一、基础javascript篇

**1. get请求传参长度的误区**

误区：我们经常说get请求参数的大小存在限制，而post请求的参数大小是无限制的。

实际上HTTP 协议从未规定 GET/POST 的请求长度限制是多少。对get请求参数的限制是来源与浏览器或web服务器，浏览器

或web服务器限制了url的长度。为了明确这个概念，我们必须再次强调下面几点:

\* HTTP 协议 未规定 GET 和POST的长度限制

\* GET的最大长度显示是因为 浏览器和 web服务器限制了 URI的长度

\* 不同的浏览器和WEB服务器，限制的最大长度不一样

\* 要支持IE，则最大长度为2083byte，若只支持Chrome，则最大长度 8182byte

====================================================================================

**2. 补充get和post请求在缓存方面的区别**

post/get的请求区别，具体不再赘述。

补充补充一个get和post在缓存方面的区别：

\*get请求类似于查找的过程，用户获取数据，可以不用每次都与数据库连接，所以可以使用缓存。

\*post不同，post做的一般是修改和删除的工作，所以必须与数据库交互，所以不能使用缓存。因此get请求适合于请求缓存。

====================================================================================

**3. 闭包**

一句话可以概括：闭包就是能够读取其他函数内部变量的函数，或者子函数在外调用，子函数所在的父函数的作用域不会被释放。

====================================================================================

**4. 类的创建和继承**

(1)类的创建（es5）：new一个function，在这个function的prototype里面增加属性和方法。

下面来创建一个Animal类：

// 定义一个动物类

Function Animal(name) {

// 属性

this.name = name ||'Animal';

// 实例方法

this.sleep = function(){

console.log(this.name + '正在睡觉！');

}

}

// 原型方法

Animal.prototype.eat = function(food) {

console.log(this.name + '正在吃：' + food);

};

//这样就生成了一个Animal类，实力化生成对象后，有方法和属性。

(2)类的继承——原型链继承

//原型链继承

function Cat(){

}

Cat.prototype = new Animal();

Cat.prototype.name = 'cat';

//&emsp;Test Code

var cat = new Cat();

console.log(cat.name);

console.log(cat.eat('fish'));

console.log(cat.sleep());

console.log(cat instanceof Animal);

//true

console.log(cat instanceof Cat);

//true

介绍：在这里我们可以看到new了一个空对象,这个空对象指向Animal并且Cat.prototype指向了这个空对象，

这种就是基于原型链的继承。

特点：基于原型链，既是父类的实例，也是子类的实例

缺点：无法实现多继承

(3)构造继承：使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）

function Cat(name) {

Animal.call(this);

this.name = name || 'Tom';

}

// Test Code

var cat = new Cat();

console.log(cat.name);

console.log(cat.sleep());

console.log(cat instanceof Animal);

// false

console.log(cat instanceof Cat);

// true

特点：可以实现多继承

缺点：只能继承父类实例的属性和方法，不能继承原型上的属性和方法。

(4)实例继承和拷贝继承

实例继承：为父类实例添加新特性，作为子类实例返回

拷贝继承：拷贝父类元素上的属性和方法

上述两个实用性不强，不一一举例。

(5)组合继承：相当于构造继承和原型链继承的组合体。通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用

function Cat(name){

Animal.call(this);

this.name = name || 'Tom';

}

Cat.prototype = new Animal();

Cat.prototype.

constructor = Cat;

// Test Code

var cat = new Cat();

console.log(cat.name);

console.log(cat.sleep());

console.log(cat instanceof Animal);

// true

console.log(cat instanceof Cat);

// true

(6)寄生组合继承：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实例方法/属性

function Cat(name){

Animal.call(this);

this.name = name || 'Tom';

}

(function(){

// 创建一个没有实例方法的类

var Super =function (){};

Super.prototype = Animal.prototype;

//将实例作为子类的原型

Cat.prototype = new Super();

})();

// Test Code

var cat = new Cat();

console.log(cat.name);

console.log(cat.sleep());

console.log(cat instanceof Animal);

// true

console.log(cat instanceof Cat);

//true

较为推荐

====================================================================================

**5. 如何解决异步回调地狱**

promise、generator、async/await

详细情况自己了解

====================================================================================

**6. 说说前端中的事件流**

HTML中与javascript交互是通过事件驱动来实现的，例如鼠标点击事件onclick、页面的滚动事件onscroll等等，

可以向文档或者文档中的元素添加事件侦听器来预订事件。想要知道这些事件是在什么时候进行调用的，

就需要了解一下“事件流”的概念。 什么是事件流：事件流描述的是从页面中接收事件的顺序,DOM2级事件流包括下面几个阶段。

\* 事件捕获阶段

\* 处于目标阶段

\* 事件冒泡阶段

addEventListener：addEventListener 是DOM2 级事件新增的指定事件处理程序的操作，这个方法接收3个参数：

要处理的事件名、作为事件处理程序的函数和一个布尔值。最后这个布尔值参数如果是true，表示在捕获阶段调用事件处理程序；

如果是false，表示在冒泡阶段调用事件处理程序。 IE只支持事件冒泡。

====================================================================================

**7. 如何让事件先冒泡后捕获**

在DOM标准事件模型中，是先捕获后冒泡。但是如果要实现先冒泡后捕获的效果，对于同一个事件，监听捕获和冒泡，

分别对应相应的处理函数，监听到捕获事件，先暂缓执行，直到冒泡事件被捕获后再执行捕获之间。

====================================================================================

**8. 事件委托**

简介：事件委托指的是，不在事件的发生地（直接dom）上设置监听函数，而是在其父元素上设置监听函数，通过事件冒泡，

父元素可以监听到子元素上事件的触发，通过判断事件发生元素DOM的类型，来做出不同的响应。

举例：最经典的就是ul和li标签的事件监听，比如我们在添加事件时候，采用事件委托机制，不会在li标签上直接添加，

而是在ul父元素上添加。

好处：比较合适动态元素的绑定，新添加的子元素也会有监听函数，也可以有事件触发机制。

====================================================================================

**9. 图片的懒加载和预加载**

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。

懒加载：懒加载的主要目的是作为服务器前端的优化，减少请求数或延迟请求数。

两种技术的本质：两者的行为是相反的，一个是提前加载，一个是迟缓甚至不加载。 懒加载对服务器前端有一定的缓解压力作用，

预加载则会增加服务器前端压力。

====================================================================================

**10. mouseover和mouseenter的区别**

mouseover：当鼠标移入元素或其子元素都会触发事件，所以有一个重复触发，冒泡的过程。对应的移除事件是mouseout

mouseenter：当鼠标移除元素本身（不包含元素的子元素）会触发事件，也就是不会冒泡，对应的移除事件是mouseleave

====================================================================================

**11. js的new操作符做了哪些事情**

new 操作符新建了一个空对象，这个对象原型指向构造函数的prototype，执行构造函数后返回这个对象。

new执行四部曲

1.new运算符执行函数的时候，首先函数内部悄悄创建一个空对象 {}

2.让函数内部的this指向这个空对象 {} === this

3.继续往下执行函数体语句 this === {xxx}

4.返回这个对象，不需要写return

====================================================================================

**12.改变函数内部this指针的指向函数（bind，apply，call的区别）**

通过apply和call改变函数的this指向，他们两个函数的第一个参数都是一样的表示要改变指向的那个对象，

第二个参数，apply是数组，而call则是arg1,arg2...这种形式。

通过bind改变this作用域会返回一个新的函数，这个函数不会马上执行。

====================================================================================

**13. js的各种位置，比如clientHeight,scrollHeight,offsetHeight ,以及scrollTop, offsetTop,clientTop的区别？**

clientHeight：表示的是可视区域的高度，不包含border和滚动条

offsetHeight：表示可视区域的高度，包含了border和滚动条

scrollHeight：表示了所有区域的高度，包含了因为滚动被隐藏的部分。

clientTop：表示边框border的厚度，在未指定的情况下一般为0

scrollTop：滚动后被隐藏的高度，获取对象相对于由offsetParent属性指定的父坐标(css定位的元素或body元素)距离顶端的高度。

====================================================================================

**14. js拖拽功能的实现**

首先是三个事件，分别是mousedown，mousemove，mouseup

当鼠标点击按下的时候，需要一个tag标识此时已经按下，可以执行mousemove里面的具体方法。

clientX，clientY标识的是鼠标的坐标，分别标识横坐标和纵坐标，并且我们用offsetX和offsetY来表示元素的元素的初始坐标，

移动的举例应该是：鼠标移动时候的坐标-鼠标按下去时候的坐标。

也就是说定位信息为：鼠标移动时候的坐标-鼠标按下去时候的坐标+元素初始情况下的offetLeft.

还有一点也是原理性的东西，也就是拖拽的同时是绝对定位，我们改变的是绝对定位条件下的left 以及top等等值。

补充：也可以通过html5的拖放（Drag 和 drop）来实现。