1. **预解释**
2. 预解释：在当前作用域下，js代码执行之前，浏览器会把带var和function进行提前声明或者定义
3. 对带var：在预解释阶段只声明，不定义；
4. 对function：在预解释阶段声明+定义；
5. 作用域：全局作用域和私有作用域；域：空间或者范围；
6. 全局作用域：在浏览器打开HTML页面的时候，会形成一个供js执行的环境；在全局作用域环境下，里面所有的变量和函数实际就是window的属性和方法
7. 私有作用域：在函数被调用的时候，会形成一个私有作用域；（形参和带var关键字是私有变量）
8. 在私有作用域下，不带var的变量表示非私有变量，但不一定是全局变量，可能是上局作用域变量；
9. 声明和定义的概念
10. 声明：告诉浏览器有这么一个名字，但是不知道值
11. 定义：给之前声明过的这个名字附值；q
12. 函数定义和执行阶段
13. 函数定义3阶段：免费课自己想
14. 函数执行3阶段：a.给形参赋值b.预解释c.js代码从上往下执行
15. 作用域链：为了保证执行环境里的函数或者变量之间有序访问；Window是最外层的作用域链，当前执行的函数是最前端的作用域链

怎样查找上级作用域？当前的这个函数是在哪个作用域下定义的，上级作用域就是谁，**跟函数在哪里执行是没有关系的**

1. 带var和不带var的区别
2. 带var：会预解释，如果带var在全局环境下，它是Window的全局属性
3. 不带var：不会预解释，首先会往上找，如果是赋值的话，找到Window还没有，就变成Window上的全局属性
4. js中的内存，按照功能的不同分为：
5. 栈内存：提供一个供JS代码执行的环境：全局作用域，私有作用域
6. 堆内存：存储引用数据类型的值；对象数据类型：存储属性和属性值；函数数据类型：代码字符串
7. **预解释之无节操**
8. 函数当值存在时，不进行预解释；
9. 自执行函数不预解释，定义+执行一起做了；
10. 不管if条件是否成立，都走预解释
11. return后面的值不进行预解释，return下面的代码不执行，但是要进行预解释
12. 在变量名相同的情况下，不重复声明，但是会重新赋值
13. **为什么要使用闭包**
14. 防止变量名冲突
15. 可以通过传参的方式，对全局变量在私有作用域里重新赋值
16. 闭包里面的值，如果想修改全局的值，可以通过window.xx=yy修改
17. 闭包可以用来封装方法，最后通过Window.方法名=方法名，把封装好的函数，让外面可以用到
18. 闭包可以实现模块化开发，但是没法实现各个模块之间的调用
19. **this关键字**

**\*this是谁和函数在哪里执行的以及在哪里定义的都没有关系**

1. 在非严格模式下
2. 自执行函数里的this，永远都是window；
3. 函数调用的时候，看前面是否有 . ,有的话，点前面是谁，this就指向谁，没有的话就指向window
4. 如果一个元素通过一个行为来触发一个方法，方法中的this就是当前这个元素

**回调函数、匿名函数中的this一般指的是window**

1. 构造函数中，this指向的是实例
2. 通过apply和call可以改变this关键字指向，优先级最高，遇到apply和call，以上四条规则都不成立
3. 在JS严格模式下（use strict）
4. 自执行函数中的this是undefined
5. 使用call和apply改变this指向：

fn.call() //this-undefined

fn,call(null) //this-null

fn.call(undefined) //this-undefined

1. 严格模式和非严格模式的区别

非严格模式下所有没有具体写上执行主体的，都默许this是Window，严格模式下写什么就是什么，没写就是undefined

1. **堆内存和栈内存的释放**
2. 堆内存的释放：当堆内存被变量占用的时候，无法释放；解决办法：变量=null；当变量指针为空的时候，浏览器会在空闲时把被释放的堆内存收回，这种机制叫做垃圾回收机制
3. 栈内存释放：
4. 全局作用域：在浏览器加载完整个页面时，形成一个共全局使用的环境叫全局作用域，只有关闭浏览器，才能彻底释放；如果不关闭浏览器，这个全局环境中的变量和方法都无法释放
5. 私有作用域：当函数执行完成时，私有作用域一般情况下会自动销毁；
6. 私有作用域不销毁的情况：a,不销毁：当函数执行完成时，如果函数里面的东西被外面的变量或者其他东西占用，就无法销毁；b.不立即销毁
7. **对象、类、实例**
8. 对象：泛指，js中万物皆对象，它是多种数据类型的复合载体；类：对象中具体事物的一个细分；实例：类中具体的一个实例
9. 面向对象编程：以对象数据类型为导向的编程；用编程的语言来描述事物的方式就是基于对象的概念
10. 面向对象的特点：1）封装：低耦合高内聚;

2）继承：子类继承父类的属性和方法；

3）多态：多种形态，包括重载和重写

重载：js中相同函数名，后面会把前面的覆盖掉，所以**js不存在重载**，但是js中有类似重载的思想，即传不同的参数实现不同的效果

重写：子类可以重写父类的属性和方法

1. 基于对象开发的编程方式如下：单例模式，工厂模式，构造函数模式，原型模式
2. **单例模式**
3. 什么是命名空间？对象类型的数据,浏览器会为其开辟一个堆内存,然后将属性和值存储在这个空间里,给这个空间起个名字(person1)->命名空间
4. 什么是单例模式？把描述同一事物的属性和方法存在相同命名空间里进行分组，避免不同事物相同属性，变量名冲突的问题
5. 单例模式本质就是对象（单个的实例）,好处：1）可以实现分组2）防止冲突3）各模块之间可以相互调用；缺点：相当于手工作业模式，一个一个创建，效率低
6. 单例模式中各模块的相互调用：模块名**.**属性名；本模块之间的相互调用：this.属性名/方法名
7. **工厂模式（低耦合高内聚）**
8. 什么是工厂模式？函数的封装，即将页面上实现相同功能的js代码封装在一个函数里，在函数里每次都会新建对象，并且把对象返回
9. 工厂模式缺点：返回的都是一个对象类型的，没法解决产品差异化，没法解决实例识别的问题
10. **构造函数**
11. 好处：通过创建自定义类，创建类的实例，解决了实例识别的问题
12. 构造函数名首字母一定要大写，为了和普通函数区分开
13. 构造函数的执行：new+函数名
14. 运行机制：1）隐式的创建一个对象，运行完之后会把这个对象返回

2）以实例未执行主体运行这个函数

3）会把对象自动返回，返回的对象就是类的一个实例

5. 知识点1）构造函数里的this指的是实例；

2）构造函数里定义的私有变量和实例没有关系，方法运行完之后，私有变量就销毁了；

3）如果手工写返回值，则可能对实例造成影响，如果return+基本数据类型，则对实例没有影响；如果return+引用数据类型，会把默认返回的对象覆盖

4）in可以判断这个属性（私有+共有）是否是这个对象的，

5）hasOwnProperty可以检测属性是否是私有的，如果是私有返回true，否则返回false；

6）constructor用来检测数据类型的

**十．原型模式**

1. 可以解决复用的问题，可以设置公共的属性和方法

2. prototype属性里存储的是共有的属性和方法，所有的函数都有prototype属性，只有在构造函数这边才有意义

3. **每一个函数天生自带prototype属性，指向浏览器默认开辟的一个堆内存，prototype上面天生自带一个constructor 属性，指向构造函数本身；每一个对象(实例，prototype)上都有\_\_proto\_\_属性,指向所属类的原型**

4. 批量设置的共有属性，prototype不再指向浏览器默认开辟的堆内存，指向重新开辟的堆内存；浏览器会在空闲时间把默认开辟的堆内存释放掉

5. 实例查找属性：首先查找私有属性，如果没有，则通过\_\_proto\_\_查找原型对象上的共有属性，如果没有,则通过\_\_proto\_\_继续往所属类的原型上查找，一直查找到Object类的原型，这样的查找过程就是基于原型链的查找

6. Oject.prototype上常用的自带属性：hasOwnproperty,toString,

Function.prototype 上常用的自带属性：call,bind,apply,

**十一.继承**

原型链是实现继承的关键，是js独有的一种机制

1. call继承：实现私有属性和私有方法的继承
2. 拷贝继承：for…in循环实现；extend()

**十二.函数的三种角色**

1. 普通函数：当方法运行的时候,形成一个私有的作用域,1.形参赋值 2.预解释 3. 代码从上往下执行
2. 构造函数(类)：new+函数名执行
3. 函数也是对象，只是函数比较特殊，因此单独独立成function类型，它可以像对象一样设置自己的属性和值（内置属性name里存函数名；length里存形参个数）

——以上这三种角色之间相互独立，没有关系

1. 函数与对象的关系？所有类都是构造函数Function的实例，Function是Object的爹，Object.prototype是Function,prototype的爹；

fn instanceof Function 结果是true；fn instanceof Object 结果也是true；

**十三.call和apply和bind**

1. bind：预处理机制，提前把this指向改了，参数也改好了，bind会返回一个函数，什么时候需要什么时候调用就可以

2. call和apply的区别：传参的方式不一样，call是散列式的传参方式，apply打包的方式；

3. 在非严格模式下，call和apply传的第一个参数是null/undefined，或者没有参数，方法里的this表示的是window，其他情况，你传的是什么，this就是什么

4. 在严格模式下(“use strict”)，如果第一个参数是null则this就是null, 如果是undefined或者没写参数则this是undefined,其他情况.你传的是什么this指的就是什么

5. call方法传参,从第二个参数开始,所有的参数相当于"."前面方法的实参,把一个参数的值赋给第一个形参,第二个参数的值赋给第二个形参....

6. apply方法,是把所有的要传的实参都放在一个数组里给"."前面的方法,但是实际上也是一个一个的传给方法的形参 ,把数组的第一项赋给第一个形参,第二项赋给第二个形参....

7. try{}catch(e){}finally{}可以捕获浏览器的异常信息，并且浏览器不会抛出异常，下面的js语句正常执行

**十三.DOM映射**

1. 获取的元素/元素集合，和页面上的标签/容器是一一对应的，当有一个发生改变，另一个也会发生变化，叫DOM映射
2. appendChild/insertBefore如果页面上已经有这个标签了，则表示移动位置，没有这个标签的时候，才是把元素添加

**十四.DOM回流和重绘**

1. DOM回流：页面的结构(添加，删除，位置移动….padding,margin,width,height)发生改变的时候，浏览器需要重新计算页面的结构，这就成为DOM回流
2. 重绘：指页面元素的样式(字体颜色，字体大小，背景颜色)发生改变的时候，浏览器只需要重新渲染当前的元素即可

**十五.JSON**

1. JSON是一种数据格式，JSON格式字符串和JSON格式的对象；JSON格式的对象zxc，对象里的属性必须以””括起来，单引号都不行
2. 为什么要解析JSON？后台给我们的数据都是JSON类型的字符串，我们必须把它转成对象才能进行数据绑定
3. JSON是Window的一个属性，属性值是一个对象,这个对象里有两个方法
4. JSON.parse()；把JSON格式的字符串转化成JSON格式的对象
5. JSON.stringify()；把JSON格式的对象转化成JSON格式的字符串
6. IE6-7中Window没有JSON属性，因此没有办法转化

**十六.AJAX的好处和实现的步骤**

1. AJAX(**A***synchronous***J***avascript****A****nd***X***ML*): 可以在不重新加载整个网页的情况下对网页的某部分进行更新

2. 创建一个XMLHttpRequest实例(xml表示可扩展的HTML语言)

var xhr=new XMLHttpRequest

3. 打开一个链接地址（这个链接地址就是向服务器端请求的数据地址）API（application programming interfact）

xhr.open(“get”,”json/userInfo.tex”,false)

1. 请求的方式：get ,post ,delete, put
2. 请求的链接地址
3. 同步还是异步 true是异步 false是同步
4. 请求的时候会一直触发onreadystatechange监听请求的状态码和响应的状态码
5. 请求的状态：xhr.readyState

readyState一共有5个状态：0 未连接/1 打开连接/2 发送请求/3 交互/4 完成交互，接手响应/不同的状态你可以有不同的操作

1. 响应的状态：xhr.status 200,203,204,205….都表示响应成功
2. 把后台返回的JSON格式的字符串转换成JSON对象
3. 最后向服务器端发送数据，没有数据也要发送

xhr.send(null)

**十七.正则**

1. 什么是正则？正则是用来处理字符串，是用来处理负责复杂的方法
2. 正则表达式：相当于定义了一个模型（模式/规则）,
3. 正则的作用？
4. 用这个模式去匹配字符串的某一部分是否和模型匹配
5. 把匹配的内容提取出来
6. 正则的组成部分：元字符(具有特殊含义的元字符和普通的字符)和修饰符组成
7. 元字符
8. 具有特殊含义的元字符

\：把\后面的字符转义成其他的含义

^：以某个字符开始

$：以某个字符结尾

\d：0-9之间的任意一个数字

\D：除了0-9，其他的字符

\n：换行符

\w：常用的单词字符a-z,A-Z,0-9\_，数字，字母，下划线

\W：除了数字，字母，下划线其他的任意字符

\s：匹配空白字符，包括空格，制表符，换页符

\S：非空白字符

\b：边界符“a b c”

\B：非边界符

**.**：除了\n符以外的其他的字符

1. 量词元字符

\*：0次或者多次

+：1次或者多次

？：0次或者1次

[ ]：1.[ ]里的字符表示元字符本身，2.[ ]内没法识别两位数

{n}：匹配n次

{n,}: 至少匹配n次

{n,m}: 至少匹配n次，最多匹配m次

x|y：x或者y中的一个

[xyz]：x或y或z中的一个

[^xyz]：不能是x或y或z

[a-z]：a-z之间的任意一个字符 **ascii码中**

[^a-z]：除了a-z之间其他的任意字符

1. 修饰符：g，i，m,
2. global全文查找 b. ignoreCase 匹配时忽略大小写 c. multiline 多行匹配

3）**( )：作用**1.分组的作用，能改变优先级

2.分组捕获：在捕获大正则匹配内容的时候，顺便每一个小分组匹配的内容捕获到

3.分组引用：规定和某一个分组出现的内容需要一模一样

4）？的作用：1.在非量词元字符右边，表示：量词元字符本身0到多次

2.在量词元字符右边，表示：在正则捕获的时候取消它的贪婪捕获

3.?: 只出现在分组中，表示：只匹配不捕获

4.？=；正向预查

5. ?!；负向预查（不管是正向预查还是负向预查其实都是在设定一些条件）

1. 正则原型上常用的属性和方法

RegExp.prototype🡪test()字符串想匹配；exec()🡪把匹配的内容捕获出来

1. **字面量方式和实例创建的方式在正则中的区别**：
2. 字面量方式中出现的一切都是元字符，所以不能进行变量值的拼接，而实例是可以的
3. 字面量方式中直接写\d就可以，而在实例中需要把它转译\\d
4. 验证中文姓名

var reg=/^[\u4e00-\u9fa5]{2,4}$/

**十八. 盒子模型**

js中没有直接获得宽度和高度的属性，但是有直接获得与宽高相关的属性

1. client系类：clientWidth; clientHeight; clientTop; clientLeft;
2. offset系列：offsetParent; offsetWidth; offseHeight; offsetLeft; offsetTop;
3. scroll系类：scrollWidth; scrollHeihgt; scrollTop; scrollLeft;
4. client系类详细解释：与内容溢出：无关系 clientWidth/clientHeight:内容的宽高+padding

clientTop/clientHeight:上border/左border的宽度

1. offset系列详细解释：与内容溢出：无关系 offfsetWidth,scrollHeight:clientWidth/height+border

offsetLeft,offsetTop:当前元素的外边距距离上级盒子的内边距的距离

offsetParent:有定位的父级

1. scroll系类详细解释：与内容溢出：有关系

scrollWidth;scrollHeight:在无内容溢出的情况下：等于clientWidth/clientHeight

在有内容溢出的情况下：约等于实际内容的宽/高+左/上padding值

约等于：1）在各大浏览器获取的值不同

2）在同一浏览器下，是否加overflow：hidden获得的值也不一样

scrollLeft,scrollTop:被浏览器卷去的距离

1. 以上属性的小知识点

1）以上属性无法获取到小数，只能得到四舍五入的整数

2）以上属性得到的都是复合值，无法取到单独的宽度高度

3）在有父级定位的情况下，只能获取到上级内边框的距离，无法获取到body顶部和left的距离

4）浏览器盒子模型的兼容处理：win（）封装

5. JS中处理兼容的几种思想：

1）try…cath…不建议使用，因为无论浏览器是否支持，都要走一遍try里的语句；性能比较差

2）属性判断,可以使用attr in obj/obj[attr];建议使用；因为if和else只会走一个

3）大招，浏览器判断；一般不使用；

6. 判断IE6-8的几种方法

1）/MSIE(6|7|8)/i.test(navigator.userAgent);**(判断浏览器版本信息：Window.navigator.userAgent)**

2) str.search(正则)；如果找到返回下标，找不到返回-1；

7. 区分结构父级和定位父级

结构父级的祖师爷 按标签找祖师爷：HTML；如果不是按标签找祖师爷：htmlDocument HTML文档

定位父级的祖师爷 body，找不到返回null

**十九.图片延迟加载**

1. 什么是图片延迟加载？ 首次打开网页的时候，不是一次性的把所有的图片都加载完成，只是先加载可视区域（首屏）的内容，其他屏的内容当出现在可视区域的时候才加载，也称之为延后加载/懒加载/缓存加载
2. 延迟加载的作用：1）加快网站打开的速度；2）可以减轻服务器的负担

**二十.JS编程：同步和异步**

1. 同步（js大部分都是同步的）：必须从上到下，一件件做事情；当前事情没有干完，无法开始下个事情

**循环都是同步编程**

1. 异步：可以同时干好多事情；后面的事情做完回过头，再执行之前既定的事情；如果后面的事情没有做完，无论前面的事情是否完成，都不会被执行

**事件、定时器、AJAX、回调函数都是异步编程**

**二十一.运动的两种思路**

1. 不设定时间，通过step++实现，当前位置=上一次位置+step，需要判断边界值
2. 设定总时间，通过time++实现，当前位置=c\*t/d+b

**二十二.事件**

1. 什么是事件？一件事情或一个行为，不管是否给其绑定方法，只要当前行为触发，那么事件就会被触发
2. 事件绑定、事件监听：
3. 事件对象：事件触发的时候，相关的方法都会被依次执行，不仅仅执行，浏览器还默认给每一个方法都传递一个参数值，这个参数值就是‘事件对象’;但是IE6-8浏览器中并没有给方法传递事件对象，只能通过window.Event来获取

事件对象分为两类：鼠标事件对象（MouseEvent）: 是用来存储当前鼠标本次操作相关的一些信息，是鼠标事件（MouseEvent）的一个实例

键盘事件对象（KeyboardEvent）

1. 关于事件对象在标准浏览器和IE6-8下的常用兼容问题总结
2. clientX/client:当前鼠标操做的点距离当前屏幕窗口左上角（可视区域左上角）的X/Y轴坐标
3. type：存储的是当前操作的事件类型

以下IE6-8都会存在兼容问题

1. pageX/pageY:当前鼠标操作的点距离body左上角的X/Y轴
2. target：事件源，当前鼠标在谁身上操作的，事件源就是谁；IE6-8下不支持target，需要用srcElement来代替
3. preventDefault:阻止默认行为；IE6-8需要用ev.returnValue来代替
4. ev.stopPropagation:阻止冒泡传播；IE6-8需要要cancelBubble;

**二十三.冒泡和捕获**

1.事件的冒泡传播：当前元素的某一个行为被触发，那么它所有父级元素的相关行为都会被触发，这种的事件的传播机制叫‘冒泡传播’

**二十四.DOM0级事件**

1. DOM0事件绑定原理：给元素对象私有的某一个事件属性附一个值（一个函数值），当元素的事件触发的时候，找到对应的属性值，并让其执行；
2. DOM0事件绑定只能给元素的某一个事件绑定一次方法，绑定多次，最后绑定的方法会把前面所有绑定的方法都覆盖；
3. DOM0级事件：
   1. onabort;
   2. onautocomplete;//自动补齐
   3. onautocompleteeerror;//
   4. onbeforcopy;//
   5. onbeforcut;//
   6. onbeforepaste//
   7. onbeforunload;//
   8. onblur;//文本框失去焦点
   9. onfocus;//文本框获取焦点
   10. oncancel
   11. oncanplay;
   12. oncanplaythrough;
   13. onchange
   14. onclick;//点击
   15. onclose;
   16. oncontextmenu;
   17. oncopy;
   18. oncuechange;
   19. oncut;
   20. ondblclick//双击（不兼容）
   21. ondrag
   22. ondragend
   23. onderagenter
   24. ondragleave;
   25. ondragover
   26. ondragstart
   27. ondrop;
   28. ondurationchange;
   29. onemptied
   30. onerror;
   31. onhashchange;
   32. oninput;
   33. oninvalid;
   34. onkeydown;//键盘按下
   35. onkeypress;//长按
   36. onkeyup;//键盘抬起
   37. onlanguagechange;
   38. onload;//加载完成
   39. onloadeddate;
   40. onloadedmetadata;
   41. onloadstart;
   42. onmessage;
   43. onmousedown;//鼠标按下
   44. onmouseenter//鼠标进入盒子
   45. onmouseleave;//鼠标离开
   46. onmousemove;//鼠标在盒子上移动
   47. onmouseout;//鼠标划出盒子
   48. onmouseover;//鼠标划上盒子
   49. onmousewheel;//鼠标滚轮滚动
   50. onffline;
   51. oninline;
   52. inpagehide;
   53. onpageshow;
   54. onpaste;
   55. onpause;
   56. onplay;
   57. onplaying;
   58. onpopstate;
   59. onprogress;
   60. onratechange;
   61. onreset;
   62. onresize;//当前浏览器窗口改变
   63. onscroll;//滚动条滚都动
   64. onsearch;
   65. onseeked;
   66. onseeking;
   67. onselect;
   68. onselectstart;
   69. onshow;
   70. onstalled;
   71. onstorage;
   72. onsubmit;
   73. onsuspend;
   74. ontimeupdata;
   75. ontoggle;
   76. onunload;
   77. onvolumechange;
   78. onwaitingl
   79. onwebkitfullscreenchange;
   80. onwebkitfullscreenerror;
   81. onwheel;

**二十五.DOM2级事件**

1. DOM2级事件绑定原理：我们把需要给当前元素的某一个事件绑定的方法都放在内置的‘事件池’中，当事件触发的时候，会把事件池中对应的所有方法依次的去执行；
2. DOM2事件绑定可以给当前元素的某一个事件行为绑定多个不同的方法，当事件触发的时候，会到对应的事件池中把绑定的方法依次执行
3. DOM2和DOM0的绑定是可以共存；
4. DOM2事件绑定中，我们一般绑定的都是实名函数，只有这样以后移除的时候才知道具体要移除谁，绑定匿名函数后期是无法移除的；
5. DOM2事件绑定在标准浏览器和IE6-8的兼容问题总结：
6. 绑定所用的方法及语法是不一样的 document.body.addEventListener(‘click’,function(){},false)

document.body.removeEventListener(‘click’,function(){},false)

document.body.attachEvent(‘onclick’,function(){})

document.body.detachEvent(‘onclick’,function(){})

1. ‘this’问题：事件触发，执行对应的每一个方法的时候，标准浏览器下方法中的this是当前的元素，IE6-8下方法中的this是window
2. ‘重复问题’：标准浏览器下，当前元素某一个事件绑定的所有方法不能重复，如果绑定过就不再重复的绑定了，所以最后执行的时候，只执行一次；IE6-8没有实现去重，哪怕方法重复了，也会给当前元素绑定上，执行的时候绑定几次就执行几次
3. ‘顺序问题’：标准浏览器下方法执行的顺序是按照绑定的顺序依次执行的，但是在IE6-8下执行的顺序和绑定的顺序是没关系的，是混乱的；
4. 常用的DOM2级事件
   1. addEventListener
   2. removeEventListener;
   3. DOMContenLoaded

**二十五.回调函数**

1. 把一个函数当做一个参数的值，传递给另一个函数，在另一个函数中，把传递进来的这个函数执行一次或多次；

**二十六.柯理化函数**

1. 柯理化函数是一种编程思想，体现出JS的预处理机制

**二十七.window.onload=function（）{}&&$(document).ready(function(){})的区别**

1. **window.onload=**：当前的HTML结构、图片、文字等所有资源都加载完成，才会触发这个事件；在同一个页面中只能使用一次；
   * + 1. window.onload采用的是DOM0级事件绑定，只能绑定一次，后面绑定的方法会把前面的覆盖掉；
       2. load事件本身就是资源都加载完成才触发
2. **$(document).ready(function(){})===$(function(){}):**当前的HTML结构完成就会触发，可以在一个页面中使用多次

1）jQuery中的这个方法采用的是DOM2级事件绑定，所以可以使用多次，并且它采用的是DOM2中独有的事件进行绑定的：DOMContentLoaded

**问答？**

1. 选项卡的实现方法？2种
2. 求数组中的最大值和最小值？3.5种方法
3. 任意数求和？2种
4. 冒泡排序、插入排序、快速排序、去重，思想
5. 数据绑定，点击表格正负排序
6. 正则部分：1）验证有效数2）18到65之间的数字3）验证身份证4）验证中文姓名5)密码
7. 让盒子在页面水平和垂直都居中的三种方式，各种方式利弊 +外加一种css3中新增的flex伸缩盒模型实现居中，但是兼容性特别差