JavaScript进阶

---JS函数及函数参数

---JS函数对象

---JS预解析

河北师范大学软件学院
Software College of Hebei Normal University

JavaScript进阶

---JS函数及函数参数

河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- **函数的定义与调用**
- **函数参数的数量问题**
- **参数类型与传递方式（值、引用）**

函数的定义与调用

- 函数定义方式（3种）
  - 通过函数声明的形式来定义（要有函数名）
  - 通过函数表达式的形式来定义（可以是没有函数名的匿名函数，有名的话方便调用栈追踪）
  - 通过Function构造函数实例化的形式来定义（JS中函数也是对象，函数对象）

```
function max(a,b){
    return a>b?a:b;
}
max(2,3);//3
```

```
var max = function(a,b){
    return a>b?a:b;
};
max(2,3);//3
```

```
var max = new Function("a","b","return a>b?a:b;");
max(2,3);//3
```


参见实例demo01 匿名函数与非匿名函数对比
查看Node.js读取文件和配置函数对象实例

函数的定义与调用

- 函数调用方式（4种）
  - 作为函数直接调用（非严格模式下this为全局对象，严格模式下为undefined）
  - 作为方法调用（this为调用此方法的对象）
  - 通过call( )和apply( )间接调用（this为函数对象的call/apply方法的首个参数，移花接木）
  - 作为构造函数调用（this为实例化出来的对象）

```
function test(){
    console.log(this);
}
test();//window
```

```
var obj = {
    x:0,
    test:function(){
        console.log(this.x);
    }
};
obj.test();//0
```


参见实例demo02 Part1 函数嵌套情况下的调用

函数的定义与调用

- 函数调用方式（4种）
  - 作为函数直接调用（this指向全局对象）
  - 作为方法调用（this指向调用此方法的对象）
  - 通过call( )和apply( )间接调用（this为函数对象的call/apply方法的首个参数，移花接木）
  - 作为构造函数调用（this指向实例化出来的对象）

```
objA = {name:"AA"};
objB = {name:"BB"};
objA.foo = function(){
    console.log(this.name);
};
objA.foo();//AA
objA.foo.call(objB);//BB
```

```
function Person(name){
    this.name = name;
}
Person.prototype.sayHi = function(){
    console.log("Hi,i'm "+this.name);
}
var p1 = new Person("Jack");
p1.sayHi();//Hi,i'm Jack
```

参见实例demo02 Part2 call/apply实例

内容提纲

- 函数的定义与调用
- **函数参数的数量问题**
- 参数类型与传递方式（值、引用）

调用参数的数量问题详解

- JS函数调用时实参数量可以与形参不一致
  - 实参数量大于形参的情况（通过函数对象属性arguments获得所有实参、类数组对象）
  - 实参数量小于形参的情况（少的参数值为undefined、可使用| |来给出默认值）

```
function test() {
    var s = "";
    for (var i = 0; i < arguments.length; i++) {
        s += arguments[i];
    }
    return s;
}
test("hello,", "world!");//"hello,world!"
```


参见实例demo03 实参数量大于形参数量

调用参数的数量问题详解

- JS函数调用时实参数量可以与形参不一致
  - 实参数量大于形参的情况（通过arguments获得所有实参、类数组对象、拥有对象属性）
  - 实参数量小于形参的情况（少的参数值为undefined、可使用| |来给出默认值）

```
var sum = function(a,b,c){
    b = b||4;
    c = c||5;
    return a+b+c;
}
console.log(sum(1,2,3));//1+2+3
console.log(sum(1,2));  //1+2+5
console.log(sum(1));    //1+4+5
```

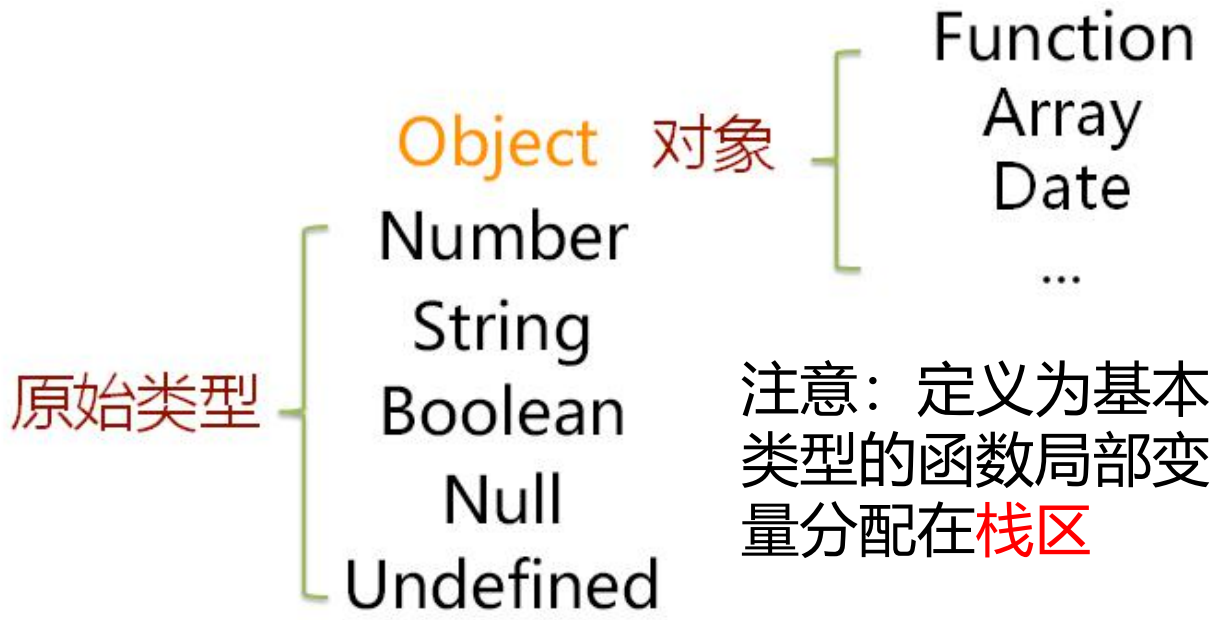

参见实例demo03 实参数量小于形参数量

内容提纲

- 函数的定义与调用
- 函数参数的数量问题
- 参数类型与传递方式（值、引用）

JavaScript数据类型-背景知识

- JavaScript（ES5）数据类型（6种）及其划分（2类）
  - 基本（原始）类型（Number、String、Boolean、Null、Undefined）
  - 引用（对象）类型（Object（Array、Function、Date、Error等））



注意：定义为引用类型的变量，其引用分配在栈区或堆区，引用的对象分配在堆区

注意：定义为基本类型的函数局部变量分配在栈区

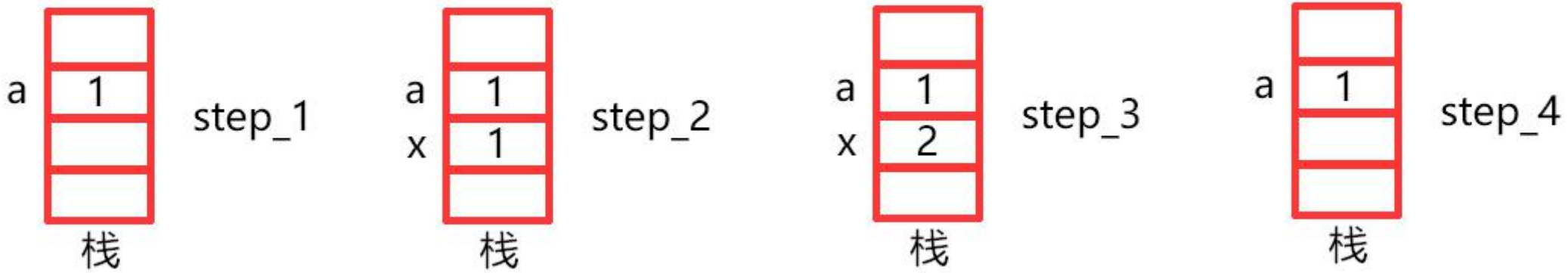
- 不同类型的数据，参数传递方式不同（值传递、引用传递）

参数类型与传递方式 - 值传递（基本数据类型的传递）

- 实参为基本数据类型时，形参改变不影响实参（值传递）

```
var a = 1;
function foo(x) {
    x = 2;  //step_2 此时a=1,x=1;
    console.log("a:",a,"x:",x); //step_3 此时a=1,x=2;
}

foo(a); //step_1 此时a=1
console.log(a); //step_4  a仍为1
```



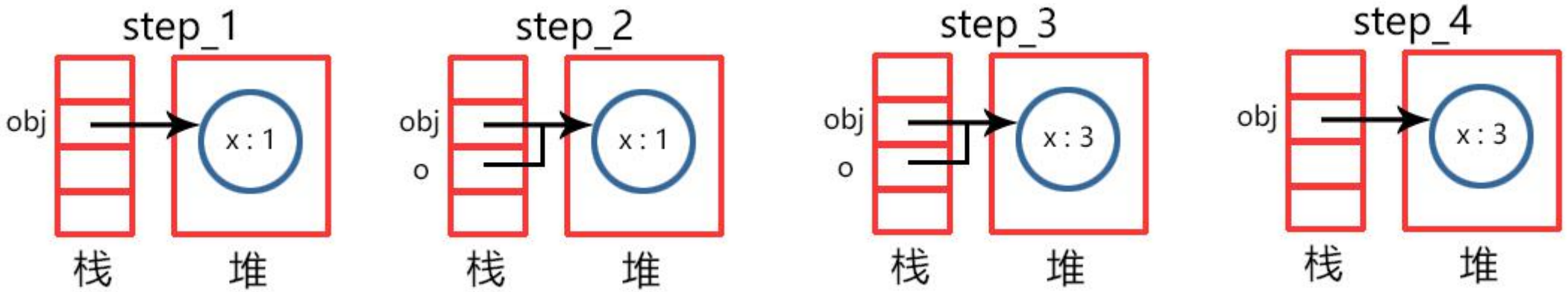
参见实例demo04 trace追踪 学会debug调试

参数类型与传递方式- 引用传递（引用数据类型的传递）

- 实参为引用类型时，形参改变影响实参（引用传递）

```
var obj = {x:1};
function fee(o){
    o.x = 3;     //step_2 此时obj.x为1，o.x为1
    console.log(obj.x,o.x); //step_3 此时obj.x为3，o.x为3
}

fee(obj);   //step_1 此时obj.x为1
console.log("obj.x :",obj.x);   //step_4 obj.x被改写为3
```



参见实例demo04 trace追踪 学会debug调试

总结

- 函数的定义与调用
- 调用参数的数量问题详解
- 参数类型与传递方式（值、引用）

Have a Break!

JavaScript进阶

---JS函数对象

河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- **函数对象**
- **函数对象的属性及方法**
- **高阶函数**

函数对象介绍

- JS中的函数也是对象
  - JS中每个函数都是作为对象来维护和运行的，即函数对象（既有属性也有方法）
  - 可以将函数（函数对象）赋值给一个变量，或将函数作为参数进行传递
  - 函数对象对应的类型是Function（类似于数组对象对应于Array、日期对象对应于Date）
  - 如果变量是函数（函数对象）时，typeof此对象，返回function，而非object
  - 内置的函数对象（Array、Function、Date等），内置的非函数对象（Math、JSON）

参见实例demo05

```
function foo(){}
console.log(foo); //function foo(){}
console.log(typeof foo); //function
console.log(foo instanceof Object); //true
console.log(foo instanceof Function); //true
console.log(foo === window.foo); //true
```

内容提纲

- 函数对象
- 函数对象的属性及方法
- 高阶函数

函数对象的属性及方法

- 函数对象的属性
  - length、arguments（隐藏的局部变量）
  - caller、callee（arguments的属性，常用于递归调用）
  - constructor、prototype

参见实例demo06 Part1函数对象属性综述

- 函数对象的方法
  - call、apply
  - bind
  - toString、valueOf

参见实例demo06 Part2 函数对象方法综述

内容提纲

- 函数对象
- 函数对象的属性及方法
- 高阶函数

高阶函数

- 高阶函数是指至少满足下列条件之一的函数
  - 函数作为参数被传递（最常见的形式：回调函数）
  - 函数作为返回值输出（与闭包有紧密联系）

```
function add(x, y, f) {
    return f(x) + f(y);
}
add(2,3,function(x){
            return x+1;
        }
);
add(2,-3,Math.abs);
add(2,3,Math.sqrt);
```

```
var obj = {x:23};
var fun1 = function () {
    return function fun2() {
        return this.x;//若改为 return this;
    }
};
obj.fun3 = fun1;
obj.fun4 = fun1();
console.log(obj.fun3());//输出什么,this是什么
console.log(obj.fun3()());//输出什么,this是什么
console.log(obj.fun4());//输出什么,this是什么
```


参见实例demo07 高阶函数Part1和Part2

Have a Break!

JavaScript进阶

---JS预解析

河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- JS解析及执行简介
- JS预解析（声明提升）
- 预解析与作用域

JS解析及执行简介

- JS脚本语言（非提前编译，由解析器边解析边执行）
  - 区别于C/C++编译成二进制和Java/C#编译成字节码（运行在跨平台虚拟机上）的解析执行
- JS代码案例（思考：是否会报错，区别于其他语言）

```
console.log(a);//undefined
var a = 2;
console.log(a);//2
```

```
//从解析器角度看到的代码
var a;
console.log(a);
a = 2;
console.log(a);
```

- JS的解析和执行过程
  - 全局预解析阶段（全局变量和函数声明前置）
  - 全局顺序执行阶段（变量赋值、函数调用等操作）
  - 当遇到函数调用时，在执行函数内代码前，进行函数范围内的预解析
  - 当存在函数嵌套时，以此类推，会进行多次函数预解析

注：解析和执行是一个不断交替的过程

内容提纲

- JS解析及执行简介
- JS预解析（声明提升）
- 预解析与作用域

JS预解析-声明提升

- 预解析主要工作（变量声明和函数声明提升）
  - 解析器在执行代码前的进行代码扫描（var、function）
  - 将变量和函数声明在当前作用域（全局、函数）内进行提升
- 变量声明提升案例

```
console.log(a);
var a = 1;
console.log(a);
```

等价于=>

```
var a;
console.log(a);//undefined
a = 1;
console.log(a);//1
```

参见实例demo08_Part1

JS预解析-声明提升

- 函数声明提升案例

```
foo();//f_2
function foo(){
    console.log("f_1");
}
function foo(){
    console.log("f_2");
}
```

等价于=>

```
function foo(){
    console.log("f_1");
}
function foo(){
    console.log("f_2");
}
foo();//f_2
```

注：ES5中函数及变量声明重复的话，相当于覆盖　参见实例demo08_Part2

JS预解析-声明提升

- 同时有var和function关键字时（情形1：函数表达式）

```
foo();//报错
var foo = function(){
    console.log("foo");
}
```

```
▶Uncaught TypeError: foo is not a function
```

- 当function前有运算符的话，认定为表达式，不提升

```
//上述代码等效如下
var foo;
foo();//报错
foo = function(){
    console.log("foo");
};
```

```
//思考以下代码是否会报错
console.log(foo);//输出什么
var foo = function(){
    console.log("foo");
};
foo();//是否会报错
```


参见实例demo08_Part3

JS预解析-声明提升

- 同时有var和function关键字时（情形2：变量名同函数名）

```
AA();
function AA(){
    console.log("AA_1");
}

var AA = function AA(){
    console.log("AA_2");
};
AA();
```

等价于=>

```
function AA(){
    console.log("AA_1");
}
var AA;//在最顶端和在这等效

AA();
AA = function AA(){
    console.log("AA_2");
};
AA();
```


参见实例demo08_Part4

内容提纲

- JS解析及执行简介
- 变量及函数声明前置
- **预解析与作用域**

JS变量作用域简介

- 变量的作用域是指变量在何处可以被访问到
  - JS采用的是静态词法作用域，代码完成后作用域链就已形成，与代码的执行顺序无关
- 全局变量与局部变量
  - 全局变量：拥有全局作用域的变量（JS代码中任何地方都可以访问）
  - 全局变量是跨域了所有函数自身作用域的自由变量，可以在函数内和函数外直接访问
  - 局部变量：函数内声明的变量，只在函数体内有定义，作用域是局部性的
  - 在函数外不能直接访问函数的局部变量
  - 函数内访问同名变量时，局部变量会覆盖全局变量
- ES5中无块作用域（ES5作用域缺陷及解决办法参见IIFE）
  - 全局作用域、函数作用域、ES5中可以使用函数立即执行表达式来模拟块作用域

参见实例demo09 作用域的案例（回顾）

JS预解析与作用域

- 声明前置与作用域的关系（全局作用域、函数作用域）

```
if(true){
    var i = 0;
}
```

等价于=>

```
var i;
if(true){
    i = 0;
}
```

```
function foo(){
    console.log("j:",j);//undefined
    var j = 10;
    console.log("j:",j);//10
}
foo();

console.log("i:",i);
console.log("j:",j);
```

等价于=>

```
function foo(){
    var j;
    console.log("j:",j);//undefined
    j = 10;
    console.log("j:",j);//10
}
foo();

console.log("i:",i);//0
console.log("j:",j);//报错
```


参见实例demo10 预解析与作用域

总结

- JS解析及执行简介
- 变量及函数声明前置
- 预解析与作用域

Thank You!

作业：

- 复习本章节课件及练习