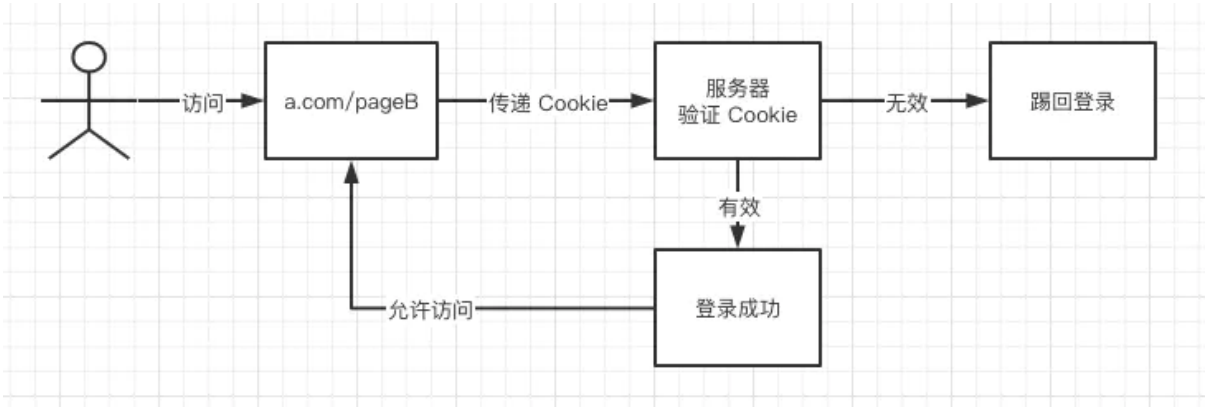
网络

前端登录的方式有哪些？各自的优缺点以及登录流程是怎样的？（字节第一次2面）

常用的前端登陆的方式：
1.Cookie + Session 登录
用户首次登陆：



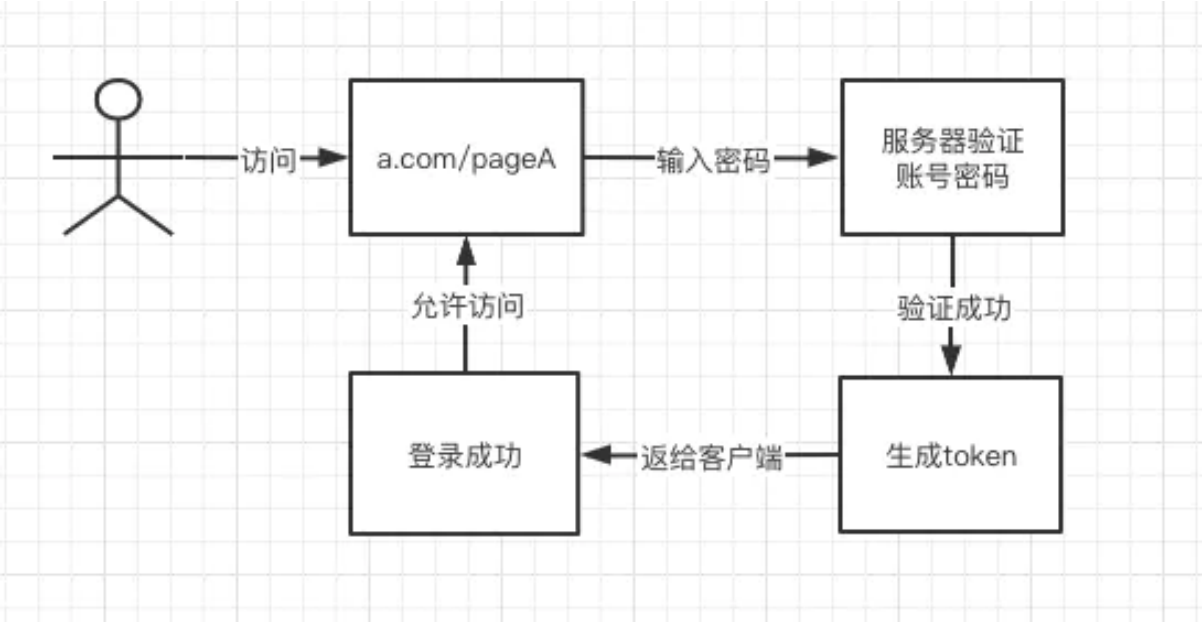
后续访问：



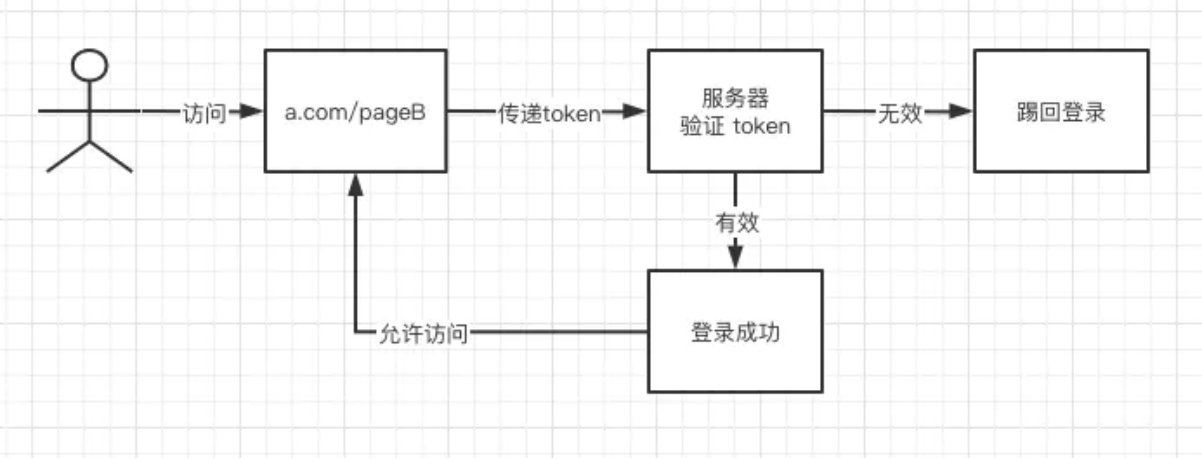
缺点：

- 需要存放SessionId，导致服务器压力过大。
- 如果服务器端是一个集群，为了同步登录态，需要将 SessionId 同步到每一台机器上，导致服务器端维护成本增大。
- 由于 SessionId 存放在 Cookie 中，所以无法避免 CSRF 攻击。

2.Token 登录
用户首次登陆：



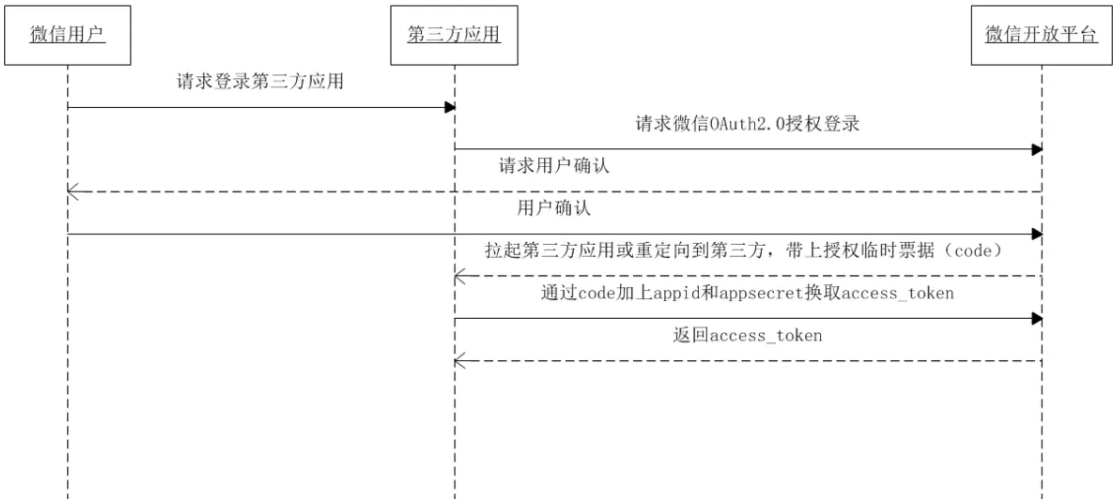
后续访问：



优缺点：

- 服务器端不需要存放 Token。
- Token 可以存放在前端任何地方，可以不用保存在 Cookie 中，提升了安全性。
- Token 下发之后，只要在生效时间之内，就一直有效，如果服务器端想收回此 Token 的权限，并不容易。

3.OAuth 第三方登录



前端登录，这一篇就够了

Cookie和session的区别？cookie和session之间如何联系？如果禁用cookie可以访问session吗？（腾讯第二次1面）

区别：

1. cookie数据保存在客户端，session数据保存在服务端。
2. cookie可以减轻服务器压力，但是不安全，容易进行cookie欺骗。
3. session更安全，但是占用服务器资源。

联系与禁用的问题：因为Session是用Session ID来确定当前对话所对应的服务器Session，而Session ID是通过Cookie来传递的，禁用Cookie相当于失去了Session ID，也就得不到Session了。
解决方法：

1. 重写url，把 sessionid 作为参数追加的原 url 中，后续的浏览器与服务器交互中携带 sessionid 参数。
2. 服务器的返回数据中包含 sessionid，浏览器发送请求时，携带 sessionid 参数。
3. 通过 Http 协议其他 header 字段，服务器每次返回时设置该 header 字段信息，浏览器中 js 读取该 header 字段，请求服务器时，js设置携带该 header 字段。

如何判断用户的使用的机型？（腾讯1面，腾讯第二次1面）

通过navigator.userAgent字段信息。

介绍从输入url到页面展示的过程？☆☆☆（拼多多1面）

1. 用户输入
2. 检查缓存
3. DNS 解析
4. 建立 TCP 连接（三次握手）
5. HTTP 请求与响应
6. 关闭 TCP 连接（四次挥手）
7. 准备渲染进程
8. 提交文档

9. 渲染阶段

为什么需要三次握手，两次握手不可以吗？（拼多多1面）

这主要是为了防止已失效的连接请求报文段突然又传送到了服务器，因而产生错误。

TCP和UDP的区别？☆☆☆

TCP特点：

1. TCP是面向连接的运输层协议。
2. 每一条TCP连接只能有两个端点，每一条TCP连接只能是点对点的。
3. TCP提供可靠交付的服务，即传输的数据无差错、不丢失、不重复、按序到达。
4. TCP提供全双工通信。
5. 面向字节流。

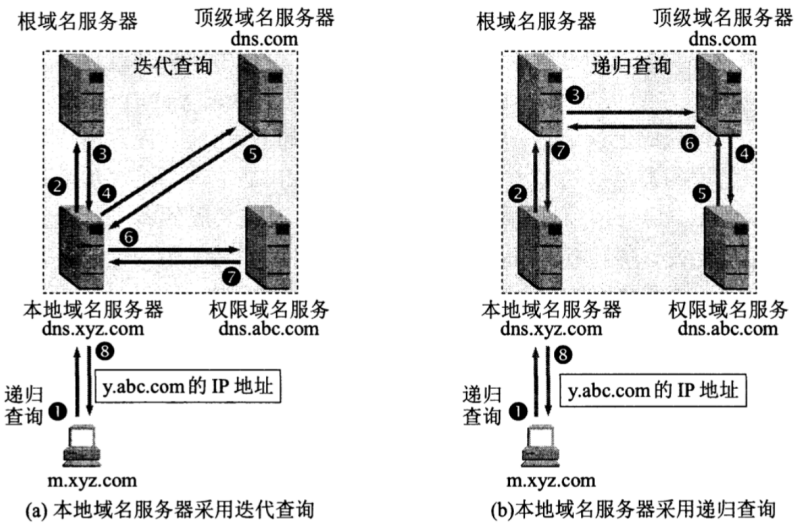
UDP的特点：

1. 无连接。
2. 尽最大努力交付。
3. 面向报文。
4. 无拥塞控制。
5. 支持一对一、一对多、多对一和多对多的交互通信。
6. 首部开销小，因为字段少。

介绍DNS实现原理？（字节第一次2面）

域名系统DNS是互联网使用的命名系统，用来把域名转换为IP地址。

DNS解析原理：
DNS解析过程中，主机向本地域名服务器的查询一般都是采用递归查询，本地域名服务器向根域名服务器的查询通常是采用迭代查询。



DNS解析步骤为：

1. 主机 m.xyz.com 先向其本地域名服务器 dns.xyz.com 进行递归查询。

2. 本地域名服务器采用迭代查询。它先向一个根域名服务器查询。
3. 根域名服务器告诉本地域名服务器，下一次应查询的顶级域名服务器 dns.com 的 IP 地址。
4. 本地域名服务器向顶级域名服务器 dns.com 进行查询。
5. 顶级域名服务器 dns.com 告诉本地域名服务器，下一次应查询的权限域名服务器 dns.abc.com 的 IP 地址。
6. 本地域名服务器向权限域名服务器 dns.abc.com 进行查询。
7. 权限域名服务器 dns.abc.com 告诉本地域名服务器，所查询的主机的 IP 地址。
8. 本地域名服务器最后把查询结果告诉主机 m.xyz.como。

DNS优化的优化方案有哪些?

1. DNS缓存： 浏览器缓存，系统缓存，路由器缓存，IPS服务器缓存，根域名服务器缓存，顶级域名服务器缓存，主域名服务器缓存。通过缓存直接读取域名相对应的IP，减去了繁琐的查找IP的步骤，大大加快访问速度。
2. DNS负载均衡：使用CDN进行负载均衡，利用DNS的重定向技术，同步服务器运行情况，然后根据该情况及时适当调整调度策略，从而使得负载均衡能力大大提高。

DNS服务器有哪些类型?

1.主服务器
2.附加的辅助服务器
3.附加的Caching-only服务器

HTTP

介绍HTTPS的加密？(腾讯第一次1面，字节data1面)☆☆

HTTP的缺点以及HTTP需要解决的问题：

1. 窃听：通信使用明文，内容可能会被窃听；
2. 篡改：无法证明报文的完整性，有可能遭到篡改；
3. 伪装：不验证通信方的身份，有可能遭遇伪装。

HTTPS=HTTP+SSL/TLS

了解一下两个概念：

1. 非对称加密：浏览器使用服务器的公钥，加密信息，然后发给服务器，服务器收到后使用私钥解密。但无法确保浏览器收到的公钥是正确的。
2. 数字证书：服务器先向CA发送公钥和相关信息，CA使用单向hash算法生成摘要，再用CA的私钥加密摘要，生成数字签名，然后CA将申请信息和数字签名整合生成数字证书，返回给服务器。服务器把公钥和数字证书发给客户端，客户端从电脑内置的根证书里的CA公钥解密数字证书，生成摘要，并且对服务器公钥使用hash算法同样生成摘要，比较二者。如果一致，说明公钥未被篡改。

HTTPS加密过程：

1. SSL握手（非对称加密）
2. 利用会话密钥传输数据（对称加密）

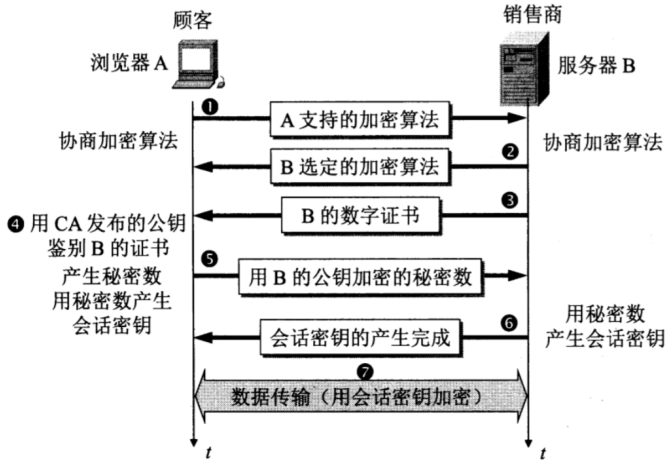


图 7-18　SSL 建立安全会话的简要过程

HTTP常见状态码？☆☆

常见状态码

HTTP2.0新特性有哪些？（腾讯第二次1面）☆

1. **二进制分帧**：HTTP/2 则是一个彻底的二进制协议，头信息和数据体都是二进制，并且统称"帧"（frame）——头信息帧和数据帧。
2. **首部压缩**：一方面，头信息使用gzip或compress压缩后再发送；另一方面，客户端和服务器同时维护一张头信息表，所有字段都会存入这个表，生成一个索引号，以后就不发送同样字段了，只发送索引号，这样就提高速度了。
3. **流量控制**
4. **多路复用**：复用TCP连接，在一个连接里，客户端和浏览器都可以同时发送多个请求或回应
5. **请求优先级**
6. **服务器推送**：允许服务器未经请求，主动向客户端发送资源。例如：服务器可以预期到客户端请求网页后，很可能会再请求静态资源，所以就主动把这些静态资源随着网页一起发给客户端了。

POST和GET的区别是什么？（腾讯第二次1面）☆☆

1. GET在浏览器后退/刷新时是无害的，而POST会再次提交请求。
2. GET产生的URL地址可以被收藏为书签，而POST不可以。
3. GET请求会被浏览器主动cache，而POST不会，除非手动设置。
4. GET请求只能进行url编码，而POST支持多种编码方式。
5. GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。
6. GET请求在URL中传送的参数是有长度限制的，而POST没有。
7. 对参数的数据类型，GET只接受ASCII字符，而POST没有限制。
8. GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。
9. GET参数通过URL传递，POST放在Request body中。

HTTP队头阻塞及解决方案？（拼多多20大题）

操作系统

进程调度算法有哪些？(腾讯团队笔试、字节data1面)

1. 先来先服务FCFS
2. 短任务优先
3. 轮转
4. 高响应比优先：响应比=（等待时间+服务时间）/服务时间
5. 优先级调度算法
6. 多级反馈队列调度算法：队内按FCFS，队间按优先级

进程和线程的区别？(字节data1面)☆☆☆

进程和线程都是CPU工作时间段的描述，不过是颗粒大小不同。
进程是系统进行资源分配和调度的一个独立单位，线程是CPU调度和分派的基本单位。

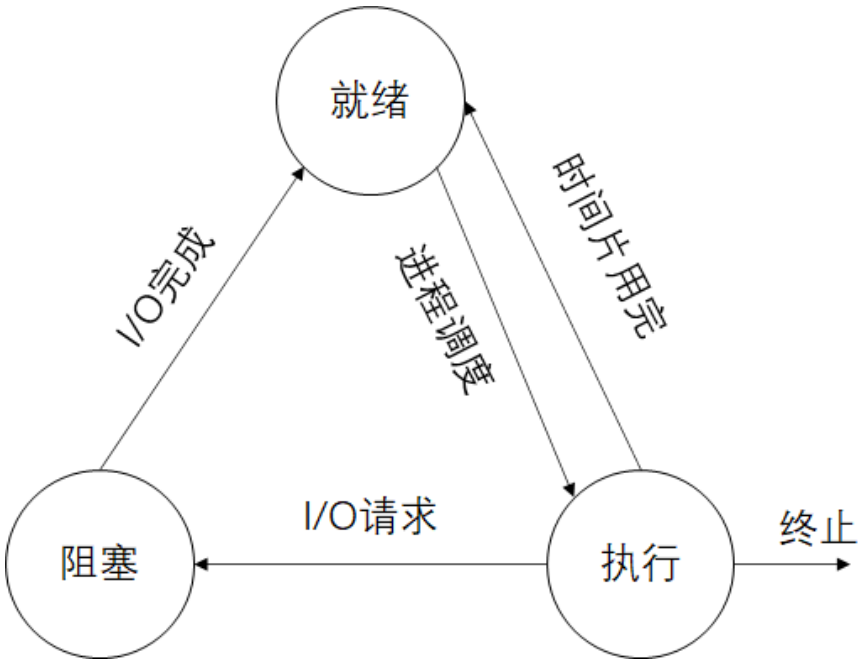
1. 任一线程出错，都会导致所在的整个进程崩溃。
2. 同一进程内的线程可以共享进程数据。
3. 不同进程间数据很难共享。

介绍chrome浏览器的进程？(字节data1面)

当前chrome浏览器中的进程：

1. 浏览器进程——负责界面显示、用户交互、子进程管理，同时提供存储等功能。
2. 渲染进程——负责将 HTML、CSS 和 JavaScript 转换为用户可以与之交互的网页，排版引擎 Blink 和 JavaScript 引擎 V8 都是运行在该进程中，默认情况下，Chrome 会为每个 Tab 标签创建一个渲染进程。
3. GPU 进程
4. 网络进程——负责页面的网络资源加载。
5. 插件进程——负责插件的运行。

进程的三种基本状态是什么？及如何转换？



进程通信的类型，主要有哪些?

1. 高级通信机制
    - 共享存储器系统
    - 管道通信系统
    - 消息传递系统
    - 客户机-服务器系统
2. 低级通信机制
    - 信号量

进程同步机制，主要有哪些?

1. 硬件同步机制
2. 信号量机制
3. 管程机制

同步机制应遵循的四个准则是什么?

1. 空闲让进
2. 忙则等待
3. 有限等待
4. 让权等待

产生死锁的必要条件有哪些？如何破坏这些条件?

产生死锁必须具备四者，缺一不可。

1. 互斥条件
2. 请求和保持——破坏方式：1.所有进程在开始运行前，一次性申请所有需要的资源。简单、安全，但是浪费资源，造成有的进程等待过久。2.只获得初期所需资源，之后逐步释放用完的资源，再去申请其他资源。

3. 不可抢占条件——破坏方式：获得不可抢占资源的进程，如果在申请新资源时无法获得，则先释放已经保持的所有资源，之后再去申请资源。
4. 循环等待条件——破坏方式：给资源编序号，进程按序号递增的顺序请求资源。

处理死锁的四种方法?

1. 预防死锁：通过设置限制，去破坏产生死锁的必要条件。
2. 避免死锁：通过在资源动态分配过程中，防止系统进入不安全状态。*引出银行家算法……*
3. 检测死锁：利用资源分配图，当且仅当它是不可完全化简时，说明发生死锁。
4. 解除死锁：1.抢占资源，分配给死锁进程。2.终止进程

如何用安全性算法和银行家算法避免死锁?

安全性算法

1. 找到一个need<available的进程P
2. 把进程P的allocation加上available，赋值给available
3. 对除了P之外的进程，重复1和2操作。
4. 最终得到的进程序列，就是一个安全序列(不唯一)。

举例：下列系统安全，因为存在一个安全序列{P1,P3,P4,P2,P0}

进程 资源情况	Max	Allocation	Need	Available
	A B C	A B C	A B C	A B C
P0	7 5 3	0 1 0	7 4 3	3 3 2 (2 3 0)
P1	3 2 2	2 0 0 (3 0 2)	1 2 2 (0 2 0)	
P2	9 0 2	3 0 2	6 0 0	
P3	2 2 2	2 1 1	0 1 1	
P4	4 3 3	0 0 2	4 3 1	

银行家算法
作用：当一个进程需要请求资源时，调用银行家算法，避免不合法的请求。
过程：

1. 如果请求资源数量request<need，则下一步，否则报错
2. 如果请求资源数量request<available，则下一步，否则报错
3. 修改进程的allocation和need，以及available
4. 执行安全性算法。合法则分配，不合法则分配作废，恢复原来的资源分配状态。

举例：上一张图里，P1请求资源(1,0,2)是合法的，分配完P1的(1,0,2)，再申请P4的(3,3,0)或者P0的(0,2,0)都是不合法的。

在检测死锁过程中，如何判断资源分配图RAG是否可化简?

步骤：

1. 找到只有分配边而没有请求边的进程节点，删除分配边。
2. 找到虽有请求边，但请求边可以立即全部转化为分配边的进程节点，将其请求边删除。
3. 经过1和2后，如果进程节点和资源节点全部转换为孤立节点，说明此RAG可化简，否则为不可完全化简得。

页面置换算法有哪些?

页面置换算法：当调入进程所请求的页面时，如果内存中已经没有空闲块了，则必须按照某种算法将内存中的若干页面淘汰至外存。

1. 最佳置换算法：淘汰一个永不用或最久不用的页面，因为无法预知，故该算法无法实现。
2. 先进先出置换算法FIFO
3. 最近最久未使用置换算法LRU
4. 最少使用置换算法LFU
5. Clock置换算法：页面被访问的时候访问位设为1。之后要淘汰页面的时候，就按照循环队列的顺序，如果访问位是1，就置0，继续看下一个页面；如果访问位是0，则淘汰这一页。

数据结构

线性表

手写链表插入元素?

```
//节点的结构
class Node {
    constructor(data) {
        this.data = data;
        this.next = null;
    }
}
//链表的结构
class LinkList {
    constructor() {
        this.head = new Node('head')
    }
    //插入节点，为了之后测试数据
    append(newElement) {
        const newNode = new Node(newElement)
        let currentNode = this.head
        while (currentNode.next) {
            currentNode = currentNode.next
        }
        currentNode.next = newNode
```

```
    }

    //查找节点，为了之后给该节点之后插入节点
    findByValue(item) {
        let currentNode = this.head.next;
        while (currentNode !== null && currentNode.data !== item) {
            currentNode = currentNode.next;
        }
        return currentNode === null ? -1 : currentNode;
    }
    //插入节点
    insert(newdata, data) {
        const currentNode = this.findByValue(data);
        if (currentNode === -1) {
            return false;
        }
        const newNode = new Node(newdata);
        newNode.next = currentNode.next;
        currentNode.next = newNode;
        return true;
    }
}
let link = new LinkList();
link.append(11);
link.append(12);
link.append(13);
console.log(link)
link.insert(666,12);
console.log(link);
```

手写链表逆置？（字节data一面）☆☆☆

头插法

```
reverseList() {
    const root = new Node('head');
    let currentNode = this.head.next;
    while (currentNode !== null) {
        const next = currentNode.next;
        currentNode.next = root.next;
        root.next = currentNode;
        currentNode = next;
    }
    this.head = root;
}
```

手写链表验环? ☆☆

验环

```
checkCircle() {
    let fast = this.head.next;
    let slow = this.head;
    while (fast !== null && fast.next !== null) {
        fast = fast.next.next;
        slow = slow.next;
        if (slow === fast) return true;
    }
    return false;
}
```

手写删除倒数第k个节点?

```
removeByIndexFromEnd(index) {
    if(this.checkCircle()) return false;
    let pos = 1;
    this.reverseList()
    let currentNode = this.head.next
    while (currentNode !== null && pos < index) {
        currentNode = currentNode.next;
        pos++;
    }
    if (currentNode === null) {
        return false;
    }
    this.remove(currentNode.element);
    this.reverseList();
}
```

树

数组转树?

```
const arr = [
    {id:1, parentId: null, name: 'a'},
    {id:2, parentId: null, name: 'b'},
    {id:3, parentId: 1, name: 'c'},
    {id:4, parentId: 2, name: 'd'},
```

```
    {id:5, parentId: 1, name: 'e'},
    {id:6, parentId: 3, name: 'f'},
    {id:7, parentId: 4, name: 'g'},
    {id:8, parentId: 7, name: 'h'},
]
function array2Tree(arr) {
    if (!Array.isArray(arr) || arr.length === 0) return;
    let map = {};
    arr.forEach(item => map[item.id] = item);
    let roots = [];
    arr.forEach(item => {
        const parent = map[item.parentId];
        if (parent) {
            (parent.children || (parent.children = [])).push(item);
        } else {
            roots.push(item);
        }
    })
    return roots;
}
let a = array2Tree(arr, null);
console.log(a);
```

基础算法

常见的排序算法？时/空复杂度？稳定性？（腾讯第一次1面、字节data1面、酷家乐笔试）☆☆☆

排序算法	平均时间复杂度	最好情况	最坏情况	空间复杂度	排序方式	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	In-place	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
希尔排序	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(1)$	In-place	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Out-place	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	In-place	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	In-place	不稳定
计数排序	$O(n + k)$	$O(n + k)$	$O(n + k)$	$O(k)$	Out-place	稳定
桶排序	$O(n + k)$	$O(n + k)$	$O(n^2)$	$O(n + k)$	Out-place	稳定
基数排序	$O(n \times k)$	$O(n \times k)$	$O(n \times k)$	$O(n + k)$	Out-place	稳定

手写常见排序算法？（腾讯广州团队笔试）

冒泡

```
function bubbleSort(arr) {
    let len = arr.length;
    for (let i = 0; i < len; i++) {
        let hasChange = false;
        for (let j = i + 1; j+1 < len; j++) {
            if (arr[j+1] < arr[j]) {
                [arr[j+1], arr[j]] = [arr[j], arr[j+1]];
                hasChange = true;
            }
        }
        if (!hasChange) break;
    }
    return arr;
}
```

选择

```
function bubbleSort(arr) {
    if (arr.length <= 1) return;
    let len = arr.length;
    for (let i = 0; i < len; i++) {
        let hasChange = false;
        for (let j = 0; j + i + 1 < len; j++) {
            if (arr[j + 1] < arr[j]) {
                [arr[j + 1], arr[j]] = [arr[j], arr[j + 1]];
                hasChange = true;
            }
        }
        if (!hasChange) break;
    }
    return arr;
}
```

插入

```
function insertionSort(arr) {
    let len = arr.length;
    let current, preIndex;
    for (let i = 1; i < len; i++) {
        current = arr[i];
        preIndex = i - 1;
```

```
        while (current < arr[preIndex] && preIndex >= 0) {
            arr[preIndex + 1] = arr[preIndex];
            preIndex--;
        }
        arr[preIndex + 1] = current;
    }
    return arr;
}
```

合并

```
function mergeSort(arr) {
    let len = arr.length;
    if (len < 2) return arr;
    let mid = Math.floor(len / 2),
        left = arr.slice(0, mid),
        right = arr.slice(mid);
    return merge(mergeSort(left), mergeSort(right));
}

function merge(left, right) {
    let result = [];
    while (left.length > 0 && right.length > 0) {
        if (left[0] <= right[0])
            result.push(left.shift());
        else
            result.push(right.shift());
    }
    while (left.length > 0)
        result.push(left.shift());
    while (right.length > 0)
        result.push(right.shift());
    return result;
}
```

快排（腾讯第一次1面）
版本一：out-place

- 空间复杂度为O(n)
- 稳定

```
function quickSort(arr) {
    const len = arr.length;
    if (len < 2) return arr;
    const partition = arr[0];
    const left = [];
```

```
    const right = [];
    for (let i = 1; i < len; i++) {
        if (arr[i] < partition)
            left.push(arr[i]);
        else
            right.push(arr[i]);
    }
    return quickSort(left).concat(partition, quickSort(right));
}
```

引出问题：如何提高空间复杂度....

版本二：in-place：

- 空间复杂度为O(logn)，因为需要产生logn层嵌套递归调用
- 非稳定

```
const partition = (arr, pivot, left, right) => {
    const pivotVal = arr[pivot];
    let startIndex = left;
    for (let i = left; i < right; i++) {
        if (arr[i] < pivotVal) {
            [arr[i],arr[startIndex]]=[arr[startIndex],arr[i]];
            startIndex++;
        }
    }
    [arr[pivot],arr[startIndex]]=[arr[startIndex],arr[pivot]];
    return startIndex;
}

const quickSort = (arr, left, right) => {
    if (left < right) {
        let pivot = right;
        let partitionIndex = partition(arr, pivot, left, right);
        quickSort(arr, left, partitionIndex - 1 < left ? left : par
titionIndex - 1);
        quickSort(arr, partitionIndex + 1 > right ? right : partiti
onIndex + 1, right);
    }
}
```

什么是数组扁平化？要求多种实现方式?

数组的扁平化，就是将一个嵌套多层的数组 array (嵌套可以是任何层数)转换为只有一层的数组。

方法一：递归

```
let arr = [1, [2, [3, 4]]];
const flatten = (arr) => {
    let result = [];
    for (let i = 0, len = arr.length; i < len; i++) {
        if (Array.isArray(arr[i])) {
            result = result.concat(flatten(arr[i]))
        } else {
            result.push(arr[i]);
        }
    }
    return result;
}
console.log(flatten(arr));
```

方法二：toString()
局限：元素必须为数值。比如，包含了”1”，则会被转换为数字1

```
[1, [2, [3, 4]]].toString() // "1,2,3,4"
```

方法三：reduce()

```
let arr = [1, [2, [3, 4]]];
const flatten = (arr) => {
    return arr.reduce((prev, next) => {
        return prev.concat(Array.isArray(next) ? flatten(next) : ne
xt)
    }, []);
}
console.log(flatten(arr));
```

方法四：ES6的扩展运算符

```
let arr = [1, [2, [3, 4]]];
const flatten = arr => {
    while (arr.some(item => Array.isArray(item))) {
        arr = [].concat(...arr);
    }
    return arr;
}
console.log(flatten(arr));
```

跳台阶? ☆

问题描述：一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个 n 级的台阶总共有多少种跳法。

```
function ways(n){
    if (n <= 1) return 1;
    let dp = [1, 1, 2];
    for (let i = 2; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }
    return dp[n];
};
```

数组去重? ☆

为 Array 对象添加一个去重的方法?
为 Array 对象添加一个取出非重复项的方法?
为Array对象添加一个取出重复项的方法?

```
//去重
Array.prototype.removeDuplication = function(){
    return [...new Set(this)];
};
//取非重
Array.prototype.unique = function(){
    return this.filter(i => this.indexOf(i) === this.lastIndexOf
(i));
};
//取重
Array.prototype.duplication = function(){
    return this.filter(i => this.indexOf(i) !== this.lastIndexOf
(i));
};

let arr = [11, 23, 26, 23, 11, 9];

console.log(arr);
console.log(arr.removeDuplication());
console.log(arr.unique());
console.log(arr.duplication());
```

大数相加?

```
function add(str1, str2) {
    str1 = str1.split("");
```

```
    str2 = str2.split("");
    let result = ""; //结果
    let flag = 0;
    while (str1.length || str2.length || flag) {
        flag += ~~str1.pop() + ~~str2.pop();
        result = flag % 10 + result;
        flag = flag > 9;
    }
    return result.replace(/^0+/, '');
}
let res = add('1111111111111111117', '1111111111111111117');
console.log(res)
```

url转对象?

```
function urlToObj(str) {
    var obj = {};
    var arr1 = str.split("?");
    var arr2 = arr1[1].split("&");
    for (var i = 0; i < arr2.length; i++) {
        var res = arr2[i].split("=");
        obj[res[0]] = res[1];
    }
    return obj;
}
console.log(urlToObj('https://www.baidu.com/s?ie=utf-8&f=3&rsv_bp=1
&tn=baidu&wd=%E7%99%BE%E5%BA%A6'))
```

验证嵌套{}()[]的合法性? （拼多多2面）