# 面向对象概述

面向对象时软件开发领域中非常重要的一种编程思想，通过面向对象可以使程序的灵活性、健壮性、可重用性、可扩展性、可维护性得到提升。

## 面向过程与面向对象

学习面向对象之前，首先需要了解面向过程与面向对象的区别。

1. 以完成一件事情来说，面向过程思想注重的是具体的步骤，只有按照步骤一步一步执行，才能够完成这件事情。而面向对象思想注重的是一个个对象，这些对象各司其职，我们只要指挥这些对象帮我们完成任务。
2. 在面向过程思想中，我们编写的代码都是一些变量和函数，随着程序功能的不断增加，变量和函数就会越来越多，此时容易遇到命名冲突的问题，由于各种功能的代码交织在一起，导致代码结构混乱，变得难以理解、维护和复用。而利用面向对象思想，我们可以将同一类事物的操作代码封装成对象，将用到的变量和函数作为对象的数学和方法，然后通过对象去调用，这样可以使代码结构清晰、层次分明。因此，面向对象可以提高开发效率。

## 面向对象的特征

面向对象的特征主要可以概括为封装性、继承性和多态性。

1. 封装性

封装性指的是隐藏内部的实现细节，只对外开发操作接口。接口就是对象的方法。

1. 继承性

继承性是指一个对象继承另一个对象的成员，从而在不改变另一个对象的前提下进行扩展。

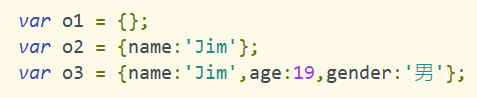
1. 多态性

多态性指的是同一操作作用于不同的对象，会产生不同的执行结果。

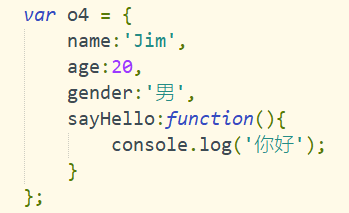
# 自定义对象

## 定义对象

在JavaScript中，使用“{ }”定义一个对象，对象的成员以键值对的形式存放在{ }中，多个成员之间使用逗号分隔。具体示例如下：



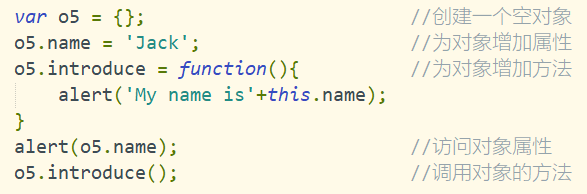
当对象的成员比较多时，为了让代码阅读更加流畅，可以对代码格式进行缩进与换行。如下：



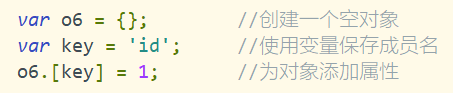
值得一提的是，使用“{ }”语法又称为对象的字面量语法，所谓字面量语法是指在源代码中直接书写一个表示数据和类型的量，如123（数值型）、‘123’（字符型）、[123]（数组）都是字面量。

## 访问对象成员

创建对象后，通过“.”可以访问对象的成员。JavaScript中对象具有动态特征，如果一个对象没有成员，用户可以手动赋值属性或方法来添加成员。具体示例如下：

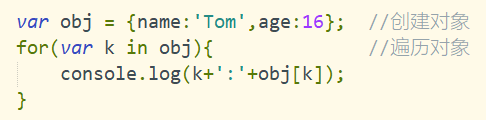


如果对象的成员名不确定时，还可以使用‘[ ]’语法实现可变成员名，即通过一个变量保存成员的名称。具体示例如下：



## 对象成员遍历

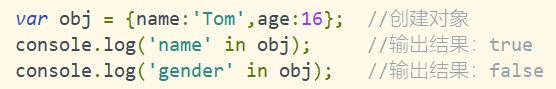
使用for…in语法不仅可以遍历数组元素，还可以遍历对象的成员。具体示例如下：



代码解读：for…in中的变量k保存了每个对象成员的名称，通过obj[k]即可访问成员的值。另外，如果对象中包含方法，则可以通过obj[k]( )进行调用。

## 判断成员存在

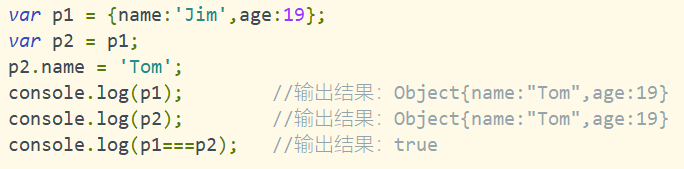
当需要判断一个对象中的某个成员是否存在时，可以使用in运算符。具体示例如下：



当成员存在时返回true，不存在时返回false。

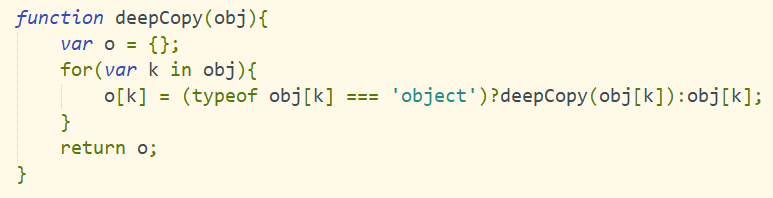
## 深拷贝与浅拷贝

拷贝（copy）是指将一个目标数据复制一份，形成两个个体。在前面的开发中，若将一个基本数据类型（数组、字符型）的变量赋值给另一个变量，就可以得到两个值相同的变量，改变其中一个变量的值，不会影响另一个变量的值（深拷贝）。但是，如果操作的目标是引用数据类型（如数组、对象），则会出现两个变量指向同一个对象的情况，如果改变其中一个对象的成员，另一个对象也会发生改变（浅拷贝）。具体示例如下：

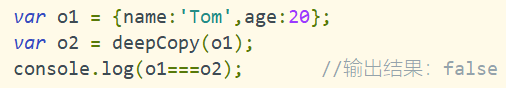


这种情况在JavaScript中称为浅拷贝（shallow copy）。在实际开发中，浅拷贝可以节省内存开销。因为一个对象可以保存大量的数据，其占用的内存会比基本数据类型高。如果没有浅拷贝机制，在将对象作为函数参数传递时，函数内部的实参就会创建对象的副本，多占用一份内存空间，尤其是进行函数嵌套调用或递归操作时，占用内存会越来越多。

与浅拷贝相对的是深拷贝（deep copy），即这正创建一个对象的副本。所要实现深拷贝的效果，可以编写代码复制对象里的成员到另一个对象。例如：



代码解读：o用来保存成员，使用for…in语句遍历传入对象的每个成员，在遍历时通过“o[k] = obj[k]”实现成员的复制。由于传入的对象obj的成员有可能还是一个对象，所以使用typeof进行了判断，如果typeof检测的类型为object（数组、对象的类型都是object），则递归调用deepCopy( )函数，进行完整的复制。测试代码如下：

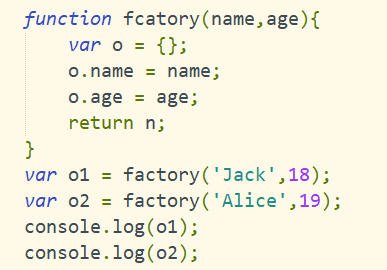


# 构造函数

构造函数是JavaScript创建对象的另一种方式。相对于字面量“{ }”的方式，构造函数可以创建出一些具有相同特征的对象。

## 为什么使用构造函数

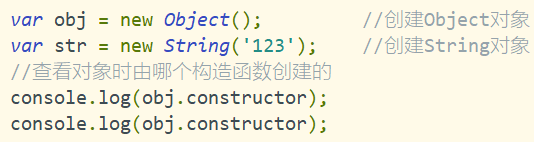
通过字面量“{}”的方式创建对象，这种方式虽然简单灵活，但是存在一些缺点。例如，当需要创建一组具有相同特征的对象时，无法通过代码指定这些对象应该具有哪些相同的成员。在以Java为代表的面向对象编程语言中，引入了类（class）的概念，用来以模板的方式构造对象。JavaScript中可以通过函数来实现相同的目的，可以创建对象的过程封装为函数，通过调用函数来创建对象。具体示例如下：



在上述示例中，我们将专门用于创建对象的factory( )函数称为工厂函数。通过工厂函数，虽然可以创建对象，但是其内部是通过字面量“{ }”的方式创建对象的，还是无法区分对象的类型。此时，可以采用JavaScript提供的另一种创建对象的方式，即通过构造函数创建对象。

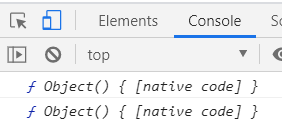
## 内置构造函数

JavaScript提供了Object、String、Number等构造函数，通过“new 构造函数名( )”即可创建对象。人们习惯将使用new关键创建对象的过程称为实例化，实例化得到的对象成为构造函数的实例。具体示例如下：



对象的constructor属性指向了该对象的构造函数。

在控制台查看输出结果：



“[native code]”表示该函数的代码是内置的，因此此函数时JavaScript的内置构造函数。

另外，通过字面量“{}”创建的对象时Object对象的实例。可进行验证：

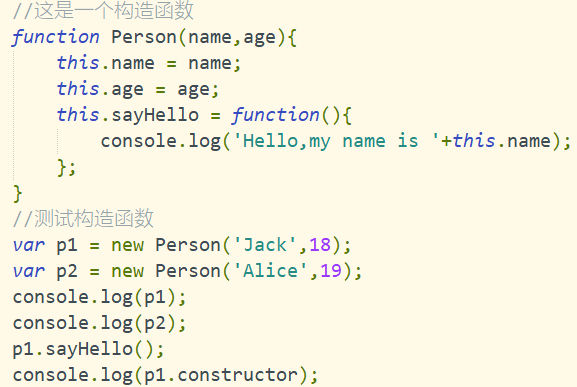


## 自定义构造函数

除了使用内置构造函数，用户也可以自己编写构造函数，在定义时应注意以下事项：

1. 构造函数的命名方式推荐采用帕斯卡命名规则，即所有的单词首字母大写。
2. 在构造函数内部，使用this来表示刚刚创建的对象。

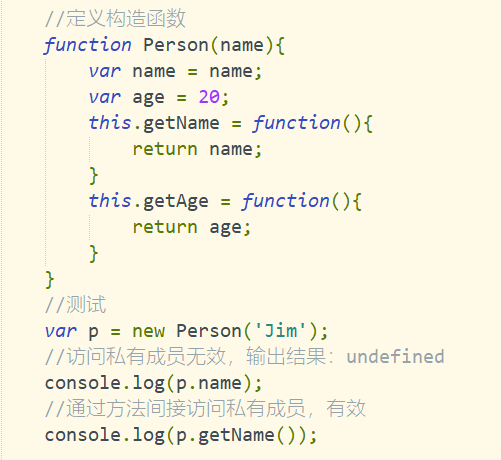
自定义构造函数的具体示例如下：



可以看出，定义构造方法时习惯使用帕斯卡命名规则，这样也方便与普通方法区分。

## 定义私有成员

在构造函数中，使用var关键字定义的变量称为私有成员，在实例化对象后无法通过“对象.成员”的方式进行访问，但是私有成员可以在对象的方法中访问。具体示例如下：



## 定义静态成员

静态成员可以通过构造函数直接使用。



实例成员需要先创建对象才能使用，而静态成员无需创建对象，直接通过构造函数即可使用。所以，在实际开发中，对于不需要创建对象即可访问的成员，推荐将其保存为静态成员。例如，构造函数的prototype属性就是一个静态成员。

## 函数中的this指向

在JavaScript中，函数有多种调用的环境，如直接通过函数名调用、作为对象的方法调用、作为构造函数调用等。根据函数不同的调用方式，函数中this指向会发生改变。

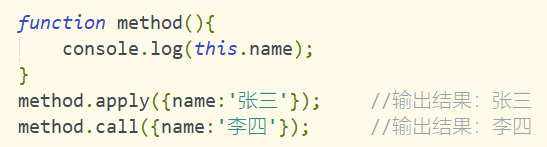
**1、分析this指向**

在JavaScript中，函数内的this关键字指向通过与以下3种情况有关：

1. 使用new关键字将函数作为构造函数调用时，构造函数内部的this指向新创建的对象。
2. 直接通过函数名调用函数时，this指向的是全局对象（在浏览器中表示window对象），如调用alert( )方法弹出警告框。
3. 将函数作为对象的方法调用，this指向该对象。

**2、更改this指向**

除了遵循默认的this指向规则，函数的调用者还可以利用JavaScript提供的两种方式手动控制this的指向。一种是apply( )方法，另一种是通过call( )方法。具体示例如下：



apply( )和call( )的第一个参数表示将this指向哪个对象。它们还有更多的用法请参考教材书或查阅相关资料。

# 内置对象

为了方便程序开发，JavaScript提供了很多常用的内置对象，包括与字符串相关的String对象，与数值相关的Number对象、与数学相关的Math对象、与日期相关的Date对象，以及与数组相关的Array对象等。

## String对象

通过前面的学习可知，利用一对单引号或双引号创建的字符型数据，可以向对象一样使用，这是因为这些对象实际上是String的实例，即String对象。String对象提供了一些用于对字符串进行处理的属性和方法。如下：

|  |  |
| --- | --- |
| 成员 | 作用 |
| length | 获取字符串的长度 |
| charAt(index) | 获取index位置的字符，位置从0开始 |
| indexOf(searchValue) | 获取searchValue在字符串中首次出现的位置 |
| lastIndexOf(searchValue) | 获取searchValue在字符串中最后出现的位置 |
| substring(start[,end]) | 从指定位置开始截取字符串或从截取指定范围内的字符串 |
| substr(start[,end]) | 从start位置开始截取字符串，并可以指定要截取的字符串长度 |
| toLowerCase( ) | 将字符串转小写 |
| toUpperCase( ) | 将字符串转大写 |
| split(separator[,limit]) | 使用separator作为分隔符将字符串分隔为数组，limit用于限制数量 |
| replace(str1,str2) | 使用str2代替字符串中的str1，返回替换后的新字符串 |

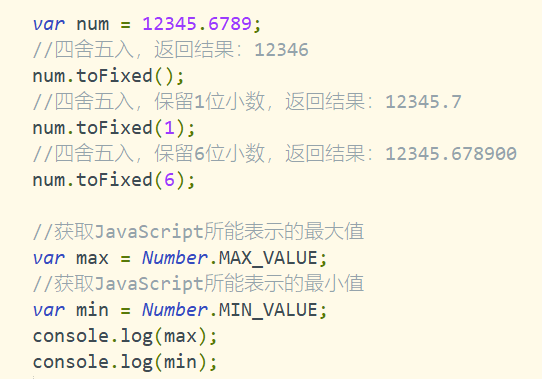
需要注意，使用以上方法对字符串进行操作时，处理结果是通过方法的返回值直接返回的，并不会改变String对象本身保存的字符串内容。在这些方法的参数中，位置指的是一个索引值，从0开始，最后一个字符的索引时字符串的长度减1。

## Number对象

Number对象用于处理整数、浮点数等数值。常用的属性和方法如下：

|  |  |
| --- | --- |
| 成员 | 作用 |
| MAX\_VALUE（静态成员） | 获取JavaScript中所能表示的最大数值 |
| MIN\_VALUE（静态成员） | 获取JavaScript中所能表示的最小正数 |
| toFixed( [digits] ) | 使用定点数表示法来格式化一个数值，可用来进行四舍五入或保留指定位数个小数的操作 |

所有的数值型数据都可以调用Number对象的成员与方法。具体示例如下：

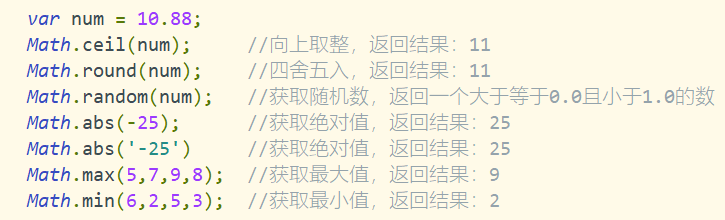


## Math对象

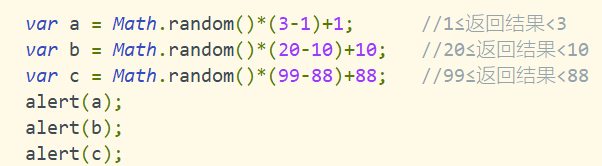
Math对象用于对数值进行数学运算，与其他对象不同的是，该对象不是一个构造函数，不需要示例化就能使用（因为该对象中成员都是静态的，可以通过Math直接调用）。常用的属性和方法如下：

|  |  |
| --- | --- |
| 成员 | 作用 |
| PI | 互殴圆周率，返回3.141592653589793 |
| abs(x) | 获取x的绝对值，x可以是数值型数据或是用字符串表示的数值 |
| max([value1[,value2,…]]) | 获取所有参数中的最大值 |
| min([value1[,value2,…]]) | 获取所有参数中的最小值 |
| pow(base,exponent) | 获取base的指数(exponent)次幂，即baseexponent |
| sqrt(x) | 获取x的平方根 |
| ceil(x) | 获取大于或等于x的最小整数，即向上取整 |
| floor(x) | 获取小于或等于x的最小整数，即向下取整 |
| round(x) | 获取x的四舍五入后的整数值 |
| random( ) | 获取大于或等于0.0且小于1.0的随机值 |

通过具体示例演示Math对象的使用：



利用Math.random( )可以获取指定范围内的随机数，公式为Math.random( )\*(n-m)+m，表示生成大于或等于m且小于n的随机值。示例代码如下：



但返回的结果是小数，如果需要整数，则需要进行取整操作（使用Math的floor函数）。

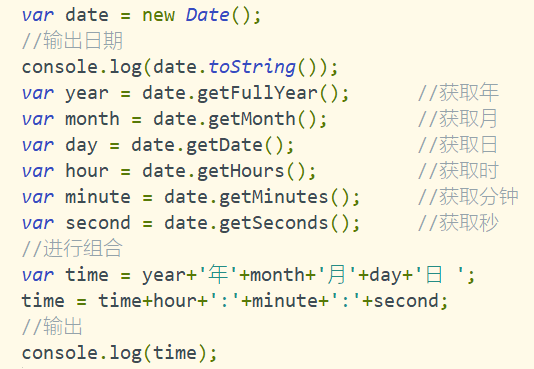
## Date对象

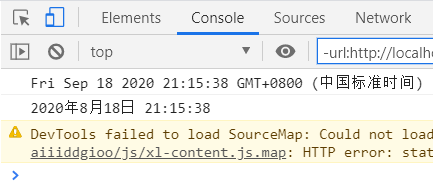
Date对象用于处理日期和时间。常用的方法如下：

|  |  |
| --- | --- |
| **方法** | **作用** |
| getFullYear( ) | 获取表示年份的4位数字，如2020 |
| setFullYear( value ) | 设置年份 |
| getMonth( ) | 获取月份。范围0~11，0表示1月，1表示二月，以此类推 |
| setMonth( value ) | 设置月份 |
| getDate( ) | 获取月份中的某一天。范围1~31 |
| setDate( value ) | 设置月份中的某一天 |
| getDay( ) | 获取星期几，返回0~6（0表示星期日，1表示星期一，以此类推） |
| getHours( ) | 获取小时，返回0~23 |
| setHours( value ) | 设置小时 |
| getMinutes( ) | 获取分钟。范围0~59 |
| setMinutes( value ) | 设置分钟 |
| getSeconds( ) | 获取秒。范围0~59 |
| setSeconds( value ) | 设置秒数 |
| getMilliseconds( ) | 获取毫秒数。范围0~999 |
| setMilliseconds( value ) | 设置毫秒数 |
| getTime( ) | 获取从1970-01-01 00:00:00距离Date对象所代表的时间的毫秒数。 |
| setTime( value ) | 通过从1970-01-01 00:00:00计时的毫秒数来设置时间 |

通过几个示例演示Date对象的使用。

1. 获取当前时间

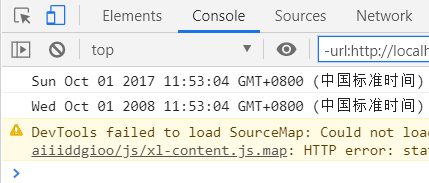




toString( )方法用来方便地查看对象保存的时间信息（也可以直接输出Date对象）。

1. 在使用Date对象时，还可以在创建的时候传入参数来指定一个日期。



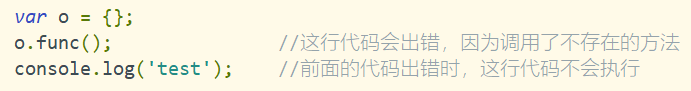


# 错误处理与代码调试

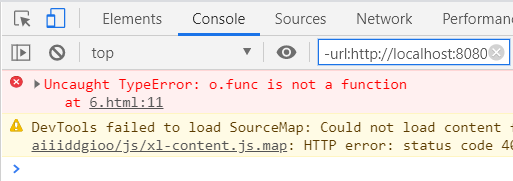
掌握JavaScript错误处理机制可以帮助我们更好地解决程序中发生的错误；掌握代码调试技术可以跟踪程序的运行流程，监听变量的值在运行过程中的改变等。

## 错误处理

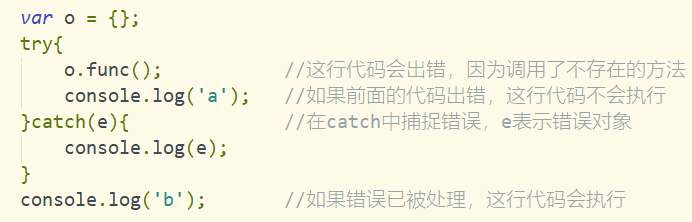
在JavaScript程序中，经常会遇到各种各样的错误，如调用了不存在的方法、引用了不存在的变量等。通过具体示例演示错误的发生：



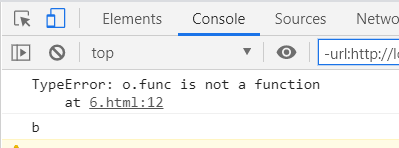
在控制台也能够看到错误并提示出错的位置：



当发生错误时，JavaScript引擎会抛出一个错误对象，利用try…catch语句可以对错误对象捕捉，捕捉后可以查看错误信息。示例如下：



查看控制台输出：



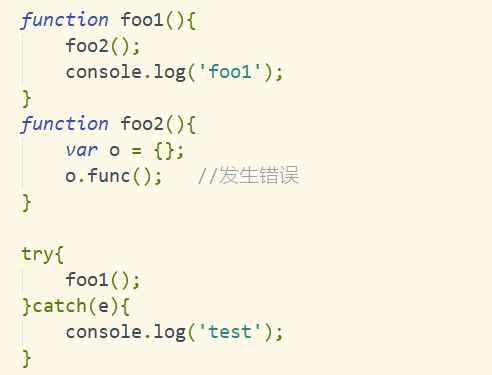
注意的是：如果try中有多行代码，只要其中一行出现错误，后面的代码都不会执行，所以try中的代码量应尽可能少。

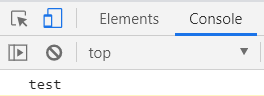
## 错误对象

在发生错误时，错误出现的位置、错误的类型、错误信息等数据，都会被封装起来，以一个对象的形式传递个给catch语句，通过catch(e)的方式来接收，其中e可看作是错误对象的一个实例。

1. **错误对象的传递**

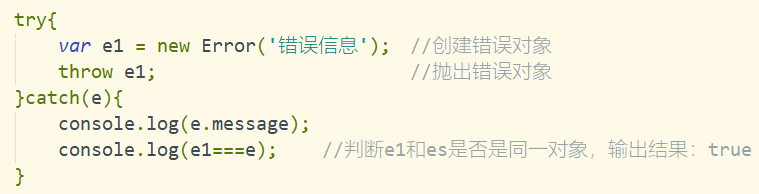
错误对象会在函数之间。当try中的代码调用了其他函数时，如果在其他函数中出现了错误，且没有使用try…catch处理时，程序就会停下来，将错误传递到调用当前函数的上一层函数，如果上一层函数仍然没有处理，则继续向上传递。具体示例如下：





1. **手动抛出错误对象**

当JavaScript程序出现错误时将自动抛出错误对象，用户也可以使用throw关键字手动抛出错误对象。具体示例如下：



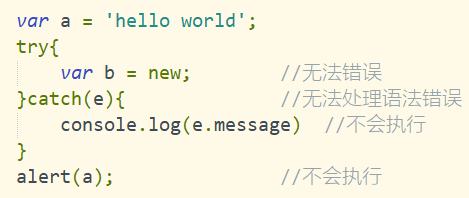
错误对象可以通过message属性获取错误信息。

## 错误类型

在JavaScript中，有7中标准错误类型，每个类型都对应一个构造函数。当发生错误时， JavaScript会根据不同的错误类型抛出不同的错误对象。

|  |  |
| --- | --- |
| 错误类型 | 说明 |
| Error | 表示普通错误，其余6种错误都继承自该对象 |
| EvalError | 调用eval( )函数错误，为了向后兼容，低版本还可以使用 |
| RangeError | 数值超出有效范围，如‘new Array(-1)’ |
| ReferenceError | 使用了不存在的变量 |
| SyntaxError | 解析过程语法错误，如“var a = new;” |
| TypeError | 变量或参数不是预期类型，如调用了不存在的函数或方法 |
| URLError | 解析URI编码出错，调用encodeURI( )、escape( )等URI函数出现 |

通过try...catch处理错误时，只能处理语义错误，无法处理语法错误。如下：

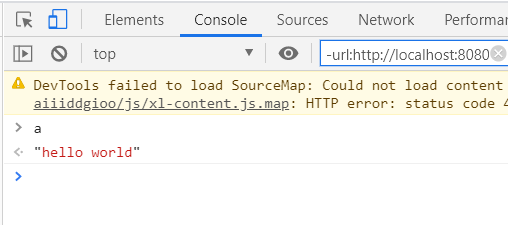


## 代码调试

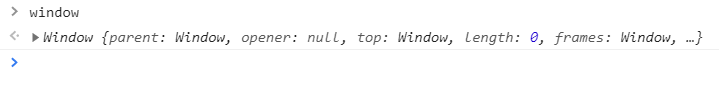
在学习JavaScript的过程中，开发者工具是帮助我们调试代码的利器，通过它可以查阅各种内置对象的成员，跟踪代码的流程。下面的学习以Chrome浏览器的开发者工具的常用功能进行讲解。

**1、在控制台中执行JavaScript代码**

打开控制台后台，会看到一个闪烁的光标，此时可以输入代码，按回车键执行。如定义变量a并赋值，在控制台查看a：

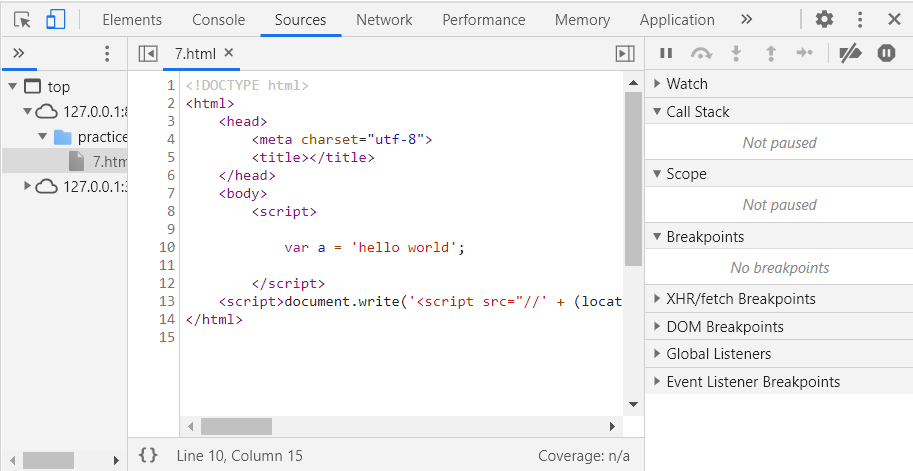


“>”表示该行代码时用户输入的，下一行的“<”表示控制台的输出结果。在控制台还可以输入一个对象，然后查看对象的成员。如下：



1. **单步调试**

在开发者工具的Sources面板可以设置断点，对代码进行单步调试。界面如下：



该面板有左、中、右3个栏目，左栏是文件目录结构，中栏可以查看网页源代码，右栏是JavaScript的调试区。在中栏显示的网页源代码中，使用鼠标单击行号可以设置断点，单击左键表示在当前行设置断点，单击右键会弹出一个菜单，用于详细控制断点。设置断点后，按F5刷新，JavaScript会执行到该位置并暂停。此时可以通过右栏中的各种工具进行调试，具体说明如下：

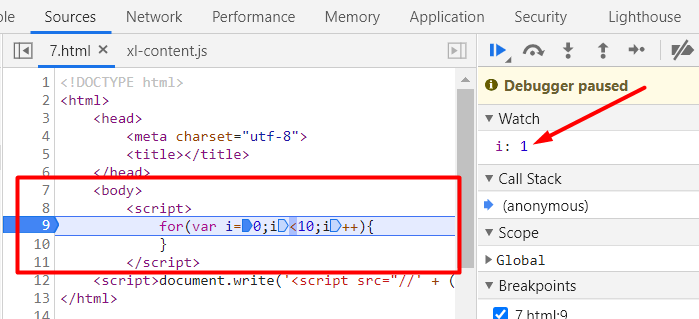
|  |  |
| --- | --- |
| 右栏上存在的按钮 | 说明 |
|  | 暂停或继续执行脚本 |
|  | 执行下一步。遇到函数时，不进入函数直接执行下一步 |
|  | 执行下一步。遇到函数时，进入函数执行 |
|  | 跳出当前函数 |
|  | 停用或启用断点 |
|  | 是否暂停错误捕获 |

1. **调试工具**

在Sources面板的右栏中除了一些调试按钮，还有一些调试工具，具体功能如下表所示：

|  |  |
| --- | --- |
| 名称 | 说明 |
| Watch | 可以对加入监听列表的变量进行监听 |
| Call Stack | 函数调用堆栈，可以在代码暂停时查看执行路径 |
| Scope | 查看当前断点所在函数执行的作用域内容 |
| Breakpoints | 查看断点列表 |
| XHR BreakPoints | 请求断点列表，可以对满足过滤条件的请求进行断点拦截 |
| DOM Breakpoints | DOM断点列表，设置DOM断点后满足条件时触发断点 |
| Global Listeners | 全局监听列表，显示绑定在window对象上的事件监听 |
| Event Listener Breakpoints | 可断点的事件监听列表，可以在触发事件时进入断点 |

例如Watch工具的使用示例如下：



可以看出，Watch工具可以对变量进行监听。

# 原型与继承

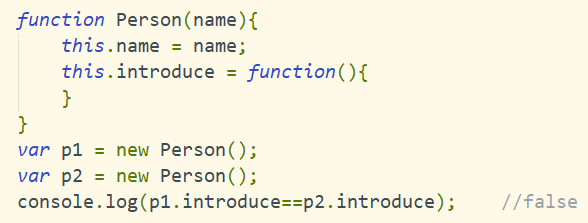
原型与继承是JavaScript语言的难点，掌握了这部分内容，才能更好地理解JavaScript中的各种对象之间的关系，从语言特性上实现面向对象编程。

## 原型

1. **为什么使用原型**

JavaScript提供了原型机制，作为JavaScript面向对象编程的一个重要体现。利用原型可以提高代码的复用性。

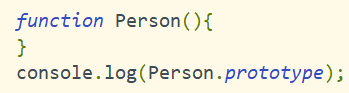
假设有p1、p2两个1对象，都是由构造函数Person常见的。如果我们在Person中定义一个introduce( )方法，则p1、p2两个方法都有了introduce( )方法，但是，这种方式存在一个缺点，就是每个基于Person创建的对象都会重复地保存这些完全相同的方法，带来不必要的浪费。代码演示如下：



JavaScript语言为函数提供了一个原型对象，通过原型对象同享成员。利用原型对象可以保存一些公共的数学和方法，也就是所，基于原型创建的对象会自动拥有原型的属性和方法。

**2、使用原型对象**

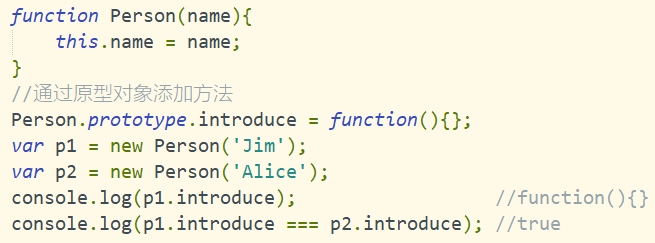
1. 在JavaScript中，每定义一个函数，就随之有一个对象产生，函数通过prototype属性指向该对象，这个对象称之为原型对象，简称原型。如下：



上述代码中，Person函数的prototype属性指向的对象就是因定义Person函数

而随之产生的对象，就是Person的原型镀锡。

1. 在利用构造函数创建对象时，创建的每个对象都默认与这个原型对象连接，连接后就可以访问到原型对象中的属性和方法。具体示例如下：



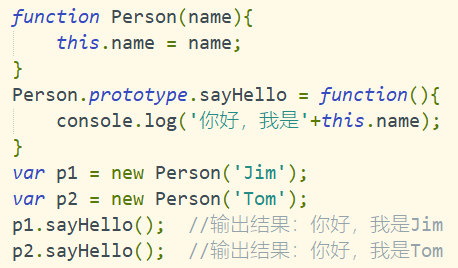
上述代码中，构造函数Person原本没有introduce( )方法，但是在为Person的原型对象添加了该方法后，基于Person函数创建的p1、p2对象都具有了相同的introduce( )方法。

## 继承

继承是在已有对象的基础上进行扩展，增加一些新的功能，得到一个新的对象。在JavaScript中实现继承有4种方式。

1. **利用原型对象实现继承**

原型对象是JavaScript实现继承的传统方式。如果一个对象中本来没有某个属性或方法，但是可以从另一个对象中获得，就实现了继承。具体示例如下：

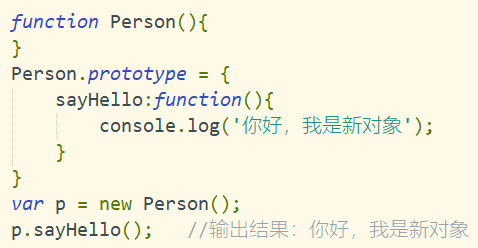


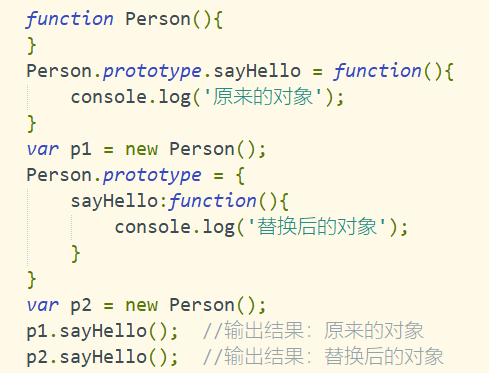
上述代码中，对象p1、p2对象原本没有sayHello( )成员，但是在为构造函数Person的原型对象添加了sayHello( )成员后，p1、p2也就拥有了sayHello( )成员。因此，

可以理解为p1、p2对象继承了原型对象中的成员。

1. **替换原型对象实现继承**

可以将构造函数的原型对象替换成另一个对象A，基于该构造函数创建的对象就会继承新的原型对象。具体示例如下：

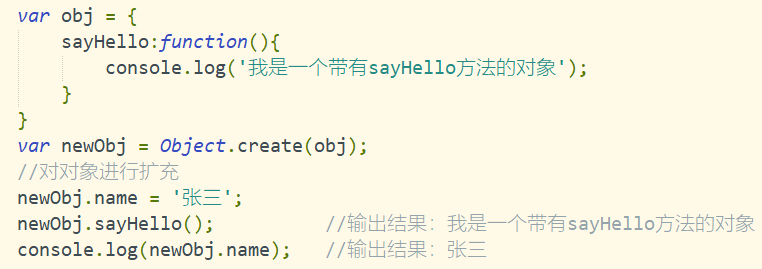


上述代码中，使用一个包含sayHello( )成员的新对象替换掉Person构造函数的原型对象，实现继承。但是需要注意：在基于构造函数创建对象时，创建对象的代码应写在替换原型对象之后，否则创建的对象仍然会继承原来的原型对象。具体示例如下：

从运行结果可知，p1对象继承了原来的原型对象的成员，p2对象继承了新原型对象的成员。

1. **利用Object.create( )实现继承**

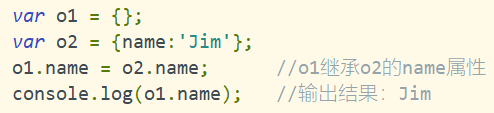
使用Object对象的create( )方法也可以实现继承。使用方法如下：



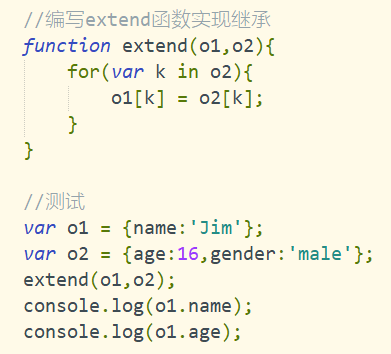
上述代码中，将obj对象作为newObj对象的原型，因此newObj对象继承了obj对象的sayHello( )方法。

1. **混入继承**

混入继承就是将一个对象的成员加入到另一个对象中，实现对象功能的扩展。实现混入继承最简单的方法就是将一个对象的成员赋值给另一个对象。具体示例如下：

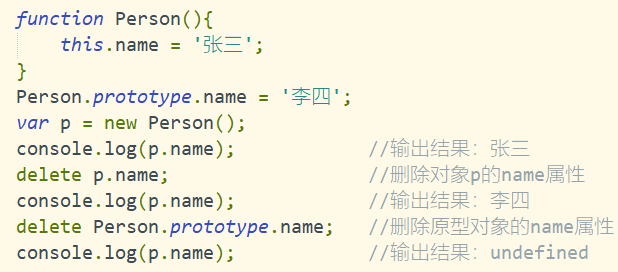


但当成员较多时，如果为每个成员都进行赋值操作，会非常麻烦，因此可以编写一个函数专门实现对象成员的赋值，函数通常名为mix（混合）或extend（扩展），具体示例如下。

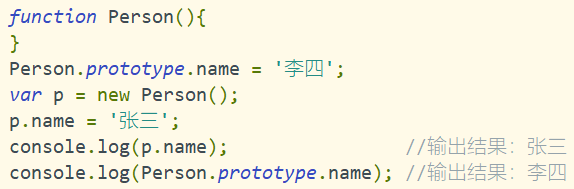


## 属性搜索原则

当对象访问某一个成员的时候，首先会在当前对象中搜索是否包含该成员，如果包含则使用如果不包含，就会自动在其原型对象中查找是否有这个成员，这就是属性搜索原则。在搜索成员时，如果当前对象没有，原型对象中也没有，就会寻找原型对象的原型对象，一直找下去。如果知道最后都没有找到，就会返回undefined。属性搜索原则体现了面向对象中继承的思想。具体示例如下：



属性搜索原则只对属性的访问操作有效，对于属性的添加或修改操作，都是在当前对象中进行的。示例如下：



从上述代码可以看出，为对象p的name属性赋值“张三”后，原型对象中同名的name属性的值没有发生改变。

## 原型链

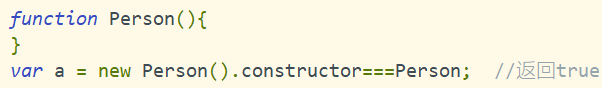
在JavaScript中，对象（函数本质上也是对象）有原型对象，原型对象也有原型对象，这就形成了一个链式结构，简称原型链。

1. **对象的构造函数**

在原型对象中，存在一个constructor属性，执行该对象的构造函数。具体示例如下：

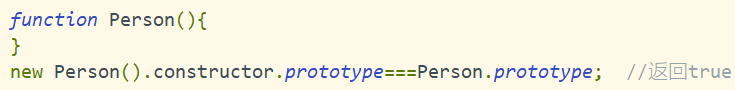


基于Person构造函数创建的实例对象，原本没有constructor属性，但因为继承了Person函数的原型对象，所以就可以访问到constructor属性。如下：



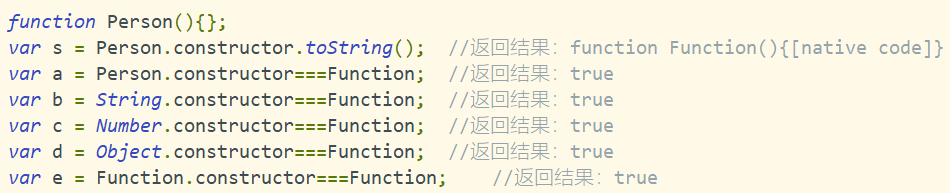
1. **对象的原型对象**

由于对象可以通过constructor属性访问构造函数，构造函数可以通过prototype属性访问原型对象，因此使用“对象.constructor.prototype”的方式即可访问对象的原型对象。具体示例如下：



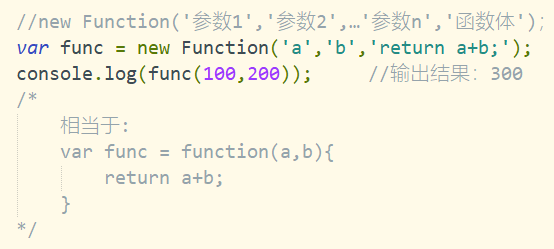
1. **函数的构造函数**

由于函数本质上就是对象，所以函数也有构造函数。在JavaScript中，自定义函数以及String、Number、Object等内置构造函数的构造函数时Function函数，而Function函数的构造函数式Function自身。通过toString( )方法可以查看函数的信息。具体示例如下：



可以看出，JavaScript中每个函数都是构造函数Function的实例，构造函数Function本身也是由自己创建出来的。

指的一提的是，用户还可以通过实例化Function构造函数的方式来创建函数。该构造函数的参数数量是不固定的，最后一个参数表示用字符串保存新创建函数的函数体，前面的参数表示新创建函数的参数名称，具体示例如下：



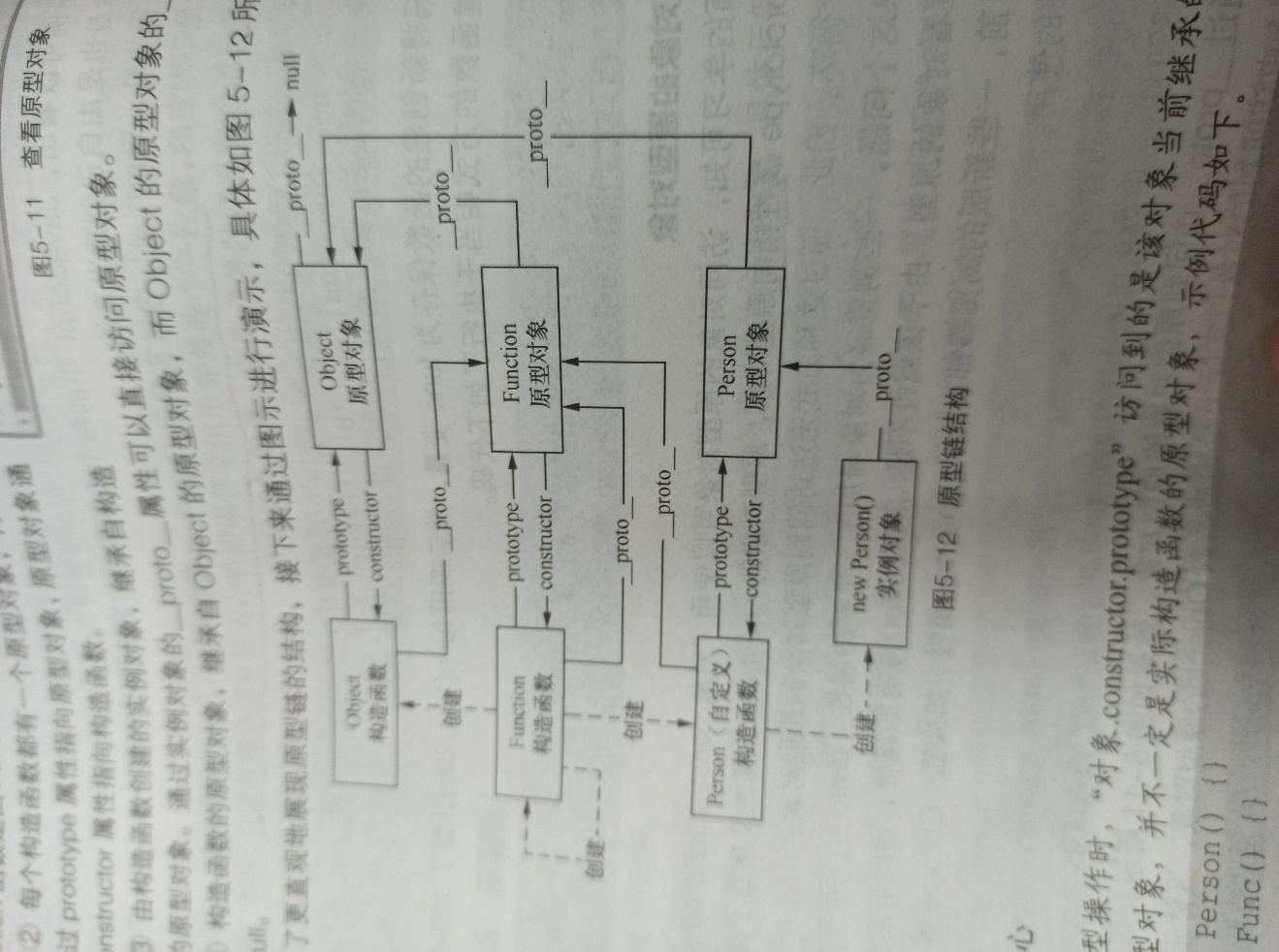
1. **原型对象的原型对象**

教材书上可能代码有误，通过相同代码无法实现相同效果，请再三认真阅读理解。

1. **原型链的结构**

通过前面的分析，关于原型链的结构可以总结为以下4点：

1. 自定义函数，以及Object、String、Number等内置函数，都是由Function函数创建的，Function函数是由Function函数自身创建的。
2. 每个构造函数都有一个原型对象，构造函数通过prototype属性指向原型对象，原型对象通过constructor属性指向构造函数。
3. 由构造函数创建的实例对象，继承自构造函数的原型对象。通过实例对象的\_proto\_属性可直接访问原型对象。
4. 构造函数的原型对象，继承自Object的原型对象，而Object的原型对象的\_proto\_属性为null。

为了更直观地展现原型链的结构，接下来通过图示进行演示，如下图所示：