临毕业摸底测验（第一部分）

@(201802)

JavaScript （前端玩家必备技能）

1. ele.getAttribute('propName') 和 ele.propName区别
   getAttribute是获取ele对应标签的行内自定义属性propName,而ele.propName中的属性propName是直接存储在ele这个对象中的。定义在标签行内的自定义属性是无法通过ele.xxx的方式获取，相反定义在ele对象上的自定义属性也无法通过getAttribute获取到。
2. mouseover和mouseenter的区别
   mouseover是鼠标滑过事情，当一个元素嵌套在另一个元素内时，鼠标指针从父级元素滑入子级元素时，会触发父级元素的mouseout事件，然后触发子级元素的mouseover事件。当鼠标离开了子级元素时，触发子级元素和父级元素的mouseout事件同时又会触发父级元素的mouseover事件。
   mouseenter属于鼠标进入，从父级元素进入子级元素时，仅仅触发子级元素的mouseenter，而不会触发父级元素的mouseleave事件。mouseenter阻止了事件冒泡传播，而mouseover存在冒泡传播机制；之前在制作轮播图效果时，使用了mouseenter来实现鼠标进入图片时停止播放的效果，如果使用mouseover会出现一些不符合预期的结果。
3. 什么是事件代理
   事件代理是通过事件冒泡传播机制实现的；当一个容器的很多后代元素有相同的事件行为需要进行处理，利用事件的冒泡机制，只需要给后代元素最外层容器对应的事件绑定方法，根据不同的事件源进行对应的业务处理即可；例如真实项目中的分类导航菜单就可以通过事件代理实现。
4. localStorage和cookie的区别，cookie和session的关系！
   localStorage生命周期是永久，这意味着除非用户显示在浏览器提供的UI上清除localStorage信息，否则这些信息将永远存在。存放

数据大小为一般为5MB,而且它仅在客户端（即浏览器）中保存，不参与和服务器的通信。
cookie生命期为只在设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭。存放数据大小为4K左右。有个数限制（各浏览器不同），一般不能超过20个。与服务器端通信：每次都会携带在HTTP头中。

Cookie和Session的方案虽然分别属于客户端和服务端，但是服务端的session的实现对客户端的cookie有依赖关系的，服务端执行session机制时候会生成session的id值，这个id值会发送给客户端，客户端每次请求都会把这个id值放到http请求的头部发送给服务端，而这个id值在客户端会保存下来，保存的容器就是cookie，因此当我们完全禁掉浏览器的cookie的时候，服务端的session也会不能正常使用。

5. 什么是闭包，你在项目中哪一块用到了闭包！
   闭包是JS中一个非常重要的机制，很多编程思想，业务逻辑和设计模式都是基于闭包机制实现的。我个人对闭包的理解是

> 闭包就是一个函数执行以后形成一个(不销毁)的私有作用域，在该作用域当中所定义的私有变量和其函数外部定义的变量互不干扰，而且只要该作用域不被销毁，这些私有变量的值就能被保存下来；闭包的主要作用就是用来保护和保存函数内私有变量的。
> 以下是我在日常项目，像循环元素做事件绑定，可以用到闭包。由于事件绑定是异步编程的，可以通过闭包机制存储索引值，后期需要使用时，向上级作用域查找即可；
> 在团队项目开发中，为了避免全局变量污染(变量冲突)，可以使用闭包机制，即单例模式封装不同功能板块的代码；

6. js中定义函数的方式有哪些，区别是什么！
   匿名函数方式，将一个没有函数名的函数赋值给一个变量，这种函数声明的方式在变量提升阶段只做声明而不进行定义
   实名函数方式，这种函数声明的方式在变量提升阶段声明和定义都已经完成
   箭头函数，不存在变量提升，而且函数中的this是继承上下文的this，也不可以通过bind,call和apply惊醒改变。

7. 说出你掌握的继承方式及优缺点，并加以改进！

   1)原型继承

   B.prototype=new A();子类B的原型就是父类A的一个实例，父类A实例具有其自己的私有属性，并可以通过**proto**可以找到父类A的Prototype上的公有方法和属性；原型继承的缺点是通过B.prototype.**proto**修改父类A原型上的方法。父类实例的私有属性/方法和公有属性/方法都成为了子类B的公有属性/方法。如果子类B原型上之前有定义过方法或属性，在让B原型重新指向父类A的实例后，之前B原型上的方法/属性就不存在了，且constructor指向了构造函数A。

2)call方法继承

通过call让父类A的构造函数以普通函数的方式在子类B的构造函数中执行并将this改为构造函数B的实例；

缺陷：只是让子类B的实例继承了父类A实例的私有属性而已，父类A原型上的公有属性方法和子类B的实例没有任何关系

方案1和方案2可以通过寄生组合继承得以改进

让子类B的私有属性继承父类A的私有属性，子类B的公有属性继承父类A的公有属性

```
function A() {
        this.x = 100;
    }

    A.prototype = {
        constructor: A,
        getX: function () {
            console.log(this.x);
        }
    };

    function B() {
        A.call(this);
        this.y = 200;
    }
```

```
    B.prototype = Object.create(A.prototype);
    B.prototype.getY = function () {
        console.log(this.Y);
    }
    let f = new B();
```

3)ES6中的继承

```
    class A {
        constructor() {
            this.x = 100;
        }

        getX() {
            console.log(this.x);
        }
    }

    class B extends A {
        constructor() {
            super();
            this.y = 200;
        }

        getY() {
            console.log(this.y);
        }
    }

    let f = new B();
```

8. 说出ES6和ES5的区别!
   -let / const
   和ES5的VAR的区别

> 1)let不存在变量提升机制(变量不允许在声明之前使用)
> 2)let不允许重复声明
> 3)在全局作用域中基于let声明的变量不是window的一个属性，和它没有关系
> 4)使用typeof检测未被声明的变量的数据类型返回的不是undefined而是报错(暂时性死区)
> 5)let会形成块级作用域(类似于私有作用域，大部分大括号都会形成块级作用域)

-解构赋值
从数组和对象中提取值

-"..."扩展、剩余、展开运算符

- 展开运算符(spread operator)，作用是和字面意思一样，就是把东西展开。可以用在array和object上都行。
  比如:
  ```
  let a = [1,2,3];
  let b = [0, ...a, 4]; // [0,1,2,3,4]
  ```

```
let obj = { a: 1, b: 2 };
let obj2 = { ...obj, c: 3 }; // { a:1, b:2, c:3 }

let obj3 = { ...obj, a: 3 }; // { a:3, b:2 }
```

- 剩余操作符(rest operator)，是解构的一种，意思就是把剩余的东西放到一个array里面赋值给它。一般只针对array的解构，其他的没见过。。。
  比如:
  ```
  let a = [1, 2, 3];
  let [b, ...c] = a;
  b; // 1

  c; // [2,3]
  ```

```
let a = [1, 2, 3];
let [b, ...[c, d, e]] = a;
b; // 1

c; // 2

d; // 3

e; // undefined

//当函数的形参个数不确定时，可以使用剩余运算符
function test(a, ...rest) {
    console.log(a); // 1

    console.log(rest); // [2,3]

}

test(1, 2, 3)
```

还有类似的

```
let array = [1, 2, 3, 4, 5];
const [x, y, ...z] = array;
// 其中z=[3, 4, 5]，注意如果由于array的length不足以完成析构，
则会导致z为[]
// 对象：

let obj = {name: 'zhangsan', age: 30, city: 'shenzhen'};
const {name, ...others} = obj;
console.log(name); // 'zhangsan'

console.log(others); // {age: 30, city: 'shenzhen'}
```

-箭头函数

和ES5普通函数的区别

> 1)没有arguments，但是可以基于...arg获取实参集合(结果是一个数组)
> 2)没有自己的this，箭头函数中的this是上下文中的this

-ES6模板字符串
让传统的字符拼接更加方便，通过使用`${}`

-Promise

-class(ES6中创建的类)

-interator (for of 循环)

-Map / Set

9. 阐述JS中的同步编程和异步编程，以及你在项目中是如何来使用异步操作的!
   JS是单线程进行执行的，在JS中分为主任务队列和等待任务队列；所用同步任务依次在主任务队列中执行，当遇到像定时器，事件绑定等异步操作时，会将其放入等待任务队列当中，只有当主任务队列中所有同步任务执行完成以后，等待任务队列中的异步操作被依次放入主任务队列中执行。
   在项目当中，异步操作用于事件绑定，发布订阅已经Ajax请求；一般讲Ajax请求设置为异步，这样位于Ajax后面的操作不需要等待Ajax请求全部完成也能正常执行。在ES6中，通过使用Promise对一些异步操作进行管控。

10. 实现一个Promise
    (自己暂时还无法实现)

```
class Promise {
    constructor(excutorCallBack) {
        this.status = 'pending';
        this.value = undefined;
        this.fulfilledAry = [];
        this.rejectedAry = [];
```

```
        //=>执行EXCUTOR（异常捕获）
        let resolveFn = result => {
            let timer = setTimeout(() => {
                clearTimeout(timer);
                if (this.status !== 'pending') return;
                this.status = 'fulfilled';
                this.value = result;
                this.fulfilledAry.forEach(item =>
item(this.value));
            }, 0);
        };
        let rejectFn = reason => {
            let timer = setTimeout(() => {
                clearTimeout(timer);
                if (this.status !== 'pending') return;
                this.status = 'rejected';
                this.value = reason;
                this.rejectedAry.forEach(item =>
item(this.value));
            }, 0);
        };
        try {
            excutorCallBack(resolveFn, rejectFn);
        } catch (err) {
            //=>有异常信息按照REJECTED状态处理
            rejectFn(err);
        }
    }

    then(fulfilledCallBack, rejectedCallBack) {
        //=>处理不传递的状况
        typeof fulfilledCallBack !== 'function' ?
fulfilledCallBack = result => result : null;
        typeof rejectedCallBack !== 'function' ?
rejectedCallBack = reason => {
            throw new Error(reason instanceof Error ?
reason.message : reason);
```

```
        } : null;

        //=>返回一个新的PROMISE实例
        return new Promise((resolve, reject) => {
            this.fulfilledAry.push(() => {
                try {
                    let x = fulfilledCallBack(this.value);
                    x instanceof Promise ? x.then(resolve,
reject) : resolve(x);
                } catch (err) {
                    reject(err);
                }
            });
            this.rejectedAry.push(() => {
                try {
                    let x = rejectedCallBack(this.value);
                    x instanceof Promise ? x.then(resolve,
reject) : resolve(x);
                } catch (err) {
                    reject(err);
                }
            });
        });
    }

    catch(rejectedCallBack) {
        return this.then(null, rejectedCallBack);
    }

    static all(promiseAry = []) {//=>Promise.all()
        return new Promise((resolve, reject) => {
            //=>INDEX:记录成功的数量 RESULT:记录成功的结果
            let index = 0,
                result = [];
            for (let i = 0; i < promiseAry.length; i++) {
                //=>promiseAry[i]:
                //每一个需要处理的PROMISE实例
```

```
                promiseAry[i].then(val => {
                    index++;
                    result[i] = val;//=>索引需要和promiseAry对
应上，保证结果的顺序和数组顺序一致
                    if (index === promiseAry.length) {
                        resolve(result);
                    }
                }, reject);
            }
        });
    }
}

module.exports = Promise;
```

HTTP && AJAX && 跨域 （18+玩家必备技能，初级玩家需要了解一些的）

1. 写出项目中经常用到的性能优化方案
   1)在JS中尽量减少闭包的使用(原因：闭包会产生不释放的栈内存)
   A:循环给元素做事件绑定的时候，尽可能地把后期需要的信息(例如索引)存储到元素的自定义属性上，而不是创建闭包存储
   B:可以在最外层形成一个闭包，把一些后续需要的公共信息进行存储，而不是每一个方法都创建闭包(例如单例模式)
   C:尽可能的手动释放不被占用的内存

2)尽量合并CSS和JS文件(把需要引入的CSS合并为一个，JS也合并为一个)，原理是可以减少HTTP请求次数，尽可能地把合并后的代码进行压缩，减小HTTP请求资源的大小
A: webpack这种自动化构建工具，可以帮我们实现代码的合并和压缩(工程化开发)
B: 在移动端开发(或者追求高性能的PC端开发[例如百度首页])，如果CSS或者JS不是需要很多，我们可以选择把CSS和js编程内嵌式(也就是代码直接写在HTML中)

3)尽量使用字体图标或者SVG图标图标，来代替传统的PNG等格式的图片(因为字体图标等是矢量图(基于代码编写写出来)，放大不会变形，而且渲染速度快，相对比位图要小一些)

4)减少对DOM的操作(主要是减少DOM重绘和回流(重排))
A:关于重排的分离读写
B:使用文档碎片或者字符串拼接做数据绑定(DOM的动态创建)

5)在JS中避免“嵌套循环”(这种会额外增加很多循环次数)和‘死循环’(一旦遇到死循环浏览器就卡壳了)

6)采用图片延迟加载
目的是为了减少页面第一次加载过程中HTTP的请求次数，让页面打开速度变快
步骤：开始加载页面的时候，所有的真实图片都不去发送HTTP请求加载，而是给一张占位的背景图，当页面加载完，并且图片在可视区域内我们再去做图片加载

7)利用浏览器和服务器端的缓存技术(304缓存)，把一些不经常更新的静态资源文件做缓存处理(例如:JS、CSS、静态图片等都可以做缓存)
原理是为了减少HTTP请求次数和请求大小

8)尽可能使用事件委托(事件代理)来处理事件绑定的操作，减少DOM的频繁操作，其中包括给每一个DOM元素做事件绑定

9)CSS雪碧图

10)减少对于COOKIE的使用(最主要的是减少本地COOKIE存储内容的大小),因为客户端操作COOKIE的时候，这些信息总是在客户端和服务器端传来传去

2. 从浏览器地址栏输入URL到显示页面，中间都经历了什么（尽可能写详细，最好回答出TCP的三次握手和四次挥手，以及浏览器加载页面的细节）
   输入URL后，会进行解析（URL的本质就是统一资源定位符）
   详细：https://mp.weixin.qq.com/s/qMsf4DcMhn2cf0fXC-PLVA
   [HTTP请求阶段：向服务器发送请求]

1.浏览器首先向DNS域名解析服务器发送请求
2.DNS反解析:根据浏览器请求地址中的域名，到DNS服务器中找到对应的服务器外网IP地址
3.通过找到的外网IP，向对应的服务器发送请求(首先访问的是服务器的WEB站点管理工具：准确来说是我们先基于工具在服务器上创建很多服务，当有客户端访问的时候，服务器会匹配出具体是请求哪个服务)
4.通过URL地址中携带的端口号，找到服务器上对应的服务，以及服务所管理的项目源文件

[HTTP响应阶段:服务器把客户端需要的内容准备好]

5.服务器根据请求地址中的路径名称、问号传参或者哈希值，把客户端需要的内容进行准备和处理
6.把准备的内容响应给客户端(如果请求的是HTML或者CSS等这样的资源文件，服务器返回的是资源文件中的源代码[不是文件本身])

[浏览器渲染阶段]

7.客户端浏览器接收到服务器返回的源代码，基于自己内部的渲染引擎(内核)开始进行页面的绘制和渲染
->首先计算DOM结构，生成DOM TREE
->自上而下运行代码，加载CSS等资源内容
->根据获取的CSS生成带样式的RENDER TREE
->开始渲染和绘制

我们把一次完整的 请求+响应称之为"HTTP事务"
事务就是请求后得到了响应的一次完整的操作，请求和响应缺一不可

一个页面完全加载完成，需要向服务器发起很多次HTTP事务操作
一般来说首先把HTML源代码拿回来，加载HTML的时候，遇到link/script/img[src]/iframe/video和audio[没有设置preload='none']...都会重新和服务器端建立HTTP事务交互 （如果出现preload属性，则视频在页面加载时进行加载，并预备播放。）

特殊情况：如果我们做了资源缓存处理(304)，而且即将加载的资源在之前已经加载过了，这样的操作和传统的HTTP事务有所不一样，他们是从服务器和浏览器的缓存中读取数据，比传统的读取快很多

3. 说出你所熟知的HTTP状态码！GET和POST有啥区别！
   200 OK 成功 (只能证明服务器成功返回信息了，但是信息不一定是业务需要的)

301 Moved Permanently 永久转移 （被请求的资源已永久移动到新位置）
=>域名更改，访问原始域名重定向到新的域名

302 / 307 Move Temporarily 临时转移
=>网站现在是基于HTTPS协议运作的，如果访问的是HTTP协议，会基于307重定向到HTTPS协议上
=>302一般用作服务器负载均衡：当一台服务器到达最大并发数的时候，会把后续访问的用户临时转移到其它的服务器机组上处理
=>偶尔真实项目会把所有的图片放到单独的服务器上“图片处理服务器”，这样减少主服务器的压力，当用户向主服务器访问图片的时候，主服务器都把它转移到图片服务器上处理

304 Not Modified 设置缓存
=>对于不经常更新的资源文件，例如：CSS/JS/HTML/IMG等，服务器会结合客户端设置304缓存，第一次加载过这些资源就缓存到客户端了，下次再获取的时候，是从缓存中获取；如果资源更新了，服务器端会通过最后修改时间来强制让客户端从服务器重新拉取；基于CTRL+F5强制刷新页面，304做的缓存就没有用了。

400 Bad Request 请求参数错误
401 Unauthorized 无权限访问
404 Not Found 找不到资源(地址不存在)
413 Request Entity Too Large 和服务器交互的内容资源超过服务器最大限制

500 Internal Server Error 未知的服务器错误
503 Service Unavailable 服务器超负荷

get和post虽然本质都是tcp/ip，但两者除了在http层面外，在tcp/ip层面也有区别。

get会产生一个tcp数据包，post两个。

具体就是：

get请求时，浏览器会把 headers和 data一起发送出去，服务器响应200（返回数据），

post请求时，浏览器先发送 headers，服务器响应 100continue，浏览器再发送 data，服务器响应200（返回数据）。

4. 什么是HTTP报文，你熟知的报文都有哪些！
   在客户端向服务器发送请求，以及服务器把内容响应给客户端的时候，中间相互传递了很多内容(客户端把一些内容传递服务器，服务器把一些内容响应给客户端)，我们把传递的内容统称为**"HTTP报文"**
   熟知的有ARP报文，PING报文，IP报文，TCP报文，UDP报文
5. 能说下304具体怎样实现吗?
   客户端在请求一个文件的时候，发现自己缓存的文件有 Last Modified ，那么在请求中会包含 If Modified Since ，这个时间就是缓存文件的 Last Modified 。因此，如果请求中包含 If Modified Since，就说明已经有缓存在客户端。服务端只要判断这个时间和当前请求的文件的修改时间就可以确定是返回 304 还是 200 。
   对于静态文件，例如：CSS、图片，服务器会自动完成 Last Modified 和 If Modified Since 的比较，完成缓存或者更新。但是对于动态页面，就是动态产生的页面，往往没有包含 Last Modified 信息，这样浏览器、网关等都不会做缓存，也就是在每次请求的时候都完成一个 200 的请求。
   因此，对于动态页面做缓存加速，首先要在 Response 的 HTTP Header 中增加 Last Modified 定义，其次根据 Request 中的 If Modified Since 和被请求内容的更新时间来返回 200 或者 304 。
6. 跨域是什么？http协议中如何判断跨域？如何解决跨域问题?
   这里说的js跨域是指通过js在不同的域之间进行数据传输或通信，比

如用ajax向一个不同的域请求数据，或者通过js获取页面中不同域的框架中(iframe)的数据。
在http协议只要协议、域名、端口有任何一个不同，都被当作是不同的域。
可以使用以下方法进行跨域:
-jsonp
-cors
-postMessage
-document.domain
-window.name
-location.hash
-http-proxy
-nginx
-websocket

7. HTTP2具体内容？SDPY了解么？
   异步连接多路复用，头部压缩，请求/响应管线化，与HTTP1.1语义向后兼容性，多路复用请求，对请求划分优先级，压缩HTTP头；
   服务器推送流（即Server Push技术);
   SPDY对比HTTP的优势:

复用连接，可在一个TCP连接上传送多个资源。应对了TCP慢启动的特性。
请求分优先级，重要的资源优先传送。
HTTP头部数据也被压缩，省流量。
服务器端可主动连接客户端来推送资源（Server Push）。
缺点:

单连接会因TCP线头阻塞（head-of-line blocking）的特性而传输速度受限。加上存在可能丢包的情况，其负面影响已超过压缩头部和优先级控制带来的好处。

8. HTTPS如何实现？tsl/ssl是什么？对称加密、非对称加密在什么时候、对什么数据加密?

HTTPS 协议（HyperText Transfer Protocol over Secure Socket Layer）：可以理解为HTTP+SSL/TLS， 即 HTTP 下加入 SSL 层，HTTPS 的安全基础是 SSL，因此加密的详细内容就需要 SSL，用于安全的 HTTP 数据传输。

TLS（Transport Layer Security，传输层安全）：其前身是 SSL，它最初的几个版本（SSL 1.0、SSL 2.0、SSL 3.0）由网景公司开发，1999年从 3.1 开始被 IETF 标准化并改名，发展至今已经有 TLS 1.0、TLS 1.1、TLS 1.2 三个版本。SSL3.0和TLS1.0由于存在安全漏洞，已经很少被使用到。TLS 1.3 改动会比较大，目前还在草案阶段，目前使用最广泛的是TLS 1.1、TLS 1.2。

SSL（Secure Socket Layer，安全套接字层）：1994年为 Netscape 所研发，SSL 协议位于 TCP/IP 协议与各种应用层协议之间，为数据通讯提供安全支持。

对称加密

有流式、分组两种，加密和解密都是使用的同一个密钥。

例如：DES、AES-GCM、ChaCha20-Poly1305等

非对称加密

加密使用的密钥和解密使用的密钥是不相同的，分别称为：公钥、私钥，公钥和算法都是公开的，私钥是保密的。非对称加密算法性能较低，但是安全性超强，由于其加密特性，非对称加密算法能加密的数据长度也是有限的。

例如：RSA、DSA、ECDSA、 DH、ECDHE

9. DNS劫持是什么?

域名劫持就是在劫持的网络范围内拦截域名解析的请求，分析请求的域名，把审查范围以外的请求放行，否则直接返回假的IP地址或者什么也不做使得请求失去响应，其效果就是对特定的网址不能访问或访问的是假网址。

10. 封装一个AJAX库!