项目组4复习文档

项目组成员名单:

|  |  |  |
| --- | --- | --- |
| 内容 | 组员 | 备注 |
|  |  |  |
| jQuery认识1-3 | 李春朋 |  |
| jQuery认识6-7 | 高原 |  |
| jQuery认识8-9 | 黄河源 |  |
| 面向对象与模块化10-11 | 苏壮晖，满春发 |  |
|  |  |  |

目录

[一、jQuery认识 4](#_Toc19178138)

[1.参考资料 4](#_Toc19178139)

[2.jQuery引用安装 5](#_Toc19178140)

[2.1在线引用地址 5](#_Toc19178141)

[2.2本地引用 5](#_Toc19178142)

[3.jQuery语法 6](#_Toc19178143)

[3.1 document ready 函数 6](#_Toc19178144)

[4.jQuery选择器 6](#_Toc19178145)

[5.jQuery事件 7](#_Toc19178146)

[5.1鼠标事件 7](#_Toc19178147)

[5.2键盘事件 9](#_Toc19178148)

[5.3表单/文档/窗口事件 9](#_Toc19178149)

[6.jq文档操作 10](#_Toc19178150)

[6.1 读取内容 10](#_Toc19178151)

[6.2 attr()函数 - 设置或返回属性 11](#_Toc19178152)

[6.3 prop()与attr()的一个坑 11](#_Toc19178153)

[6.4添加新内容的四个 jQuery 方法 11](#_Toc19178154)

[6.5删除元素的两个方法 12](#_Toc19178155)

[6.6jQuery 操作 CSS 12](#_Toc19178156)

[6.7 animate() 12](#_Toc19178157)

[7.jq效果和样式 13](#_Toc19178158)

[7.1显示和隐藏 ----hide() 和 show() 13](#_Toc19178159)

[7.2淡入淡出 13](#_Toc19178160)

[7.3动画-- animate()方法--用于建立自定义动画 14](#_Toc19178161)

[8.jQuery ajax()方法 15](#_Toc19178162)

[8.1参考资料 15](#_Toc19178163)

[8.2Ajax简介 15](#_Toc19178164)

[9.jQuery遍历 17](#_Toc19178165)

[参考资料 17](#_Toc19178166)

[9.1什么是遍历? 17](#_Toc19178167)

[9.2 jQuery 遍历 - 祖先 18](#_Toc19178168)

[9.2.1 jQuery parent() 方法 18](#_Toc19178169)

[9.2.2 jQuery parents() 方法 18](#_Toc19178170)

[9.2.3jQuery parentsUntil() 方法 19](#_Toc19178171)

[9.3jQuery 遍历 - 后代 20](#_Toc19178172)

[9.3.1jQuery children() 方法 20](#_Toc19178173)

[9.3.2jQuery find() 方法 21](#_Toc19178174)

[二、JavaScript面向对象 23](#_Toc19178175)

[1.参考资料 23](#_Toc19178176)

[2.JavaScript面向对象简介 23](#_Toc19178177)

[3.实例对象与new命令 23](#_Toc19178178)

[4.this关键字 24](#_Toc19178179)

[5.Object对象的相关方法 24](#_Toc19178180)

[6.严格模式 24](#_Toc19178181)

[三、模块的写法 25](#_Toc19178182)

[1.原始写法 25](#_Toc19178183)

[2.立即执行函数的写法 26](#_Toc19178184)

[3.放大模式 26](#_Toc19178185)

[4.输入全局变量 27](#_Toc19178186)

[5.主模块的写法 28](#_Toc19178187)

[6.模块的加载 28](#_Toc19178188)

[7.AMD模块的写法 28](#_Toc19178189)

[8.加载方式 29](#_Toc19178190)

# 一、jQuery认识

# 1.参考资料

菜鸟网站: https://www.runoob.com/

jQuery API在线手册 : https://www.runoob.com/manual/jquery/

# 2.jQuery引用安装

## 2.1在线引用地址

<script src="https://cdn.staticfile.org/jquery/1.10.2/jquery.min.js"></script>

<script src="https://apps.bdimg.com/libs/jquery/2.1.4/jquery.min.js"></script>

<script src="https://upcdn.b0.upaiyun.com/libs/jquery/jquery-2.0.2.min.js"></script>

<script src="https://lib.sinaapp.com/js/jquery/2.0.2/jquery-2.0.2.min.js"></script>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>

<script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-1.9.0.min.js"></script>

## 2.2本地引用

下载地址: <https://jquery.com/download/>

使用 HTML 的 <script> 标签引用它：

<script src="jquery-1.10.2.min.js"></script>

安装jquery/ 引用jquery (尽量使用1.11 ~ 1.20 之间的版本 , 最新的浏览器没有限制, 目前最高版本是 1.3.1)

# 3.jQuery语法

## 3.1 document ready 函数

为了防止文档在完全加载（就绪）之前运行 jQuery 代码，即在 DOM 加载完成后才可以对 DOM 进行操作。我们需要将jquery代码写在document ready 函数里.

一般写法:

$(document).ready(function(){

// 开始写 jQuery 代码...

});

简单写法:

$(function(){

// 开始写 jQuery 代码...

});

**jquery有一个基本语法 $(selector[选择器]).action[操作]()**

# 4.jQuery选择器

参考网址: <https://www.runoob.com/jquery/jquery-ref-selectors.html>

1.基本选择器,常用的一些

$("#id") //ID选择器

$("div") //元素选择器

$(".classname") //类选择器

$(".classname,.classname1,#id1") //组合选择器

$("#id>.classname ") //子元素选择器

$("#id .classname ") //后代元素选择器

$("li:first") //第一个li

$("li:last") //最后一个li

$("li:even") //挑选下标为偶数的li

$("li:odd") //挑选下标为奇数的li

$("li:eq(4)") //下标等于 4 的li(第五个 li 元素)

$("li:gt(2)") //下标大于 2 的li

$("li:lt(2)") //下标小于 2 的li

$("li:not(#runoob)") //挑选除 id="runoob" 以外的所有li

$("div:contains('Runob')") // 包含 Runob文本的元素

$("td:empty") //不包含子元素或者文本的空元素

$("div:has(selector)") //含有选择器所匹配的元素

$("td:parent") //含有子元素或者文本的元素

$("li:hidden") //匹配所有不可见元素，或type为hidden的元素

$("li:visible") //匹配所有可见元素

# 5.jQuery事件

参考网址: <https://www.runoob.com/jquery/jquery-ref-events.html>

<https://www.w3school.com.cn/jquery/jquery_ref_events.asp>

## 5.1鼠标事件

1.click--点击

2.dblclick--双击

3.mouseenter--划过

4.mouseleave--划出

5.hover--悬停在上面

**以click为例:**

//绑定方法1 - 直接绑定

$("#bt1").click(function btClick(e){console.info("触发了点击事件");});

$("#bt1").click(btClick);

//绑定方法2 - on函数

$("#bt1").on("click" , function(){console.info("触发了点击事件");});

$("#bt1").on("click", btClick);

//绑定方法3 - 原生方式

$("#bt1")[0].onclick = function(e){

btClick();

}

// 绑定方法4 - bind

$("#bt1").bind("click" , function(){console.info("触发了点击事件");});

// 事件解绑 - 重要概念!!! ------原生js的做法是 事件函数 - null

$("#bt1").unbind("click"); // 如果不传参数, 所有事件函数都解绑

// 模拟事件 - 对已经绑定了事件函数的事件,可以执行,代替触发

$("p").mouseover();

// 未来元素事件绑定(通过他爹操作它)

$("#father").delegate("p" , "click" , function(){

$(this).css({

color:"blue"

});

});

## 5.2键盘事件

1.keydown--键按下的过程

2.keypress--键被按下

3.keyup--键被松开

// keypress

$("input").keypress(function(){

$("span").text(i+=1);

});

// keydown

$("input").keydown(function(){

$("input").css("background-color","yellow");

});

// keyup

$("input").keyup(function(){

$("input").css("background-color","pink");

});

## 5.3表单/文档/窗口事件

1.submit() -- 提交表单

2.change()  -- 当 <input> 字段改变时

3.focus() -- 当 <input> 字段获得焦点时发生 focus

4.blur() -- 当 <input> 字段失去焦点时发生 blur

5.load() -- 加载完成时运行的函数

6.resize() -- 当调整浏览器窗口大小时，发生 resize

7.scroll() -- 当用户滚动指定的元素时，会发生 scroll

8.unload() -- 关闭网页时

# 6.jq文档操作

## 6.1 读取内容

* text() - 设置或返回所选元素的文本内容

text() 方法设置或返回被选元素的文本内容。

当该方法用于**返回**内容时，则返回所有匹配元素的文本内容（会删除 HTML 标记）。

当该方法用于**设置**内容时，则重写所有匹配元素的内容。

例：

$("button").click(function(){ $("p").text("Hello world!"); });

* html() - 设置或返回所选元素的内容（包括 HTML 标记）

html() 方法设置或返回被选元素的内容（innerHTML）。

当该方法用于**返回**内容时，则返回第一个匹配元素的内容。

当该方法用于**设置**内容时，则重写所有匹配元素的内容。

例：

$("button").click(function(){ $("p").html("Hello <b>world</b>!"); });

* val() - 设置或返回表单字段的值

val() 方法返回或设置被选元素的 value 属性。

**当用于返回值时：**该方法返回第一个匹配元素的 value 属性的值。

**当用于设置值时：**该方法设置所有匹配元素的 value 属性的值。

**注意：**val() 方法通常与 HTML 表单元素一起使用。

例：

$("button").click(function(){ $("input:text").val("Glenn Quagmire"); });

## 6.2 attr()函数 - 设置或返回属性

定义和用法：

attr() 方法设置或返回被选元素的属性和值。

当该方法用于**返回属性值**，则返回第一个匹配元素的值。

当该方法用于**设置属性值**，则为匹配元素设置一个或多个属性/值对。

例：$("button").click(function(){ $("img").attr("width","500"); });

## 6.3 prop()与attr()的一个坑

prop()函数的结果:

1.如果有相应的属性，返回指定属性值。

2.如果没有相应的属性，返回值是空字符串。

attr()函数的结果:

1.如果有相应的属性，返回指定属性值。

2.如果没有相应的属性，返回值是 undefined。

对于HTML元素本身就带有的固有属性，在处理时，使用prop方法。

对于HTML元素我们自己自定义的DOM属性，在处理时，使用 attr 方法。

具有 true 和 false 两个属性的属性，如 checked, selected 或者 disabled 使用prop()

## 6.4添加新内容的四个 jQuery 方法

append() - 在被选元素的结尾插入内容

prepend() - 在被选元素的开头插入内容

after() - 在被选元素之后插入内容

before() - 在被选元素之前插入内容

## 6.5删除元素的两个方法

remove() - 方法移除被选元素，包括所有的文本和子节点该方法也会移除被选元素的数据和事件。

empty() - 从被选元素中删除子元素（该方法不会移除元素本身，或它的属性）

## 6.6jQuery 操作 CSS

addClass() - 向被选元素添加一个或多个类

例：$("button").click(function(){ $("p:first").addClass("intro"); });

removeClass() - 从被选元素删除一个或多个类

例：$("button").click(function(){ $("p").removeClass("intro"); });

toggleClass() - 对被选元素进行添加/删除类的切换操作

例：$("button").click(function(){ $("p").toggleClass("main"); });

css() - 设置或返回样式属性

例：返回css属性--$("p").css("background-color");

设置 CSS 属性**--**$("p").css("background-color","yellow");

设置多个CSS属性--$("p").css({"background-color":"yellow","font-size":"200%"});

## 6.7 animate()

animate() 方法执行 CSS 属性集的自定义动画。

该方法通过 CSS 样式将元素从一个状态改变为另一个状态。CSS属性值是逐渐改变的，这样就可以创建动画效果。

只有数字值可创建动画（比如 "margin:30px"）。字符串值无法创建动画（比如 "background-color:red"）。

**提示：**请使用 "+=" 或 "-=" 来创建相对动画。

例：$("button").click(function(){ $("#box").animate({height:"300px"}); });

# 7.jq效果和样式

## 7.1显示和隐藏 ----hide() 和 show()

hide()：隐藏文本

例：

$("#hide").click(function(){ $("p").hide(); });

show()：显示文本

例：

$("#hide").click(function(){ $("p").show(); });

## 7.2淡入淡出

jQuery 拥有下面四种方法：

* fadeIn()

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。可选的callback 参数是 fading 完成后所执行的函数名称。

例：

$("button").click(function(){ $("#div1").fadeIn(); $("#div2").fadeIn("slow"); $("#div3").fadeIn(3000); });

* fadeOut()

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。可选的 callback 参数是 fading 完成后所执行的函数名称。

例：

$("button").click(function(){ $("#div1").fadeOut(); $("#div2").fadeOut("slow"); $("#div3").fadeOut(3000); });

* fadeToggle()

jQuery fadeToggle() 方法可以在 fadeIn() 与 fadeOut() 方法之间进行切换。

如果元素已淡出，则 fadeToggle() 会向元素添加淡入效果。

如果元素已淡入，则 fadeToggle() 会向元素添加淡出效果。

例：

$("button").click(function(){ $("#div1").fadeToggle(); $("#div2").fadeToggle("slow"); $("#div3").fadeToggle(3000); });

* fadeTo()

必需的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。fadeTo() 方法中必需的 opacity 参数将淡入淡出效果设置为给定的不透明度（值介于 0 与 1 之间）。可选的 callback 参数是该函数完成后所执行的函数名称。

例:

$("button").click(function(){ $("#div1").fadeTo("slow",0.15); $("#div2").fadeTo("slow",0.4); $("#div3").fadeTo("slow",0.7); });

## 7.3动画-- animate()方法--用于建立自定义动画

必需的 params 参数定义形成动画的 CSS 属性。可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。可选的 callback 参数是动画完成后所执行的函数名称。

---把<div>元素往右边移动250像素

例：

$("button").click(function(){ $("div").animate({left:'250px'}); });

1--jQuery animate() - 操作多个属性

例：

$("button").click(function(){ $("div").animate({ left:'250px', opacity:'0.5', height:'150px', width:'150px' }); });

2--jQuery animate() - 使用相对值

也可以定义相对值（该值相对于元素的当前值）需要在值的前面加上 += 或 -=：

例：

$("button").click(function(){ $("div").animate({ left:'250px', opacity:'0.5', height:'150px', width:'150px' }); });

3--jQuery animate() - 使用预定义的值

您甚至可以把属性的动画值设置为 "show"、"hide" 或 "toggle"：

例:

$("button").click(function(){ $("div").animate({ height:'toggle' }); });

4--jQuery animate() - 使用队列功能

如果在彼此之后编写多个 animate() 调用，jQuery 会创建包含这些方法调用的"内部"队列然后逐一运行这些 animate 调用。

例：

$("button").click(function(){ var div=$("div"); div.animate({height:'300px',opacity:'0.4'},"slow"); div.animate({width:'300px',opacity:'0.8'},"slow"); div.animate({height:'100px',opacity:'0.4'},"slow"); div.animate({width:'100px',opacity:'0.8'},"slow"); });

# 8.jQuery ajax()方法

## 8.1参考资料

菜鸟教程:https://www.runoob.com/jquery/ajax-ajax.html

W3C:<https://www.w3school.com.cn/jquery/jquery_ref_ajax.asp>

## 8.2Ajax简介

什么是 AJAX？

AJAX = 异步 JavaScript 和 XML（Asynchronous JavaScript and XML）。通过在后台与服务器进行少量数据交换，AJAX 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

示例代码

执行异步ajax请求:

$.ajax({

url: '',

type: "",

data: {},

dataType: "",

success: function(res) {

},

error: function(a, b) {

}

})

Url : 请求的地址.

Type : 请求的类型,一般为”get”或”post”.

Data : 要发送到服务器的数据.

dataType : 服务器响应(返回)的数据类型,我们用过的有”text”和”json”.

Success : 回调函数,请求成功时运行的函数,res为请求成功后服务器响应(返回)的结果.

Error : 回调函数,请求失败时运行的函数,一般参数为a,b两个,a表示错误的类型,b表示错误的信息.

# 9.jQuery遍历

## 参考资料

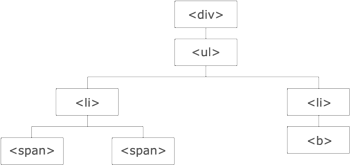
菜鸟教程 : <https://www.runoob.com/jquery/jquery-traversing.html>

W3C : <https://www.w3school.com.cn/jquery/jquery_traversing.asp>

## 9.1什么是遍历?

jQuery 遍历，意为"移动"，用于根据其相对于其他元素的关系来"查找"（或选取）HTML 元素。以某项选择开始，并沿着这个选择移动，直到抵达您期望的元素为止。

下图展示了一个家族树。通过 jQuery 遍历，您能够从被选（当前的）元素开始，轻松地在家族树中向上移动（祖先），向下移动（子孙），水平移动（同胞）。这种移动被称为对 DOM 进行遍历。



图示解析：

<div> 元素是 <ul> 的父元素，同时是其中所有内容的祖先。

<ul> 元素是 <li> 元素的父元素，同时是 <div> 的子元素

左边的 <li> 元素是 <span> 的父元素，<ul> 的子元素，同时是 <div> 的后代。

<span> 元素是 <li> 的子元素，同时是 <ul> 和 <div> 的后代。

两个 <li> 元素是同胞（拥有相同的父元素）。

右边的 <li> 元素是 <b> 的父元素，<ul> 的子元素，同时是 <div> 的后代。

<b> 元素是右边的 <li> 的子元素，同时是 <ul> 和 <div> 的后代。

提示：祖先是父、祖父、曾祖父等等。后代是子、孙、曾孙等等。同胞拥有相同的父。

## 9.2 jQuery 遍历 - 祖先

### 9.2.1 jQuery parent() 方法

parent() 方法返回被选元素的直接父元素。

该方法只会向上一级对 DOM 树进行遍历。

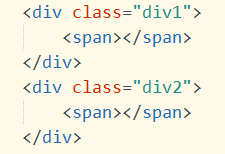
实例:

$(document).ready(function(){

$("span").parent();

});

这个例子返回的是每个<span>标签的直接父元素,例如



通过$("span").parent();这段代码在控制台输出我们可以得到所有<span>标签的直接父元素



### 9.2.2 jQuery parents() 方法

parents() 方法返回被选元素的所有祖先元素，它一路向上直到文档的根元素 (<html>)。

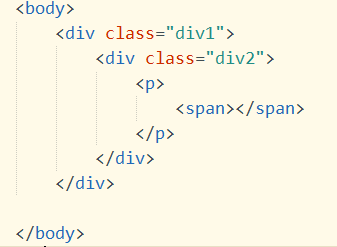
实例:

$(document).ready(function(){

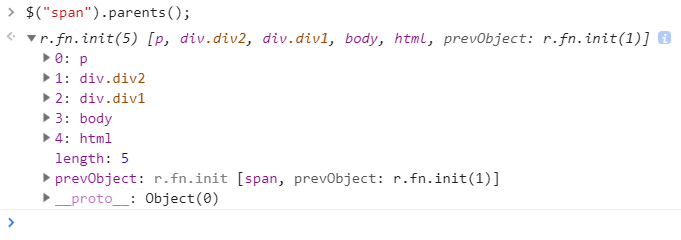
$("span").parents();

});

这个例子返回所有 <span> 元素的所有祖先,例如



通过$("span").parents();这段代码在控制台输出我们可以得到<span>标签的所有祖先元素,一直找到最上层的根元素<html>.



### 9.2.3jQuery parentsUntil() 方法

parentsUntil() 方法返回介于两个给定元素之间的所有祖先元素。

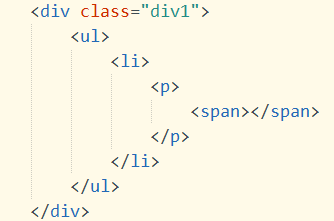
实例:

$(document).ready(function(){

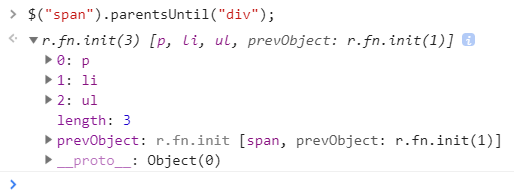
$("span").parentsUntil("div");

});

这个例子返回介于 <span> 与 <div> 元素之间的所有祖先元素,例如



通过$("span").parentsUntil("div");这段代码在控制台输出我们可以得到介于 <span> 与 <div> 元素之间的所有祖先元素.



## 9.3jQuery 遍历 - 后代

### 9.3.1jQuery children() 方法

children() 方法返回被选元素的所有直接子元素。

该方法只会向下一级对 DOM 树进行遍历。

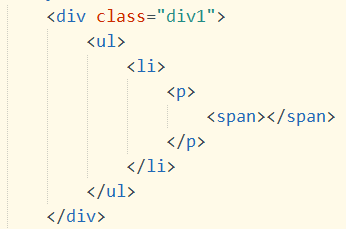
实例:

$(document).ready(function(){

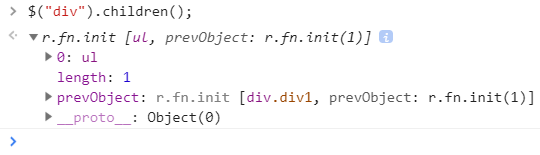
$("div").children();

});

这个例子返回每个 <div> 元素的所有直接子元素,例如



通过$("div").children();这段代码在控制台输出我们可以得到每个 <div> 元素的所有直接子元素.



### 9.3.2jQuery find() 方法

find() 方法返回被选元素的后代元素，一路向下直到最后一个后代。

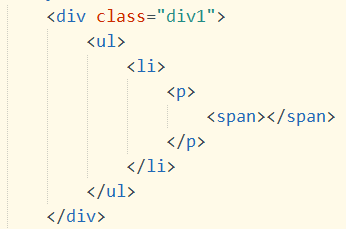
实例:

$(document).ready(function(){

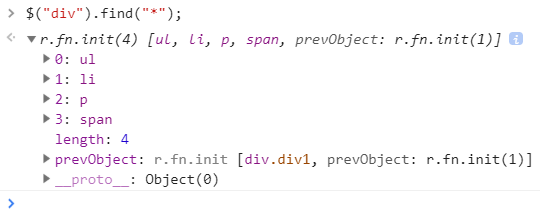
$("div").find("\*");

});

这个例子返回 <div> 的所有后代,例如



通过$("div").find("\*");这段代码我们可以得到<div>的所有后代元素,例如



参考资料

// 教程地址(菜鸟教程)

https://www.runoob.com/nodejs/nodejs-tutorial.html

// 官网

https://nodejs.org/zh-cn/

中文文档

<http://nodejs.cn/api/>

// 目前版本 10.16.3 (lts)

// 接口文档

https://nodejs.org/api/

https://nodejs.org/dist/latest-v10.x/docs/api/

# 二、JavaScript面向对象

## 1.参考资料

[**https://developer.mozilla.org/zhCN/docs/Web/JavaScript/Introduction\_to\_Object-Oriented\_JavaScript**](https://developer.mozilla.org/zhCN/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript)

**http://wangdoc.com/javascript/oop/strict.html**

## 2.JavaScript面向对象简介

JavaScript 的核心是支持面向对象的，同时它也提供了强大灵活的 OOP 语言能力。本文从对面向对象编程的介绍开始，带您探索 JavaScript 的对象模型，最后描述 JavaScript 当中面向对象编程的一些概念.

## 3.实例对象与new命令

构造函数作为模板，可以生成实例对象。但是，有时拿不到构造函数，只能拿到一个现有的对象。我们希望以这个现有的对象作为模板，生成新的实例对象，这时就可以使用Object.create()方法。

使用new命令时，它后面的函数依次执行下面的步骤。

1.创建一个空对象，作为将要返回的对象实例。

2.将这个空对象的原型，指向构造函数的prototype属性。

3.将这个空对象赋值给函数内部的this关键字。

4.开始执行构造函数内部的代码。

也就是说，构造函数内部，this指的是一个新生成的空对象，所有针对this的操作，都会发生在这个空对象上。构造函数之所以叫“构造函数”，就是说这个函数的目的，就是操作一个空对象（即this对象），将其“构造”为需要的样子。

如果构造函数内部有return语句，而且return后面跟着一个对象，new命令会返回return语句指定的对象；否则，就会不管return语句，返回this对象。

## 4.this关键字

this主要有以下几个使用场合:

（1）**全局环境:**全局环境使用this，它指的就是顶层对象window。

（2）**构造函数:**构造函数中的this，指的是实例对象。

（3）**对象的方法:**如果对象的方法里面包含this，this的指向就是方法运行时所在的对象。该方法赋值给另一个对象，就会改变this的指向。

## 5.Object对象的相关方法

Object.getPrototypeOf方法返回参数对象的原型。这是获取原型对象的标准方法。

Object.setPrototypeOf方法为参数对象设置原型，返回该参数对象。它接受两个参数，第一个是现有对象，第二个是原型对象。

JavaScript 提供了Object.create方法，用来满足这种需求。该方法接受一个对象作为参数，然后以它为原型，返回一个实例对象。该实例完全继承原型对象的属性。

Object.prototype.isPrototypeOf() : 实例对象的isPrototypeOf方法，用来判断该对象是否为参数对象的原型。

Object.prototype.\_\_proto\_\_ : 实例对象的\_\_proto\_\_属性（前后各两个下划线），返回该对象的原型。该属性可读写。

Object.getOwnPropertyNames方法返回一个数组，成员是参数对象本身的所有属性的键名，不包含继承的属性键名。

Object.prototype.hasOwnProperty() : 对象实例的hasOwnProperty方法返回一个布尔值，用于判断某个属性定义在对象自身，还是定义在原型链上。

## 6.严格模式

除了正常的运行模式，JavaScript 还有第二种运行模式：严格模式（strict mode）。顾名思义，这种模式采用更加严格的 JavaScript 语法。

同样的代码，在正常模式和严格模式中，可能会有不一样的运行结果。一些在正常模式下可以运行的语句，在严格模式下将不能运行。

# 三、模块的写法

## 1.原始写法

模块就是实现特定功能的一组方法。

只要把不同的函数（以及记录状态的变量）简单地放在一起，就算是一个模块

　　function m1(){  
　　　　//...  
　　}

　　function m2(){  
　　　　//...  
　　}

上面的函数m1()和m2()，组成一个模块。使用的时候，直接调用就行了。

这种做法的缺点很明显："污染"了全局变量，无法保证不与其他模块发生变量名冲突，而且模块成员之间看不出直接关系。

对象写法

为了解决上面的缺点，可以把模块写成一个对象，所有的模块成员都放到这个对象里面。

var module1 = new Object({

　　　　\_count : 0,

　　　　m1 : function (){  
　　　　　　//...  
　　　　},

　　　　m2 : function (){  
　　　　　　//...  
　　　　}

　　});

上面的函数m1()和m2(），都封装在module1对象里。使用的时候，就是调用这个对象的属性。

module1.m1();

但是，这样的写法会暴露所有模块成员，内部状态可以被外部改写。比如，外部代码可以直接改变内部计数器的值。

　　module1.\_count = 5;

## 2.立即执行函数的写法

使用"立即执行函数"（Immediately-Invoked Function Expression，IIFE），可以达到不暴露私有成员的目的。

var module1 = (function(){

　　　　var \_count = 0;

　　　　var m1 = function(){  
　　　　　　//...  
　　　　};

　　　　var m2 = function(){  
　　　　　　//...  
　　　　};

　　　　return {  
　　　　　　m1 : m1,  
　　　　　　m2 : m2  
　　　　};

　　})();

使用上面的写法，外部代码无法读取内部的\_count变量。

console.info(module1.\_count); //undefined

module1就是Javascript模块的基本写法。下面，再对这种写法进行加工。

## 3.放大模式

如果一个模块很大，必须分成几个部分，或者一个模块需要继承另一个模块，这时就有必要采用"放大模式"（augmentation）。

var module1 = (function (mod){

　　　　mod.m3 = function () {  
　　　　　　//...  
　　　　};

　　　　return mod;

　　})(module1);

上面的代码为module1模块添加了一个新方法m3()，然后返回新的module1模块。

宽放大模式

在浏览器环境中，模块的各个部分通常都是从网上获取的，有时无法知道哪个部分会先加载。如果采用上一节的写法，第一个执行的部分有可能加载一个不存在空对象，这时就要采用"宽放大模式"。

　　var module1 = ( function (mod){

　　　　//...

　　　　return mod;

　　})(window.module1 || {});

与"放大模式"相比，＂宽放大模式＂就是"立即执行函数"的参数可以是空对象。

## 4.输入全局变量

独立性是模块的重要特点，模块内部最好不与程序的其他部分直接交互。

为了在模块内部调用全局变量，必须显式地将其他变量输入模块。

　　var module1 = (function ($, YAHOO) {

　　　　//...

　　})(jQuery, YAHOO);

AMD规范

模范的规范

因为有了模块，我们就可以更方便地使用别人的代码，想要什么功能，就加载什么模块。但是，这样做有一个前提，那就是大家必须以同样的方式编写模块，考虑到Javascript模块现在还没有官方规范，这一点就更重要了。

目前，通行的Javascript模块规范共有两种：CommonJS和AMD。

在浏览器环境下，没有模块也不是特别大的问题，毕竟网页程序的复杂性有限；但是在服务器端，一定要有模块，与操作系统和其他应用程序互动，否则根本没法编程。

node.js的模块系统，就是参照CommonJS规范实现的。在CommonJS中，有一个全局性方法require()，用于加载模块。

require .js的用法

require.js的诞生实现js文件的异步加载,避免网页失去响应

管理模块之间的依懒性,便于代码的编写和维护

require.js的加载

　<script src="js/require.js"></script>

## 5.主模块的写法

　　// main.js

　　require(['moduleA', 'moduleB', 'moduleC'], function (moduleA, moduleB, moduleC){

　　　　// some code here

　　});

require()函数接受两个参数。第一个参数是一个数组，表示所依赖的模块，上例就是['moduleA', 'moduleB', 'moduleC']，即主模块依赖这三个模块；第二个参数是一个回调函数，当前面指定的模块都加载成功后，它将被调用。加载的模块会以参数形式传入该函数，从而在回调函数内部就可以使用这些模块。

require()异步加载moduleA，moduleB和moduleC，浏览器不会失去响应；它指定的回调函数，只有前面的模块都加载成功后，才会运行，解决了依赖性的问题。

## 6.模块的加载

默认情况下，require.js假定这三个模块与main.js在同一个目录，文件名分别为jquery.js，underscore.js和backbone.js，然后自动加载。

使用require.config()方法，我们可以对模块的加载行为进行自定义。require.config()就写在主模块（main.js）的头部。参数就是一个对象，这个对象的paths属性指定各个模块的加载路径。

　　require.config({

　　　　paths: {

　　　　　　"jquery": "jquery.min",  
　　　　　　"underscore": "underscore.min",  
　　　　　　"backbone": "backbone.min"

　　　　}

　　});

## 7.AMD模块的写法

require.js加载的模块，采用AMD规范。也就是说，模块必须按照AMD的规定来写。

具体来说，就是模块必须采用特定的define()函数来定义。如果一个模块不依赖其他模块，那么可以直接定义在define()函数之中。

假定现在有一个math.js文件，它定义了一个math模块。那么，math.js就要这样写：

　　// math.js

　　define(function (){

　　　　var add = function (x,y){

　　　　　　return x+y;

　　　　};

　　　　return {

　　　　　　add: add  
　　　　};

　　});

## 8.加载方式

　　// main.js

　　require(['math'], function (math){

　　　　alert(math.add(1,1));

　　});

如果这个模块还依赖其他模块，那么define()函数的第一个参数，必须是一个数组，指明该模块的依赖性。

　　define(['myLib'], function(myLib){

　　　　function foo(){

　　　　　　myLib.doSomething();

　　　　}

　　　　return {

　　　　　　foo : foo

　　　　};

　　});

当require()函数加载上面这个模块的时候，就会先加载myLib.js文件。