# 今日内容

1. JavaScript基础

## JavaScript：

\* 概念： 一门客户端脚本语言

\* 运行在客户端浏览器中的。每一个浏览器都有JavaScript的解析引擎

\* 脚本语言：不需要编译，直接就可以被浏览器解析执行了

\* 功能：

\* 可以来增强用户和html页面的交互过程，可以来控制html元素，让页面有一些动态的效果，增强用户的体验。

\* JavaScript发展史：

1. 1992年，Nombase公司，开发出第一门客户端脚本语言，专门用于表单的校验。命名为 ： C-- ，后来更名为：ScriptEase

2. 1995年，Netscape(网景)公司，开发了一门客户端脚本语言：LiveScript。后来，请来SUN公司的专家，修改LiveScript，命名为JavaScript

3. 1996年，微软抄袭JavaScript开发出JScript语言

4. 1997年，ECMA(欧洲计算机制造商协会)，制定出客户端脚本语言的标准：ECMAScript，就是统一了所有客户端脚本语言的编码方式。

\* JavaScript = ECMAScript + JavaScript自己特有的东西(BOM+DOM)

\* ECMAScript：客户端脚本语言的标准

1. 基本语法：

1. 与html结合方式

1. 内部JS：

\* 定义<script>，标签体内容就是js代码

2. 外部JS：

\* 定义<script>，通过src属性引入外部的js文件

\* 注意：

1. <script>可以定义在html页面的任何地方。但是定义的位置会影响执行顺序。

2. <script>可以定义多个。

2. 注释

1. 单行注释：//注释内容

2. 多行注释：/\*注释内容\*/

3. 数据类型：

1. 原始数据类型(基本数据类型)：

1. number：数字。 整数/小数/NaN(not a number 一个不是数字的数字类型)

2. string：字符串。 字符串 "abc" "a" 'abc'

3. boolean:

4. null：一个对象为空的占位符

5. undefined：未定义。如果一个变量没有给初始化值，则会被默认赋值为undefined

2. 引用数据类型：对象

4. 变量

\* 变量：一小块存储数据的内存空间

\* Java语言是强类型语言，而JavaScript是弱类型语言。

\* 强类型：在开辟变量存储空间时，定义了空间将来存储的数据的数据类型。只能存储固定类型的数据

\* 弱类型：在开辟变量存储空间时，不定义空间将来的存储数据类型，可以存放任意类型的数据。

\* 语法：

\* var 变量名 = 初始化值;

\* typeof运算符：获取变量的类型。

\* 注：null运算后得到的是object

5. 运算符

1. 一元运算符：只有一个运算数的运算符

++，-- ， +(正号)

\* ++ --: 自增(自减)

\* ++(--) 在前，先自增(自减)，再运算

\* ++(--) 在后，先运算，再自增(自减)

\* +(-)：正负号

\* 注意：在JS中，如果运算数不是运算符所要求的类型，那么js引擎会自动的将运算数进行类型转换

\* 其他类型转number：

\* string转number：按照字面值转换。如果字面值不是数字，则转为NaN（不是数字的数字）

\* boolean转number：true转为1，false转为0

2. 算数运算符

+ - \* / % ...

3. 赋值运算符

= += -+....

4. 比较运算符

> < >= <= == ===(全等于)

\* 比较方式

1. 类型相同：直接比较

\* 字符串：按照字典顺序比较。按位逐一比较，直到得出大小为止。

2. 类型不同：先进行类型转换，再比较

\* ===：全等于。在比较之前，先判断类型，如果类型不一样，则直接返回false

5. 逻辑运算符

&& || !

\* 其他类型转boolean：

1. number：0或NaN为假，其他为真

2. string：除了空字符串("")，其他都是true

3. null&undefined:都是false

4. 对象：所有对象都为true

6. 三元运算符

? : 表达式

var a = 3;

var b = 4;

var c = a > b ? 1:0;

\* 语法：

\* 表达式? 值1:值2;

\* 判断表达式的值，如果是true则取值1，如果是false则取值2；

6. 流程控制语句：

1. if...else...

2. switch:

\* 在java中，switch语句可以接受的数据类型： byte int shor char,枚举(1.5) ,String(1.7)

\* switch(变量):

case 值:

\* 在JS中,switch语句可以接受任意的原始数据类型

3. while

4. do...while

5. for

7. JS特殊语法：

1. 语句以;结尾，如果一行只有一条语句则 ;可以省略 (不建议)

2. 变量的定义使用var关键字，也可以不使用

\* 用： 定义的变量是局部变量

\* 不用：定义的变量是全局变量(不建议)

8. 练习：99乘法表

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>99乘法表</title>

<style>

td{

border: 1px solid;

}

</style>

<script>

document.write("<table align='center'>");

//1.完成基本的for循环嵌套，展示乘法表

for (var i = 1; i <= 9 ; i++) {

document.write("<tr>");

for (var j = 1; j <=i ; j++) {

document.write("<td>");

//输出 1 \* 1 = 1

document.write(i + " \* " + j + " = " + ( i\*j) +"&nbsp;&nbsp;&nbsp;");

document.write("</td>");

}

/\*//输出换行

document.write("<br>");\*/

document.write("</tr>");

}

//2.完成表格嵌套

document.write("</table>");

</script>

</head>

<body>

</body>

</html>

2. 基本对象：

1. Function：函数(方法)对象

1. 创建：

1. var fun = new Function(形式参数列表,方法体); //忘掉吧

2.

function 方法名称(形式参数列表){

方法体

}

3.

var 方法名 = function(形式参数列表){

方法体

}

2. 方法：

3. 属性：

length:代表形参的个数

4. 特点：

1. 方法定义是，形参的类型不用写,返回值类型也不写。

2. 方法是一个对象，如果定义名称相同的方法，会覆盖

3. 在JS中，方法的调用只与方法的名称有关，和参数列表无关

4. 在方法声明中有一个隐藏的内置对象（数组），arguments,封装所有的实际参数

5. 调用：

方法名称(实际参数列表);

2. Array:数组对象

1. 创建：

1. var arr = new Array(元素列表);

2. var arr = new Array(默认长度);

3. var arr = [元素列表];

2. 方法

join(参数):将数组中的元素按照指定的分隔符拼接为字符串

push() 向数组的末尾添加一个或更多元素，并返回新的长度。

3. 属性

length:数组的长度

4. 特点：

1. JS中，数组元素的类型可变的。

2. JS中，数组长度可变的。

3. Boolean

4. Date：日期对象

1. 创建：

var date = new Date();

2. 方法：

toLocaleString()：返回当前date对象对应的时间本地字符串格式

getTime():获取毫秒值。返回当前如期对象描述的时间到1970年1月1日零点的毫秒值差

5. Math：数学对象

1. 创建：

\* 特点：Math对象不用创建，直接使用。 Math.方法名();

2. 方法：

random():返回 0 ~ 1 之间的随机数。 含0不含1

ceil(x)：对数进行上舍入。

floor(x)：对数进行下舍入。

round(x)：把数四舍五入为最接近的整数。

3. 属性：

PI

6. Number

7. String

8. RegExp：正则表达式对象

1. 正则表达式：定义字符串的组成规则。

1. 单个字符:[]

如： [a] [ab] [a-zA-Z0-9\_]

\* 特殊符号代表特殊含义的单个字符:

\d:单个数字字符 [0-9]

\w:单个单词字符[a-zA-Z0-9\_]

2. 量词符号：

?：表示出现0次或1次

\*：表示出现0次或多次

+：出现1次或多次

{m,n}:表示 m<= 数量 <= n

\* m如果缺省： {,n}:最多n次

\* n如果缺省：{m,} 最少m次

3. 开始结束符号

\* ^:开始

\* $:结束

2. 正则对象：

1. 创建

1. var reg = new RegExp("正则表达式");

2. var reg = /正则表达式/;

2. 方法

1. test(参数):验证指定的字符串是否符合正则定义的规范

9. Global

1. 特点：全局对象，这个Global中封装的方法不需要对象就可以直接调用。 方法名();

2. 方法：

encodeURI():url编码

decodeURI():url解码

encodeURIComponent():url编码,编码的字符更多

decodeURIComponent():url解码

parseInt():将字符串转为数字

\* 逐一判断每一个字符是否是数字，直到不是数字为止，将前边数字部分转为number

isNaN():判断一个值是否是NaN

\* NaN六亲不认，连自己都不认。NaN参与的==比较全部问false

eval():讲 JavaScript 字符串，并把它作为脚本代码来执行。

3. URL编码

传智播客 = %E4%BC%A0%E6%99%BA%E6%92%AD%E5%AE%A2

\* BOM

\* DOM

# 今日内容：

1. JavaScript：

1. ECMAScript：

2. BOM：

3. DOM：

1. 事件

## DOM简单学习：为了满足案例要求

\* 功能：控制html文档的内容

\* 获取页面标签(元素)对象：Element

\* document.getElementById("id值"):通过元素的id获取元素对象

\* 操作Element对象：

1. 修改属性值：

1. 明确获取的对象是哪一个？

2. 查看API文档，找其中有哪些属性可以设置

2. 修改标签体内容：

\* 属性：innerHTML

1. 获取元素对象

2. 使用innerHTML属性修改标签体内容

## 事件简单学习

\* 功能： 某些组件被执行了某些操作后，触发某些代码的执行。

\* 造句： xxx被xxx,我就xxx

\* 我方水晶被摧毁后，我就责备对友。

\* 敌方水晶被摧毁后，我就夸奖自己。

\* 如何绑定事件

1. 直接在html标签上，指定事件的属性(操作)，属性值就是js代码

1. 事件：onclick--- 单击事件

2. 通过js获取元素对象，指定事件属性，设置一个函数

\* 代码：

<body>

<img id="light" src="img/off.gif" onclick="fun();">

<img id="light2" src="img/off.gif">

<script>

function fun(){

alert('我被点了');

alert('我又被点了');

}

function fun2(){

alert('咋老点我？');

}

//1.获取light2对象

var light2 = document.getElementById("light2");

//2.绑定事件

light2.onclick = fun2;

</script>

</body>

\* 案例1：电灯开关

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>电灯开关</title>

</head>

<body>

<img id="light" src="img/off.gif">

<script>

/\*

分析：

1.获取图片对象

2.绑定单击事件

3.每次点击切换图片

\* 规则：

\* 如果灯是开的 on,切换图片为 off

\* 如果灯是关的 off,切换图片为 on

\* 使用标记flag来完成

\*/

//1.获取图片对象

var light = document.getElementById("light");

var flag = false;//代表灯是灭的。 off图片

//2.绑定单击事件

light.onclick = function(){

if(flag){//判断如果灯是开的，则灭掉

light.src = "img/off.gif";

flag = false;

}else{

//如果灯是灭的，则打开

light.src = "img/on.gif";

flag = true;

}

}

</script>

</body>

</html>

# BOM:

1. 概念：Browser Object Model 浏览器对象模型

\* 将浏览器的各个组成部分封装成对象。

2. 组成：

\* Window：窗口对象

\* Navigator：浏览器对象

\* Screen：显示器屏幕对象

\* History：历史记录对象

\* Location：地址栏对象

3. Window：窗口对象

1. 创建

2. 方法

1. 与弹出框有关的方法：

alert() 显示带有一段消息和一个确认按钮的警告框。

confirm() 显示带有一段消息以及确认按钮和取消按钮的对话框。

\* 如果用户点击确定按钮，则方法返回true

\* 如果用户点击取消按钮，则方法返回false

prompt() 显示可提示用户输入的对话框。

\* 返回值：获取用户输入的值

2. 与打开关闭有关的方法：

close() 关闭浏览器窗口。

\* 谁调用我 ，我关谁

open() 打开一个新的浏览器窗口

\* 返回新的Window对象

3. 与定时器有关的方式

setTimeout() 在指定的毫秒数后调用函数或计算表达式。

\* 参数：

1. js代码或者方法对象

2. 毫秒值

\* 返回值：唯一标识，用于取消定时器

clearTimeout() 取消由 setTimeout() 方法设置的 timeout。

setInterval() 按照指定的周期（以毫秒计）来调用函数或计算表达式。

clearInterval() 取消由 setInterval() 设置的 timeout。

3. 属性：

1. 获取其他BOM对象：

history

location

Navigator

Screen:

2. 获取DOM对象

document

4. 特点

\* Window对象不需要创建可以直接使用 window使用。 window.方法名();

\* window引用可以省略。 方法名();

4. Location：地址栏对象

1. 创建(获取)：

1. window.location

2. location

2. 方法：

\* reload() 重新加载当前文档。刷新

3. 属性

\* href 设置或返回完整的 URL。

5. History：历史记录对象

1. 创建(获取)：

1. window.history

2. history

2. 方法：

\* back() 加载 history 列表中的前一个 URL。

\* forward() 加载 history 列表中的下一个 URL。

\* go(参数) 加载 history 列表中的某个具体页面。

\* 参数：

\* 正数：前进几个历史记录

\* 负数：后退几个历史记录

3. 属性：

\* length 返回当前窗口历史列表中的 URL 数量。

## DOM：

\* 概念： Document Object Model 文档对象模型

\* 将标记语言文档的各个组成部分，封装为对象。可以使用这些对象，对标记语言文档进行CRUD的动态操作

\* W3C DOM 标准被分为 3 个不同的部分：

\* 核心 DOM - 针对任何结构化文档的标准模型

\* Document：文档对象

\* Element：元素对象

\* Attribute：属性对象

\* Text：文本对象

\* Comment:注释对象

\* Node：节点对象，其他5个的父对象

\* XML DOM - 针对 XML 文档的标准模型

\* HTML DOM - 针对 HTML 文档的标准模型

\* 核心DOM模型：

\* Document：文档对象

1. 创建(获取)：在html dom模型中可以使用window对象来获取

1. window.document

2. document

2. 方法：

1. 获取Element对象：

1. getElementById() ： 根据id属性值获取元素对象。id属性值一般唯一

2. getElementsByTagName()：根据元素名称获取元素对象们。返回值是一个数组

3. getElementsByClassName():根据Class属性值获取元素对象们。返回值是一个数组

4. getElementsByName(): 根据name属性值获取元素对象们。返回值是一个数组

2. 创建其他DOM对象：

createAttribute(name)

createComment()

createElement()

createTextNode()

3. 属性

\* Element：元素对象

1. 获取/创建：通过document来获取和创建

2. 方法：

1. removeAttribute()：删除属性

2. setAttribute()：设置属性

\* Node：节点对象，其他5个的父对象

\* 特点：所有dom对象都可以被认为是一个节点

\* 方法：

\* CRUD dom树：

\* appendChild()：向节点的子节点列表的结尾添加新的子节点。

\* removeChild() ：删除（并返回）当前节点的指定子节点。

\* replaceChild()：用新节点替换一个子节点。

\* 属性：

\* parentNode 返回节点的父节点。

\* HTML DOM

1. 标签体的设置和获取：innerHTML

2. 使用html元素对象的属性

3. 控制元素样式

1. 使用元素的style属性来设置

如：

//修改样式方式1

div1.style.border = "1px solid red";

div1.style.width = "200px";

//font-size--> fontSize

div1.style.fontSize = "20px";

2. 提前定义好类选择器的样式，通过元素的className属性来设置其class属性值。

## 事件监听机制：

\* 概念：某些组件被执行了某些操作后，触发某些代码的执行。

\* 事件：某些操作。如： 单击，双击，键盘按下了，鼠标移动了

\* 事件源：组件。如： 按钮 文本输入框...

\* 监听器：代码。

\* 注册监听：将事件，事件源，监听器结合在一起。 当事件源上发生了某个事件，则触发执行某个监听器代码。

\* 常见的事件：

1. 点击事件：

1. onclick：单击事件

2. ondblclick：双击事件

2. 焦点事件

1. onblur：失去焦点

2. onfocus:元素获得焦点。

3. 加载事件：

1. onload：一张页面或一幅图像完成加载。

4. 鼠标事件：

1. onmousedown 鼠标按钮被按下。

2. onmouseup 鼠标按键被松开。

3. onmousemove 鼠标被移动。

4. onmouseover 鼠标移到某元素之上。

5. onmouseout 鼠标从某元素移开。

5. 键盘事件：

1. onkeydown 某个键盘按键被按下。

2. onkeyup 某个键盘按键被松开。

3. onkeypress 某个键盘按键被按下并松开。

6. 选择和改变

1. onchange 域的内容被改变。

2. onselect 文本被选中。

7. 表单事件：

1. onsubmit 确认按钮被点击。

2. onreset 重置按钮被点击。

## 今日内容

1. JQuery 基础：

1. 概念

2. 快速入门

3. JQuery对象和JS对象区别与转换

4. 选择器

5. DOM操作

6. 案例

# JQuery 基础：

1. 概念： 一个JavaScript框架。简化JS开发

\* jQuery是一个快速、简洁的JavaScript框架，是继Prototype之后又一个优秀的JavaScript代码库（或JavaScript框架）。jQuery设计的宗旨 是“write Less，Do More”，即倡导写更少的代码，做更多的事情。它封装JavaScript常用的功能代码，提供一种简便的JavaScript设计模式，优 化HTML文档操作、事件处理、动画设计和Ajax交互。

\* JavaScript框架：本质上就是一些js文件，封装了js的原生代码而已

2. 快速入门

1. 步骤：

1. 下载JQuery

\* 目前jQuery有三个大版本：

1.x：兼容ie678,使用最为广泛的，官方只做BUG维护，

功能不再新增。因此一般项目来说，使用1.x版本就可以了，

最终版本：1.12.4 (2016年5月20日)

2.x：不兼容ie678，很少有人使用，官方只做BUG维护，

功能不再新增。如果不考虑兼容低版本的浏览器可以使用2.x，

最终版本：2.2.4 (2016年5月20日)

3.x：不兼容ie678，只支持最新的浏览器。除非特殊要求，

一般不会使用3.x版本的，很多老的jQuery插件不支持这个版本。

目前该版本是官方主要更新维护的版本。最新版本：3.2.1（2017年3月20日）

\* jquery-xxx.js 与 jquery-xxx.min.js区别：

1. jquery-xxx.js：开发版本。给程序员看的，有良好的缩进和注释。体积大一些

2. jquery-xxx.min.js：生产版本。程序中使用，没有缩进。体积小一些。程序加载更快

2. 导入JQuery的js文件：导入min.js文件

3. 使用

var div1 = $("#div1");

alert(div1.html());

3. JQuery对象和JS对象区别与转换

1. JQuery对象在操作时，更加方便。

2. JQuery对象和js对象方法不通用的.

3. 两者相互转换

\* jq -- > js : jq对象[索引] 或者 jq对象.get(索引)

\* js -- > jq : $(js对象)

4. 选择器：筛选具有相似特征的元素(标签)

1. 基本操作学习：

1. 事件绑定

//1.获取b1按钮

$("#b1").click(function(){

alert("abc");

});

2. 入口函数

$(function () {

});

window.onload 和 $(function) 区别

\* window.onload 只能定义一次,如果定义多次，后边的会将前边的覆盖掉

\* $(function)可以定义多次的。

3. 样式控制：css方法

// $("#div1").css("background-color","red");

$("#div1").css("backgroundColor","pink");

2. 分类

1. 基本选择器

1. 标签选择器（元素选择器）

\* 语法： $("html标签名") 获得所有匹配标签名称的元素

2. id选择器

\* 语法： $("#id的属性值") 获得与指定id属性值匹配的元素

3. 类选择器

\* 语法： $(".class的属性值") 获得与指定的class属性值匹配的元素

4. 并集选择器：

\* 语法： $("选择器1,选择器2....") 获取多个选择器选中的所有元素

2. 层级选择器

1. 后代选择器

\* 语法： $("A B ") 选择A元素内部的所有B元素

2. 子选择器

\* 语法： $("A > B") 选择A元素内部的所有B子元素

3. 属性选择器

1. 属性名称选择器

\* 语法： $("A[属性名]") 包含指定属性的选择器

2. 属性选择器

\* 语法： $("A[属性名='值']") 包含指定属性等于指定值的选择器

3. 复合属性选择器

\* 语法： $("A[属性名='值'][]...") 包含多个属性条件的选择器

4. 过滤选择器

1. 首元素选择器

\* 语法： :first 获得选择的元素中的第一个元素

2. 尾元素选择器

\* 语法： :last 获得选择的元素中的最后一个元素

3. 非元素选择器

\* 语法： :not(selector) 不包括指定内容的元素

4. 偶数选择器

\* 语法： :even 偶数，从 0 开始计数

5. 奇数选择器

\* 语法： :odd 奇数，从 0 开始计数

6. 等于索引选择器

\* 语法： :eq(index) 指定索引元素

7. 大于索引选择器

\* 语法： :gt(index) 大于指定索引元素

8. 小于索引选择器

\* 语法： :lt(index) 小于指定索引元素

9. 标题选择器

\* 语法： :header 获得标题（h1~h6）元素，固定写法

5. 表单过滤选择器

1. 可用元素选择器

\* 语法： :enabled 获得可用元素

2. 不可用元素选择器

\* 语法： :disabled 获得不可用元素

3. 选中选择器

\* 语法： :checked 获得单选/复选框选中的元素

4. 选中选择器

\* 语法： :selected 获得下拉框选中的元素

5. DOM操作

1. 内容操作

1. html(): 获取/设置元素的标签体内容 <a><font>内容</font></a> --> <font>内容</font>

2. text(): 获取/设置元素的标签体纯文本内容 <a><font>内容</font></a> --> 内容

3. val()： 获取/设置元素的value属性值

2. 属性操作

1. 通用属性操作

1. attr(): 获取/设置元素的属性

2. removeAttr():删除属性

3. prop():获取/设置元素的属性

4. removeProp():删除属性

\* attr和prop区别？

1. 如果操作的是元素的固有属性，则建议使用prop

2. 如果操作的是元素自定义的属性，则建议使用attr

2. 对class属性操作

1. addClass():添加class属性值

2. removeClass():删除class属性值

3. toggleClass():切换class属性

\* toggleClass("one"):

\* 判断如果元素对象上存在class="one"，则将属性值one删除掉。 如果元素对象上不存在class="one"，则添加

4. css():

3. CRUD操作:

1. append():父元素将子元素追加到末尾

\* 对象1.append(对象2): 将对象2添加到对象1元素内部，并且在末尾

2. prepend():父元素将子元素追加到开头

\* 对象1.prepend(对象2):将对象2添加到对象1元素内部，并且在开头

3. appendTo():

\* 对象1.appendTo(对象2):将对象1添加到对象2内部，并且在末尾

4. prependTo()：

\* 对象1.prependTo(对象2):将对象1添加到对象2内部，并且在开头

5. after():添加元素到元素后边

\* 对象1.after(对象2)： 将对象2添加到对象1后边。对象1和对象2是兄弟关系

6. before():添加元素到元素前边

\* 对象1.before(对象2)： 将对象2添加到对象1前边。对象1和对象2是兄弟关系

7. insertAfter()

\* 对象1.insertAfter(对象2)：将对象2添加到对象1后边。对象1和对象2是兄弟关系

8. insertBefore()

\* 对象1.insertBefore(对象2)： 将对象2添加到对象1前边。对象1和对象2是兄弟关系

9. remove():移除元素

\* 对象.remove():将对象删除掉

10. empty():清空元素的所有后代元素。

\* 对象.empty():将对象的后代元素全部清空，但是保留当前对象以及其属性节点

## 今日内容：

1. JQuery 高级

1. 动画

2. 遍历

3. 事件绑定

4. 案例

5. 插件

## JQuery 高级

1. 动画

1. 三种方式显示和隐藏元素

1. 默认显示和隐藏方式

1. show([speed,[easing],[fn]])

1. 参数：

1. speed：动画的速度。三个预定义的值("slow","normal", "fast")或表示动画时长的毫秒数值(如：1000)

2. easing：用来指定切换效果，默认是"swing"，可用参数"linear"

\* swing：动画执行时效果是 先慢，中间快，最后又慢

\* linear：动画执行时速度是匀速的

3. fn：在动画完成时执行的函数，每个元素执行一次。

2. hide([speed,[easing],[fn]])

3. toggle([speed],[easing],[fn])

2. 滑动显示和隐藏方式

1. slideDown([speed],[easing],[fn])

2. slideUp([speed,[easing],[fn]])

3. slideToggle([speed],[easing],[fn])

3. 淡入淡出显示和隐藏方式

1. fadeIn([speed],[easing],[fn])

2. fadeOut([speed],[easing],[fn])

3. fadeToggle([speed,[easing],[fn]])

2. 遍历

1. js的遍历方式

\* for(初始化值;循环结束条件;步长)

2. jq的遍历方式

1. jq对象.each(callback)

1. 语法：

jquery对象.each(function(index,element){});

\* index:就是元素在集合中的索引

\* element：就是集合中的每一个元素对象

\* this：集合中的每一个元素对象

2. 回调函数返回值：

\* true:如果当前function返回为false，则结束循环(break)。

\* false:如果当前function返回为true，则结束本次循环，继续下次循环(continue)

2. $.each(object, [callback])

3. for..of: jquery 3.0 版本之后提供的方式

for(元素对象 of 容器对象)

3. 事件绑定

1. jquery标准的绑定方式

\* jq对象.事件方法(回调函数)；

\* 注：如果调用事件方法，不传递回调函数，则会触发浏览器默认行为。

\* 表单对象.submit();//让表单提交

2. on绑定事件/off解除绑定

\* jq对象.on("事件名称",回调函数)

\* jq对象.off("事件名称")

\* 如果off方法不传递任何参数，则将组件上的所有事件全部解绑

3. 事件切换：toggle

\* jq对象.toggle(fn1,fn2...)

\* 当单击jq对象对应的组件后，会执行fn1.第二次点击会执行fn2.....

\* 注意：1.9版本 .toggle() 方法删除,jQuery Migrate（迁移）插件可以恢复此功能。

<script src="../js/jquery-migrate-1.0.0.js" type="text/javascript" charset="utf-8"></script>

4. 案例

1. 广告显示和隐藏

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>广告的自动显示与隐藏</title>

<style>

#content{width:100%;height:500px;background:#999}

</style>

<!--引入jquery-->

<script type="text/javascript" src="../js/jquery-3.3.1.min.js"></script>

<script>

/\*

需求：

1. 当页面加载完，3秒后。自动显示广告

2. 广告显示5秒后，自动消失。

分析：

1. 使用定时器来完成。setTimeout (执行一次定时器)

2. 分析发现JQuery的显示和隐藏动画效果其实就是控制display

3. 使用 show/hide方法来完成广告的显示

\*/

//入口函数，在页面加载完成之后，定义定时器，调用这两个方法

$(function () {

//定义定时器，调用adShow方法 3秒后执行一次

setTimeout(adShow,3000);

//定义定时器，调用adHide方法，8秒后执行一次

setTimeout(adHide,8000);

});

//显示广告

function adShow() {

//获取广告div，调用显示方法

$("#ad").show("slow");

}

//隐藏广告

function adHide() {

//获取广告div，调用隐藏方法

$("#ad").hide("slow");

}

​

</script>

</head>

<body>

<!-- 整体的DIV -->

<div>

<!-- 广告DIV -->

<div id="ad" style="display: none;">

<img style="width:100%" src="../img/adv.jpg" />

</div>

<!-- 下方正文部分 -->

<div id="content">

正文部分

</div>

</div>

</body>

</html>

2. 抽奖

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>jquery案例之抽奖</title>

<script type="text/javascript" src="../js/jquery-3.3.1.min.js"></script>

<script language='javascript' type='text/javascript'>

/\*

分析：

1. 给开始按钮绑定单击事件

1.1 定义循环定时器

1.2 切换小相框的src属性

\* 定义数组，存放图片资源路径

\* 生成随机数。数组索引

2. 给结束按钮绑定单击事件

1.1 停止定时器

1.2 给大相框设置src属性

\*/

var imgs = ["../img/man00.jpg",

"../img/man01.jpg",

"../img/man02.jpg",

"../img/man03.jpg",

"../img/man04.jpg",

"../img/man05.jpg",

"../img/man06.jpg",

];

var startId;//开始定时器的id

var index;//随机角标

$(function () {

//处理按钮是否可以使用的效果

$("#startID").prop("disabled",false);

$("#stopID").prop("disabled",true);

//1. 给开始按钮绑定单击事件

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

// 1.1 定义循环定时器 20毫秒执行一次

startId = setInterval(function () {

//处理按钮是否可以使用的效果

$("#startID").prop("disabled",true);

$("#stopID").prop("disabled",false);

//1.2生成随机角标 0-6

index = Math.floor(Math.random() \* 7);//0.000--0.999 --> \* 7 --> 0.0-----6.9999

//1.3设置小相框的src属性

$("#img1ID").prop("src",imgs[index]);

},20);

});

//2. 给结束按钮绑定单击事件

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

//处理按钮是否可以使用的效果

$("#startID").prop("disabled",false);

$("#stopID").prop("disabled",true);

// 1.1 停止定时器

clearInterval(startId);

// 1.2 给大相框设置src属性

$("#img2ID").prop("src",imgs[index]).hide();

//显示1秒之后

$("#img2ID").show(1000);

});

});

​

​

</script>

</head>

<body>

<!-- 小像框 -->

<div style="border-style:dotted;width:160px;height:100px">

<img id="img1ID" src="../img/man00.jpg" style="width:160px;height:100px"/>

</div>

<!-- 大像框 -->

<div

style="border-style:double;width:800px;height:500px;position:absolute;left:500px;top:10px">

<img id="img2ID" src="../img/man00.jpg" width="800px" height="500px"/>

</div>

<!-- 开始按钮 -->

<input

id="startID"

type="button"

value="点击开始"

style="width:150px;height:150px;font-size:22px">

<!-- 停止按钮 -->

<input

id="stopID"

type="button"

value="点击停止"

style="width:150px;height:150px;font-size:22px">

</body>

</html>

5. 插件：增强JQuery的功能

1. 实现方式：

1. $.fn.extend(object)

\* 增强通过Jquery获取的对象的功能 $("#id")

2. $.extend(object)

\* 增强JQeury对象自身的功能 $/jQuery

## 今日内容

1. Filter：过滤器

2. Listener：监听器

# Filter：过滤器

1. 概念：

\* 生活中的过滤器：净水器,空气净化器，土匪、

\* web中的过滤器：当访问服务器的资源时，过滤器可以将请求拦截下来，完成一些特殊的功能。

\* 过滤器的作用：

\* 一般用于完成通用的操作。如：登录验证、统一编码处理、敏感字符过滤...

2. 快速入门：

1. 步骤：

1. 定义一个类，实现接口Filter

2. 复写方法

3. 配置拦截路径

1. web.xml

2. 注解

2. 代码：

@WebFilter("/\*")//访问所有资源之前，都会执行该过滤器

public class FilterDemo1 implements Filter {

@Override

public void init(FilterConfig filterConfig) throws ServletException {

}

@Override

public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {

System.out.println("filterDemo1被执行了....");

//放行

filterChain.doFilter(servletRequest,servletResponse);

}

@Override

public void destroy() {

}

}

3. 过滤器细节：

1. web.xml配置

<filter>

<filter-name>demo1</filter-name>

<filter-class>cn.itcast.web.filter.FilterDemo1</filter-class>

</filter>

<filter-mapping>

<filter-name>demo1</filter-name>

<!-- 拦截路径 -->

<url-pattern>/\*</url-pattern>

</filter-mapping>

2. 过滤器执行流程

1. 执行过滤器

2. 执行放行后的资源

3. 回来执行过滤器放行代码下边的代码

3. 过滤器生命周期方法

1. init:在服务器启动后，会创建Filter对象，然后调用init方法。只执行一次。用于加载资源

2. doFilter:每一次请求被拦截资源时，会执行。执行多次

3. destroy:在服务器关闭后，Filter对象被销毁。如果服务器是正常关闭，则会执行destroy方法。只执行一次。用于释放资源

4. 过滤器配置详解

\* 拦截路径配置：

1. 具体资源路径： /index.jsp 只有访问index.jsp资源时，过滤器才会被执行

2. 拦截目录： /user/\* 访问/user下的所有资源时，过滤器都会被执行

3. 后缀名拦截： \*.jsp 访问所有后缀名为jsp资源时，过滤器都会被执行

4. 拦截所有资源：/\* 访问所有资源时，过滤器都会被执行

\* 拦截方式配置：资源被访问的方式

\* 注解配置：

\* 设置dispatcherTypes属性

1. REQUEST：默认值。浏览器直接请求资源

2. FORWARD：转发访问资源

3. INCLUDE：包含访问资源

4. ERROR：错误跳转资源

5. ASYNC：异步访问资源

\* web.xml配置

\* 设置<dispatcher></dispatcher>标签即可

5. 过滤器链(配置多个过滤器)

\* 执行顺序：如果有两个过滤器：过滤器1和过滤器2

1. 过滤器1

2. 过滤器2

3. 资源执行

4. 过滤器2

5. 过滤器1

\* 过滤器先后顺序问题：

1. 注解配置：按照类名的字符串比较规则比较，值小的先执行

\* 如： AFilter 和 BFilter，AFilter就先执行了。

2. web.xml配置： <filter-mapping>谁定义在上边，谁先执行

4. 案例：

1. 案例1\_登录验证

\* 需求：

1. 访问day17\_case案例的资源。验证其是否登录

2. 如果登录了，则直接放行。

3. 如果没有登录，则跳转到登录页面，提示"您尚未登录，请先登录"。

2. 案例2\_敏感词汇过滤

\* 需求：

1. 对day17\_case案例录入的数据进行敏感词汇过滤

2. 敏感词汇参考《敏感词汇.txt》

3. 如果是敏感词汇，替换为 \*\*\*

\* 分析：

1. 对request对象进行增强。增强获取参数相关方法

2. 放行。传递代理对象

\* 增强对象的功能：

\* 设计模式：一些通用的解决固定问题的方式

1. 装饰模式

2. 代理模式

\* 概念：

1. 真实对象：被代理的对象

2. 代理对象：

3. 代理模式：代理对象代理真实对象，达到增强真实对象功能的目的

\* 实现方式：

1. 静态代理：有一个类文件描述代理模式

2. 动态代理：在内存中形成代理类

\* 实现步骤：

1. 代理对象和真实对象实现相同的接口

2. 代理对象 = Proxy.newProxyInstance();

3. 使用代理对象调用方法。

4. 增强方法

\* 增强方式：

1. 增强参数列表

2. 增强返回值类型

3. 增强方法体执行逻辑

## Listener：监听器

\* 概念：web的三大组件之一。

\* 事件监听机制

\* 事件 ：一件事情

\* 事件源 ：事件发生的地方

\* 监听器 ：一个对象

\* 注册监听：将事件、事件源、监听器绑定在一起。 当事件源上发生某个事件后，执行监听器代码

\* ServletContextListener:监听ServletContext对象的创建和销毁

\* 方法：

\* void contextDestroyed(ServletContextEvent sce) ：ServletContext对象被销毁之前会调用该方法

\* void contextInitialized(ServletContextEvent sce) ：ServletContext对象创建后会调用该方法

\* 步骤：

1. 定义一个类，实现ServletContextListener接口

2. 复写方法

3. 配置

1. web.xml

<listener>

<listener-class>cn.itcast.web.listener.ContextLoaderListener</listener-class>

</listener>

\* 指定初始化参数<context-param>

2. 注解：

\* @WebListener