

# 4.3 JavaScript 基础

---

- JavaScript 概述
- JavaScript 基本语法
- JavaScript 网页操作
- jQuery



# 目录

1

JavaScript 概述

2

JavaScript 基本语法

3

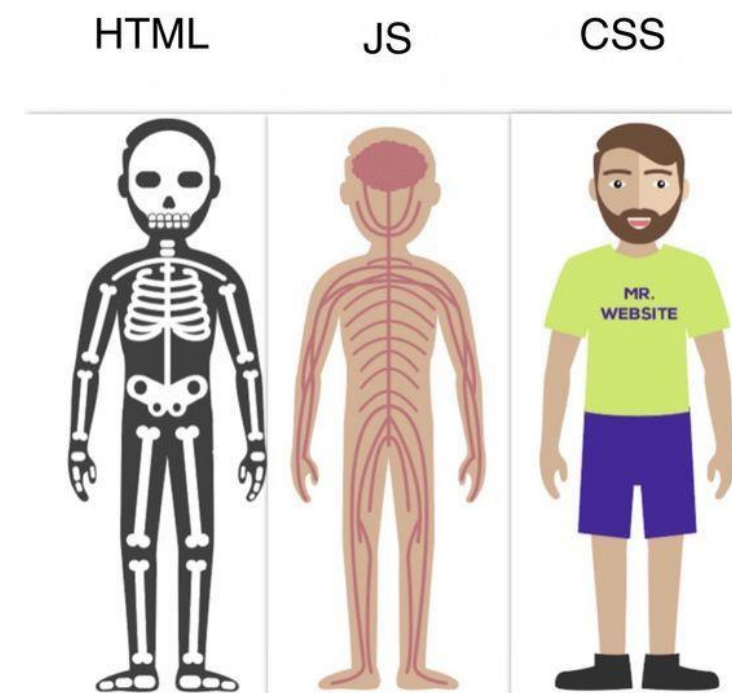
JavaScript 网页操作

4

jQuery

# JavaScript

- **JavaScript** 是网络相关开发最流行的语言之一。
- JavaScript 最主要的应用场景就是编写网页行为。
- JavaScript 可以直接与 HTML 页面上的元素交互，对用户操作给出动态反馈。



# JavaScript 的应用场景

- JavaScript 是一种**脚本语言**[スクリプト言語]。相比起 Java，它的基本语法更简单、并可以直接嵌入 HTML 中。因此，它可以非常高效地开发一些小规模的程序，如网页上的行为。
- 但 JavaScript 也能用于大规模程序的开发，如**服务器**和**桌面应用**。
- 比如，**Node.js** 是最流行的 JavaScript 服务器开发框架之一。很多日本互联网企业（**合同会社 DMM.com**、**株式会社ディー・エヌ・エー**）都在使用 Node.js 开发服务器。



# JavaScript 和 Java

- 虽然名称相似，但 JavaScript 和 Java 是两种截然不同的语言。
- JavaScript 的命名还要追溯到 20 世纪 90 年代网景（Netscape）和微软（Microsoft）的浏览器大战。为了抢占更大的市场，与 Sun 公司达成合作的网景公司选择把自己设计的浏览器脚本语言起名为 JavaScript，因为当时 Java 是热度最高的新兴语言。
- 因此，尽管两者有很多不同之处，但 JavaScript 在设计时考虑了与 Java 语法风格的统一性，这也使 Java 程序员对 JavaScript 的学习轻松了许多。

# 外部导入 JavaScript 文件

- 和 CSS 类似，你也可以选择导入外部 JavaScript 文件或在 HTML 文档内直接嵌入 JavaScript 代码。
- 要导入外部的 JavaScript 代码，使用 HTML 的 **<script>** 标签，设置其 **src** 属性为代码文件的 **URL**：

```
<script src="js/code.js"></script>
```

**Note**



不要忘了书写结束标签。

- JavaScript 代码文件的后缀名一般为 **“.js”**。

# JavaScript 代码嵌入

- 要嵌入 JavaScript 代码，直接在 `<script>` 标签内书写 JavaScript 代码：

```
1 <script>
2   let str = "Hello, world!";
3   console.log(str);
4 </script>
```

- 和 CSS 一样，建议将 `<script>` 写在 `<head>` 标签内。

# JavaScript 的运行顺序

- 如果同一个文档内有多段导入 / 嵌入的 JavaScript 代码，它们会按照导入 / 嵌入的顺序依次执行：

```
1 <script src="1.js"></script>
2 <script>
3   console.log("2");
4 </script>
5 <script src="3.js"></script>
```

- 如无特殊要求，推荐使用**外部代码**文件。这样做有以下好处：
  - 将描述页面结构的 HTML 代码和描述动作的 JavaScript 代码分离开来，使阅读和扩展更容易。
  - JavaScript 代码可以单独被浏览器**缓存**，加快页面加载速度。



Q & A

*Question and answer*

# 目录

1 JavaScript 概述

2 JavaScript 基本语法

3 JavaScript 网页操作

4 jQuery

# 从 Java 到 JavaScript

- 虽然 JavaScript 和 Java 是不同的语言，但 JavaScript 的许多基本语法保持了对 Java 语法的统一。我们学习 JavaScript 时可以先只记忆和 Java 语法不同的地方：

语法	区别
变量	使用 var、let 或 const 定义；可以存储不同类型的值
字符串	可以使用单引号 ' ' 定义
数组	使用 [] 或 new Array() 定义；可以存储不同类型的数据
对象	使用 {} 定义；可以作为关联数组使用
循环语句	for-of、for-in 语句
方法	使用 function 定义；lambda 表达式使用 => 符号定义



# 一个基本的 JavaScript 程序

- 将以下 JavaScript 代码导入或嵌入 HTML 文档中并打开页面：

```
console.log("Hello, world!");
```

- 打开你浏览器的**控制台**[\[コンソール\]](#)。各浏览器的快捷键如下：
  - Chrome: Shift + Ctrl + J (Windows) ; ⌘ + ⌘ + J (macOS) 。
  - Firefox: Shift + Ctrl + J (Windows) ; ⌘ + ⌘ + J (macOS) 。
  - Safari: ⌘ + ⌘ + C。
  - Edge: Shift + Ctrl + I。
- 如果代码书写无误，你应该能看到“Hello, world!”被显示在控制台中。

# 基本语法

- JavaScript 语句的基本运行方式和 Java 类似：
  - 从上到下依次运行。
  - 用分号 “;” 分隔每一个语句。
  - 可以使用空格或 Tab 分隔单词或排版。
- 注意以下不同之处：
  - **不要**使用换行分隔单词。
  - 每级缩进使用 **2** 个空格。

# 标准输出

- JavaScript 可以通过以下多种方法输出数据：

- 使用 **document.write()** 写入 HTML 文档。
- 使用 **window.alert()** 显示警报框[アラート]：



- 使用 **console.log()** 输出至浏览器控制台。





# 变量

- 要在 JavaScript 中声明一个变量<sup>[变数]</sup>，你可以使用 **var**、**let** 或 **const** 关键字：

```
var str = "This is a string.";
let num = 120;
const PI = 3.14;
```


- **var** 关键字声明一个可以不被初始化的变量。该变量会被自动初始化为“undefined”。
- **let** 关键字声明一个必须被初始化的变量。
- **const** 关键字声明一个常量。
- 为了增强代码的可读性和实行效率，尽量使用 **const** 和 **let** 进行声明。

# 变量的类型

- 和 Java 中不同，你~~不需要~~在声明变量时指定变量的类型。同一个变量也可以用于保存~~不同类型~~的值，比如变量 `a` 首先被用于保存一个数字，其后被用于保存一个字符串：

```
1 let a = 42;  
2 console.log(a); // => 42  
3 a = "This is the answer.";  
4 console.log(a); // => This is the answer.
```

# 变量名

- JavaScript 的标识符**起名规则**和 Java 中类似：
  - 名称只能包含英文字母、阿拉伯数字、下划线（\_）和美元符号（\$）。
  - 名称不能以数字开头。
  - 名称不能和 JavaScript 自带的关键字[キーワード]相同。
  - 变量和方法名都应为小写驼峰型。
- JavaScript 的关键字和 Java 的**不同**，可以查阅以下资料：
  -  [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical\\_grammar#reserved\\_words](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#reserved_words)



# 算术运算符

- 算术运算符<sup>[算術演算子]</sup>用于对数字执行运算：

运算符	功能
+	加法
-	减法
*	乘法
**	乘方
/	小数除法
%	求余
++	增加 1
--	减少 1

# 赋值运算符

- 赋值运算符<sub>[代入演算子]</sub>将改变变量的值:

运算符	功能	等价于
=	赋值	
+=	增加	$a = a + b$
-=	减少	$a = a - b$
*=	乘以	$a = a * b$
**=	变为乘方	$a = a ** b$
/=	除以	$a = a / b$
%=	变为余数	$a = a \% b$

# 比较运算符

- 比较运算符[比較演算子]比较两个变量间的关系：

运算符	功能
==	比较值是否相等
===	比较值和类型是否相等
!=	比较值是否不等
!==	比较值和类型是否不等
>	大于
<	小于
>=	大于等于
<=	小于等于



# === 与 ==

- JavaScript 中的相等运算符 “==” 和 “===” 的表现略有不同：

- == 比较两个对象的值是否相同。
- === 比较两个对象的值和类型是否相同。

```
1 let a = "321";  
2 let b = 321.0;  
3  
4 console.log(a == b); // => true  
5 console.log(a === b); // => false
```

- == 可以类比 Java 中对象的 **equals()** 方法。
- 与 Java 中的 equals 方法等价的 JavaScript 语法则是 Object.is() 方法。

# 逻辑运算符

- 逻辑运算符<sup>[論理演算子]</sup>用于组合布尔值以表达复杂逻辑:

运算符	功能
&&	逻辑与（并且）
	逻辑或（或者）
!	逻辑非（不是）

# 类型运算符

- **typeof** 用于获取变量的类型:

```
1 let a = "123";  
2 console.log(typeof a); // => string  
3 console.log(typeof 10); // => number
```

- **instanceof** (注意没有空格) 用于判断某个对象是否是某个类型:

```
1 let a = "123";  
2 console.log(a instanceof String); // => true  
3 console.log(a instanceof Number); // => false
```



# 数字

- JavaScript 中的变量可以存储任意**整数或小数**:

```
let num1 = 1.0;  
const num2 = -123;
```

- 也可以使用和 Java 类似的表示法表达指数表现、十六进制表现、八进制表现或二进制表现:

```
1 console.log(1.23E5); // => 123000  
2 console.log(0xCAFE); // => 51966  
3 console.log(036021); // => 15377  
4 console.log(0b1101); // => 13  
5 console.log(123_456_789); // => 123456789
```

# 布尔值

- JavaScript 中也使用布尔值[ブール値] **true** 或 **false** 进行条件判断:

```
1 let bool1 = 1 + 1 == 2;  
2 console.log(bool1); // => true  
3  
4 let bool2 = false;  
5 console.log(bool2); // => false  
6  
7 console.log(bool2 || true) // => true
```

# 字符串

- JavaScript 也使用字符串[文字列]存储文本数据。
- 注意：JavaScript 的字符串可以使用双引号 “” 或单引号 “” 进行定义：

```
let str1 = "quoted by double";  
let str2 = 'quoted by single';
```

- 这种语法的一个好处是如果使用 ' 括起字符串，你可以直接在字符串中写入 " 而无需转义：

```
let str1 = 'Tom says: "Hello!"';
```

- JavaScript 中不存在 Java 中的字符类型（char）。字符用单个字符构成的字符串表达。



# 数组

- JavaScript 中数组<sup>[配列]</sup>的使用方法和 Java 中大有不同。
  1. 创建方式不同。JavaScript 数组的两种创建语法都与 Java 不同。
  2. 存储规则不同。JavaScript 数组可以存储多种类型的数据。
  3. 提供方法不同。JavaScript 数组提供了一些方便的方法。

# 创建数组

- 要创建一个定义好初始值的数组，使用 `[]` 符号：

```
1 let arr = [1, 2, 3];  
2 console.log(arr); // => [1, 2, 3]
```

- 要创建一个确定初始长度的数组，使用 `new Array()` 构造方法：

```
1 let arr = new Array(5);  
2 arr[1] = 0;  
3 console.log(arr); // => [empty, 0, empty x 3]
```

Try 

arrays.html

# 存储数据

- 和 Java 不同，同一个数组中可以存储不同类型的数据：

```
1 let arr = ["string", 10, 1.5, [1, 2, 3]];
2 console.log(arr); // => ['string', 10, 1.5, Array(3)]
```

- 数据的存取方式则和 Java 中一样，用 `[]` 符号通过索引<sup>[添字]</sup>操作。

```
1 let arr = ["string", 10, 1.5, [1, 2, 3]];
2
3 console.log(arr[0]); // => string
4 console.log(arr[3][1]); // => 2
5
6 arr[1] = "alpha";
7 console.log(arr[1]); // => alpha
```



# 数组属性及方法

- 和 Java 一样，**length** 属性可以获取数组的长度：

```
console.log([1, 2, 3].length); // => 3
```

- **push()** 方法可以向数组**最后**插入一个数据（数组“变长”），**pop()** 方法可以删除**最后**一个数据（数组“变短”）：

```
1 let arr = ['a', 'b', 'c', 'd'];  
2  
3 arr.push('e');  
4 console.log(arr); // => ['a', 'b', 'c', 'd', 'e']  
5 console.log(arr.pop()); // => 'e'  
6 console.log(arr); // => ['a', 'b', 'c', 'd']
```

- 和 Java 不同，JavaScript 的数组可以直接被打印为可读的字符串。

# 对象

- JavaScript 中的对象[オブジェクト]和 Java 中的对象有相似之处，但 JavaScript 中的对象更常被用作关联数组储存数据。
- 要创建一个对象，使用 {} 符号括起一些键值对：

```
1 let obj = {name: "Alice", id: 148, isStudent: true};  
2 console.log(obj); // => {name: 'Alice', id: 148, isStudent: true}
```

- 对象的键[キー]（也被称为属性[プロパティ]）只能为字符串。你书写的任何属性名都会被自动转化为字符串。
- 对象的属性值可以为任何类型。



# 对象的使用

- 和 Java 类似，可以使用 “.” 运算符通过属性名直接访问对象的某个属性值：

```
1 let obj = {name: "Alice", id: 148, isStudent: true};  
2 obj.id = 120;  
3 console.log(obj.id); // => 120
```

- 你也可以使用 [] 符号，通过属性名的字符串进行访问：

```
1 let obj = {name: "Alice", id: 148, isStudent: true};  
2 obj["id"] = 120;  
3 console.log(obj["id"]); // => 120
```

- JavaScript 中的对象也可以保存方法实现类似 Java 的面向对象编程，但我们这里省略面向对象相关的内容。

Q & A

*Question and answer*



# 分支语句

- JavaScript 中的这些分支语句[分岐文]语法和 Java 是一样的：
  1. if 语句、if-else 语句与它们间的嵌套。
  2. switch-case 语句与其中的 break 语句。
  3. 三元运算符[三項演算子] “?:”。

# 循环语句

- JavaScript 中的这些循环语句[繰り返し文]与 Java 中是一样的：
  1. for 循环。
  2. while 循环。
  3. do-while 循环。
- 但 JavaScript 中存在 2 种不同的循环语法：
  1. for-of 循环。
  2. for-in 循环。

# for-of 循环

- **for-of** 循环类似 Java 中的 **for-each** 循环，它可以用于遍历一个可迭代对象（如数组或字符串）中的每一个值：

```
1 const arr = [2, 3, 5, 7];
2 let sum = 0;
3 for (let x of arr) {
4     sum += x;
5 }
6 console.log(sum); // => 17
```

```
1 const str = "abc";
2 for (let c of str) {
3     console.log(c); // => a b c
4 }
```

**Note**

变量名前要书写 **let**。

# for-in 循环

- **for-in** 循环遍历一个对象中的所有属性名:

```
1 const obj = {name: "Alice", id: 148, isStudent: true};
2 for (let prop in obj) {
3   console.log(prop); // => name id isStudent
4 }
```

- 也可用于遍历数组的所有索引名, 也就是 0 到 length - 1:

```
1 const arr = [1, 2, 3, 4];
2 for (let i in arr) {
3   console.log(i); // => 0 1 2 3
4 }
```





# break 和 continue

- 和 Java 中一样，你也可以在循环中使用 **break** 或 **continue** 语句。
- break 语句跳过整个循环。
- continue 语句跳过单次循环。
- 同样，你可以使用标签`[ラベル]`指定跳出哪一层循环：

```
1 outer: for (let i = 0; i < 10; i++) {  
2   for (let j = 0; j < 10; j++) {  
3     console.log(i + " " + j); // => 0 0  0 1  0 2  0 3  
4     if (j == 3) break outer;  
5   }  
6 }
```

# 方法

- JavaScript 中的方法[メソッド]在使用上和 Java 类似，但定义语法有所不同。
- 要定义方法，使用 **function** （函数）关键字：

```
1 function sum(a, b) {  
2     return a + b;  
3 }
```

- 和 Java 的不同点在于：
  1. 要使用 function 关键字。
  2. 不定义返回值[戻り値]类型。
  3. 不定义参数[引数]类型。



# 方法的默认参数

- JavaScript 中的方法参数可以比较简单地设置**默认值**。要设置默认值，在定义时的参数名后加上“**=默认值**”：

```
1 function print(str1, str2="World", str3="!") {  
2     console.log(str1 + " " + str2 + str3);  
3 }
```

- 在使用时，如果你没有传入默认参数，默认值将会被使用：

```
1 print("Hello"); // => Hello World!  
2 print("Hello", "Bob"); // => Hello Bob!  
3 print("Bye", "Alice", "...") // Bye Alice...
```

# 默认参数的注意点

- 注意：只有**最靠后**的几个参数可以被设置默认值：

## Example ✓

```
1 function method(a, b, c=0)
2 function method(a, b=0, c=0)
3 function method(a=0, b=0, c=0)
```

## Example ✗

```
1 function method(a, b=0, c)
2 function method(a=0, b=0, c)
3 function method(a=0, b, c=0)
```



# 变量的作用域

- 和 Java 类似，JavaScript 中变量的作用域[スコープ]可以根据它定义于哪个大括号 `{ }` 看出：

```
1 let a = 5;
2
3 function method( ) {
4     let a = 10;
5     console.log(a); // => 10
6 }
7
8 method( );
9 console.log(a); // => 5
```

## Tips

全局变量在同一个页面的其他 JavaScript 代码中也可以被使用！

- 在不考虑面向对象的前提下，我们可以简单地称方法内的变量为**局部变量**[ローカル変数]，方法外的变量为**全局变量**[グローバル変数]。

# Lambda 表达式

- JavaScript 中也有类似 Java 中的 **lambda 表达式**[\[ラムダ式\]](#)，它可以被用于传递一个方法作为参数。
- 和 Java 中**不同**，JavaScript 中的 lambda 表达式使用 **=>** 符号：

```
1 const arr = ["Alice", "Bob", "Dave", "Alexander"];
2 arr.sort((a, b) => {
3   return a.length - b.length;
4 });
5 console.log(arr); // => ["Bob", "Dave", "Alice", "Alexander"]
```

- 你也可以使用**方法名**直接传递一个现成的方法：

```
1 function compare(a, b) {return a.length - b.length;}
2 arr.sort(compare);
```

# 注释

- 和 Java 中一样，你可以使用 // 符号书写单行注释，或使用 /\* \*/ 符号书写多行注释：

```
1 let a = 120 + 240; // a becomes 360
2 /*
3  * This is a multi-line comment.
4  * これは複数行コメントです。
5  */
6 console.log(a) // => 360
```

# 总结：JavaScript 与 Java 的重要区别

## Sum Up

语法	区别
变量	使用 var、let 或 const 定义；可以存储不同类型的值
字符串	可以使用单引号 ' ' 定义
数组	使用 [ ] 或 new Array() 定义；可以存储不同类型的数据
对象	使用 { } 定义；可以作为关联数组使用
循环语句	for-of、for-in 语句
方法	使用 function 定义；lambda 表达式使用 => 符号定义



Q & A

*Question and answer*

# 目录

1 JavaScript 概述

2 JavaScript 基本语法

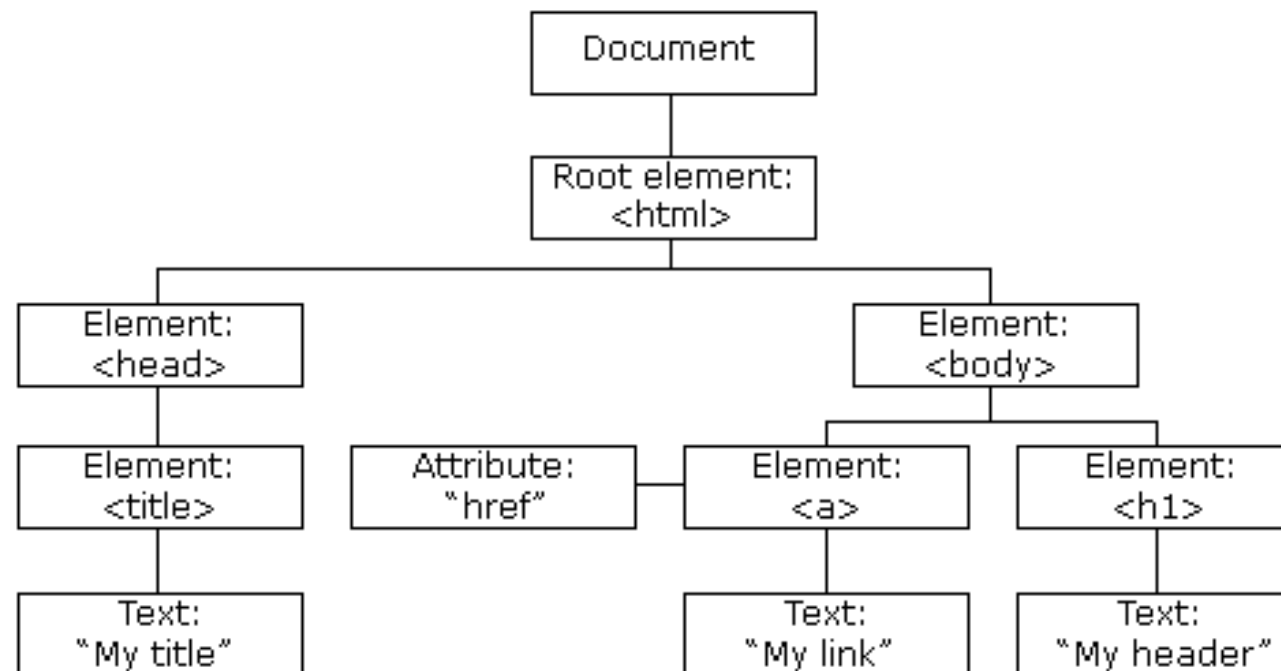
3 JavaScript 网页操作

4 jQuery

# 文档对象模型

- 当 HTML 网页被加载时，JavaScript 会为每个元素创建 1 个对应的对象。这些对象构成一些**文档对象模型**（Document Object Model, **DOM**）。
- DOM 中的元素构成一颗树。其每个节点都代表了 1 个 HTML 元素：

HTML DOM对象树





# document 对象

- **document** 对象是整个 DOM 对象树的根节点。页面上的所有元素都作为 document 的子孙节点存储。
- 在 JavaScript 中，你可以直接使用名为“document”的对象：

```
document.getElementById( "text-1" ).innerHTML = "Hello";
```



# 获得 HTML 元素

- 要获取 HTML 页面上某一个元素的 DOM 对象，你可以使用 document 对象的以下方法：

方法	功能
getElementById(id)	根据 id 指定元素
getElementsByTagName(tag)	根据标签名指定元素
getElementsByClassName(class)	根据类名指定元素

- 可以发现，这和我们在 CSS 中指定元素的方式类似：使用元素的 **id**（如 #text-1）、**标签名**（如 p）或**类名**（如 .class-1）选择元素。

# 访问元素内容

- 在获得元素后，你可以通过其 **innerHTML** 属性访问它的内容（开始标签 `<>` 和结束标签 `</>` 之间的部分）：

HTML:

```
1 <body>
2   <p id="text-1">fuga</p>
3 </body>
```

Try  DOM/edit-html.html

JavaScript:

```
1 console.log(document.getElementById("text-1").innerHTML); // => "fuga"
2
3 document.getElementById("text-1").innerHTML = "hoge";
```

# 访问元素 HTML 属性

- 你也可以使用同名属性直接访问 **HTML 属性**。例如，如果元素存在 href 属性，你可以通过其 DOM 对象的 href 属性进行获取和修改：

HTML:

```
1 <a id="link-1" href="https://developer.mozilla.org">link</a>
```

JavaScript:

```
1 // => https://developer.mozilla.org
2 console.log(document.getElementById("link-1").href);
3 document.getElementById("link-1").href = "https://google.com";
```

**Note**



特别的，class 属性对应的属性名为 **className**。

# 获得元素列表的情况

- 要注意，如果使用 `getElementsByTagName()` 或 `getElementsByClassName()` 方法，有可能获得多个元素的 DOM 对象。JavaScript 将会返回一个元素的**列表**。
- 我们可以使用循环语句遍历列表中的元素，获取或修改其中的值：

HTML:

```
1 <p id="text-1">fuga</p>
2 <p id="text-2">hoge</p>
3 <p id="text-3">piyo</p>
```

Try 

[DOM/edit-multiple.html](#)

JavaScript:

```
1 const ps = document.getElementsByTagName("p");
2 for (let p of ps) {
3   console.log(p.innerHTML); // => fuga hoge piyo
4 }
```



# 创建元素

- 要往页面上添加新的元素，首先使用 **document.createElement()** 方法创建一个新元素的 DOM 对象：

```
let newTitle = document.createElement("h1");
```

- 其中，传入的参数为元素的**标签名**（p、h1、div 等）。
- 你可以使用刚刚说明的方法设置它的**内容**和**属性**：

```
1 newTitle.title = "Hi!";  
2 newTitle.innerHTML = "This Is A Title";
```

# 添加元素

- 接下来，你可以通过使用页面上已有元素的各种方法将新建好的元素添加至页面：

方法	功能
append()	添加至元素（内部）的末尾
prepend()	添加至元素（内部）的开头
after()	添加至元素后
before()	添加至元素前

- 比如，下面的代码将刚刚新建的元素插入到 id 为 “text-1” 的元素之后：

```
document.getElementById("text-1").after(newTitle);
```

# 删除元素

- 你也可以使用 **remove()** 方法删除某个元素：

```
document.getElementById( "text-1" ).remove( );
```

Try 

DOM/add-remove.html

# DOM 事件

- DOM 还允许我们设置各个元素对用户操作的反应。这通过一种叫做**事件**[イベント]的机制实现。每一种事件都会在进行不同类型的操作时被触发。
- 以下是常见事件的列表：

事件名	何时触发
click	元素被点击时
dblclick	元素被双击时
mouseover	鼠标移入元素时
mousemove	鼠标在元素上移动时
mouseout	鼠标移出元素时
focus	元素获得焦点时
blur	元素失去焦点时



# 设置事件行为

- 你可以设置当某个元素的某个事件触发时，运行怎样的 JavaScript 代码。
- 要设置事件行为，设置元素的“**on事件名**”属性。比如 click 事件对应 onclick 属性：

```
1 document.getElementById("text-1").onclick = () => {  
2   console.log("Hello");  
3 };
```

- 该属性需要被设置为某个**方法**。上例中，我们使用 lambda 表达式来传递方法。你也可以直接使用现成的方法名：

```
1 function printHello() { console.log("Hello"); }  
2 document.getElementById("text-1").onclick = printHello;
```

# 在 HTML 中设置事件行为

- 你也可以直接在 HTML 标签中使用“on事件名”**属性**定义事件行为：

```
<p id="text-1" onmouseover="printHello( )">fuga</p>
```

# 添加事件行为

- 使用“on事件名”属性设置事件行为时，会覆盖之前已被设置的行为。因此，只有**最近 1 次**设置是有用的。
- 要避免覆盖，只是添加新的事件行为，可以使用元素的 **addEventListener()** 方法：

```
1 const text = document.getElementById("text-1");  
2 text.addEventListener("dblclick", () => console.log("Hello"));  
3 text.addEventListener("dblclick", () => console.log("World"));
```

- 其中，第 1 个参数为事件名，第 2 个参数为新增的行为（方法）。

# load 事件

- **load** 事件会在元素被加载完成后被触发。
- 特殊对象 window（窗口对象）的 load 将会在整个页面加载完成后被触发。因此，我们可以把需要执行的 JavaScript 代码都添加至这个事件，以防止代码运行时文档还未被加载完：

```
1 window.addEventListener("load", () => {  
2   const text = document.getElementById("text-1");  
3   text.addEventListener("dblclick", () => console.log("Hello"));  
4   text.addEventListener("dblclick", () => console.log("World"));  
5 });
```

Try   
DOM/events.html



Q & A

*Question and answer*

# 目录

1 JavaScript 概述

2 JavaScript 基本语法

3 JavaScript 网页操作

4 jQuery


# jQuery

- jQuery 是一个帮助我们处理 HTML 操作的 JavaScript 工具库。
- jQuery 的目标是实现“少写、多做”。原有的 DOM API 提供的方法使用起来大多繁琐、重复，而 jQuery 旨在帮助用户用最小限的代码实现同样的操作。
- 许多著名公司都在使用 jQuery，如 Google、Microsoft 等。很多其它网页开发库也依存于 jQuery，如 Bootstrap（5 以前版本）等。





# 导入 jQuery

- 有两种方法可以向你的网页中导入 jQuery：下载文件或使用其 **CDN** 版本（在线版本）。
- jQuery 库可以通过以下官方链接下载：  
 <https://code.jquery.com/jquery-3.6.0.min.js>  
下载后，你需要使用 `<script>` 标签进行导入。
- 你也可以直接在 `<script>` 标签中使用其 CDN 链接导入：

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
```



## Coffee ☕ Break

# JavaScript 代码的最小化

在下载许多 JavaScript 库时，我们会发现有些库提供了一种特殊的代码版本。比如 jQuery 的 3.6.0 版提供了 `jquery-3.6.0.js` 和 `jquery-3.6.0.min.js` 这两个版本。它们有什么区别呢？

以 **.min.js** 结尾的通常是原文件的**最小化**[ミニファイ]版本。这种代码文件通常具有和原版**完全一样的功能**，只是在**书写上进行了压缩**（比如去掉多余的空格、换行；缩短标识符的名称等）。使用最小化代码可以加快浏览器加载代码的时间，优化用户体验。

你也可以使用一些工具简单地最小化自己的代码，如 **Google Closure Compiler** 等。

# 基本语法

- jQuery 的所有操作都基于一个特殊的对象“\$”。我们可以使用 \$ 对象替代绝大部分之前使用 document 对象进行的操作。
- \$ 的基本使用形式如下：

```
$("selector").method( );
```

- 其中， selector 参数用于**选择**需要进行操作的元素， method 指定需要进行的**操作**。

# 选择器

- jQuery 使用 **CSS 选择器**来替代 DOM 中繁琐的“getElementById()”等方法。
- 比如之前使用 DOM 方法指定元素的代码：

```
const text = document.getElementById( "text-1" );  
const ps = document.getElementsByTagName( "p" );  
const divs = document.getElementsByClassName( "main" );
```

- 如果用 jQuery 则可以写成这样：

```
const text = $( "#text-1" );  
const ps = $( "p" );  
const divs = $( ".main" );
```

# 获取元素内容

- 使用选择器获得指定元素后，便可以使用其 **html()** 方法获得其内容：

```
1 let text = $("#text-1");  
2 console.log(text.html()); // => fuga
```



# 修改元素内容

- 修改内容同样使用 **html()** 方法。只需要向该方法里传入 1 个数：

```
1 let text = $( "#text-1" );  
2 text.html( "hoge" );
```

# 获取元素 HTML 属性

- 要获取元素 **HTML 属性**，使用 **attr()** 方法：

HTML:

```
<a id="link-1" href="https://google.com">fuga</a>
```

JavaScript:

```
1 let link = $("#link-1");  
2 console.log(link.attr("href")); // => https://google.com
```

- 该方法接受 1 个参数，指定**属性**的名称。

# 修改元素 HTML 属性

- 要修改元素 HTML 属性，同样使用 **attr()** 方法：

```
1 let link = $("#link-1");  
2 link.attr("href", "https://developer.mozilla.com");
```

- 该方法的第 2 个参数指定修改后的值。

# 获取或修改元素 CSS 属性

- 要获取或修改元素的 **CSS 属性**，使用 **css()** 方法：

```
1 console.log(text.css("color")); // => rgb(0, 0, 0)
2 text.css("color", "red");
```

- 和 `attr()` 类似，传入 1 个参数可以获取属性值，传入 2 个参数则可以修改属性值。

Try   
[jquery/edit-html.html](#)



# 获得元素列表的情况

- 大多数 jQuery 方法都可以让你在**不**遍历元素列表的前提下批量操作元素。
- 比如，以下代码可以直接设置所有 `<p>` 元素的文字颜色为红色：

```
1 let allText = $("p");  
2 allText.css("color", "red");
```

Try 

[jquery/edit-multiple.html](#)

# 添加元素

- jQuery 新建元素的方法也非常简单，只需要在 `$()` 中传入元素的标签：

```
let newText = $( "<p>" );  
newText.html( "Added text" );  
newText.css( "color", "red" );
```

- 新建好后，你可以用和 DOM 类似的方法（`append()`、`prepend()`、`before()`、`after()`）把它插入到页面中：

```
$( "div.main" ).append( newText );
```

# 删除元素

- 你同样可以使用 **remove()** 方法删除某个元素：

```
1 $( "#text-1" ).remove( );
```

- 还可以使用 **empty()** 方法清空元素的内容（删除所有子元素）：

```
$( "div.main" ).empty( );
```

Try 

[jquery/add-remove.html](#)

# 设置事件行为

- 想要设置事件行为，使用和事件名相同的方法：

```
1 $( "#text-1" ).click( ( ) => {  
2     console.log( "Fizz" );  
3 } );
```



# 添加事件行为

- 添加事件行为（DOM 中的 `addEventListener()` 方法）在 jQuery 中则是使用 `on()` 方法：

```
1 $( "#text-1" ).on( "click", () => {  
2     console.log( "Buzz" );  
3 } );
```

# load 事件

- DOM 中的 window.load() 在 jQuery 中可以直接使用 **\$()** 方法：

```
$(method);
```

```
1 $(( ) => {  
2     $( "#text-1" ).click(( ) => {  
3         console.log( "Fizz" );  
4     });  
5 });
```

- 你可以把大多数 jQuery 代码都放在这个 lambda 表达式中以保证其在页面完成加载后运行。

Try 

[jquery/events.html](#)

# 特效

- jQuery 还提供了很多元素的**动态特效**，比如：

方法	效果
hide()	隐藏元素
show()	显示元素
fadeOut()	淡出元素
fadeIn()	淡入元素
slideUp()	向上移出元素
slideDown()	向下移入元素
animate()	各种动画效果

Try   
jquery/effects.html

## Tips

你还可以下载扩展库 jQuery UI，轻松添加更多动画效果：

 <https://jqueryui.com/>

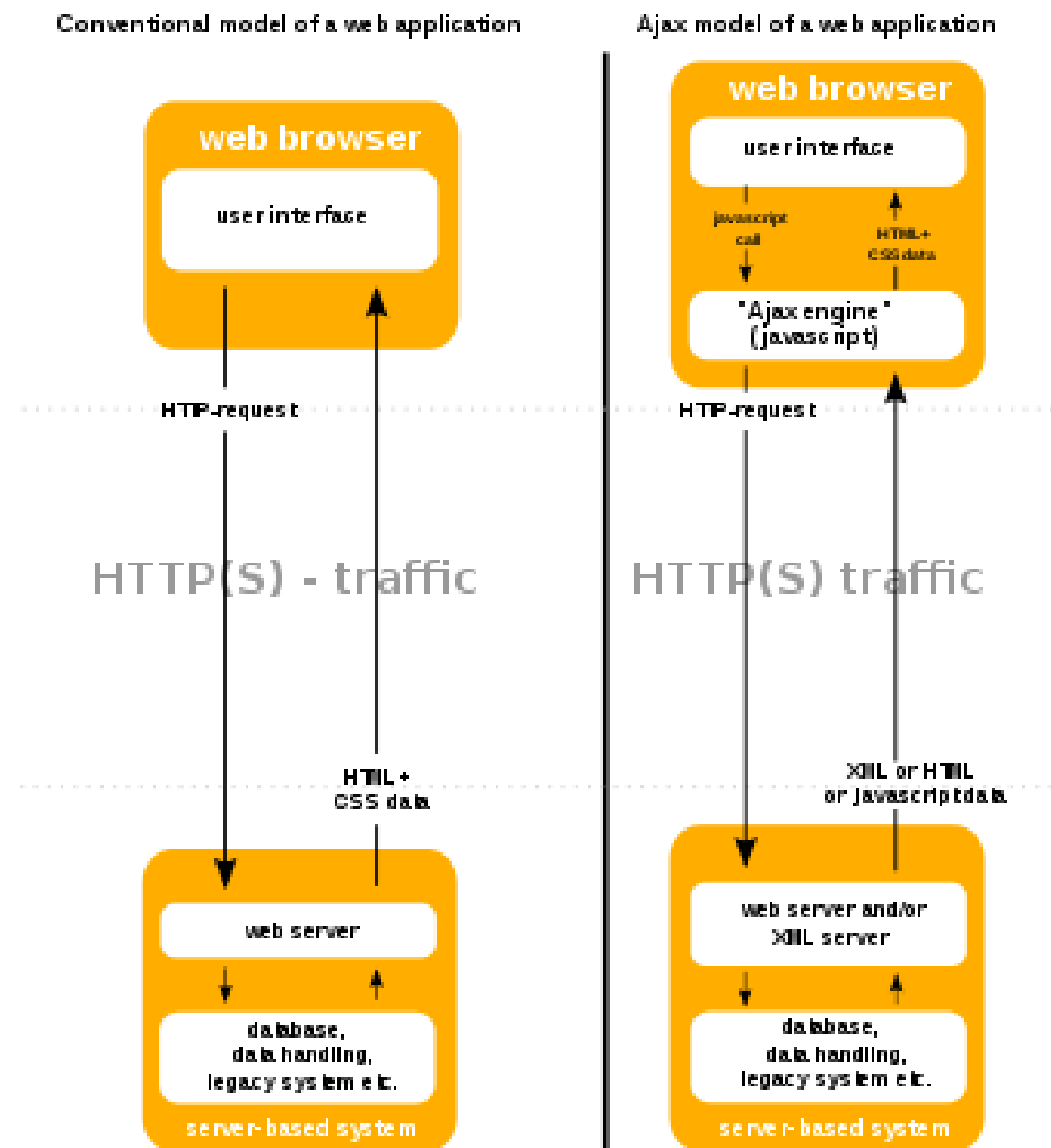
Q & A

*Question and answer*



# AJAX

- **AJAX** (Asynchronous JavaScript and XML) 指通过 JavaScript 等嵌入式语言，与服务器交换数据并更新部分网页的技术。
- 与传统动态网页模式不同，用户无需等待服务端创建并传递整个 HTML 页面，而只需在客户端更新页面的一部分，大大提高了服务端的负载和动态页面的反馈速度。



# jQuery 的 ajax() 方法

- jQuery 提供了 **ajax()** 方法，它可以发送访问服务器的 **HTTP** 请求，并返回服务器的**响应**。你可以使用该响应修改页面上的某些信息，实现一些动态效果：

```
1 $.ajax({  
2   url: "http://ip-api.com/json/",  
3   data: {fields: "country,regionName,city,lat,lon"},  
4   success: (result) => {  
5       console.log(result);  
6   }  
7 });
```

## 其他 AJAX 常用方法

- jQuery 还提供了一些使用 AJAX 的便利方法：
- **get()** 方法简单地向服务器发送一个 HTTP **GET** 请求。
- **post()** 方法简单地向服务器发送一个 HTTP **POST** 请求。
- **load()** 方法直接将服务器的应答作为 HTML 代码载入某个元素。



Q & A

*Question and answer*



# 总结

## Sum Up

1. 导入 JavaScript 的 2 种方法。
2. JavaScript 的基本语法：重点记忆与 Java 不同的地方。
3. JavaScript 中对 HTML 文档的操作方法：DOM。
4. 使用 jQuery 简化 DOM 操作的方法：
  - ① 使用**选择器**获取元素。
  - ② 使用元素方法修改元素的内容、**HTML 属性**、**CSS 属性**。
  - ③ 处理**事件**的方法。

**THANK YOU!**