页面布局
css盒模型
DOM事件
HTTP协议
面向对象
原型链
通信
安全 xss csrf
算法

页面布局
1. 浮动
2.定位
3. flex
4. 表哥布局
5. 网格布局

flex布局
容器
flex-direction 设置的主轴方向
row 水平 左 --> 右
row-reverse 右 --> 左
column 上-->下
column-reverse 下 --> 上

flex-wrap 换行规则
nowrap 不换行 默认
wrap 换行
wrap-reverse 后面再上

flex-flow
flex-direction flex-wrap的缩写

justify-content 元素在主轴上的排列方式
flex-start 默认 开始位置排列
flex-end 默认 末尾开始排列
center 居中
space-between 两边分布
space-around 平均分配资源

align-items 元素在交叉轴上如何排列
flex-start 开始位置
flex-end 结束位置开始排列
center 居中
stretch 高度和父级一样 如果元素没有设置高度或者设置auto，默认沾满容器

baseline 项目的第一行文字的基线对齐

align-center 多个基线的对齐方式
flex-start
flex-end
center
stretch
space-between
space-around

元素上的属性
order
flex-grow
flex-shrink
flex-basis
flex
align-self 单个元素和其他的排列不一样
auto 默认
flex-start
flex-end
center
stretch
baseline

display: table-cell;
让元素按照单元格的形式展现，类似td标签
对margin无响应
统一行元素都是等高的

```
table    { display: table } 块级表格 inline-table 内联表格
tr       { display: table-row } 表格的行
td, th   { display: table-cell } 表格单元格
thead    { display: table-header-group } 头
tbody    { display: table-row-group } 主题
tfoot    { display: table-footer-group } 底部
col      { display: table-column } 表格的列
colgroup { display: table-column-group }
caption  { display: table-caption }
```

垂直居中布局
1. 三种定位形式
2. table布局

```
html,body {
    height: 100%;
}
body {
    display: table;
    width: 100%;
}
.box {
```

```
    display: table-cell;
    vertical-align: center;
    text-align:center;
}
<div class="box">
    <p>Double Line</p>
    <p>Double Line</p>
</div>
```

3. flex

4.grid

5. inline的方式 这个需要查

双飞翼布局

1. float
2. position
3. flex
4. table布局

```
<style>
html,body {
    height: 100%;
}
.boxes {
    display: table;
    width: 100%;
    height: 100%;
}
.box {
    height: 100%;
    display: table-cell;
}
.box1 {
    background: red;
    width: 100px;
}
.box2 {
    background: blue;
}
.box3 {
    background: orange;
    width: 100px;
}
</style>
<div class="boxes">
<div class="box box1"></div>
<div class="box box2"></div>
<div class="box box3"></div>
</div>
```

5. grid布局

flex布局是轴线布局，可以看作是一维布局

grid布局则是将容器划分为行和列，产生单元格，然后指定项目所在的单元格，被称为二维布局

```
.container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}
```
grid-template-columns 按顺序定义列宽
grid-template-rows 定义行高

1. 五个方案的优缺点
2. 高度未知，两边如何自适应中间高度
3. 上面方案兼容性，实际业务中最优方案

1. 五个方案的优缺点
a. 浮动要清除浮动，兼容性好
b. 定位可以是实现，但是定位导致元素脱离文档流
c. flex布局，在移动端完美，ie8不支持
d. 表格布局，兼容性很好，i8支持，中间超出，两边会自适应
e. 网格布局，通过标准化，代码简化，但是现在浏览器支持比较差 ie11

2. 去掉高度已知，那个还能适用
flex 表格布局 可用

3. 浮动的出现本质上是解决实现文本环绕的问题，所以当文字高度超出浮动元素的高度时候，会出现文本偏移现象
如何解决
创建BFC

布局要求
1. 语意化清楚
2. 页面布局理解深刻，掌握方案原理
3. css基础扎实
4. 表现思维灵活积极上进
5. 代码书写规范

衍生
三栏布局
    左右固定，中间自适应
    上下高度固定，中间自适应
两栏和三栏没有区别

==========>
grid布局就是将网页划分成一个个网格，然后通过网格之间的组合，做出各种布局
flex布局是通过指定项目在轴线上的排列顺序，达到布局需要
grid布局是通过将页面划分成列和行，然后在执行单元格中绘制，达到布局要求

采用网格布局的区域，成为容器。每个网格的元素称为项目

水平是行(row)
垂直是列(column)

单元格 行和列的交叉区域成为单元格
网格线 水平网格划分出行(row) 垂直网格划分出列(column)

属性分类
定义在容器上，称为容器属性
定义在项目上，称为项目属性

容器属性
display: grid; 指定一个容器采用网格布局
默认情况下容器都是块级元素，也可以是display: inline-grid;

网格布局中，子级元素设置float, table-cell, inline-block,vertical-align都将失败
设置每个列的宽度 grid-template-columns: 100px 100px 200px;
设置每个行的宽度 grid-template-row: 100px 100px 200px;
也可以设置相对百分比

关键字 repeat() auto-fill fr minmax() auto
重复写很麻烦，可以使用repeat()
grid-template-columns: repeat(2, 100px 300px);

单元格大小固定，但是容器大小不是固定的，这时候需要尽可能多的容纳元素，这时候auto-fill
grid-template-columns: repeat(auto-fill, 100px)

fr 比例， 剩余空间的比例
grid-template-columns: 100px 1fr 3fr;

minmax(min, max) 最小值和最大值的一个范围，元素的长度在这个范围之类
grid-template-columns: 1fr 1fr minmax(100px 1fr);

grid-template-columns: 100px auto 100px; 宽度自适应

布局例子
两格布局
```
.box {
    display: grid;
    grid-template-columns: 30% 70%;
}
```

传统的12列布局
```
.box {
    display: grid;
    grid-template-columns: repeat(12, 1fr);
}
```

grid-column-gap 设置列之间的间距
grid-row-gap: 设置行之间的间距
grid-gap: <grid-row-gap> <grid-column-gap>

```
.container {
    grid-gap: 20px 20px;
}
```

grid-auto-flow 设置排列规则，默认row, 一行排满之后，折行
grid-auto-flow: column; 现排列，从第一列从上到下，满了之后再来一次
=====================>

css盒模型
谈谈你对css盒模型的理解
1. 基本概念 标准模型 + IE模型
2. 基本模型和IE模型之间的区别
3. css如何设置这两种模型
4. js如何设置获取盒模型对应的宽和高
5. 实例题：解释边距重叠
6. BFC(边距重叠解决方案) / IFC

标准模型大小 box-sizing: content-box; 默认
margin + border + padding + content width/height

IE模型 box-sizing: border-box;
margin + width(border + padding + content)

js获取盒模型宽高
1.dom.style.width/height 只能获取获取内联样式宽高
2.dom.currentStyle.width/height 获取渲染以后的宽高 只是ie支持
3.window.getComputedStyle(dom).width/height 获取渲染以后的宽高 chrome/firfox支持
4.dom.getBoundingClientRect().width/height left/top/width/height

css盒模型
父子边距重叠问题(margin 塌陷)
1. 添加overflow:hidden 创建BFC

BFC基本概念 块级格式化上下文
原理(渲染规则)
1. BFC元素内子级垂直方向的间距会发生重叠
2. BFC区域不会与浮动元素发生重叠(布局)
3. BFC在页面上是一个独立的容器，外面元素不会影响里面的元素，里面也不会影响外面的元素
4. 计算BFC元素高度的时候，浮动元素也会参与计算(清除浮动)

如何创建BFC

1. float 不为none
2. position 不是static/relative
3. display 为 table table-cell table-columns
4. overflow auto hidden 都可以创建

BFC使用场景
1. 给边界重叠元素加一个父级，父级创建一个BFC，就能解决边距重叠问题
2. 解决浮动元素覆盖正常元素问题
3. 父级设置BFC，子级浮动元素也会参与父级的高度计算(清除浮动)

DOM事件类
基本概念： DOM事件的级别
DOM事件模型： 捕获和冒泡
DOM事件流
描述DOM事件捕获的集体流程
Event对象的常见应用
自定义事件

DOM事件的级别(DOM事件标准定义的级别)
DOM0 ele.onclick = function() {}
DOM2 ele.addEventListener('click', function(){}, false) ie attach
false 冒泡阶段触发， true 捕获阶段触发
DOM3 ele.addEventListener('keyup', function(){}, false) 增加一些事件类型

DOM事件模型：捕获和冒泡
捕获： 从上到目标元素
冒泡： 从目标元素向上

事件流
浏览器在和用户进行交互的过程中，事件是如何传递到页面上的
阶段1 捕获
阶段2 目标阶段
阶段3 冒泡
事件通过捕获到达目标元素，这个阶段就是目标阶段
从目标元素再上传到document对象，这是一个冒泡的过程

描述DOM事件捕获的具体流程
window --> document --> html --> body -->目标元素
获取html标签： document.documentElement
获取body标签： document.body

Event对象的常见应用
event.preventDefault() 阻止默认行为
event.stopPropagation() 阻止冒泡
event.stopImmediatePropagation() 一个元素注册了两个事件，在a中添加这个，

会阻止B的执行
event.currentTarget 事件委托，当前哪个元素被点击
event.target 绑定当前事件的元素

自定义事件

```
var eve = new Event('custome');
oEle.addEventListener('custome', function() {
    console.log('custome')
})
oEle.dispatchEvent(eve)
```

CustomeEvent 用法相同，但是可以指定参数

HTTP协议
HTTP协议的主要特点
HTTP报文的组成部分
HTTP协议方法
POST和GET的区别
HTTP状态码
什么是持久化
什么是管线化

HTTP协议的主要特点
简单快速 每一个资源的URI是固定的
灵活 通过http，可以传输任意的资源
无连接 没链接一次就会断掉，不会保持链接
无状态 服务端不能区分两次连接的身份

HTTP报文的组成部分
请求报文
- 请求行 http方法 页面地址 http协议和版本
- 请求头 key:value值，告诉服务端需要哪些内容
- 空行 请求头和请求体隔开区
- 请求体

响应报文
- 状态行
- 响应头
- 空行
- 响应体

HTTP协议方法
GET 获取资源
POST 传输资源
PUT 更新资源
DELETE 删除资源

HEAD 获取报文首部

POST和GET的区别
GET在浏览器会退时是无害的，而POST会再次提交请求 *
GET产生的URI地址可以被收藏，而POST不可以
GET请求会被浏览器主动缓存，而POST不会，除非手动设置 *
GET请求只能进行URL编码，而POST支持多种编码方式
GET请求参数会被完整保存在浏览器历史记录中，而POST中的参数不会被保留 * 防止CSRF攻击
GET请求再URL中传送的参数是有长度限制的，而POST没有限制 *
对参数的数据类型，GET只接受ASCII字符，而POST没有限制
GET比POST更不安全，因为参数会直接暴露再URL上，所以不能用来传递敏感信息
GET参数通过URL传递，POST方在Requst body中 *

HTTP状态码
1xx: 指示信息--表示请求已接收，继续处理
2xx: 成功--表示请求已被成功接收
3xx: 重定向--要完成请求必须进行更进一步的操作
4xx: 客户端错误--请求有语法错误或者请求无法实现
5xx: 服务器错误--错误未能实现合法的请求

200 OK 客户端请求成功
206 客户端发送一个带有range头的get请求，服务器按照range头上的要求，返回文件对应的部分， 也就是在响应体中，只有range头指定的那部分内容
301 永久重定向，所有页面请求已经转移至新的URL
302 临时重定向
304 客户端有缓冲的文档并发出了一个条件性的请求，服务器告诉客户，原来缓冲文档还可以继续使用
400 客户端有语法错误，不能被服务器理解(参数格式错误)
401 请求未经授权
403 对被请求页面的访问被禁止(权限不足)
404 资源不存在
500 服务器错误
503 服务器临时过载或者当机，一段时间可能恢复正常

什么是持久化
http持久链接 1.1版本支持
http采用请求-应答模式，当使用普通模式。即非keep-alive模式时，每个请求/应答客户和服务器都要创建一个新的连接，完成之后立即断开连接(http 协议为无连接的协议)
当使用keep-alive模式(又称持久连接，连接重用)时，keep-alive功能使客户端到服务端的连接有效，当出现对服务器的后续请求时，keep-alive功能避免了建立或者重新建立连接

什么是管线化
持久连接就是连一次之后，其他请求公用这一个通道，这时候请求就类似这种模式

请求1-->响应1-->请求2-->响应2-->请求3-->响应3

管线化就是
通道持久建立，把现在的请求一次性打包发出去，然后服务器把响应打包一次性发送回来
请求1,请求2,请求3-->响应1,响应2,响应3

管线话特点
管线化机制通过持久连接完成，http/1.1支持 *
只有GET和HEAD请求可以进行管线化，而POST请求则会有所限制 *
初次创建http连接的时候，不应该启动管线化，因为对方服务器不一定支持HTTP/1.1 *
管线化不会影响响应到来的顺序响应返回的顺序也不会改变
因为服务器对管线化支持不好，所以chrome，firefox默认没有开启

原型链
创建对象有几种方法
原型 构造函数 实例 原型链 分别是什么
instanceof 原理
new 运算符干了什么，如何实现

创建对象有几种方法
法1 字面量
```
var o1 = {a: 1};
var o2 = new Object(o1);
```

法2 使用显示的构造函数
```
var M = function() {this.name = 1};
var m1 = new M();
```

法3
```
var p = {name: 1};
var p1 = Object.create(p)
```

new 后面的函数就叫构造函数
构造函数也是以函数，构造函数通过new创建一个实例
只要是函数，就有prototype属性，指向当前函数的原型
构造函数原型的constructor属性指向函数本身
实例的__proto__指向实例构造函数的原型

原型链
实例对象向上找构造这个实例的相关联的对象，这个相关联的对象想上查找，还能找到创建他的构造函数的原型，一直找到Obeject.prototype
多个实例都想具有一个方法，直接写在原型上，省去了每次创建所占用的内存

原型链工作原理
任何一个实例对象，通过原型链找到他上面的原型对象，那上面的方法和属性都是被实例所共享的
实例本身没有找到属性，会在原型对象上找，如果还没找到，会沿着原型链继续找到原型的原

型对象，直到Obeject.prototype

instanceof 原理
实例的__proto__指向的是构造函数的原型
判断对象是不是某一个构造函数的实例的时候，实际上判断的是实例对象的__proto__和构造函数的prototype引用的是不是同一个地址，如果是返回true， 不是返回false
但是 实例的构造函数的原型的构造函数，也会返回true,只要是在那一条链上的构造函数都会返回true

判断谁是亲儿子 obj.__proto__.constructor === M

new运算符工作原理
1. 创建一个新的对象，他继承自构造函数的原型
2. 构造函数执行，执行的时候，相应的传参会被传入，同时上下文(this)会被指定为这个新实例
new Foo() 等同于new Foo，只能用于在不传递任何参数的情况
3. 如果构造函数返回了一个对象，那么这个对象会取代new出来的结果，
如果没有返回对象，那么new出来的结果为步骤1创建的对象

var o = Object.create(p) 创建对象原理
因为生成的对象o的__proto__指向的是p
o.__proto__ === p

面向对象
a. 类与实例
    类的申明 function class
    生成实例 new
b. 类与继承
    如何实现继承 基于原型链
    继承的几种方式

    构造函数实现继承
    原型链实现继承
    构造函数➕原型链实现继承

通信类
什么是同源策略和限制
前后端如何通信
如何创建ajax
跨域通信的几种方式

同源策略和限制
同源策略限制从一个源加载的文档或者脚本如何与来自另一个源的资源进行交互，这是一个用于隔离潜在恶意文件的关键的安全机制
源: 协议，域名，端口 http www.baidu.com 80没有指定端口，默认端口是80
协议，域名，端口有一个不一样就是跨域了

限制：不是一个源的文档，没有权利操作另一个源的文档
cookie localStorage/sessionStorage和indexDB 无法读取
DOM无法获得
Ajax请求无法发送

前后端如何通信
ajax
WebSocket
Form表单
CORS
CORS 跨域资源共享 它允许浏览器向跨源服务器，发出XMLHttpRequest请求，从而客服了ajax同源的限制
CORS需要浏览器和服务器同时支持，他的通信过程都是浏览器自动完成，不需要用户参与，对于开发者来说，CORS通信与同源的Ajax没有区别，代码完全一样
浏览器一旦发现Ajax请求跨域，就会自动添加一些附加的头信息，有时还会多出一次附加请求，但是用户不会有感知
因此，实现CORS的关键是服务器，只要服务器实现了CORS接口，就可以实现跨源通信

CORS请求的两类：简单请求和非简单请求
只要同时满足下面的两个条件，就属于简单请求
1. 请求方法是下面三个中的之一的：HEAD GET POST
2. HTTP的头信息不超出以下几种字段
   Accept
   Accept-Language
   Content-Language
   Last-Event-ID
   Content-Type：只限于三个值，application/x-www-form-urlencoded、multipart/form-date、text/plain
凡是不同时满足上面连个条件，就属于非简单请求
简单请求就是简单的Http方法与简单的Http头信息的结合
这种划分的原因是：表单在历史上一直可以跨域发送请求，
简单请求就是表单请求，浏览器沿袭了传统的处理方式，不把行为复杂化，否则开发者可能会转而使用表单，规避CORS的限制
对非简单请求，浏览器会采用新的处理方式

简单请求
对于简单请求，浏览器直接发出CORS请求，具体来说，就是在头信息之中，增加一个Origin字段
下面就是一个浏览器发现这次跨域ajax请求是一个简单请求，就自动在头信息之中加了一个Origin的字段

```
GET /cors HTTP/1.1
Origin: http://api.bob.com
Host: api.alice.com
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla/5.0...
```

Origin字段用来说明这次请求来自哪个域下(协议、域名、端口)，服务器根据这个值，决定是否同意这次请求

如果Origin指定的源，不再许可范围内，服务器会返回一个正常的HTTP回应，浏览器发现，这个回应的头信息没有包含Access-Control-Allow-Origin字段，就知道出错了，从而抛出一个错误，被XMLHttpRequest的onerror回调函数捕获
**注意：这种错误无法通过状态码识别，因为Http回应的状态码有可能是200
如果Origin指定的域名在许可范围内，服务器返回的响应，会多几个头信息字段
Access-Control-Allow-Origin: http://api.bob.com
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: FooBar
和CORS相关的三个响应头字段都以Access-Control-开头
(1)Access-Control-Allow-Origin
这个字段是必须的。它的值要么是请求时Origin字段的值，要么是一个*，表示接受任意域名的请求
(2)Access-Control-Allow-Credentials
这个字段是可选字段，他是一个boolean值，表示是否可以发送cookie，默认情况下，cookie不包括在CORS请求之中
设置为true，即表示服务器明确许可，浏览器可以把cookie包含在请求中，一起发送给服务器。
这个值也只能为true，如果服务器不需要浏览器发送cookie，不发送该字段即可

如何创建一个ajax
XMLHttpRequest对象的工作流程
兼容性
事件的触发条件
事件的触发顺序

写一个ajax

跨域通信的几种方式
JSONP
Hash hash变动页面不会刷新 search的变动的引起页面的刷新
postMessage h5新加
WebSocket WebSocket不受同源限制
CORS 支持跨域通信的ajax

JSONP原理 如何实现
script标签的异步加载实现的

前端安全
CSRF
OSS

CSRF

基本概念和缩写
攻击原理
防御措施

基本概念和缩写
CSRF 跨站请求伪造(Cross-site request forgery)

XSS: 跨域脚本攻击
http://www.imooc.com/learn/812

渲染机制
js运行机制
页面性能
错误监控

渲染机制
什么是doctype及其作用
浏览器渲染过程
重排(reflow)
重绘(repaint)
布局Layout(浏览器布局方式)

什么是doctype及其作用
DTD(文档类型定义)就是告诉浏览器，当前文件是什么文档类型，浏览器根据这个来判断用哪种方式或者引擎来来渲染解析

doctype
就是直接告诉浏览器当前是哪种DTD，用来帮助浏览器判断文件的合法性验证，如果文件代码不合法，那么浏览器在解析的时候便会出现一些差错

常见doctype
HTML5 <!doctype html>
html 4.01 strict strict.dtd 严格模式
该DTD包含所有HTML元素和属性，但是不包含展示性和弃用的元素 如font
html 4.01 transitional loose.dtd 传统模式
该DTD包含所有HTML元素和属性，包含展示性的和弃用的元素

浏览器的渲染过程
html生成DOM tree
css 生成 cssOM tree
生成render tree
通过layout就能精确的显示元素的宽高
paint 画图

重排(reflow)
DOM结构中的各个元素都有自己的盒子模型，这些都需要浏览器根据何种样式来计算并根据计

算结果讲元素放在它该出现的位置，这个过程称为reflow

触发reflow
当你添加、删除、修改DOM节点的时候，会

重绘repaint

如何避免回流和重绘?

js运行机制 eventloop
如何理解js单线程
什么是任务队列
什么是eventloop

错误监控(如何保证产品质量体系)
前端错误分类
错误的捕获方式
上报错误的基本原理

前端错误分类
及时运行错误: 代码错误
资源加载错误
运行错误
1. try{} catch
2. window.onerror

资源加载错误
1. ele.onerror 错误不会冒泡，所以window.onerror不能捕获资源加载错误，只能捕获及时运行错误
2. performance.getEntries()
获取到所有已经加载到的资源加载时长
所有图片的集合减去上面已经存在资源加载时长的元素的集合，剩下的就是没有成功加载的
3. window上Error事件捕获可以获取到资源加载错误

```
window.addEventListener('error', function(err) {
    console.log('捕获',  err)
}, true);
```

延伸: 跨域的js运行错误可以捕获吗，错误提示是什么，应该如何处理
跨域的js运行错误是可以获取到的
1. 所有跨域文件的js代码错误
错误信息: script error
不能获取到行号和列号
如何做
1. 在script标签增加crossorigin属性
2. 在js资源响应头上增加Access-Control-Allow-Origin: */指定域名

上报错误的基本原理

1.采用ajax上报

2. 利用Image对象上报

```
(new Image()).src = 'http://www.baidu.com/test?err=123'
```