目　录

第二部分

1. JavaScript

1.1. 简要描述 JavaScript 的数据类型？

参考答案：

JavaScript 的数据类型可以分为原始类型和对象类型。

原始类型包括 string、number 和 boolean 三种。其中，字符串是使用一对单引号或者一对双引号括起来的任意文本；而数值类型都采用 64 位浮点格式存储，不区分整数和小数；布尔（逻辑）只能有两个值：true 或 false。

复杂类型指其他对象，如 Array、Date、Object 等。

除此之外，JavaScript 中还有两个特殊的原始值：null（空）和 undefined（未定义），它们代表了各自特殊类型的唯一成员。

1.2. 读代码，写结果

写出下列表达式的计算结果：

```
var a = [];
var b = a;
b[0] = 1;
console.log(a[0]);
console.log(b[0]);
console.log(a===b);
```

参考答案：

上述代码的结果分别为：

```
1
1
true
```

将对象赋值给变量，仅仅是赋值的引用，对象本身并没有赋值一次，因此，变量 a 和 b 指向同一个数组

1.3. 简要描述 null 和 undefined 的区别

参考答案：

null 是 JavaScript 的关键字，用于描述“空值”，对其执行 typeof 操作，返回“object”，即为一个特殊的对象值，可以表示数字、字符串和对象是“无值”的。

undefined：是预定义的全局变量，其值为“未定义”，它是变量的一种取值，表示变量没有初始化。当查询对象属性、数组元素的值时，如果返回 undefined 则表示属性或者元素不存在；如果函数没有任何返回值，也返回 undefined。

需要注意的是，虽然 null 和 undfined 是不同的，但是因为都表示“值的空缺”，两者可以互换。因此，使用“==”认为二者是相等的，需要使用“===”来区分它们。

1.4. 读代码，写结果

写出下列表达式的计算结果：

```
10 + "objects"
"7" * "4"
1 - "x"
1 + {}
true + true
2 + null
```

参考答案：

上述代码的结果分别为：

```
10objects           //转换为字符串
28                  //转换为字符
NaN                 //无法转换为数值进行计算，因此返回 NaN
1[object Ojbect]    //返回对象的 toString()结果，按照字符串相加
2                   //bool 类型转换为数值类型
2                   //null 转换为数值 0
```

1.5. 读代码，写结果

写出下列代码的输出结果：

```
var a = 2;
var obj = { x: 1, y: { z: 2} };
var n = [obj,3,[4,5]];
console.log(a << 2);
console.log(obj["y"].z);
console.log(n[0].y["z"]);
console.log(n[2]["1"]);
delete n[0];
console.log(n[0].x);
```

参考答案：

上述代码的输出结果分别为：

```
8
2
2
5
console.log(n[0].x);此行代码错误，无法输出有效信息，会输出 Error 信息
```

1.6. 阅读如下代码：

```
   var x = 10;
   var y = 20;
   var z = x < y ? x++ : ++y;
```

上述代码运行后，变量 x、y 和 z 的值为多少？

参考答案：

上述代码运行后，变量 x 的值为 11；变量 y 的值为 20；变量 z 的值为 10。

这是因为，运行第三行代码时，只执行？后的第一个语句，因此，y 的值不发生变化，仍为 20，并返回 x 的值赋值给变量 z，因此 z 的值为 10，然后将 x 的值增加 1，变为 11。

1.7. 什么是“逻辑短路”？

参考答案：

逻辑短路是对于逻辑运算而言，是指，仅计算逻辑表达式中的一部分便能确定结果，而不对整个表达式进行计算的现象。

对于“&&”运算符，当第一个操作数为 false 时，将不会判断第二个操作数，因为此时无论第二个操作数为何，最后的运算结果一定是 false；

对于“||”运算符，当第一个操作数为 true 时，将不会判断第二个操作数，因为此时无论第二个操作数为何，最后的运算结果一定是 true。

1.8. 阅读如下代码

```
var empAge = 20;
var empName;
if (empAge > 20 && empName.length > 0) {
   console.log(1);
}
else {
   console.log(2);
}
```

上述代码运行后，将产生有效输出，还是代码错误？

参考答案：

上述代码运行，会输出 2，而不会发生错误。

这是因为， if 条件中的逻辑表达式的第一个条件（empAge > 20）不满足，会返回 false，此时，将发生“逻辑短路”，而不会继续判断下一个条件。因此，即使下一个条件中的变量 empName 没有赋值，此时如果计算 empName.length 将发生错误；但是，因为发生了“逻辑短路”，不会计算此表达式，因此不会发生错误。

既然 if 条件中的逻辑表达式返回 false，则运行 else 语句：输出 2。

1.9. 解释一下 JavaScript 中的局部变量与全局变量的区别

参考答案：

全局变量拥有全局作用域，在 JavaScript 代码的任何地方都可以访问；在函数内声明的变量只在函数体内有定义，即为局部变量，其作用域是局部性的。

需要注意的是，在函数体内声明局部变量时，如果不使用 var 关键字，则将声明全局变量。

1.10. 读代码，写结果

写出下列代码的输出结果：

```
var x = "global";
function test() {
var x = "local";
return x;
}
console.log(test());
```

参考答案：

上述代码的输出结果为 local。这是因为，在函数 test() 体内，局部变量的优先级高于同名的全局变量。因此，如果在函数体内声明的局部变量和全局变量重名，则以局部变量优先。因此，调用函数 test() 时，返回的是局部变量 x，其值为 local。

1.11. 什么是 JavaScript 中的函数作用域

参考答案：

JavaScript 中的函数作用域是指：变量在声明它的函数体以及这个函数体嵌套的任意函数体内都是有定义的。这意味着，在函数体内声明的所有变量在整个函数体内始终是可见的，这种特性也被称为“声明提前”，即，函数内声明的所有变量（不涉及到赋值）都被提前至函数的顶部声明。

比如，查看如下代码：

```
function test() {
   console.log(x);
   var x = 10;
   console.log(x);
}
test();
```

上述代码运行，将先输出 undefined，再输出 10。

这是因为，虽然变量 x 在第二行声明并赋值，但其有效范围为整个函数体，因此，第一行代码输出时，表示变量 x 已经声明但是没有赋值，因此输出 undefined；第三行代码运行时，因为变量 x 已经赋值为 10，则输出 10。

1.12. 读代码，写结果

写出下列代码的输出结果：

```
function test() {
   var sum = 0;
   for (var i = 0; i < 10; i++) {
      sum += i;
   }
   console.log(sum);
   console.log(i);
}
test();
```

参考答案：

上述代码中，输出 sum 的值为 45；输出 i 的值为 10。

这是因为，在函数 test() 体内，循环计算完毕后，变量 sum 的值为从 0 累加到 10 的和，即 45，变量 i 虽然是在 for 循环中声明，但是在整个函数体内都有效（函数作用域），因此，循环完毕后，变量 i 的值为 10。

1.13. 读代码，写结果

写出下列代码的输出结果：

```
var x = "global";
function test() {
   console.log(x);
   var x = "local";
   console.log(x);
}
test();
```

参考答案：

上述代码中，先输出 undefined，再输出 local。

函数 test() 体内声明了与全局变量同名的局部变量 x，则将覆盖全局变量，即局部变量优先。因此，第一次输出变量 x 时，为输出局部变量 x，此时变量 x 只有声明而没有赋值，因此输出 undefined；第二次输出变量 x 时，局部变量 x 已经赋值，因此输出字符串 local。

1.14. 简述 arguments 对象的作用

参考答案：

在函数代码中，使用特殊对象 arguments 可以访问函数的参数。即，开发者在定义函数时，无需明确的为方法声明参数，也可以在方法体中使用 arguments 来访问参数。这是因为， arguments 是一种特殊对象，在函数代码中，表示函数的参数数组。

正因为 arguments 表示参数组成的数组，因此，首先可以使用 arguments.length 检测函数的参数个数，其次，可以通过下标（arguments[index]）来访问某个参数。这样，可以用 arguments 对象判断传递给函数的参数个数并获取参数，适用于函数参数无法确定个数的情况下。

1.15. 简要描述 JavaScript 中定义函数的几种方式

参考答案：

JavaScript 中，有三种定义函数的方式：

1、函数语句：即使用 function 关键字显式定义函数。如：

```
function f(x){
    return x+1;
}
```

2、函数定义表达式：也称为“函数直接量”。形如：

```
var f = function(x){return x+1;};
```

3、使用 Function() 构造函数定义，形如：

```
Var f = new Function(“x”,”return x+1;”);
```

1.16. 读代码，写结果

写出下列代码的输出结果：

```
var f = function (x) { return x * x; }
```

```
console.log(f);
console.log(f(10));
```

参考答案：

上述代码运行时，先输出 function (x) { return x * x; }；再输出 100。

这是因为，变量 f 代表一个函数对象，因此直接输出变量时，将输出函数体对应的字符文本；f（10）表示调用变量 f 所对应的函数，返回 100 并输出。

1.17. 阅读如下代码

```
function f() {
   console.log("function");
}
function test() {
   console.log(f);
   f();
   f = "hello";
   console.log(f);
   f();
}
test();
```

上述代码运行后，输出结果为？

参考答案：

上述代码运行，会先输出

```
function f() {
    console.log("function");
};
```

然后输出 function；再输出 hello；然后会输出异常信息：string is not a function。

这是因为，定义函数时，函数名称作为一个全局变量，该变量指向函数对象。因此，执行函数 test 中的第一行代码时，将输出变量 f 所对应的函数对象，即输出函数体中代码的字符串形式；然后执行 f（）表示调用方法 f，则输出字符串 “function”；执行代码 f = “hello” 意味着将变量 f 的值改为字符串 因此再输出变量 f 时 将输出字符串“Hello”；而如果试图再执行 f（），会引发错误。这是因为，此时，变量 f 不再是一个函数对象，而是一个普通字符串。

1.18. 列举几个 JavaScript 中常用的全局函数，并描述其作用

参考答案

JavaScript 中常用的全局函数及其作用如下：

1. parseInt：解析一个字符串并返回一个整数；

2. parseFloat：解析一个字符串并返回一个浮点数；
3. isNaN：检查某个值是否是数字，返回 true 或者 false；
4. encodeURI ：把字符串作为 URI 进行编码；
5. decodeURI ：对 encodeURI() 函数编码过的 URI 进行解码；
6. eval：计算某个字符串，以得到结果，或者用于执行其中的 JavaScript 代码。

1.19. 阅读如下代码

```
function printArray(arr) {
    for (var i in arr) {
        if (arr[i] instanceof Array) {
            printArray(arr[i]);
        } else {
            document.write(arr[i] + ' ');
        }
    }
}
var data = [1, [20, 21], [[301, 302], [310, 311]]];
printArray(data);
```

上述代码运行后，页面的输出结果为？

参考答案：

上述代码运行，会在页面输出：1 20 21 301 302 310 311。
函数 printArray 使用了递归方式，逐一输出数组中的每个成员，中间以空格隔开。

1.20. 编写函数，实现冒泡排序

参考答案：
使用 JavaScript 编写的冒泡排序函数如下所示：

```
function bubbleSort(arr) {
    for (var i = 0; i < arr.length; i++) {
        for (var j = 0; j < arr.length - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                var temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

测试函数 bubbleSort，代码如下：

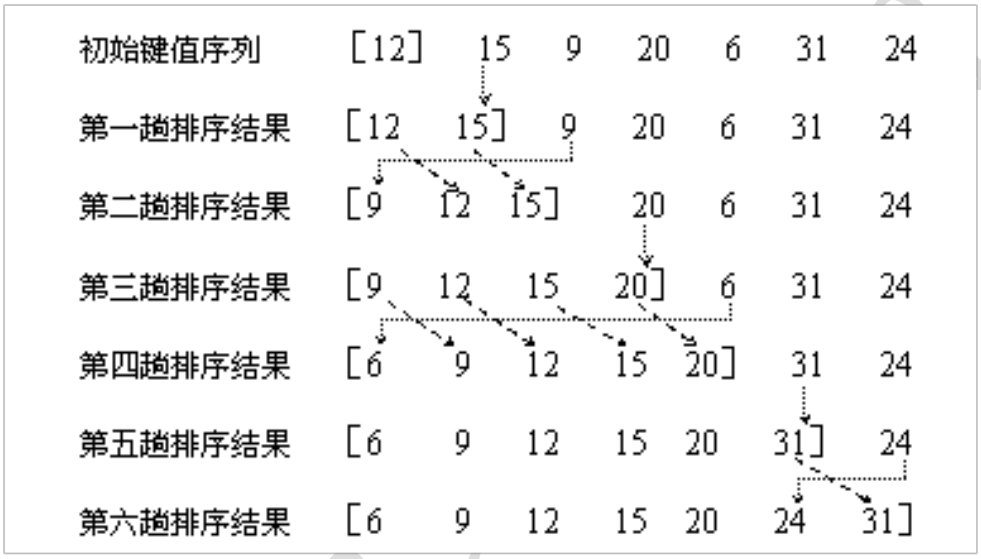
```
var arr = [12, 4, 9, 21, 43, 3];
bubbleSort(arr);
console.log(arr);
```

上述代码运行时，将输出排序后的结果：[3, 4, 9, 12, 21, 43]。

1.21. 编写函数，实现插入排序

参考答案：

插入排序是指，先假定将 n 个元素的数列分为已有序和无序两个部分；然后将无序数列的第一个元素与有序数列的元素从后往前逐个进行比较，找出插入位置，将该元素插入到有序数列的合适位置中。过程如下图所示：



使用 JavaScript 编写的插入排序函数如下所示：

```
function insertionSort(arr) {
    //从第二个元素开始
    for (var i = 1; i < arr.length; i++) {
        // 取出待比较的元素
        var k = arr[i];
        // 向前找，找到比当前元素大的位置
        var j;
        for (j = i - 1; j >= 0 && k < arr[j]; j--) {
            //向后移动一位
            arr[j + 1] = arr[j];
        }
        // 插入元素
        arr[j + 1] = k;
    }
}
```

测试函数 insertionSort，代码如下：

```
var arr = [12, 4, 9, 21, 43, 3];
insertionSort(arr);
console.log(arr);
```

上述代码运行时，将输出排序后的结果：[3, 4, 9, 12, 21, 43]。

1.22. 编写函数，实现对身份证号码最后一位的验证

二代身份证号码为 18 位，其最后一位（第 18 位）的计算方法为：

1、 将前面的身份证号码 17 位数分别乘以不同的系数。从第一位到第十七位的系数分别为：

7 - 9 - 10 - 5 - 8 - 4 - 2 - 1 - 6 - 3 - 7 - 9 - 10 - 5 - 8 - 4 - 2

2、 将这 17 位数字和系数相乘的结果相加

3、 用加出来和除以 11，看余数是多少？

4、 余数只可能有 0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 这 11 个数字。每个数字所对应的最后一位身份证的号码为：1 - 0 - X - 9 - 8 - 7 - 6 - 5 - 4 - 3 - 2。即，如果余数是 2，就会在身份证的第 18 位数字上出现罗马数字的 X。如果余数是 10，身份证的最后一位号码就是 2。

例如：某男性的身份证号码是 34052419800101001X。验证其最后一位是否正确时，首先需要得出前 17 位的乘积和是 189，然后用 189 除以 11 得出的结果是 17+2/11，也就是说其余数是 2。最后通过对应规则就可以知道余数 2 对应的数字是 x。所以，可以判定此身份证号码的最后一位是合格的。

参考答案：

编写验证方法如下：

```
//验证方法
function verifyCode(id){
  if(id.length !=18 )
    return false;
  /*1、从第一位到第十七位的系数分别为：
        7,9,10,5,8,4,2,1,6,3,7,9,10,5,8,4,2
        将这 17 位数字和系数相乘的结果相加。    */
  var arr = [7,9,10,5,8,4,2,1,6,3,7,9,10,5,8,4,2];
  var sum = 0;
  for(var i=0; i<arr.length; i++){
    sum += parseInt(id.charAt(i)) * arr[i];
  }
  //2、用加出来和除以 11，看余数，
  var c = sum%11;
  //3、分别对应的最后一位身份证的号码为：1 - 0 - X - 9 - 8 - 7 - 6 - 5 - 4 - 3 - 2
  var ch = ['1', '0', 'X', '9', '8', '7', '6', '5', '4', '3', '2'];
  var code = ch[c];
  var last = id.charAt(17);
  last = last=='x' ? 'X': last;
  return last == code;
}
```

测试该方法：

```
var id = "340524198001010O1X";
console.log(verifyCode(id));
```

1.23. 读代码，写结果

写出下列代码的输出结果：

```
var arr1 = [10, 20];
arr1.push(30);
arr1.push([40, 50]);
var data = arr1.pop();
console.log(data);
```

参考答案：

上述代码的输出结果为 [40, 50]。

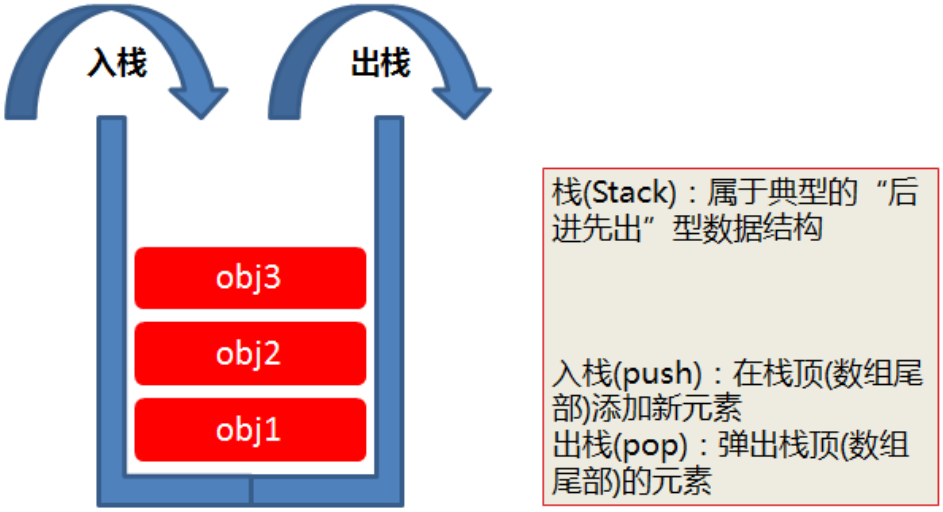
数组的方法 push() 表示入栈，即在栈顶（数组尾端）添加指定的元素；方法 pop() 表示出栈，删除并返回栈顶（数组尾端）的元素。

代码中，第一次入栈为数字 30；第二次入栈为数组 [40,50]，且该数组排在栈顶。因此，调用方法 pop() 时，将删除并返回栈顶元素 [40,50]，这是一个数组，因此输出结果为[40, 50]。

1.24. 什么是栈？在 JavaScript 中，如何模拟栈操作？

参考答案：

栈（stack）是一种运算受限的线性表，其限制是仅允许在表的一端进行插入和删除运算。这一端被称为栈顶，相对的把另一端称为栈底。向一个栈插入新元素又称作进栈、入栈或压栈，它是把新元素放到栈顶元素的上面，使之成为新的栈顶元素；从一个栈删除元素又称作出栈或退栈，它是把栈顶元素删除掉，使其相邻的元素成为新的栈顶元素。栈的操作如下图所示：



在 JavaScript 中，可以使用数组及其相关操作来模拟栈操作。首先，使用数组存储一列元素，然后使用数组的 push() 方法在数组的尾部添加指定的元素，类似于在栈顶添加元素，即顶部入；然后使用数组的 pop() 删除并返回数组尾部的元素，类似于顶部出栈，即后入的元素先出。

1.25. 读代码，写结果

写出下列代码的输出结果：

```
var arr1 = [10, 20, 30, 40];
arr1.push(50);
arr1.shift();
console.log(arr1);
```

参考答案：

上述代码的输出结果为[20, 30, 40, 50]。

数组的方法 push() 表示入栈，即在栈顶（数组尾端）添加指定的元素，因此，数字 50 将作为数组的最后一个元素；方法 shift() 表示删除并返回栈底（数组头部）的元素，因此，将从数组删除数值 10。此时，输出数组，将输出剩余的 4 个数值，即[20, 30, 40, 50]。

1.26. 什么是正则表达式？在 JavaScript 中，如何应用正则表达式？

参考答案：

正则表达式(Regular Expression) 本身就是一个字符串，由一些普通字符和特殊字符组成的，用以描述一种特定的字符规则的表达式。

正则表达式常用于在一段文本中搜索、匹配或替换特定形式的文本。如：词语出现频率

统计、验证字符串是否符合邮箱格式、屏蔽一篇帖子中的限制性词语等。许多程序设计语言都支持利用正则表达式进行字符串操作。

在 JavaScript 中，正则表达式的应用分为两种：

1、结合 String 对象的 replace、search 和 match 方法，实现对字符串的替换、查找和匹配；

2、定义正则表达式对象，实现对字符串的复杂匹配操作。

1.27. 读代码，写结果

写出下列代码的输出结果：

```
var regexp = /\bdo\b/ig;
var data = 'He does told to Do,do.';
console.log(data.search(regexp));
```

参考答案：

上述代码的输出结果为 3。

String 的 search(regexp) 方法，用于返回第一次出现匹配指定正则表达式的下标，若没有匹配则返回-1。

试题中,正则表达式 \bdo\b 表示匹配完整的单词 do,且不区分大小写。而变量 data 中，第一次出现单词 do（不区分大小写）的位置为 16。

1.28. 阅读如下代码

```
function add(num) {
    try {
        num = Number(num);
        if (isNaN(num)) {
            throw new Error('Arguments is NaN');
        }
        console.log('try block end');
    } catch (e) {
        console.log('catch block');
        return;
    } finally {
        console.log('finally block');
    }
    console.log('function end');
}

add('10x');
```

上述代码运行后，输出结果为？

参考答案：

上述代码运行，会先输出 catch block；再输出 finally block。

这是因为，执行代码 num = Number(num); 时，因为传入的参数值为字符串“10x”，无法转换为 number 类型，则产生错误，运行到 catch 语句块中，输出“catch bloc”；而 finally 块始终会运行，因此继续输出“finally block”。程序发生异常后，将退出，因此不再执行其他语句。

1.29. 简要描述 JavaScript 中的匿名函数

参考答案：

匿名函数是指在定义时没有指定名字的函数，且定义后往往直接调用。如：

```
function(num1, num2){
 console.log( num1 + num2 );
}
```

这种方式所定义的匿名函数，往往需要直接调用，如：

```
   (function (num1, num2) {
      console.log(num1 + num2);
   })(10,20);
```

匿名函数常用于定义不需要重复使用的函数，用完即释放。另外，对于直接调用的匿名函数而言，可以看成是一个临时的命名空间，其区域内定义的所有变量，不会污染到全局命名空间。

1.30. 简要描述 JavaScript 中的作用域链

参考答案：

任何一段 JavaScript 代码都对应一个作用域链，作用域链中存放一系列对象，代码中声明的变量将作为对象的属性存放。

在 JavaScript 的顶层代码中，作用域链由一个全局对象组成；当定义一个函数时，它保存一个作用域链，作用域链上有两个对象，一个是函数对象，一个是全局对象。

每当一个函数被调用时，会创建一个活动对象（也叫上下文对象），函数中的局部变量将作为该对象的属性存放。

当需要使用一个变量时，将从作用域链中逐个查找对象的属性。比如：要使用变量 a，将先查找作用域中的第一个对象是否有属性 a，如果有就使用；如果没有就查找作用域链中下一个对象的属性，以此类推。如果作用域链上没有任何一个对象含有属性 x，则认为这段代码的作用域链上不存在 x，将抛出引用错误异常。

当函数调用完成后，如果没有其他引用指向为此次调用所创建的上下文对象，该对象将被回收。

1.31. 阅读如下代码

```
var a = 100;
function f() {
    var a = 200;
    function g() {
        return a;
    }
    return g;
}
console.log(f()());
```

上述代码运行后，输出结果为？

参考答案：

上述代码运行，会输出 200。

上述代码意味着，先执行 f()，得到该函数内嵌套的一个函数对象 g，然后调用这个嵌套的函数。

定义函数 f 时，它保存一个作用域链，作用域链上有两个对象，一个是函数对象，一个是全局对象，此时，全局变量 a 是全局对象的属性，和函数对象 f 无关。

执行函数 f 时，会创建一个活动对象，其中保存变量 a 作为函数 f 的属性而存在。而函数 g 嵌套定义在函数 f 里，对其访问变量 a 时，它没有变量 a，则继续往下查找函数 f。找到函数 f 的属性 a 并返回。因此，输出结果为 200。

这种函数将变量包裹起来，外部代码可以通过内嵌函数 g 来访问函数 f 内的局部变量的方式，也称为闭包。

1.32. 简要描述你对闭包的理解

参考答案：

函数对象可以通过作用域链相互关联起来，函数体内部的变量都可以保存在函数作用域内，这种特性称为闭包。这意味着函数变量可以隐藏于作用域链之内，看起来好像是函数将变量包裹了起来。这种方式常用于共享函数内的私有变量。

闭包有如下应用特征：

1、局部变量：在函数中定义有共享意义（如：缓存、计数器等）的局部变量（注：定义成全局变量会对外造成污染）；

2、内嵌函数：在函数中声明有内嵌函数，内嵌函数对函数中的局部变量进行访问；

3、外部使用：函数向外返回此内嵌函数，外部可以通过此内嵌函数持有并访问声明在函数中的局部变量，而此变量在外部是通过其他途径无法访问的。

1.33. 阅读如下代码

```
var n = 10;
function counter() {
    var n = 0;
    var g = function () {
                    return ++n;
              };
    return g;
 }

 var c1 = counter();
 console.log(c1());
 console.log(c1());
 console.log(c1());
```

上述代码运行后的输出为？

参考答案：

上述代码运行后，分别输出数字 1、2、3。

首先从应用上分析，这是一个闭包的典型应用：模拟计数器。调用一次函数 counter，则得到一个计数器，即代码中的变量 c1，用于统计被调用的次数；每运行一次计数器 c1，则访问次数加 1。因此分别输出数字 1、2、3。

其次从代码原理来分析：函数 counter 返回其内嵌函数 g，该函数中包裹了函数 counter 的局部变量 n，其初始值为 0；每次运行函数 g，均访问相应的局部变量 n，从而实现累加计数的效果。

1.34. 阅读如下代码

```
for (var i = 0; i < 3; i++) {
       setTimeout(function () { alert(i); }, 3000);
}
```

上述代码希望实现：3s 后弹出三次警告对话框，对话框中显示的数字分别为 0、1、2。问，上述代码能否实现所需要的效果？如果不能，应该如何实现？

参考答案：

上述代码不能实现所需要的效果。会在 3s 后弹出三次警告对话框，对话框中的数字均为 3。

这是因为，循环结束后，变量 i 的值为 3；当 3s 后运行循环中内嵌的函数，执行语句 alert(i) 时，访问的是循环中的变量 i。因此，三次弹出均为数值 3。

如果希望实现分别弹出数字 0、1、2，需要将代码修改如下：

```
for (var i = 0; i < 3; i++) {
   (function (n) {
      setTimeout(function () { alert(n); }, 3000);
   })(i);
}
```

修改后的代码中，使用匿名函数封装一次性定时器的代码，并在调用匿名函数时，将变量 i 的值作为参数传入。每次循环中，调用一次匿名函数，则启动一个定时器对象，将相应的数字传入。3s 后执行函数时，不再访问循环中的局部变量 i，则会依次弹出 0、1、2。

1.35. 简述 JavaScript 中创建自定义对象的方式

参考答案：

自定义对象（user-defined object）指由用户创建的对象，兼容性问题需要由编写者注意。创建自定义对象的方式有：

1、对象直接量

2、new Object()

3、function 对象模板

4、Object.create()

1.36. 查看如下 JavaScript 代码：

```
var p = new User("mary");
alert(p.name);          //弹出 mary
p.introduce();          //弹出 i am mary
```

为使上段代码正确运行，请定义对象 User。

参考答案：

需要使用 function 模板创建对象 User，代码如下：

```
function User(name) {
    this.name = name;
    this.introduce = function () {
       alert("i am " + this.name);
    };
 }
```

1.37. JavaScript 中，this 关键字的作用是什么？

参考答案：

笼统的说，关键字 this 指向当前对象。比如，顶级代码中的 this 指向全局对象；在指定元素事件的时候，this 指定当前发生事件的元素对象。

对于嵌套函数，如果嵌套函数作为方法被调用，其 this 指向调用它的对象；如果作为函数调用，this 是全局对象或者为 undefined（严格模式下）。

1.38. 查看如下 JavaScript 代码：

```
var o = {
    m: function () {
        console.log(this);
        f();

        function f() {
            console.log(this);
        }
    }
};
o.m();
```

请描述上述代码的运行结果。

参考答案：

上述代码在浏览器中运行后，第一行的输出结果类似于：Object {m: function}，表示 this 关键字指向调用它的当前对象；第二行的输出结果为 Window，表示 this 关键字指向全局变量。

这是因为，调用对象 o 的方法 m，在该方法中直接使用 this 关键字，则指向调用它的对象；然后运行嵌套函数 f 时，this 指向全局对象，在浏览器中运行时，全局对象为 Window 对象。需要注意的是，如果使用严格模式，则第二行会输出 undefined。

1.39. 查看如下 JavaScript 代码：

```
var a = 10;
var foo = {
    a: 20,
    bar: function () { var a = 30; return this.a }
}
console.log(foo.bar());
console.log((foo.bar = foo.bar)());
```

上述代码运行后，输出结果是？

参考答案：

上述代码运行后，先输出 20。这是因为调用对象 foo 的 bar() 方法时，this 关键字指向调用它的对象，即 foo，因此，输出该对象中所定义的属性值。

第二行将输出 10。这是属于函数调用，此时，非严格模式下 this 指代全局对象 window，因此 a 的值为 10。

1.40. 简述你对 JavaScript 中原型的理解

参考答案：

在 JavaScript 中，函数本身就是一个包含了方法和属性的对象。每个函数中都有一个 prototype 属性，该属性所存储的就是原型对象。

原型对象用来保存共享属性和方法，可以通过原型来实现为对象扩展属性，实现继承。

1.41. 查看如下 JavaScript 代码：

```
function Emp(ename, salary) {
   this.ename = ename;
   this.salary = salary;
   this.toString = function(){
      return this.ename + ":" + this.salary;
   };
}
var emp1 = new Emp("mary",3500);
var emp2 = new Emp("john", 5500);
Emp.prototype.hireDate = "2015/05/01";
delete emp1.ename;
delete Emp.prototype.hireDate;
console.log(emp1.toString() + ":" + emp1.hireDate);
console.log(emp2.toString() + ":" + emp2.hireDate);
```

上述代码运行后，输出结果是？

参考答案：

第一行输出为：undefined:3500:undefined

第二行输出为：john:5500:undefined

JavaScript 中，每个函数都有一个 prototype 属性，该属性引用着该函数的原型对象，原型对象中包含着当前函数所有实例共享的方法和属性。

使用函数的 prototype 属性可以向函数的原型对象添加或删除属性和方法，并且这些属性和方法是这一类对象所共享的。

可以使用 delete 关键字删除对象的属性，包括自有属性和原型属性。

1.42. 简要描述 JavaScript 中的自有属性和原型属性

参考答案：

自有属性是指，通过对象的引用添加的属性，此时，其它对象可能无此属性。对于自有属性，是各个对象所特有的、彼此独立的属性。比如：

```
emp1.job = 'Coder';
```

原型属性是指从原型对象中继承来的属性，一旦原型对象中属性值改变，所有继承自该原型的对象属性均改变。比如：

```
Emp.prototype.dept = '研发部';
```

当需要检测对象的自有属性时，可以使用 hasOwnProperty()方法。另，还可以使用 in 操作检测对象及其原型链中是否具备指定属性。

需要注意的是，在检测对象属性时，先检测自有属性，再检测原型属性。

1.43. 查看如下 JavaScript 代码：

```
function Hero() {
    this.name = "unknown";
}

Hero.prototype.age = 20;

var hero = new Hero();
console.log('name' in hero);
console.log(hero.hasOwnProperty('name'));
console.log('age' in hero);
console.log(hero.hasOwnProperty('age'));
hero.age = 30;
console.log(hero.hasOwnProperty('age'));
```

上述代码运行后，输出结果是？

参考答案：

输出结果分别为 true、true、true 、false 和 true。

因为在构造函数中定义了属性 name，因此变量 hero 拥有自有属性 name。因此在对 hero 用 in 操作符检测 name 属性时，先检测 name 属性为自有属性还是原型属性，则输出 true；使用 hasOwnProperty()进行判断也是输出 true。

使用函数的 prototype 向函数的原型添加属性时，添加的属性 age 为原型属性。因此，下面代码：

```
console.log('age' in hero);
console.log(hero.hasOwnProperty('age'));
```

使用 in 操作符判断时，先判断是否为自有属性，如果没有，则在原型链中查找属性 age，因此第一行输出 true；但是 age 并非自有属性，所以第二行输出 false。

此时，再添加代码：

```
hero.age = 30;
```

这是为 hero 定义了同名的自有属性 age，其值为 30。因此，此时再检测 hero 对象是否有自有属性 age 时，将输出 true。

1.44. 简要描述你对原型链的理解

参考答案：

每个函数中都有 prototype 属性，该函数被 new 操作符用于创建对象。将一个函数的 prototype 属性指向某个对象，由此形成了一条链，称之为原型链。

可以使用 isPrototypeOf()方法判定一个 prototype 对象是否存在于另一个对象的原型链中。如果是，返回 true，否则返回 false。

1.45. 简要描述 JavaScript 中的继承

参考答案：

在 JavaScript 中，继承都是源于原型，有多种实现方式。比如：

1. 修改构造函数的原型，为该构造函数创建的对象指定统一的父级对象。代码如下：

```
B.prototype = new A();
```

2. 只继承于原型（尽可能地将可重用的属性和方法添加到原型中），代码如下：

```
B.prototype = A.prototype;
```

3. 单独修改一个对象的原型，而不影响其他对象的原型。代码如下：

```
var b1 = new B();
Object.setPrototypeOf(b1, new A());
```

4. 修改构造函数，这将影响使用该构造函数创建的所有对象。代码如下：

```
function B() {
    Object.setPrototypeOf(this, new A());
}
```

1.46. 查看如下 JavaScript 代码：

```
function Hero() {
    this.name = "unknown";
}

Hero.prototype.name = "Caesar";

var hero = new Hero();
console.log(hero.name);

delete hero.name;
console.log(hero.name);

delete Hero.prototype.name;
console.log(hero.name);
```

上述代码运行后，输出结果是？

参考答案：

上述代码运行后，首先输出 unknown；然后输出 Caesar；最后输出 undefined。

查找对象属性时，先查找自有属性，因此，第一次输出的是 hero 的自有属性 name 的值，该属性的值从构造函数中继承而来，即为“unknown “；删除 hero 的自有属性后，再试图输出 name 属性时，则查找其原型链中的属性 name 的值并输出，即为“Caesar”；继续删除 hero 的原型属性后，则输出“undefined”。

1.47. 查看如下 JavaScript 代码：

```
function A() {
   this.name = "a";
   this.toString = function () { return this.name };
}
function B() {
   this.name = "b";
}

var b1 = new B();
console.log(b1.toString());

B.prototype = new A();
var b2 = new B();
console.log(b2.toString());
```

上述代码运行后，输出结果是？

参考答案：

上述代码运行后，首先输出[object Object]；然后输出 b。

这是因为，虽然修改了 B 的原型，但是，只影响修改之后的对象。因此，对象 b1 依然使用默认的从 Object 继承而来的 toString() 方法，因此输出对象的类型和名称。

对象 b2 是在修改了 B 的原型之后创建的，因此将使用 A 中所定义的 toString() 方法，将输出字符“b”。

1.48. 简要描述 Call 和 apply 的区别

参考答案：

call()和 apply()都用于间接调用函数。

call 方法用于调用一个对象的一个方法，并以另一个对象替换当前对象。即，任何函数可以作为任何对象的方法来调用，哪怕这个函数并非那个对象的方法。语法如：

```
call([thisObj[,arg1[, arg2[,   [,.argN]]]]])
```

其中，第一个参数 thisObj 要调用函数的上下文，即将被用作当前对象的对象。其他参

数为可选参数，表示将被传递方法参数序列。

apply()和 call()在作用上是相同的 ,但两者在参数上有区别的。它俩的第一个参数相同，不同的是第二个参数。对于 apply()，第二个参数是一个参数数组，也就是将多个参数组合成为一个数组传入。如：

```
func.call(func1,var1,var2,var3)
func.apply(func1,[var1,var2,var3])
```

1.49. 查看如下 JavaScript 代码：

```
function A() {
    this.name = "a";
    this.introduce = function () { console.log("My name is " + this.name) };
}
function B() {
    this.name = "b";
}
var a1 = new A();
var b1 = new B();
a1.introduce.call(b1);
```

上述代码运行后，输出结果是？

参考答案：

上述代码运行后，输出 My name is b。

代码 a1.introduce.call(b1); 表示对于对象 b1 调用 a1 的 introduce() 方法。

1.50. 查看如下 JavaScript 代码：

```
var a = b = { n: 1 };
a.x = a = { n: 2 };
alert(a.x);
alert(b.x.n);
```

上述代码运行后，输出结果是？

参考答案：

上述代码运行后，先弹出显示 undefined，再弹出显示 2。

第一行代码运行后，变量 a 和 b 都指向对象{n:1}，第二行代码运行，先为对象 {n:1} 添加新属性 x ,其值为 {n:2}。此时 ,对象{n:1}形如 { n:1,x:{n:2} }。然后创建新对象 {n:2}，并将变量 a 指向这个新对象，而变量 b 依然指向原有对象。

因此，a.x 将输出 undefined，因为此时新对象中只有属性 n 没有属性 x；b.x.n 将表示向原有对象的中的新属性，因此输出 2。

2. DOM

2.1. 谈谈 innerHTML、nodeValue 与 textContent 之间的区别

参考答案：

innerHTML 属性读取或设置节点起始和结束标签中的 HTML 内容；

nodeValue 属性读取或设置指定节点的文本内容，适用于文本类型的节点；

textContent 属性读取或设置指定节点的文本内容，对于元素节点而言，会返回所包含的所有子节点中的文本内容的组合。

2.2. DOM 操作中，如何获取元素的属性值？

参考答案：

对于元素节点，获取其某属性的值有多种方式，如下所示：

1、element.attributes[下标].value

2、element.attributes['属性名'].value

3、element.getAttributeNode('属性名').value

4、element.getAttribute('属性名')

2.3. 简要描述 DOM 操作中查找元素的方式

参考答案：

1、 通过 HTML 中的信息选取元素，比如：

a) getElementById() 方法：根据元素的 id 属性值查询单个节点；

b) getElementsByTagName() 方法：根据元素标签的名称查询节点；

c) getElementsByName() 方法：根据元素 name 属性的值查询节点 。

2、 通过 CSS 类选取元素

a) getElementsByClassName('className')方法：根据 class 名称选取元素；

b) querySelector('selector') 和 querySelectorAll('selector')方法：根据 CSS 选择器选取元素。

3、 通过 document 对象选取，如 document.all、document.body 等；

4、 通过节点遍历选取节点，如 parentNode、firstChild 等。

2.4. 为 html 页面上的一个按钮添加 onclick 事件处理，有几种方法？

参考答案：

1、 直接在 HTML 代码中添加，如：

```
<input type=”button” onclick=”funcA();” />
```

其中，funcA() 为一个有效函数。

2、 在 js 代码中添加，如：

```
btn.onclick = funcA;
```

或者

```
btn.onclick = function(){};
```

其中，btn 表示按钮对象。

3、 定义监听函数，代码如：

```
btn.addEventListener('click', function( ){ });
```

2.5. 简述 window 对象除 document 以外的一些常用子对象，并描述其作用？

参考答案：

window 对象有很多子对象，除了 document 以外，还有如下常用子对象：

- screen 对象：此对象包含有关客户端显示屏幕的信息，常用于获取屏幕的分辨率和色彩；
- history 对象：此对象包含用户（在浏览器窗口中）访问过的 URL；
- location 对象：此对象包含有关当前 URL 的信息，常用于获取和改变当前浏览的网址；
- navigator 对象：此对象包含有关浏览器的信息，常用于获取客户端浏览器和操作系统信息；
- event 对象：任何事件触发后将会产生一个 event 对象，该对象记录事件发生时的鼠标位置、键盘按键状态和触发对象等信息。

2.6. 查看如下代码：

```
<html>
```

```
    <head>
       <title>bubble</title>
       <script type="text/javascript" language="javascript">
          function clickP(e) {
             var target = e.target || e.srcElement;
             alert("clickP, target=" + target.nodeName);
          }
          function clickDIV(e) {
             var target = e.target || e.srcElement;
             alert("clickDIV, target=" + target.nodeName);
          }
       </script>
    </head>
    <body>
        <div onclick="clickDIV(event);">
           <p onclick="clickP(event);">ClickMe</p>
        </div>
    </body>
</html>
```

该代码在浏览器中运行时，点击段落中的文本“ClickMe ”，页面效果为？

参考答案：

先弹出“clickP, target=P”，再弹出“clickDIV, target=P”。

单击段落中的文本“ClickMe”，先触发 <P> 元素的 onclick 事件，且触发事件的元素为段落 P ,因此 ,先弹出“clickP, target=P”;然后事件冒泡 ,触发 元素的 onclick 事件，再弹出“clickDIV, target=P”。

2.7. 实现每隔一秒钟弹出一个对话框，且此弹出过程持续 5 秒钟

参考答案：

代码如下所示：

```
var id = setInterval(function(){alert("Hello JavaScript");},1000)
setTimeout(function(){clearInterval(id);},5*1000)
```

2.8. 为页面动态添加按钮

使用 js 代码为页面动态添加 5 个按钮 ,每个按钮上的文本为“button1”、“button2”…“button5”。单击每个按钮时，分别弹出数字 1、2…5。代码如下：

```
for (var i = 1; i < 6; i++) {
   var input = document.createElement('input');
               （空白处）
   document.body.appendChild(input);
}
```

为空白处填上代码，实现所需功能

参考答案：

代码如下所示：

```
for (var i = 1; i < 6; i++) {
    var input = document.createElement('input');
    input.setAttribute("type", "button");
    input.setAttribute("value", "button" + i);

    (function (n) {
        input.onclick = function () { alert(n); };
    })(i);

    document.body.appendChild(input);
}
```

2.9. 文本框验证（考虑浏览器兼容性）

页面有文本框，需要限制该文本框中只能录入数字，即：如果用户按下数字以外的其他键，文本框中无法录入；如果用户粘贴进非数字字符，也需要进行过滤。

请设计 HTML 代码以及 JS 代码，实现上述功能。

参考答案：

首先，需要为文本框定义 onkeypress 事件，用于在录入文本时做出判断；并为文本框定义 onkyeup 事件，用于过滤粘贴进来的文本。HTML 代码如下所示：

```
<h2>输入框中只能输入数字</h2>
请输入您的年龄：
     <input onkeypress="digitOnly(event)" onkeyup="filterChar(this)"/>
```

JS 代码如下所示：

```
//当按键按下时，阻止非数字字符的输入
function digitOnly(event){
 var code = event.keyCode || event.which; //ie||firefox
 if((code<48)||(code>57)){
     if(event.preventDefault){  //firefox
         event.preventDefault();
     }else{                     //ie
         event.returnValue = false;
     }
 }
}
//当按键弹起时，剔除输入中的非中文字符——解决复制粘贴进来的非数字
function filterChar(input){
 input.value = input.value.replace(/[^0-9]/g,'');
}
```

2.10. 总结可以实现页面跳转和刷新的方法

参考答案：

1、 使用超级链接，代码如：

```
<a href=”url”></a>
```

2、 表单提交，代码如：

```
<form action=”url”></form>
```

3、 JS 代码，代码如：

```
location.href=”url”;
location.assign(‘url’);
location.replace();
location.reload();
window.open(‘url’);
history.go();
```