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
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；私有变量（形参和带var关键字的变量）
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；
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**
20. 自执行函数里的this，永远都是window
21. 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
22. 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
23. 构造函数中，this指向的是实例
24. 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
25. **堆内存和栈内存的释放**
26. 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
27. 栈内存释放：
28. 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
29. 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；
30. 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
31. **对象、类、实例**
32. 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
33. 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
34. 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**
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

**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，知识函数比较特殊，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

——以上这三种角色之间相互独立，没有关系

1. 所有的函数都是Function类的实例；所有的对象都是Object的实例；所以：

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

**十三.call和apply**

1. call和apply方法运行的时候：1）让”. ”前面的方法里的this关键字变成第一个实参

2）让”.”前面的方法运行

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，对象里的属性必须以””括起来，单引号都不行
2. Window天生有个属性是JSON，属性值是一个对象，这个对象里有两种方法：
3. parse()；把JSON格式的字符串转化成JSON格式的对象
4. stringify()；把JSON格式的对象转化成JSON格式的字符串
5. IE6-7中Window没有JSON属性，因此没有办法转化

**十六.AJAX实现的步骤**

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

var xhr=new XMLHttpRequest

1. 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. 请求的状态码：xhr.readyState
6. 响应的状态码：xhr.status 200,203,204,205….都表示响应成功
7. 把后台返回的JSON格式的字符串转换成JSON对象
8. 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

{n}：出现n次

{n,}: 至少出现n次

{n,m}: 至少出现n次，最多出现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

[^a-z]：除了a-z之间其他的任意字符

( )：分组的作用，能改变优先级

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