1. JavaScript的基本类型有几种?

JavaScript 中分为七种内置类型，七种内置类型又分为两大类型：基本类型和对象（Object）

基本类型有六种： null，undefined，boolean，number，string，symbol

2. 使用 typeof bar === "object" 来确定 bar 是否是对象的潜在陷阱是什么？如何避免这个陷阱?

尽管 typeof bar === "object" 是检查 bar 是否对象的可靠方法，但在JavaScript中 null 也被认为是对象！所 以可用以下方法进行判断：

```
(bar !== null) && (typeof bar === "object")
```

如果bar是一个函数,也会得到false ,所以我们改进一下：

```
(bar !== null) && ((typeof bar === "object") || (typeof bar === “function"))
```

如果bar是数组的话会得到true,但我们大多数情况希望数组判断为false,所以最终的代码为：

```
(bar !== null) && (typeof bar === "object") && (toString.call(bar) !== "[object Array]")
```

3. 请写出下面运算结果

```
[]+[]           //""(空字符串)
[]+{}           //"[object object]"
{}+[]           //0
{}+{}           //"[object object][object object](firefox下是NaN)
true + true      //2
1+{a:1}         //"1[object object]"
```

4. 请描述一下new一个对象的过程

新生成了一个对象 ==> 链接到原型 ==> 绑定 this ==> 返回新对象

```
//代码实现
function create() {
    let obj = new Object()                          // 创建一个空的对象
    let Con = [].shift.call(arguments)               // 获得构造函数
    obj.__proto__ = Con.prototype                    // 链接到原型
    let result = Con.apply(obj, arguments)           // 绑定 this，执行构造函数
    return typeof result === 'object' ? result : obj   // 确保 new 出来的是个对象
}
```

5. 介绍一下 JavaScript 原型，原型链，它们有何特点?

每个对象都会在其内部初始化一个属性，就是prototype(原型)，当我们访问一个对象的属性时，如果这个对象内部不存在这个属性，那么他就会去prototype里找这个属性，这个prototype又会有自己的prototype， 于是就这样一直找下去，也就是我们平时所说的原型链的概念。

特点：JavaScript对象是通过引用来传递的，我们创建的每个新对象实体中并没有一份属于自己的原型副本，当我们修改原型时，与之相关的对象也会继承这一改变。

6. JavaScript 如何实现继承?

既然要实现继承，那么首先我们得有一个父类，代码如下：

```
// 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);
};
```

1.原型链继承

```
function Cat(){
}
Cat.prototype = new Animal();
Cat.prototype.name = 'cat';
```

2.构造继承

```
function Cat(name){
  Animal.call(this);
  this.name = name || 'Tom';
}
```

3.实例继承

```
function Cat(name){
  var instance = new Animal();
  instance.name = name || 'Tom';
  return instance;
}
```

4.拷贝继承

```
function Cat(name){
  var animal = new Animal();
  for(var p in animal){
    Cat.prototype[p] = animal[p];
  }
  Cat.prototype.name = name || 'Tom';
}
```

7. JavaScript如何判断函数是 new 调用还是普通调用

第一种方式：通过 instanceof 判断

```
function Person() {
  if(this instanceof arguments.callee) {
    console.log('new 调用');
  }else {
    console.log('普通调用');
  }
}
let p1 = new Person();            // new 调用
let p2 = Person();              // 函数调用
```

第二种方式：通过 constructor

```
function Person() {
  if(this.constructor === arguments.callee) {
    console.log('new 调用');
  }else {
    console.log('普通调用');
  }
}

let p1 = new Person();            // new 调用
let p2 = Person();              // 函数调用
```

8. 以下代码的结果是什么？请解释你的答案。

```
var fullname = 'John Doe';
var obj = {
   fullname: 'Colin Ihrig',
   prop: {
      fullname: 'Aurelio De Rosa',
      getFullname: function() {
         return this.fullname;
      }
   }
};

console.log(obj.prop.getFullname());
```

```
var test = obj.prop.getFullname;

console.log(test());
//结果是 Aurelio De Rosa 和 John Doe
//原因是，JavaScript中关键字this所引用的是函数上下文，取决于函数是如何调用的，而不是怎么被定义的。
```

9. 以下代码执行结果是什么?

```
    alert(a);
    a();
    var a=3;
    function a(){
      alert(a);
      alert(1);
    }
    alert(a);
    a=6;
    a();
```

第一次弹出a函数源代码

第二次弹出a函数源代码

第三次弹出1

第四次弹出3

第五次报错