**2012-03-06 Edited by: xuzhezhao Last edited:**

**JavaScript 学习笔记**

# JavaScript How To

Scripts in HTML must be inserted between **<script>** and **</script>** tags.

Scripts can be put in the **<body>** and in the **<head>** section of an HTML page.

## JavaScript in <body>

In this example, JavaScript writes into the HTML **<body>** while the page loads:

<!DOCTYPE html>

<html>

<body>

<script>

document.write("<h1>This is a heading</h1>");

document.write("<p>This is a paragraph</p>");

</script>

</body>

</html>

## JavaScript in <head> or <body>

You can place **an unlimited number of scripts** in an HTML document.

Scripts can be in the **<body>** or in the **<head>** section of HTML, and/or in both.

It is a common practice to put functions in the **<head>** section, or at the bottom of the page. This way they are all in one place and do not interfere with page content.

## External JavaScripts

Scripts can also be placed in **external files**. External files often contain code to be used by several different web pages.

External JavaScript files have the file extension **.js.**

To use an external script, point to the .js file in the "src" attribute of the <script> tag:

<!DOCTYPE html>

<html>

<body>

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

</body>

</html>

You can place the script in the <head> or <body> as you like. The script will behave as if it was located exactly where you put the <script> tag in the document.

**External scripts cannot contain <script> tags.**

# JavaScript Output

## Manipulating HTML Elements

To access an HTML element from JavaScript, you can use the **document.getElementById(id)** method.

Use the "id" attribute to identify the HTML element:

**Example**

Access the HTML element with the **specified id**, and change its content:

<!DOCTYPE html>

<html>

<body>

<h1>My First Web Page</h1>

<p id="demo">My First Paragraph</p>

<script>

document.getElementById("demo").innerHTML="My First JavaScript";

</script>

</body>

</html>

## Writing to The Document Output

The example below writes a **<p>** element directly into the HTML document output:

<!DOCTYPE html>

<html>

<body>

<h1>My First Web Page</h1>

<script>

document.write("<p>My First JavaScript</p>");

</script>

</body>

</html>

## Warning

Use document.write() only to write directly into the document output.

If you execute document.write after the document has finished loading, the entire HTML page will be overwritten:

<!DOCTYPE html>

<html>

<body>

<h1>My First Web Page</h1>

<p>My First Paragraph.</p>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

document.write("Oops! The document disappeared!");

}

</script>

</body>

</html>

# JavaScript Statements

JavaScript statements are **"commands"** to the browser. The purpose of the statements is to tell the browser what to do.

This JavaScript statement tells the browser to write "Hello Dolly" inside an HTML element with id="demo":

document.getElementById("demo").innerHTML="Hello Dolly";

## JavaScript is Case Sensitive

## Break up a Code Line

You can break up a code line within a text string with a backslash. The example below will be displayed properly:

document.write("Hello \

World!");

# JavaScript Comments

* Single line comments start with //.
* Multi line comments start with /\* and end with \*/.

# JavaScript Variables

**Example**

var x=5;

var y=6;

var z=x+y;

## JavaScript Data Types

**Example**

var pi=3.14;

var name="John Doe";

var answer='Yes I am!';

## Declaring (Creating) JavaScript Variables

Creating a variable in JavaScript is most often referred to as "declaring" a variable.

You declare JavaScript variables with the **var** keyword:

var carname;

**NOTE:** It's a good programming practice to declare all the variables you will need, in one place, at the beginning of your code.

## One Statement, Many Variables

You can declare many variables in one statement. Just start the statement with var and separate the variables by comma:

var name="Doe", age=30, job="carpenter";

## Re-Declaring JavaScript Variables

If you **re-declare** a JavaScript variable, it will not lose its value:.

The value of the variable carname will still have the value "Volvo" after the execution of the following two statements:

var carname="Volvo";

var carname;

## JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +:

**Example**

y=5;

x=y+2;

# JavaScript Data Types

**String, Number, Boolean, Array, Object, Null, Undefined.**

## JavaScript Has Dynamic Types

JavaScript has **dynamic types**. This means that the same variable can be used as different types:

**Example**

var x // Now x is undefined

var x = 5; // Now x is a Number

var x = "John"; // Now x is a String

## JavaScript Arrays

The following code creates an Array called cars:

var cars=new Array();

cars[0]="Saab";

cars[1]="Volvo";

cars[2]="BMW";

or (condensed array):

var cars=new Array("Saab","Volvo","BMW");

or (literal array):

**Example**

var cars=["Saab","Volvo","BMW"];

## JavaScript Objects

An object is delimited by curly braces. Inside the braces the object's properties are defined as name and value pairs **(name : value)**. The properties are separated by commas:

var person={firstname:"John", lastname:"Doe", id:5566};

The object (person) in the example above has 3 properties: firstname, lastname, and id.

You can address the object properties in two ways:

**Example**

name=person.lastname;

name=person["lastname"];

## Undefined and Null

Undefined is the value of a variable with no value.

Variables can be emptied by setting the value to null;

Example

cars=null;

person=null;

## Declaring Variable Types

When you declare a new variable, you can declare its type using the "**new**" keyword:

var carname=new String;

var x= new Number;

var y= new Boolean;

var cars= new Array;

var person= new Object;

**JavaScript variables are all objects. When you declare a variable you create a new object.**

# JavaScript Objects

"Everything" in JavaScript is an Object: a String, a Number, an Array, a Date....

In JavaScript, **an object is data, with properties and methods.**

## Properties and Methods

**Properties** are values associated with an object.

**Methods** are actions that can be performed on objects.

## Objects in JavaScript:

In JavaScript, objects are data (variables), with properties and methods.

When you declare a JavaScript variable like this:

var txt = "Hello";

You actually create a JavaScript String object. The String object has a **built-in property** called **length**. For the string above, length has the value 5. The String object also have several **built-in methods**.

## Creating JavaScript Objects

Almost "everything" in JavaScript is an object. Strings, Dates, Arrays, Functions.

You can also create your own objects.

This example creates an object called "person", and adds four properties to it:

**Example**

person=new Object();

person.firstname="John";

person.lastname="Doe";

person.age=50;

person.eyecolor="blue";

## Accessing Object Properties

The syntax for accessing the property of an object is:

objectName.propertyName

This example uses the length property of the String object to find the length of a string:

var message="Hello World!";

var x=message.length;

The value of x, after execution of the code above will be:

12

## Accessing Object Methods

You can call a method with the following syntax:

objectName.methodName()

This example uses the toUpperCase() method of the String object, to convert a text to uppercase:

var message="Hello world!";

var x=message.toUpperCase();

The value of x, after execution of the code above will be:

HELLO WORLD!

# JavaScript Functions

## JavaScript Function Syntax

function functionname()

{

some code to be executed

}

## Calling a Function with Arguments

function myFunction(var1,var2)

{

some code

}

<script>

function myFunction(name,job)

{

alert("Welcome " + name + ", the " + job);

}

</script>

## Functions With a Return Value

function myFunction()

{

var x=5;

return x;

}

## Local JavaScript Variables

## Global JavaScript Variables

## Assigning Values to Undeclared JavaScript Variables

If you assign a value to variable that has not yet been declared, **the variable will automatically be declared as a GLOBAL variable.**

This statement:

carname="Volvo";

will declare the variable carname as a global variable , even if it is executed inside a function.

# JavaScript Operators

## The + Operator Used on Strings

The + operator can also be used to add string variables or text values together.

## Adding Strings and Numbers

Adding two numbers, will return the sum, but **adding a number and a string will return a string:**

**Example**

x=5+5;

y="5"+5;

z="Hello"+5;

The result of x,y, and z will be:

10

55

Hello5

# JavaScript Comparison and Logical Operators

|  |
| --- |
|  |
| **Operator** | **Description** | **Comparing** | **Returns** | **Try it** |
| == | is equal to | x==8 | *false* | [**Try it »**](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison1) |
| x==5  x==”5” | *true* | [**Try it »**](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison2) |
| === | is exactly equal to (value and type) | x==="5" | *false* | [**Try it »**](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison3) |
| x===5 | *true* | [**Try it »**](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison4) |
| != | is not equal | x!=8 | *true* | [**Try it »**](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison5) |
| !== | is not equal (neither value nor type) | x!=="5" | *true* | [**Try it »**](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison6) |
| x!==5 | *false* | [**Try it »**](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison7) |
| > | is greater than | x>8 | *false* | [**Try it »**](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison8) |
| < | is less than | x<8 | *true* | [**Try it »**](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison9) |
| >= | is greater than or equal to | x>=8 | *false* | [**Try it »**](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison10) |
| <= | is less than or equal to | x<=8 | *true* |  |

## Logical Operators

**&&, ||, !**

## Conditional Operato

**(condition)?value1:value2**

# JavaScript If...Else Statements

**The same as C!**

# JavaScript Switch Statement

**The same as C!**

# JavaScript For Loop

## The same as C!

## The For/In Loop

**The JavaScript for/in statement loops through the properties of an object:**

**Example**

var person={fname:"John",lname:"Doe",age:25};

for (x in person)

{

txt=txt + person[x];

}

txt=JohnDoe25

## The While Loop

**The same as C!**

# JavaScript Break and Continue

The **break** statement "jumps out" of **a loop**.

The **continue** statement "jumps over" **one iteration in the loop.**

## JavaScript Labels

As you have already seen, in the chapter about the switch statement, JavaScript statements can be labeled.

To label JavaScript statements you precede the statements with a colon:

label:

statements

The break and the continue statements are the only JavaScript statements that can "jump out of" a code block.

**Syntax:**

break labelname;

continue labelname;

The continue statement (**with or without a label reference**) can only be used inside a loop.

The break statement, **without** a label reference, can only be used inside a loop or a switch.

**With a label reference, it(break) can be used to "jump out of" any JavaScript code block:**

**Example**

cars=["BMW","Volvo","Saab","Ford"];

list:

{

document.write(cars[0] + "<br>");

document.write(cars[1] + "<br>");

document.write(cars[2] + "<br>");

break list;

document.write(cars[3] + "<br>");

document.write(cars[4] + "<br>");

document.write(cars[5] + "<br>");

}

# JavaScript Errors - Throw and Try to Catch

The **try** statement lets you to test a block of code for errors.

The **catch** statement lets you handle the error.

The **throw** statement lets you create custom errors.

## Errors Will Happen!

When the JavaScript engine is executing JavaScript code, different errors can occur:

It can be syntax errors, typically coding errors or typos made by the programmer.

It can be misspelled or missing features in the language (maybe due to browser differences).

It can be errors due to wrong input, from a user, or from an Internet server.

And, of course, it can be many other unforeseeable things.

## JavaScript Throws Errors

When an error occurs, when something goes wrong, the JavaScript engine will normally stop, and generate an error message.

The technical term for this is: JavaScript will **throw** an error.

## JavaScript try and catch

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements try and catch come **in pairs.**

**Syntax**

try

{

//Run some code here

}

catch(err)

{

//Handle errors here

}

## Examples

<!DOCTYPE html>

<html>

<head>

<script>

var txt="";

function message()

{

try

{

adddlert("Welcome guest!");

}

catch(err)

{

txt="There was an error on this page.\n\n";

txt+="Error description: " + err.message + "\n\n";

txt+="Click OK to continue.\n\n";

alert(txt);

}

}

</script>

</head>

<body>

<input type="button" value="View message" onclick="message()">

</body>

</html>

## The Throw Statement

The throw statement allows you to **create a custom error.**

The correct technical term is to create or **throw an exception.**

If you use the throw statement together with try and catch, you can control program flow and generate custom error messages.

**Syntax**

throw exception

The **exception** can be a JavaScript **String, a Number, a Boolean or an Object.**

<script>

function myFunction()

{

try

{

var x=document.getElementById("demo").value;

if(x=="") throw "empty";

if(isNaN(x)) throw "not a number";

if(x>10) throw "too high";

if(x<5) throw "too low";

}

catch(err)

{

var y=document.getElementById("mess");

y.innerHTML="Error: " + err + ".";

}

}

</script>

<h1>My First JavaScript</h1>

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">

<button type="button" onclick="myFunction()">Test Input</button>

<p id="mess"></p>

# JavaScript Form Validation

JavaScript can be used to validate data in HTML forms before sending off the content to a server.

Form data that typically are checked by a JavaScript could be:

has the user left required fields empty?

has the user entered a valid e-mail address?

has the user entered a valid date?

has the user entered text in a numeric field?

## Required Fields

The function below checks if a field has been left empty. If the field is blank, an alert box alerts a message, the function returns false, and the form will not be submitted:

function validateForm()

{

var x=document.forms["myForm"]["fname"].value;

if (x==null || x=="")

{

alert("First name must be filled out");

return false;

}

}

<form name="myForm" action="demo\_form.asp" onsubmit="return validateForm()" method="post">

First name: <input type="text" name="fname">

<input type="submit" value="Submit">

</form>

## E-mail Validation

The function below checks if the content has the general syntax of an email.

This means that the input data must contain an @ sign and at least one dot (.). Also, the @ must not be the first character of the email address, and the last dot must be present after the @ sign, and minimum 2 characters before the end:

function validateForm()

{

var x=document.forms["myForm"]["email"].value;

var atpos=x.indexOf("@");

var dotpos=x.lastIndexOf(".");

if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)

{

alert("Not a valid e-mail address");

return false;

}

}

The function above could be called when a form is submitted:

**Example**

<form name="myForm" action="demo\_form.asp" onsubmit="return validateForm();" method="post">

Email: <input type="text" name="email">

<input type="submit" value="Submit">

</form>

# JavaScript HTML DOM

With the HTML DOM, JavaScript can access all the elements of an HTML document.

## The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The HTML DOM model is constructed as **a tree of Objects:**



With a programmable object model, JavaScript gets all the power it needs to create dynamic HTML:

* JavaScript can change all the HTML elements in the page
* JavaScript can change all the HTML attributes in the page
* JavaScript can change all the CSS styles in the page
* JavaScript can react to all the events in the page

## Finding HTML Elements by Id

var x=document.getElementById("intro");

## Finding HTML Elements by Tag Name

This example finds the element with id="main", and then finds all <p> elements inside "main":

var x=document.getElementById("main");

var y=x.getElementsByTagName("p");

**y is a array!**

# JavaScript HTML DOM - Changing HTML

## Changing the HTML Output Stream

JavaScript can create dynamic HTML content:

**Date: Wed Mar 06 2013 23:30:41 GMT+0800 (中国标准时间)**

In JavaScript, **document.write()** can be used to write directly to the HTML output stream:

**Example**

<!DOCTYPE html>

<html>

<body>

<script>

document.write(Date());

</script>

</body>

</html>

**Note:** Never use **document.write()** after the document is loaded. It will overwrite the document.

## Changing HTML Content

The easiest way to modify the content of an HTML element is by using the innerHTML property.

To change the content of an HTML element, use this syntax:

document.getElementById(id).innerHTML=new HTML

**Example**

<html>

<body>

<p id="p1">Hello World!</p>

<script>

document.getElementById("p1").innerHTML="New text!";

</script>

</body>

</html>

## Changing an HTML Attribute

To change the attribute of an HTML element, use this syntax:

document.getElementById(id).attribute=new value

This example changes the src attribute of an <img> element:

**Example**

<!DOCTYPE html>

<html>

<body>

<img id="image" src="smiley.gif">

<script>

document.getElementById("image").src="landscape.jpg";

</script>

</body>

</html>

# JavaScript HTML DOM - Changing CSS

## Changing HTML Style

To change the style of an HTML element, use this syntax:

document.getElementById(id).style.property=new style

**Example**

<html>

<body>

<p id="p2">Hello World!</p>

<script>

document.getElementById("p2").style.color="blue";

</script>

<p>The paragraph above was changed by a script.</p>

</body>

</html>

This example changes the style of the HTML element with id="id1", when the user clicks a button:

<!DOCTYPE html>

<html>

<body>

<h1 id="id1">My Heading 1</h1>

<button type="button"

onclick="document.getElementById('id1').style.color='red'">

Click Me!</button>

</body>

</html>

# JavaScript HTML DOM Events

## Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=JavaScript

**Examples of HTML events:**

* When a user clicks the mouse
* When a web page has loaded
* When an image has been loaded
* When the mouse moves over an element
* When an input field is changed
* When an HTML form is submitted
* When a user strokes a key

In this example, the content of the <h1> element is changed when a user clicks on it:

**Example**

<!DOCTYPE html>

<html>

<body>

<h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>

</body>

</html>

In this example, a **function** is called from the event handler:

**Example**

<!DOCTYPE html>

<html>

<head>

<script>

**function changetext(id)**

**{**

**id.innerHTML="Ooops!";**

**}**

</script>

</head>

<body>

<h1 onclick="changetext(this)">Click on this text!</h1>

</body>

</html>