函数进阶

目录 Contents

1. 函数的定义和调用

1.1 函数的定义方式

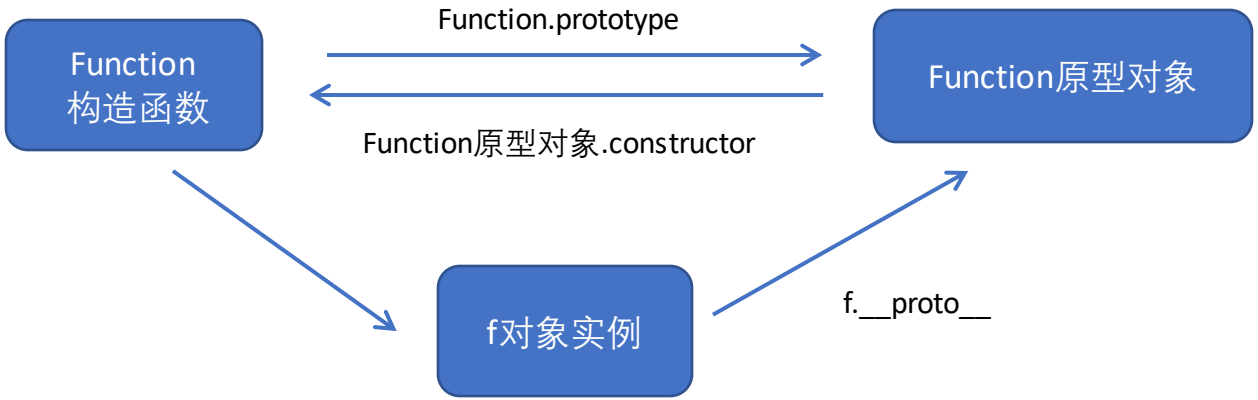
1. 函数声明方式 function 关键字 (命名函数)
2. 函数表达式 (匿名函数)
3. new Function()

```
var fn = new Function('参数1','参数2'..., '函数体')
```

- Function 里面参数都必须是字符串格式
- 第三种方式执行效率低，也不方便书写，因此较少使用
- 所有函数都是 Function 的实例(对象)
- 函数也属于对象

1. 函数的定义和调用

1.1 函数的定义方式



1. 函数的定义和调用

1.2 函数的调用方式

1. 普通函数
2. 对象的方法
3. 构造函数
4. 绑定事件函数
5. 定时器函数
6. 立即执行函数

目录 Contents

2. this

2.1 函数内 this 的指向

这些 this 的指向，是当我们调用函数的时候确定的。 调用方式的不同决定了this 的指向不同

一般指向我们的调用者.

调用方式	this指向
普通函数调用	window
构造函数调用	实例对象 原型对象里面的方法也指向实例对象
对象方法调用	该方法所属对象
事件绑定方法	绑定事件对象
定时器函数	window
立即执行函数	window

2. this

2.1 改变函数内部 this 指向

JavaScript 为我们专门提供了一些函数方法来帮我们更优雅的处理函数内部 this 的指向问题，常用的有 bind()、call()、apply() 三种方法。

1. call 方法

call() 方法调用一个对象。简单理解为调用函数的方式，但是它可以改变函数的 this 指向。

```
fun.call(thisArg, arg1, arg2, ...)
```

- thisArg：在 fun 函数运行时指定的 this 值
- arg1，arg2：传递的其他参数
- 返回值就是函数的返回值，因为它就是调用函数
- 因此当我们想改变 this 指向，同时想调用这个函数的时候，可以使用 call，比如继承

2. this

2.1 改变函数内部 this 指向

JavaScript 为我们专门提供了一些函数方法来帮我们更优雅的处理函数内部 this 的指向问题，常用的有 bind()、call()、apply() 三种方法。

2. apply 方法

apply() 方法调用一个函数。简单理解为调用函数的方式，但是它可以改变函数的 this 指向。

```
fun.apply(thisArg, [argsArray])
```

- thisArg：在fun函数运行时指定的 this 值
- argsArray：传递的值，必须包含在数组里面
- 返回值就是函数的返回值，因为它就是调用函数
- 因此 apply 主要跟数组有关系，比如使用 Math.max() 求数组的最大值

2. this

2.1 改变函数内部 this 指向

JavaScript 为我们专门提供了一些函数方法来帮我们更优雅的处理函数内部 this 的指向问题，常用的有 bind()、call()、apply() 三种方法。

3. bind 方法

bind() 方法不会调用函数。但是能改变函数内部this 指向

```
fun.bind(thisArg, arg1, arg2, ...)
```

- thisArg：在 fun 函数运行时指定的 this 值
- arg1，arg2：传递的其他参数
- 返回由指定的 this 值和初始化参数改造的原函数拷贝
- 因此当我们只是想改变 this 指向，并且不想调用这个函数的时候，可以使用 bind

2.2 call apply bind 总结

相同点:

都可以改变函数内部的this指向.

区别点:

1. call 和 apply 会调用函数, 并且改变函数内部this指向.
2. call 和 apply 传递的参数不一样, call 传递参数 aru1, aru2..形式 apply 必须数组形式[arg]
3. bind 不会调用函数, 可以改变函数内部this指向.

主要应用场景:

1. call 经常做继承.
2. apply 经常跟数组有关系. 比如借助于数学对象实现数组最大值最小值
3. bind 不调用函数,但是还想改变this指向. 比如改变定时器内部的this指向.

目录 Contents

3. 严格模式

3.1 什么是严格模式

JavaScript 除了提供正常模式外，还提供了严格模式（strict mode）。ES5 的严格模式是采用具有限制性 JavaScript 变体的一种方式，即在严格的条件下运行 JS 代码。

严格模式在 IE10 以上版本的浏览器中才会被支持，旧版本浏览器中会被忽略。

严格模式对正常的 JavaScript 语义做了一些更改：

1. 消除了 Javascript 语法的一些不合理、不严谨之处，减少了一些怪异行为。
2. 消除代码运行的一些不安全之处，保证代码运行的安全。
3. 提高编译器效率，增加运行速度。
4. 禁用了在 ECMAScript 的未来版本中可能会定义的一些语法，为未来新版本的 Javascript 做好铺垫。比如一些保留字如：class, enum, export, extends, import, super 不能做变量名

3. 严格模式

3.2 开启严格模式

严格模式可以应用到整个脚本或个别函数中。因此在使用时，我们可以将严格模式分为为脚本开启严格模式和为函数开启严格模式两种情况。

1. 为脚本开启严格模式

为整个脚本文件开启严格模式，需要在所有语句之前放一个特定语句“use strict”；(或‘use strict’;)。

```
<script>
    "use strict";
    console.log("这是严格模式。");
</script>
```

因为"use strict"加了引号，所以老版本的浏览器会把它当作一行普通字符串而忽略。

3. 严格模式

3.2 开启严格模式

严格模式可以应用到整个脚本或个别函数中。因此在使用时，我们可以将严格模式分为为脚本开启严格模式和为函数开启严格模式两种情况。

1. 为脚本开启严格模式

有的 script 基本是严格模式，有的 script 脚本是正常模式，这样不利于文件合并，所以可以将整个脚本文件放在一个立即执行的匿名函数之中。这样独立创建一个作用域而不影响其他 script 脚本文件。

```
<script>
  (function (){
       "use strict";
       var num = 10;
       function fn() {}
    })();
</script>
```

3. 严格模式

3.2 开启严格模式

严格模式可以应用到整个脚本或个别函数中。因此在使用时，我们可以将严格模式分为为脚本开启严格模式和为函数开启严格模式两种情况。

2. 为函数开启严格模式

要给某个函数开启严格模式，需要把“use strict”；(或 'use strict';) 声明放在函数体所有语句之前。

```
function fn(){
    "use strict";
    return "这是严格模式。";
}
```

将 "use strict" 放在函数体的第一行，则整个函数以 "严格模式" 运行。

3.4 严格模式中的变化

严格模式对 Javascript 的语法和行为，都做了一些改变。

1. 变量规定

① 在正常模式中，如果一个变量没有声明就赋值，默认是全局变量。严格模式禁止这种用法，变量都必须先用 var 命令声明，然后再使用。

② 严禁删除已经声明变量。例如，delete x; 语法是错误的。

3. 严格模式

3.4 严格模式中的变化

严格模式对 Javascript 的语法和行为，都做了一些改变。

2. 严格模式下 this 指向问题

① 以前在全局作用域函数中的 this 指向 window 对象。

② 严格模式下全局作用域中函数中的 this 是 undefined。

③ 以前构造函数时不加 new也可以 调用,当普通函数，this 指向全局对象

④ 严格模式下,如果 构造函数不加new调用, this 指向的是undefined 如果给他赋值则 会报错

⑤ new 实例化的构造函数指向创建的对象实例。

⑥ 定时器 this 还是指向 window 。

⑦ 事件、对象还是指向调用者。

3. 严格模式

3.4 严格模式中的变化

严格模式对 Javascript 的语法和行为，都做了一些改变。

3. 函数变化

① 函数不能有重名的参数。

② 函数必须声明在顶层.新版本的 JavaScript 会引入“块级作用域”（ES6 中已引入）。为了与新版本接轨，不允许在非函数的代码块内声明函数。

更多严格模式要求参考：https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Strict_mode

目录 Contents

4. 高阶函数

高阶函数是对其他函数进行操作的函数，它接收函数作为参数或将函数作为返回值输出。

```
<script>
function fn(callback){
  callback&&callback();
}
fn(function(){alert('hi')})
</script>
```

```
<script>
function fn(){
    return function() {}
}
 fn();
</script>
```

此时fn 就是一个高阶函数

函数也是一种数据类型，同样可以作为参数，传递给另外一个参数使用。 最典型的就是作为回调函数。

同理函数也可以作为返回值传递回来

目录 Contents

5. 闭包

5.1 变量作用域

变量根据作用域的不同分为两种：全局变量和局部变量。

1. 函数内部可以使用全局变量。

2. 函数外部不可以使用局部变量。

3. 当函数执行完毕，本作用域内的局部变量会销毁。

5. 闭包

5.2 什么是闭包

闭包（closure）指有权访问另一个函数作用域中变量的函数。 ----- JavaScript 高级程序设计

简单理解就是， 一个作用域可以访问另外一个函数内部的局部变量。

```
<script>
 function fn1(){     // fn1 就是闭包函数
       var num = 10;
       function fn2(){
           console.log(num);  // 10
       }
       fn2()
  }
  fn1();
</script>
```

5. 闭包

5.3 在 chrome 中调试闭包

1. 打开浏览器，按 F12 键启动 chrome 调试工具。
2. 设置断点。
3. 找到 Scope 选项（Scope 作用域的意思）。
4. 当我们重新刷新页面，会进入断点调试，Scope 里面会有两个参数（global 全局作用域、local 局部作用域）。
5. 当执行到 fn2() 时，Scope 里面会多一个 Closure 参数，这就表明产生了闭包。

5. 闭包

5.3 闭包的作用

提问：我们怎么能在 fn() 函数外面访问 fn() 中的局部变量 num 呢？

```
<script>
 function fn() {
    var num = 10;
    return function {
         console.log(num);  // 10
     }
  }
  var f = fn();
  f()
</script>
```

闭包作用：延伸变量的作用范围。

5. 闭包

5.5 闭包案例

1. 循环注册点击事件。

2. 循环中的 setTimeout()。

3. 计算打车价格。

5. 闭包

5.6 闭包总结

1. 闭包是什么?

闭包是一个函数 （一个作用域可以访问另外一个函数的局部变量）

2. 闭包的作用是什么?

延伸变量的作用范围

目录 Contents

6. 递归

6.1 什么是递归?

如果一个函数在内部可以调用其本身，那么这个函数就是递归函数。

简单理解:函数内部自己调用自己, 这个函数就是递归函数

递归函数的作用和循环效果一样

由于递归很容易发生“栈溢出”错误（stack overflow），所以必须要加退出条件 return。

6. 递归

6.2 利用递归求数学题

1. 求 1 * 2 *3 ... * n 阶乘。

2. 求斐波那契数列 。

3. 根据id返回对应的数据对象

6. 递归

6.3 利用递归求:根据id返回对应的数据对象

6.4 浅拷贝和深拷贝

1. 浅拷贝只是拷贝一层, 更深层次对象级别的只拷贝引用.
2. 深拷贝拷贝多层, 每一级别的数据都会拷贝.
3. Object.assign(*target*, ...*sources*)　　es6 新增方法可以浅拷贝

传智播客旗下高端IT教育品牌