# 迭代语句

## do-while 语句

**do** {*statement*} **while** (*expression*);

例子：

var i = 0;

do {i += 2;} while (i < 10);

## for 语句

for 语句是前测试循环，而且在进入循环之前，能够初始化变量，并定义循环后要执行的代码。

它的语法如下：

**for** (initialization; expression; post-loop-expression) statement

注意：post-loop-expression 之后不能写分号，否则无法运行。

例子：

iCount = 6;

for (var i = 0; i < iCount; i++) {

alert(i);

}

这段代码定义了初始值为 0 的变量 i。只有当条件表达式（i < iCount）的值为 true 时，才进入 for 循环，这样循环主体可能不被执行。如果执行了循环主体，那么将执行循环后表达式，并迭代变量 i。

## for-in 语句

for-in 语句是严格的迭代语句，用于枚举对象的属性。

它的语法如下：

**for** (property **in** expression) statement

例子：

for (sProp in window) {

alert(sProp);

}

**JS的基本数据类型和引用数据类型**

* 基本数据类型：undefined、null、boolean、number、string、symbol
* 引用数据类型：object、array、function

**介绍JS有哪些内置对象？**

* 数据封装类对象：Object、Array、Boolean、Number、String
* 其他对象：Function、Arguments、Math、Date、RegExp、Error
* ES6新增对象：Symbol、Map、Set、Promises、Proxy、Reflect

**JavaScript如何实现一个类，怎么实例化这个类？**

* 构造函数法（this + prototype） -- 用 new 关键字 生成实例对象
  + 缺点：用到了 this 和 prototype，编写复杂，可读性差

function Mobile(name, price){

this.name = name;

this.price = price;

}

Mobile.prototype.sell = function(){

alert(this.name + "，售价 $" + this.price);

}

var iPhone7 = new Mobile("iPhone7", 1000);

iPhone7.sell();

* Object.create 法 -- 用 Object.create() 生成实例对象
* 缺点：不能实现私有属性和私有方法，实例对象之间也不能共享数据

var Person = {

firstname: "Mark",

lastname: "Yun",

age: 25,

introduce: function(){

alert('I am ' + Person.firstname + ' ' + Person.lastname);

}

};

var person = Object.create(Person);

person.introduce();

// Object.create 要求 IE9+，低版本浏览器可以自行部署：

if (!Object.create) {

　 Object.create = function (o) {

　　　 function F() {}

　　　 F.prototype = o;

　　　 return new F();

　　};

　}

**DOM操作——怎样添加、移除、移动、复制、创建和查找节点?**

* + （1）创建新节点
  + createDocumentFragment() //创建一个DOM片段
  + createElement() //创建一个具体的元素
  + createTextNode() //创建一个文本节点
  + （2）添加、移除、替换、插入
  + appendChild()
  + removeChild()
  + replaceChild()
  + insertBefore() //在已有的子节点前插入一个新的子节点
  + （3）查找
  + getElementsByTagName() //通过标签名称
  + getElementsByName() // 通过元素的Name属性的值(IE容错能力较强，会得到一个数组，其中包括id等于name值的)
  + getElementById() //通过元素Id，唯一性

**cookie 和session 的区别：**

* 1、cookie数据存放在客户的浏览器上，session数据放在服务器上。
* 2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗
  + 考虑到安全应当使用session。
* 3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能
  + 考虑到减轻服务器性能方面，应当使用COOKIE。
* 4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。
* 5、所以个人建议：
  + 将登陆信息等重要信息存放为SESSION
  + 其他信息如果需要保留，可以放在COOKIE中

**数组的创建及操作方法**

//字面量方法

var ary1=[];//建个空数组

var ary2=[1,2,3];//建个有值的数组

//构造函数的方法

var ary3=new Array()//建个空数组

var ary4=new Array(1,2,3)//建个有值的数组

var ary5=new Array(3) //建个预知长度的数组 元素都为 undefined

//pop() 方法将删除 arrayObject 的最后一个元素，把数组长度减 1，并且返回它删除的元素的值。如果数组已经为空，则 pop() 不改变数组，并返回 undefined 值。

//用法：arrayObject.pop()

　　var arr = [1,2,3]

　　console.log(arr)//[1,2,3]

　　console.log(arr.pop())//3

　　console.log(arr)//[1,2]

//push() 直接操作原数组，向数组的末尾添加一个或更多元素，并返回新的长度。

//reverse() 颠倒数组中元素的顺序。返回翻转后的数组

//shift() 删除并返回数组的第一个元素

//sort() 对数组的元素进行排序

　　//用法： arrayObject.sort(sortby) 在原数组上进行排序,返回排序后的新数组

//splice() 删除元素，并向数组添加新元素。

　　//用法： arrayObject.splice(index,howmany,item1,.....,itemX) 删除从 index 处开始的零个或多个元素，并且用参数列表中声明的一个或多个值来替换那些被删除的元素。

　　//如果从 arrayObject 中删除了元素，则返回的是含有被删除的元素的数组。

//unshift() 向数组的开头添加一个或更多元素，并返回新的长度。

//用法：arrayObject.unshift(newelement1,newelement2,....,newelementX) 将把它的参数插入 arrayObject 的头部，并将已经存在的元素顺次地移到较高的下标处，以便留出空间。该方法的第一个参数将成为数组的新元素 0，如果还有第二个参数，它将成为新的元素 1，以此类推。

**Jquery选择器**

jQuery 元素选择器

jQuery 使用 CSS 选择器来选取 HTML 元素。

$("p") 选取 <p> 元素。

$("p.intro") 选取所有 class="intro" 的 <p> 元素。

$("p#demo") 选取所有 id="demo" 的 <p> 元素。

jQuery 属性选择器

jQuery 使用 XPath 表达式来选择带有给定属性的元素。

$("[href]") 选取所有带有 href 属性的元素。

$("[href='#']") 选取所有带有 href 值等于 "#" 的元素。

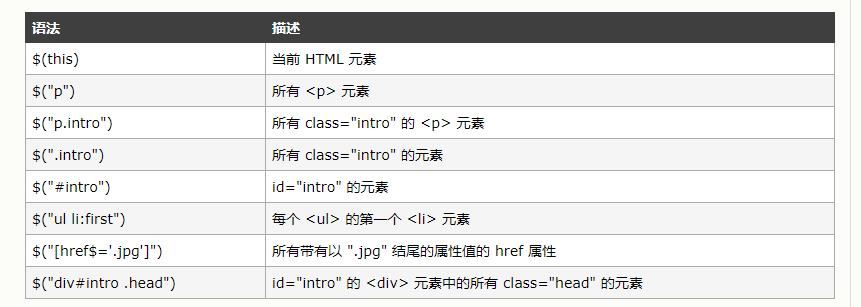
$("[href!='#']") 选取所有带有 href 值不等于 "#" 的元素。

$("[href$='.jpg']") 选取所有 href 值以 ".jpg" 结尾的元素。

## jQuery CSS 选择器

jQuery CSS 选择器可用于改变 HTML 元素的 CSS 属性。

下面的例子把所有 p 元素的背景颜色更改为红色：



## jQuery DOM 操作

## 获得内容 - text()、html() 以及 val()

三个简单实用的用于 DOM 操作的 jQuery 方法：

* text() - 设置或返回所选元素的文本内容
* html() - 设置或返回所选元素的内容（包括 HTML 标记）
* val() - 设置或返回表单字段的值

下面的例子演示如何通过 jQuery text() 和 html() 方法来获得内容：

### 实例

$("#btn1").click(function(){

alert("Text: " + $("#test").text());

});

$("#btn2").click(function(){

alert("HTML: " + $("#test").html());

});

$("#btn1").click(function(){

alert("Value: " + $("#test").val());

});

$("button").click(function(){

alert($("#w3s").attr("href"));

});

# jQuery - 设置内容和属性

* text() - 设置或返回所选元素的文本内容
* html() - 设置或返回所选元素的内容（包括 HTML 标记）
* val() - 设置或返回表单字段的值
* $("#btn1").click(function(){
* $("#test1").text("Hello world!");
* });
* $("#btn2").click(function(){
* $("#test2").html("<b>Hello world!</b>");
* });
* $("#btn3").click(function(){
* $("#test3").val("Dolly Duck");
* });

# jQuery - 添加元素

* 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
* $("p").append("Some appended text.");

$("p").prepend("Some prepended text.");

## 通过 append() 和 prepend() 方法添加若干新元素

function appendText()

{

var txt1="<p>Text.</p>"; // 以 HTML 创建新元素

var txt2=$("<p></p>").text("Text."); // 以 jQuery 创建新元素

var txt3=document.createElement("p"); // 以 DOM 创建新元素

txt3.innerHTML="Text.";

$("p").append(txt1,txt2,txt3); // 追加新元素

}

## jQuery after() 和 before() 方法

jQuery after() 方法在被选元素之后插入内容。

jQuery before() 方法在被选元素之前插入内容。

$("img").after("Some text after");

$("img").before("Some text before");

# jQuery - 删除元素

* remove() - 删除被选元素（及其子元素）
* empty() - 从被选元素中删除子元素

$("#div1").remove();

$("#div1").empty();

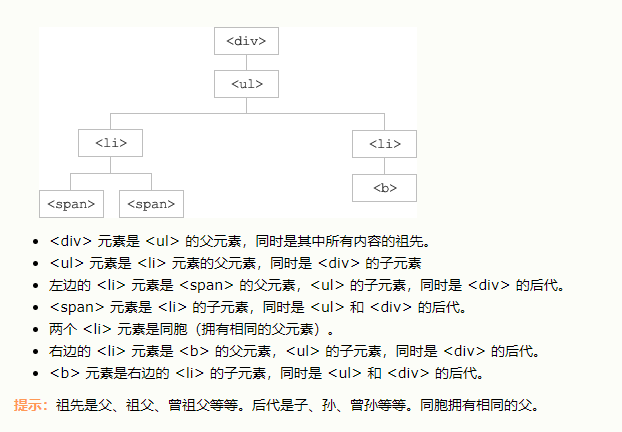
$("p").remove(".italic");

# jQuery 遍历

## 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。



## 向上遍历 DOM 树

这些 jQuery 方法很有用，它们用于向上遍历 DOM 树：

* parent()
* parents()
* parentsUntil()
* $(document).ready(function(){
* $("span").parent();
* });

## 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>)。

下面的例子返回所有 <span> 元素的所有祖先：

$(document).ready(function(){

$("span").parents();

});

下面的例子返回所有 <span> 元素的所有祖先，并且它是 <ul> 元素：

$(document).ready(function(){

$("span").parents("ul");

});

## jQuery 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。

下面的例子返回介于 <span> 与 <div> 元素之间的所有祖先元素：

$(document).ready(function(){

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

});

## 向下遍历 DOM 树

下面是两个用于向下遍历 DOM 树的 jQuery 方法：

* children()
* find()

## jQuery 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。

下面的例子返回每个 <div> 元素的所有直接子元素：

$(document).ready(function(){

$("div").children();

});

下面的例子返回类名为 "1" 的所有 <p> 元素，并且它们是 <div> 的直接子元素：

$(document).ready(function(){

$("div").children("p.1");

});

## jQuery 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。

下面的例子返回属于 <div> 后代的所有 <span> 元素：

$(document).ready(function(){

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

});

# jQuery 遍历 - 同胞

## 在 DOM 树中水平遍历

有许多有用的方法让我们在 DOM 树进行水平遍历：

* siblings()
* next()
* nextAll()
* nextUntil()
* prev()
* prevAll()
* prevUntil()

## jQuery siblings() 方法

siblings() 方法返回被选元素的所有同胞元素。

下面的例子返回 <h2> 的所有同胞元素：

### 实例

$(document).ready(function(){

$("h2").siblings();

});

下面的例子返回属于 <h2> 的同胞元素的所有 <p> 元素：

### 实例

$(document).ready(function(){

$("h2").siblings("p");

});

## jQuery next() 方法

next() 方法返回被选元素的下一个同胞元素。

该方法只返回一个元素。

下面的例子返回 <h2> 的下一个同胞元素：

### 实例

$(document).ready(function(){

$("h2").next();

});

## jQuery nextAll() 方法

nextAll() 方法返回被选元素的所有跟随的同胞元素。

下面的例子返回 <h2> 的所有跟随的同胞元素：

### 实例

$(document).ready(function(){

$("h2").nextAll();

});

# jQuery 遍历 - 过滤

三个最基本的过滤方法是：first(), last() 和 eq()，它们允许您基于其在一组元素中的位置来选择一个特定的元素。

## jQuery filter() 方法

filter() 方法允许您规定一个标准。不匹配这个标准的元素会被从集合中删除，匹配的元素会被返回。

## jQuery not() 方法

not() 方法返回不匹配标准的所有元素。