HTTP – HyperText Transfer Protocol(超文本传输协议)

1. What is hypertext?
2. What is transfer protocol?

HTTP : It is a protocol, developed by Tim Berners at CERN,1989.

RFC specifications; RFC 7230.(课上使用的协议版本)

A protocol is a plan for how cooperating components of a system should interact.

(协议是关于系统的协作组件应如何交互的计划。)

Protocols: Status Communications.

Responses are not just the data requested.
It will have different status.

HTTP Status Communications

HTTP transfers hypertext (of course). This is the data the protocol is concerned with. Like the previous toy example, HTTP is also a **client-server protocol** with **request-response semantics**.
But HTTP also transfers *metadata* about the status of communicating parties. There are *two key* mechanisms for this: request and response headers & status codes.
*The *metadata* is an important part of the HTTP *protocol*. However, *metadata* is separate from the *data*. A hypertext document does not *have to* arrive via HTTP.
HTTP 协议中的**元数据**是指与传输的数据相关的信息，比如数据的大小、类型、来源等。**这些元数据是独立于实际数据内容的**，因此即使是通过其他方式传输的超文本文档，也可以有元数据。**HTTP 协议是一种常用的传输超文本文档的方式，但并不是唯一的方式，**其他协议或方法也可以用于传输超文本文档，但仍然可以附加元数据来描述这些文档的特征和属性。

HTTP message structure

```
HTTP-message   = start-line
                  *( header-field CRLF )
                  CRLF
```

```
                 [ message-body ]
```
- Strat line can be request line(for a request) or a status-line(for a response)
- Headers: optional
- Internally(内部): field-name : field-value

HTTP Requests

Per RFC 7230, the format for a HTTP request-line is:
method SP request-target SP HTTP-version CRLF
- SP = space(空格)
- CRLF = carriage return(回车)

The HTTP-version **rarely changes** (though it can!).
Key elements to understand are **method** and **request-target.**

HTTP Methods

- GET : retrieve a copy of the target resource 检索目标资源的副本
- POST : submit payload data to a target resource 向目标资源提交有效负载数据
- HEAD : retrieve metadata for corresponding GET 检索对应 GET 的元数据
- PUT : replace target resource with payload 用有效负载替换目标资源
- DELETE: delete the target resource 删除目标资源

Many servers do not implement or will ignore DELETE or PUT requests in favor of custom semantics using POST requests.
服务器会忽略 DELETE 和 PUT，更倾向使用 POST 请求。

HTTP request-target – The recourse you are targeting with your request.

Relates to the path and query components of a URI (Uniform Resource Identifier).
与 URI（统一资源标识符）的路径和查询组件相关。

- path: e.g., /, or /files/index.html or /user/george/
- query: e.g., ?name=welcome&action=view
    - formed of a series of ***parameters*** (name, action) with **values** (welcome, view)

The same resource at a **path** might **respond differently** to **different query** strings.

HTTPS Status Line

Format: ` status-line = HTTP-version SP status-code SP reason-phrase CRLF `
- **HTTP-version:** we covered this already in the request
- **status code:** 3-digit code with specific meaning
- **reason-phrase:** description to explain status code

HTTP Status Codes

Can have very **specific** meanings. Can grouped by **first digit** with semantic meaning.
- 1xx: information(100 Continue)
- 2xx: success(200 OK)
- 3xx: redirect(301 Moved Permanently)
- 4xx: client error(403 forbidden 404page not found)
- 5xx: server error(500 internal server error)

Example HTTP Exchange

Request
```
GET /index.html HTTP/1.1
Host: www.bristol.ac.uk
Connection: close
```
Response
```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 1009


<!DOCTYPE html>
<html lang="en">
...
```

Content-Type – metadata/ important header

- An important **header** in the **response** is the **Content-type**.
- Metadata. Tells the client what **type of data** the response body will contain.
- Very important for browsers. Clients interpret response bodies for humans. (浏览器需要根据 Content-Type 来确定如何解释响应体的内容，以便正确地呈现给用户。例如，如果响应的 Content-Type 是"text/html"，那么浏览器就知道响应体包含的是 HTML 格式的数据，因此会以网页的形式展示给用户；如果 Content-Type 是"application/json"，浏览器就知

道响应体是 JSON 格式的数据，可能会以适当的方式展示给用户或者交由 JavaScript 进一步处理。因此，Content-Type 对于客户端来说是非常关键的元数据，它帮助客户端正确地解释和处理响应的内容，以提供良好的用户体验。）

HTML – hypertext

Interactive text. Text with hyperlinks.(核心)
HTML is HyperText Markup Language. A language for 'marking up' text to make it interactive (and structured). (HTML 是超文本标记语言。一种用于"标记"文本以使其具有交互性（和结构化）的语言。)
Key elements:
- Tags: indicate **meaningful document components.** (标签指示文档中的有意义的组件。在 HTML 中，标签用于标识文档中的不同部分，比如段落、标题、链接等。每个标签都有其特定的含义和作用。)
- Nesting of tags and text within tags organizes document structure. (标签的嵌套和文本在标签内部的排列组织文档结构。HTML 中的标签可以嵌套在其他标签内部，这种嵌套关系构成了文档的层次结构。例如，一个段落可以包含在 `<p>` 标签中，而段落内部可以包含文本或其他标签，如 `<strong>`、`<em>` 等，这样的嵌套和排列方式帮助组织文档内容。)
- Tags can have **attributes** that affect interpretation of their semantics. (标签可以具有影响其语义解释的属性。**HTML 中的标签可以带有属性**，这些属性可以提供额外的信息或控制标签的行为。这些属性可以改变标签的语义含义或者影响其在浏览器中的呈现方式。例如，`<a>` 标签可以用 `href` 属性指定链接目标，`<img>` 标签可以用 `src` 属性指定图像源等。这些属性可以影响标签的解释和行为，进而影响文档的呈现和交互。)

Important HTML tags

- 所有的都被<html>包裹住。
- <head> VS <body> HTML 的<head>和 HTTP 的 header 毫无关系！
  <head>里包含修饰<body>的<title>和<meta>tag。
  <body>里包含文档的可见部分。

大部分正文放在<body>里。
常见的<body>里的组件有<p>,<div>,<span>…

HTML Document Example

```
<!DOCTYPE html>
    <html lang="en">
      <head>
        <meta charset="utf-8" />
        <title>A web page</title>
      </head>
      <body>
         <h1>An example web-page</h1>
         <p>A paragraph of text here, perhaps with <a href='./another_page/'>a link</a></p>
      </body>
    </html>
```

HTML Presentation

There are common visual defaults for GUI browsers (e.g., blue underline).
- 可见化的默认设置。 比如蓝色下划线超链接。

But the semantic structure of a document is (meant to be) separate from its presentation.
- HTML 文档的语义结构与其呈现方式应该是分开的。这意味着 HTML 的标记应该更关注文档内容的语义，而不是具体的呈现效果。HTML 应该描述文档的内容结构和含义，而不应该包含太多关于样式和布局的信息。

In a browser, the presentation of elements in HTML documents is governed by *stylesheets*. ->CSS
- 文档元素在浏览器中的呈现效果由样式表控制。CSS（层叠样式表）提供了一种机制，让开发者可以为 HTML 文档定义样式，包括颜色、字体、布局等方面的样式。通过 CSS，开发者可以将文档的呈现方式与其语义结构分离开来，实现更灵活的设计和布局。

CSS - Cascading style sheets

It is a set of rules that define how HTML elements are displayed.
Most cases: setting of visual properties for a HTML document element.(To the DOM objects derived from the HTML) 从 HTML 派生的 DOM 对象。

CSS compostition

syntax:
```
selector {
    property : value;
}
```

- 'property': is any display property that is meaningful for this element type. (CSS 中的属性是要样式化的 HTML 元素的属性。例如，color、font-size、background-color 等都是属性。文本意思是这里可以使用任何与所要样式化的元素类型相关的显示属性。)
- 'value': depend on the property: permitted values. (属性的值是你设置的值。例如，对于 `color` 属性，值可以是 "red"、"blue"、"green"，或者 CSS 支持的任何其他颜色。文本意思是允许的值取决于所使用的属性)
- 'selector': in this rule which elements can be applied to. (: 在 CSS 中，选择器用于定位特定

的 HTML 元素以应用样式。例如，你可以选择所有 `<p>` 元素、所有具有特定类的元素，或者具有特定 ID 的特定元素。文本意思是选择器定义了 CSS 规则可以应用于哪些元素。)

CSS implementation(CSS 实现)

CSS rules should be placed in a **stylesheet document**. e.g. mystyle.css
Then browser can be informed to use a particular stylesheet by a reference in the HTML document.

```
<html>
  <head>
      <link rel="stylesheet" href="mystyle.css"/>
  </head>
  <body>
    ...
```

You can also just instruct your browser to apply a custom stylesheet to HTML documents **by default** (‘user’ or ‘custom’ styles).

CSS simple example

```
p {
  color : red;
}
```

Explain: for all *p* elements, set the color (font color) to red.
Selectors(选择器)
Wide range of selection capabilities. (CSS 有很多不同的选择器。)
Simplest selector: name of a single tag(p, a, div). Applies to **all elements** of that kind.(应用到所有有那种标签名的元素)
Next-simplest selector is a list of tags:（有一系列 tags 的选择器）:

```
p, div, main {
      color : red;
}
```

效果：将<p>,<div>,<main>三种类型的 HTML 元素文本颜色都设置为红色。

Selectors: Class(类选择器)

例子 1：选中所有 p（特定）里 important 类的

```
p.important {
      color: red;
  }
```

```
<p class='important'>This is red</p>
  <p>This is not</p>
```

例子 2：无论元素类型，选中所有类

```
.important {
      color: red;
  }
```

```
<p class='important'>This is red</p>
  <p>This is not</p>
  <span class='important'>But this is</span>
```

Selectors: ID(ID 选择器)

将样式应用于特定文档

```
p#uniquebox {
      color: red;
  }
```

```
<p id='uniquebox'>This is red</p>
  <p class='uniquebox'>This is not</p>
  <p>This certainly is not</p>
```

在 CSS 中，`#`符号用于选择具有特定 ID 属性的 HTML 元素，这称为 ID 选择器。所以`#uniquebox` 将应用于具有 ID 属性为`uniquebox` 的元素。在这个例子中，`<p id='uniquebox'>This is red</p>`是唯一符合条件的元素，其文本颜色将被设置为红色。

Same as class – #uniquebox by itself would apply to?

答：如果只有`#uniquebox`，那么它只会应用于具有 ID 属性为`uniquebox`的元素。

Selectors: Attribute(属性选择器)

Select elements by any attribute. 按任何元素选择属性。

```
p[name=tim] {
      color:red;
  }
  div[border=none] {
      color: blue;
  }
```

`p[class='important']` = `p.important`.

Selectors: Positional(各种元素位置概念详解)

```
<div class="container">
  <p>direct child</p>
  <div>
    <p>descendant</p>
  </div>
```

```
</div>
<p>para one</p>
<p>para two</p>
```

- A descendant is an element that is ‘inside’ another element, at any level. (后代*是*在任何级别上位于另一个元素“内部”的元素)
  - The child is an element that is directly contained inside the parent. (子元素是**直接包含**在父元素内的元素。)

  - An element precedes another if it comes at any point earlier at the same level of the document. (如果一个元素在文档**同一级别**的任何位置较早出现，则该元素 *位于另一个元素之前。*)

  - An element follows another if it is the very next element at that level of the document. (如果一个元素是**文档该级别**的 *下一个元素*，则该元素 *紧随另一个元素。*)

Selectors: Positional（选择器以及使用规则）

- this that (space): selects all elements that which are *descendants* of this. : 这是一种基本的选择器形式，使用空格分隔两个选择器。它选择所有作为第一个选择器的后代的第二个选择器。例如，div p 选择了所有 <p> 元素，只要它们是 <div> 元素的后代。
- this > that: selects all elements that which are *direct children* of the parent this. 这是一个选择器形式，使用 **>** 符号表示直接子元素关系。它选择作为第一个选择器的父元素的**直接子元素**中匹配第二个选择器的元素。
- this ~ that: selects all elements that which are *preceded by* an element this. 这是一个选择器形式，使用 ~ 符号表示同级元素中的前置关系。它选择在同级中，作为第一个选择器的元素之后出现的所有匹配第二个选择器的元素。
- this + that: selects all elements that which directly *follow* an element this. 这是一个选择器形式，使用 **+** 符号表示同级元素中的后继关系。它选择在同级中，作为第一个选择器的元素之后紧跟着出现的匹配第二个选择器的元素。

这些规则可以**组合使用**，以创建更复杂的选择器。

```
div.important > p, h1#main, [title=nowred] ~ span {
    color: red;
 }
```

多个不一样的规则加起来，可以组成复合规则。

Cascading

哪条规则适用?

```
<p class='important'>If you want to pass this unit then...</p>
p {
    font-size: 12pt;
}

p.important {
    color: red;
}
```
第一个规则：p，设置了所有 <p> 元素的字体大小为 12pt。
第二个规则：p.important，设置了所有带有 important 类的 <p> 元素的文本颜色为红色。
由于 CSS 的层叠规则（Cascading Rules），具有相同特 specificity (特异性)的规则中，后定义的规则将覆盖先定义的规则。所以，在这种情况下，第二个规则将**覆盖**第一个规则。因此，带有 important 类的 <p> 元素的文本颜色将被设置为红色。

Values

Some common elements relate to *colour* and *element layout*.

Color values.

Can set text-color, background-color and border-color.
- Red. Bule 等等 keywords。
- 十六进制格式 #rrggbb 它可以 accept 00-FF 对每个 R G B。
- 调用函数 rgba(r, g, b, a). 范围: r&g&b:1-255 A:0-1
- Red = #FF0000.   #FF0001 或者#FF1111 看起来都是红色的。(尽管它们的具体颜色值有所不同，但它们都足够接近红色，以至于人类的肉眼几乎无法分辨它们之间的差异。)

Layout

common issue: dealing with space ‘around’ an element (or between elements).
每个页面元素都可以被看作是一个“盒子”，具有几个层次:
- The **content** is the raw material of the element itself 内容是元素本身的原始材料（例如，<p> 元素中文本的空间，或者 <img> 元素中图像的空间）。
- The **padding** is the space between the content and the border.内边距是内容和边框之间的空间。
- The **border** is a (**sometimes** invisible) line ‘around’ the element, marking its bounds. It can have a thickness.边框是“围绕”元素的（有时是不可见的）线，标记着它的边界。它可以有一个厚度。
- The **margin** is the space required to be kept clear outside the border – other elements must not intrude on this space.外边距是保持在边框外部的必需空间 - 其他元素不得侵入此空间。

很容易混淆内边距和外边距（使边框可见有助于理解）。

Layout values

Both **margin** and **padding** can be specified for individual sides, or collectively in clockwise order.
可以单独为每个边指定（第一种），也可以按顺时针顺序整体指定（第二种）。
```
    margin-top: 10px;
     margin-right: 20px;
```

```
margin-bottom: 10px;
margin-left: 5px;

margin: 10px 20px 10px 5px;
```

Units of measurement(指定测量单位的各种方式)

Absolute' units try to produce a specific **real size(**特定的实际尺寸**):**
1 px - 1 'pixel' (however that is interpreted: 1/96th inch). | 1 px：1 像素(1/96 英寸)。
1 pt - 1 'point' (1/72th of an inch) | 1 pt：1 点（1/72 英寸）
1 cm - 1 centimetre (also 10 mm) | 1 cm：1 厘米（也是 10 毫米）。
1 in - 1 inch | 1 in：1 英寸。
'**Relative' units** produce dimensions relative to either the viewport or some reference element of the page.(**'相对'单位**产生相对于视口或页面的某个参考元素的尺寸)
1 vh - 1% of the viewport's height (also 1 vw for width) | 1 vh：视口高度的 1%（宽度也有 1 vw）。
1 em - 1 x whatever the size of the font (width of an 'm') is. | 1 em：字体大小的 1 倍（一个 'm'的宽度）。
1 ex - 1 x whatever the height of an x would be. | 1 ex：1 倍 x 的高度。
1 rem - 1 x whatever the size of the font of the document's root element is. | 1 rem：文档根元素的字体大小的 1 倍。
1 % - 1% of the size of the parent element's corresponding dimension. | 1 %：父元素相应维度的 1%。

Design is hard.

This unit is trying to teach you some **fundamental understanding** of CSS.
CSS can be hard to debug and understand – technical issues. 难以调试和理解。
But successfully designing styles for real websites can also be hard in a non-technical sense. There are **key principles** (links to fundamentals of ergonomics, audience expectations, etc.) but fundamentally a lot comes down to questions of taste, style, fashion – web design is an art.
Some concepts you may find handy(有用的建议):
- grid-based page layouts (**big focus** in this week's lab) 基于网格的页面布局，重点。
- let **designers create frameworks** which you can apply (also in the lab)
- stealing ideas from **other websites.**

JavaScript

Why need JavaScript:

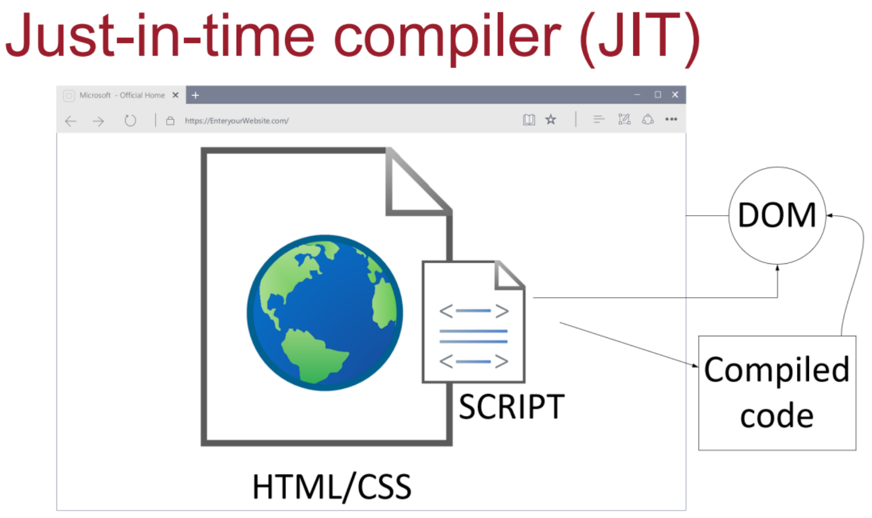
- High level language like Python and Java.
- JavaScript, HTML, CSS 是前端三件套。
- 所有浏览器都包含一个 JavaScript 引擎来执行代码。Engine to execute the code.
- JavaScript 可以用于使用 Node.js 进行服务器端操 be used for server-side operations.

Attributes & properties(JavaScript)

- **Dynamically typed** languages(动态型语言)
- **Weakly typed** languages(**弱类型语言)**
- **Muti-paradigm** language **: functional** & **event-driven behavior** 支持函数式和事件驱动行为的多范式语言
- **Has application interface(API) : handling text, arrays, and HTML Document Object Mode(DOM)** 有应用程序编程接口(API):处理**文本**、**数组**和 HTML **文档对象模型**(DOM)
- does not **include input/output (I/O) module for networking, storage or any graphics interface. 不包括**用于网络、存储或任何图形界面的**输入/输出（I/O）模块**。

Just-in-time compiler (JIT)

"Just-in-time compiler" (JIT) 是一种编译器，它在**运行时**（即程序执行时）将源代码或中间代码（通常是字节码）编译成本地机器代码，以提高程序的执行效率。与传统的解释器不同，JIT 编译器会**在需要时动态地将代码编译成机器码**，而不是一次性解释整个程序.
现代的 JavaScript 引擎：V8（Chrome 和 Node.js 使用的引擎）、SpiderMonkey（Firefox 使用的引擎）和 Chakra（Edge 使用的引擎），都使用了 JIT 编译器来执行 JavaScript 代码，以**提高其性能**。



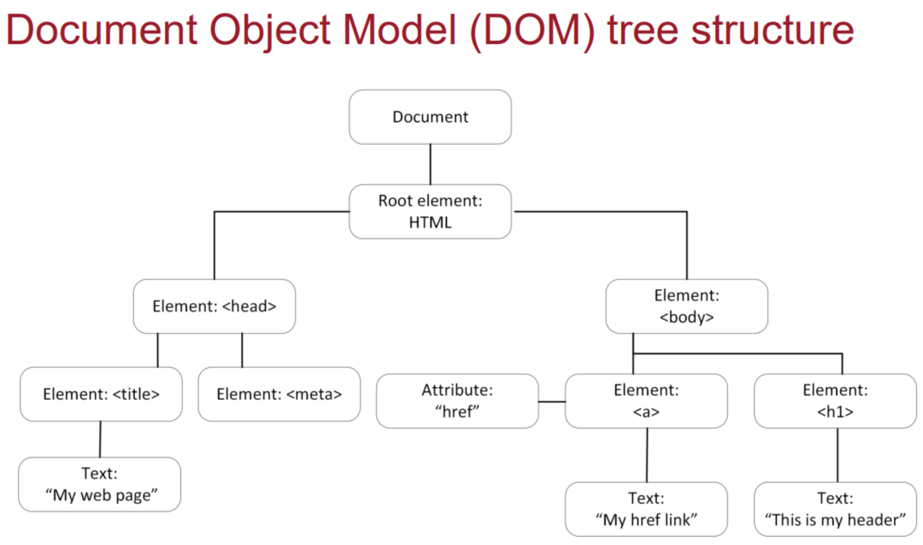
JavaScript communication with user examples

• console.log("Hello world!");
- 这行代码将字符串 "Hello world!" 输出到控制台，通常用于调试和记录信息。
• window.console.error(“Error");
- 这行代码将字符串 "Error" 作为错误消息输出到控制台的错误级别。
• window.alert("Hello world!"); //referencing the global object directly
- 这行代码将弹出一个包含 "Hello world!" 的警告框，用于向用户显示消息。
• alert("Hello world!"); //shorthand way
这行代码是 window.alert("Hello world!"); 的简写形式，直接调用了全局对象的 alert 方法。
• window.prompt("Are you feeling ok");
- 这行代码将弹出一个提示框，询问用户是否感觉好，并且等待用户输入。用户可以在提示框中输入文本，然后点击确定或取消。

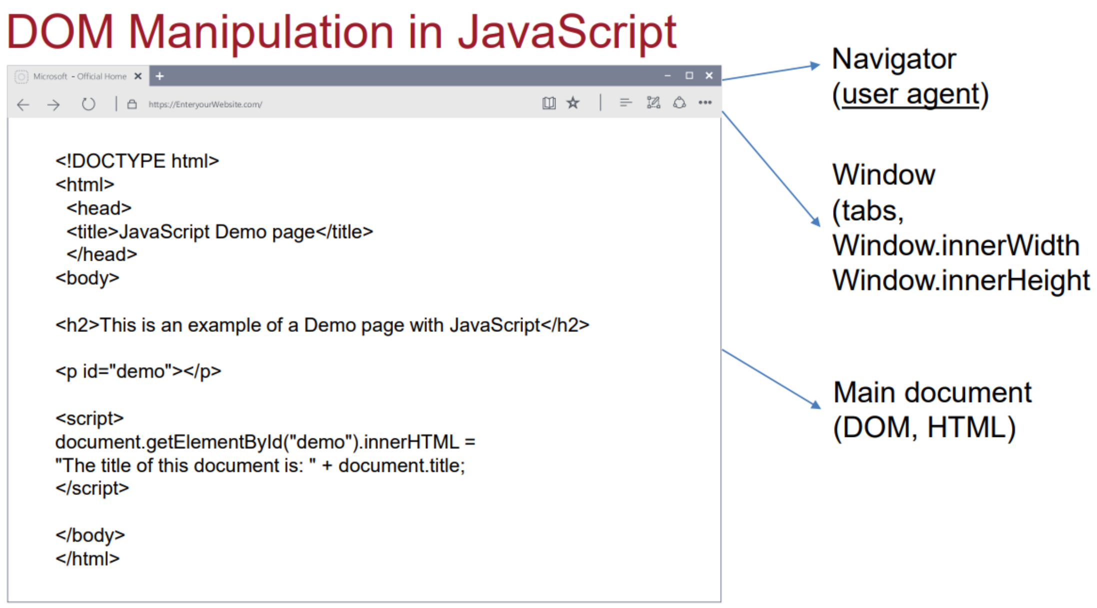
JavaScript engines – JavaScript 引擎

Every browser starts two processes for every web page；
1.Render Engine: It is responsible to **display the web page**. Parses the HTML and CSS and displays the content on the screen. 渲染引擎：负责显示网页。解析 HTML 和 CSS，并将内容显示在屏幕上。
2. JavaScript engine: where JavaScript **code gets executed. JavaScript is interpreted,** not a compiled language. (**Modern browsers** use a technology called **Just-In-Time (JIT)** compilation that compiles the code **to an executable bytecode**). JavaScript 引擎：这是 JavaScript 代码执行的地方。JavaScript 是解释型语言，而不是编译型语言。现代浏览器使用一种称为即时（Just-In-Time，JIT）编译的技术，将代码编译为可执行的字节码。

DOM 树形结构 – DOM tree structure



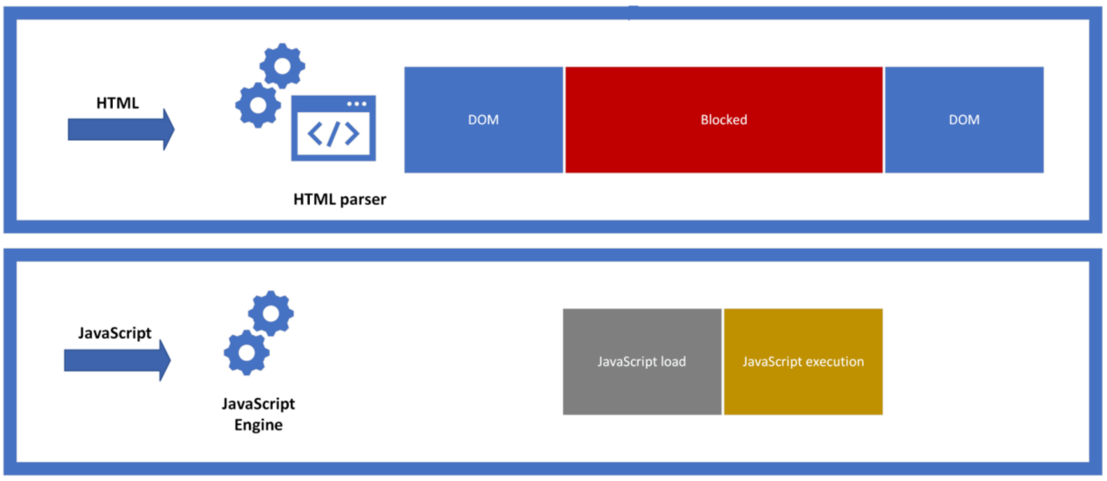
DOM Manipulation in JavaScript　DOM 操作



```
<!DOCTYPE html>
<html>
  <head>
  <title>JavaScript Demo page</title>
  </head>
<body>

<h2>This is an example of a Demo page with JavaScript</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"The title of this document is: " + document.title;
</script>

</body>
</html>
```

How to Make JavaScript Execute After HTML Load onload Event
If we use JavaScript to handle a Document Object Model (DOM) then our code **executes after HTML** (blocked code until OK). 如果我们使用 JavaScript 处理文档对象模型（DOM），那么我们的代码**将在 HTML 加载完成后执行**(直到 OK！ 之前都会阻塞)。

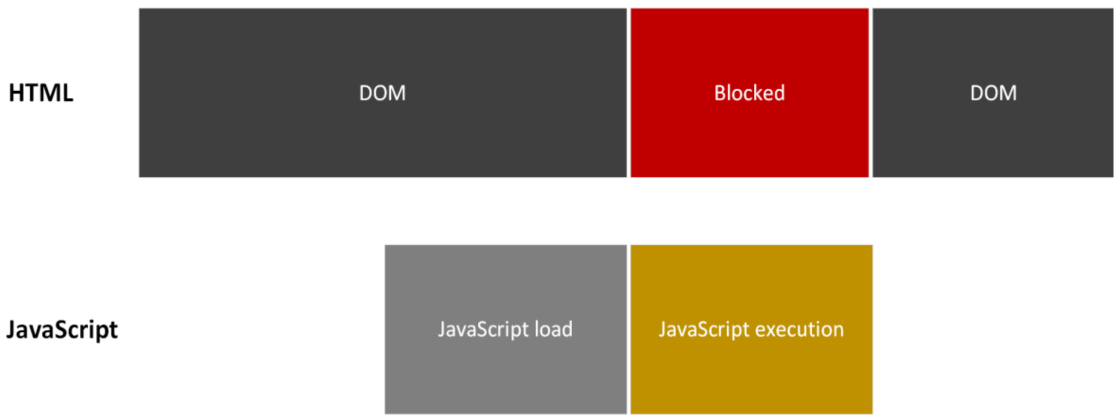
Execute JavaScript: parser-blocking scripts (synchronous) -解析器阻塞脚本（同步）



Asynchronous JavaScript – 异步 JavaScript

We can run function in parallel **Asynchronously vs Synchronous** coding: is executed line by line (JavaScript has single thread execution) each line waits for previous line to finish long time operation. 与同步编程相比：它按行执行（JavaScript 采用单线程执行），每行都等待前一行

完成，对长时间操作造成阻塞。



Variable Names（变量名）

• JavaScript variable names cannot include any Unicode character 不能包含任何 Unicode 字符。
• **First** character can be any of **a-z, A-Z or (_) or $** 第一个

```
let $4 = 1;      → Block
let _4 = 10;
var $_$ = "money"; → Function
var I_AM_HUNGRY = true;
```

typeof Operator – typeof 运算符

We can use the typeof operator to **identify** the **data type** of a Javascript variable: 来确定 JavaScript 变量的数据类型
let x= 10; ：
console.log(typeof x);
>number

String in JavaScript

- 和 Java 用法差不多，1. 用+来连接两个字符串 2. "\" 是 escape point(z 转义字符)
- 在 javaScript 中的可以用的有关 String 的方法（method）

```
let myName = 'Manolis
> myName.length;
7
> myName[0];
'M'
> myName[myName.length-1];
's'
> myName.slice(0,3);
'Man'
```

```
> myName.indexOf('lis');
4
> myName.slice(3);
'olis'
> myName.toLowerCase();
'manolis'
> myName.toUpperCase();
'MANOLIS'
>
myName.replace('lis','s');
'Manos'
```

比如这个例子:
String.length -> 计算 string 字符串长度
String [index] -> 打出指定地点的 char 字符（index=0：首个字符|| index = value.length-1:末尾字符）
String.slice(start_index, need_length) -> 打印出 String 的一个指定为 need_length 长度的片段
String.indexOf('字符串片段') -> 这个片段的第一个字母在字符串中的下标
String.slice(index) -> 从 index 起之后到末尾这个字符串所有的东西
String.toLowerCase(); & String.toUpperCase(); -> 返回大小写转换之后的值
String.replace('Original_Slice','aim_Slice') 替换 Stirng 中的特定片段

Arrays in JavaScript(数组)

Arrays are objects which can hold multiple values. Example:

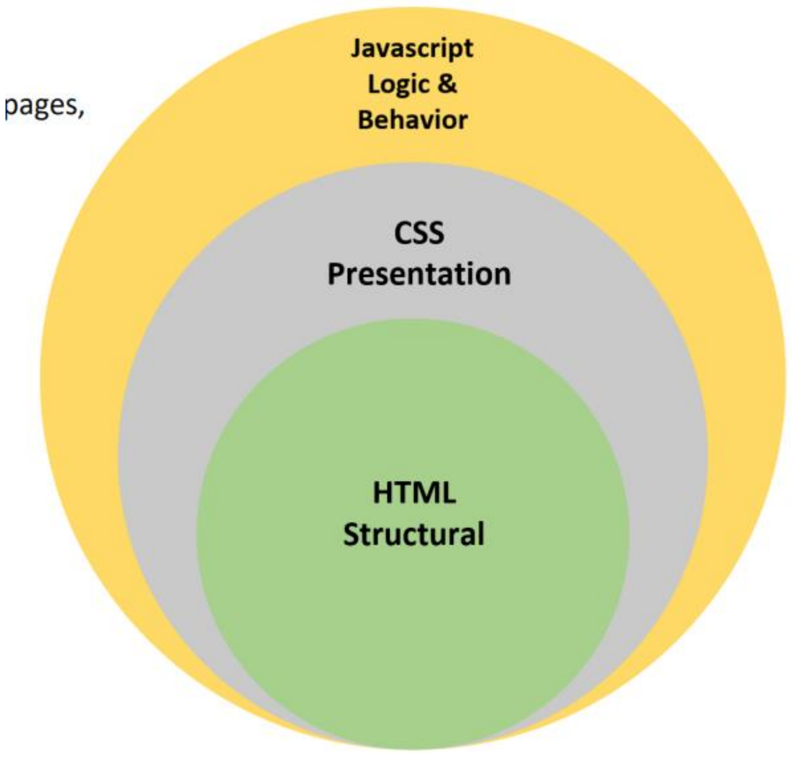
```
> let numbers = ['one', 'two', 'three'];
undefined
> numbers
[ 'one', 'two', 'three' ]
> let values = [1, 2, 3, 4, 5, 6, 10];
undefined
> let variety = ['one', 7896, [0, 1, 2]];
undefined
> variety
[ 'one', 7896, [ 0, 1, 2 ] ]
```

JavaScript & html

JavaScript is a **scripting language**（脚本语言） utilized within an HTML document, to **add functionality to the document** or if you prefer to make a document **dynamic**. With JavaScript, what we a**dd complex features on web pages**, specifically, we can（JavaScript 是一种脚本语言，用于在 HTML 文档中添加功能，使文档具有动态性。）：

- Dynamically update html page content 动态更新 HTML 页面内容
- Control multimedia\images content of a page ， 控制多媒体/图像内容
- Control, change text 控制，改变文本

HTML& CSS& JavaScript 范围对比图：



JavaScript Functions 函数

Every function contains a set of instructions in a logical sequence so that a specific result is always achieved. For function in JavaScript（每个函数都包含一系列逻辑顺序的指令，以确保始终实现特定的结果。对于 JavaScript 中的函数）：

• Every function must have a **specific and unique** name (e.g., myFunction) 每个函数必须有一个特定且唯一的名称

• Same rules for naming variables apply to function names 同变量命名规则适用于函数（特定且唯一）名称

• Word **"function"** must precede the name of a function (e.g., function myFunction).（函数名称前**必须有单词"function"**）

• Each function name is followed by a **pair of parentheses** (e.g., function myFunction()).每个函数名称后面跟着一对括号

• The **set of instructions for each function** is contained within curly braces { and } (e.g., function myFunction() { . . . })每个函数的一组指令都包含在花括号{和}内
• The **function parameters** may be included within the parentheses, and there may be 0, 1, or more parameters 函数参数可以包含在括号内，可以是 0 个、1 个或多个参数
• Each function is **called by its name** (e.g., myFunction())通过其名称调用
• **Instructions within a function** are executed when the function is called by its name 函数内的指令在**函数通过其名称调用**时执行
• Each function could **be called multiple times** within a script 函数可以在脚本中被多次调用。
• An HTML document **may contain more than one function** 可能包含多个函数
• All the functions **are contained within** the <script> and </script> tags or wherever else code is inserted 所有函数都包含在<script>和</script>标签中或者其他代码插入的位置
一般格式以及举例:

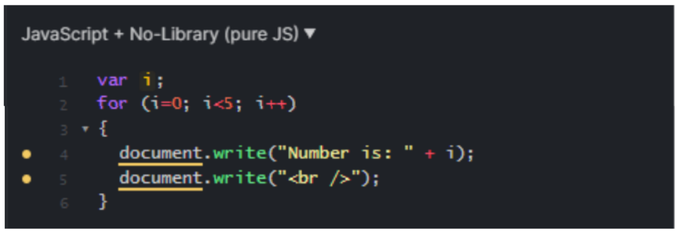
General syntax:
```
function FunctionName([param[, param[, ... param]]])
{
...
}
```

Example:
```
<script>
    function Hello() {
      alert("Welcome!");
    }
  </script>
```

JavaScript Loops – for & while Loop

就和 java 的 for 循环差不多。

```
for (Initial counter value; Condition; Counting step) {
  // code block to be executed
}
```
Example:

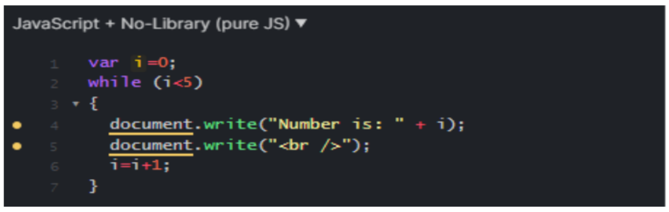


```
var i;
for (i=0; i<5; i++)
{
  document.write("Number is: " + i);
  document.write("<br />");
}
```

```
Number is: 0
Number is: 1
Number is: 2
Number is: 3
Number is: 4
```

```
while (condition) {
  // code block to be executed
}
```

Example:



```
var i=0;
while (i<5)
{
  document.write("Number is: " + i);
  document.write("<br />");
  i=i+1;
}
```

```
Number is: 0
Number is: 1
Number is: 2
Number is: 3
Number is: 4
```

JavaScript Cookies

定义：Cookies are **small text files** that are **stored on the computer** and **contain data** in the form of name=value pairs. For example: visitor=vist123. Cookie 是存储在计算机上的小型文本文件，以 name=value 对的形式包含数据。

To write (or store) and read cookies on the machine, the command used is **document.cookie**. 要在计算机上写入（或存储）和读取 cookie，使用的命令是 document.cookie

格式： document.cookie = "cookieName=cookieValue";

Overall, the cookie is stored as a string, and any additional parameters you can pass are separated by a question mark (;).

总的来说，cookie 被存储为一个字符串，可以通过问号分隔任何额外的参数（;）。

Example: document.cookie = "userid=Fe80gRCCijyH4mgdO; expires=Sun, 13 Jun 2021 20:31:59 GMT; path=/“

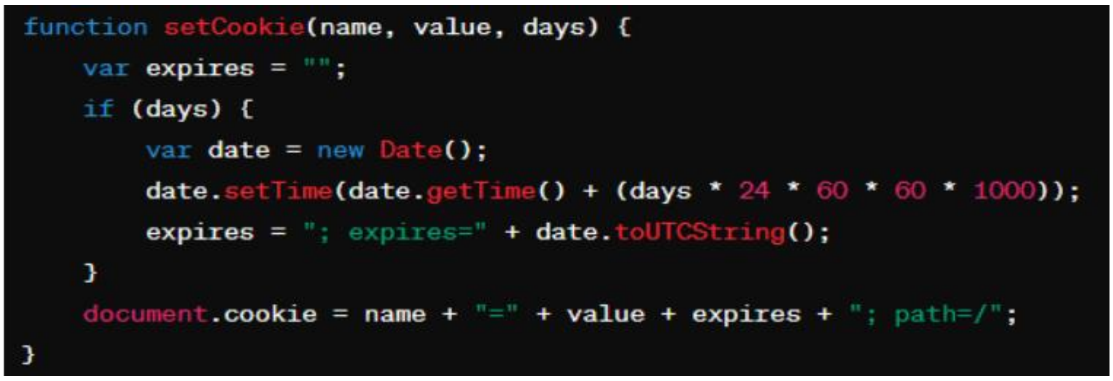
This is a JavaScript function to create a cookie: 这是一个用于创建 cookie 的 JavaScript 函数：

Cookies **store user/visitor information Cookies 存储用户/访问者信息**

When a browser requests a web page from the server, page cookies are added to that request.

当浏览器从服务器请求网页时，页面的 cookie 会被添加到该请求中。

Cookie 的 JavaScript 函数



```
function setCookie(name, value, days) {
    var expires = "";
    if (days) {
        var date = new Date();
        date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
        expires = "; expires=" + date.toUTCString();
    }
    document.cookie = name + "=" + value + expires + "; path=/";
}
```

Cookie – Privacy concerns

- Tracking 追踪
- Profiling 个人资料化
- Data Collection 数据收集
- Third-Party vs First-Party Cookies 第三方与第一方 Cookie（？）

Cookie – How to protect

- Browser Privacy Settings: private browsing, cookie blocking, and thirdparty cookie restrictions 浏览器隐私设置：私密浏览、阻止 cookie 以及第三方 cookie 限制
- Cookie Blockers, extensions or browsers Cookie 阻止工具、扩展或浏览器
- Anonymization with Tor (not 100% safe) 使用 Tor 进行匿名化（不是 100%安全）
- Use of Tails, a portable operating system that protects against surveillance and hides your identity. 使用 Tails，一种可移动操作系统，可保护免受监视并隐藏您的身份

Object-Oriented Programming with JavaScript (OOP)

The JavaScript language has been **designed on an object-oriented basis**. With OOP a developer can（写出 JavaScript 使用 OOP 的优点）：

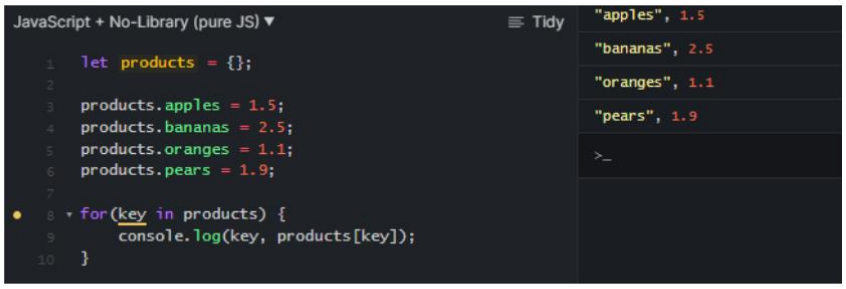
- Create their **own objects** and organize better the code by making it **more flexible and maintainable** 创建自己的对象，并通过使代码更加灵活和可维护来更好地组织代码
- Use the **pre-made built-in objects** provided by JavaScript (JSON, Date, Math) 使用 JavaScript 提供的预制内置对象（JSON、Date、Math 等）
- We use OOP to model or **better describe real-world examples** 使用面向对象编程来模拟或更好地描述现实世界的示例
- The objects can contain **data or methods**. With objects we **incorporate** the data and their behavior into a block of code 可包含数据或方法。通过对象将数据及其行为合并到一块代码中
- Objects can **interact** with one another 对象可以相互交互
- We can use an **API methods** to access and communicate with the object 可以使用 API 方法访问和与对象通信

JavaScript Built-in objects

JSON (JavaScript Object Notation) is used for **storing and transmitting data**, primarily in web applications. JSON (JavaScript 对象表示法) 用于存储和传输数据，主要用于 Web 应用程序。

- JSON objects share many similarities with Array objects. 对象与数组对象有许多相似之处
- Data is recorded in the format key: value 键值对贮存格式

Example:

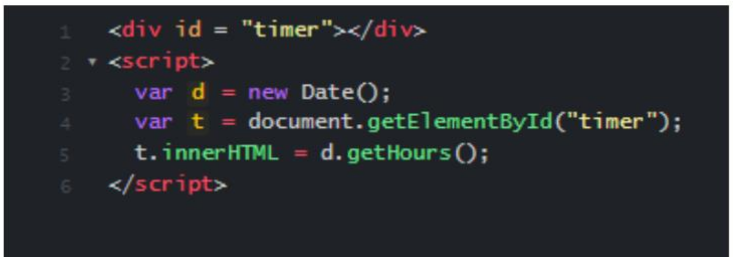


```
let products = {};

products.apples = 1.5;
products.bananas = 2.5;
products.oranges = 1.1;
products.pears = 1.9;

for(key in products) {
    console.log(key, products[key]);
}
```

```
"apples", 1.5
"bananas", 2.5
"oranges", 1.1
"pears", 1.9
>_
```

The date object is used for managing dates and times

Example:



```
<div id = "timer"></div>
<script>
    var d = new Date();
    var t = document.getElementById("timer");
    t.innerHTML = d.getHours();
</script>
```

Objects and Instances (对象和实例)

In JavaScript, anything stored in a variable is an object. To use an object, we first need to create an instance of that object. 在 JavaScript 中，存储在变量中的任何东西都是对象。要使用对象，我们首先需要创建该对象的一个实例

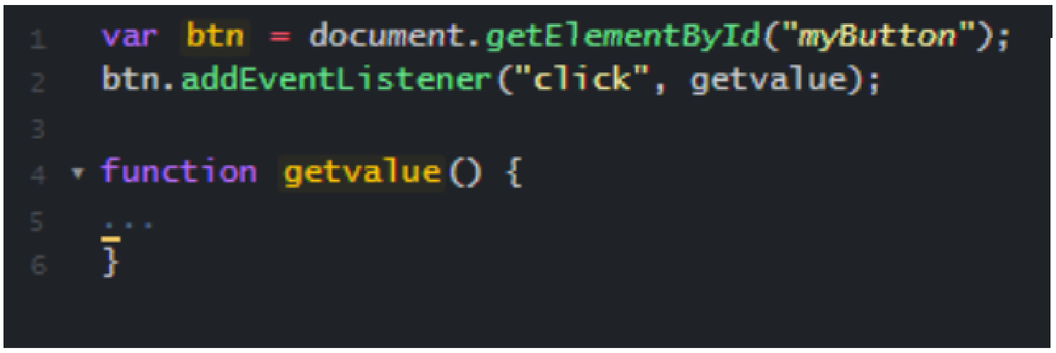
Creating an object is done using the new operator -> **var d = new Date();使用 new 运算符创建对象**

Each instance (or object, as we will call it) has:（每个实例（或对象，我们将其称为对象）具有）

- Properties or characteristics (features, properties, or fields) 属性或特征（特性、属性或字段）
- Functions or methods 函数或方法

- Ability to handle events 处理事件的能力

An object can have **sub objects** (child objects) -> **window.document** 一个对象可以有子对象（子对象）
To add an event handler, we use the method -> **addEventListener("eventname", functionname);** 要添加事件处理程序，我们使用方法，如下图：

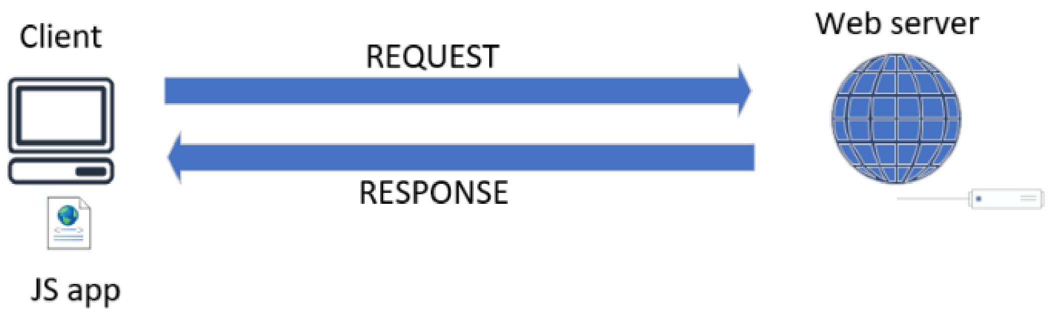


```
var btn = document.getElementById("myButton");
btn.addEventListener("click", getvalue);

function getvalue() {
    ...
}
```

Asynchronous JavaScript and AJAX

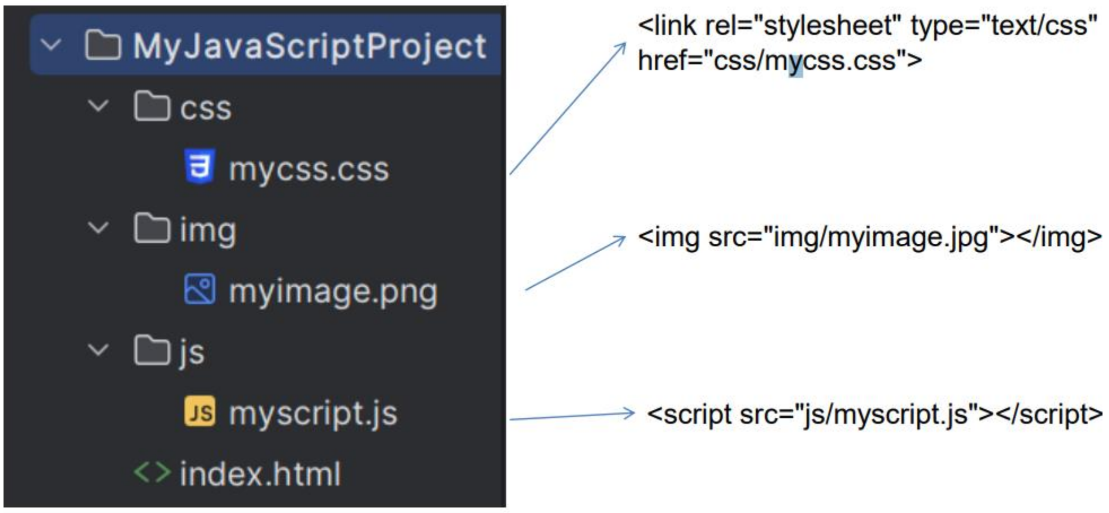
AJAX (Asynchronous JavaScript And XML): 异步 JavaScript 和 XML）

- It can **communicate with remote web** servers in an asynchronous manner 以异步方式与远程 Web 服务器通信。
- It requests data from web servers dynamically 动态请求来自 Web 服务器的数据



解释这种异步技术： AJAX 是一种技术，允许网页在不重新加载整个页面的情况下，通过 JavaScript 与服务器进行通信和交换数据。与传统的同步方式不同，AJAX 允许页面在发送请求的同时继续执行其他操作，从而提高了用户体验和页面的响应速度。通过使用 AJAX，网页可以动态地从服务器获取数据，并将其显示或处理，而无需刷新整个页面。这使得网页更加灵活、动态，并且可以更有效地与用户交互。
额外阅读资料：https://developer.mozilla.org/zh-CN/docs/Learn/JavaScript

How to organize your project folder and file structure（VS code 里的）



```
<link rel="stylesheet" type="text/css"
href="css/mycss.css">

<img src="img/myimage.jpg"></img>

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

Web Scraping（网页抓取）

Reason for learning web scraping 理由

Previously: writing and serving websites, presenting data in a website.
Now: collecting webpages, extracting data from them.
The web is a massive data resource – why would we only access it manually?
以前：编写和服务网站，在网站中呈现数据。
现在：收集网页，从中提取数据。

Swarms of robots 机器人

Approximately half of web traffic is not human 大约一半的网络流量不是人为的.
Robots visit websites for **a variety of reasons**:
- **Indexing websites** for search or archive purposes. 为搜索或存档目的对网站建立索引
- Carrying out **specific tasks** for their owners (e.g., newsreader-like tools). 为其所有者执行特定任务（例如，类似新闻阅读器的工具）
- Probing for **security vulnerabilities**. 探测安全漏洞
- Pretending to be humans for **nefarious purposes** (e.g., in order to advertise something on social media). 出于**邪恶目的**冒充人类（例如，为了在社交媒体上做广告）
- Generating **false 'traffic'** so website owners can **scam ad networks**. 生成虚假"流量"，以便网站所有者可以欺骗广告网络。

Crawlers（爬虫者）

Depending on the usage, also a ‘spider’ – because it **traverses the web**: 根据用途，也可以是“蜘蛛”——因为它遍历网络：

- Spiders are indexing content, usually for **search purposes** 正在对内容建立索引，通常用于搜索目的
- Other ‘crawlers’ may have other uses for the content. 其他“爬虫”可能对该内容有其他用途。

A crawler is **a HTTP client**, like a browser, but **automated**. Two approaches:类似一个自动化 HTTP 浏览器

- write your own purpose-specific software - 编写您自己的专用软件
- use an existing tool – 使用现有工具

记住：**wget <url>**

wget is a crawler

理念: do one thing well 把一件事情做到最好
only used the bare minimum of wget’s capacity:
Downloading webpages for offline consumption: `wget` https://news.ycombinator.com/news
解释： 这条命令会下载指定网址的 HTML 页面，并将其保存在当前工作目录下，文件名为 `news`。
Resources required to display the page correctly: `wget -p` https://news.ycombinator.com/new
解释：这条命令中的`-p`选项表示“下载页面所需的所有资源”，包括 HTML 页面、CSS 样式、JavaScript 文件、图片等。这意味着除了网页本身的内容，还会下载页面中引用的所有资源文件，以确保页面在离线情况下能够正确显示。
前者**只下载 HTML 页面**，而后者下载 **HTML 页面以及页面中引用的所有资源文件**。

Robot.txt

Standard for websites to use to tell HTTP or FTP crawlers which parts of a site can be accessed. 网站用于告诉 HTTP 或 FTP 爬网程序可以访问网站的哪些部分的标准。
爬虫可以访问网站哪些部分的规则列表:
List of rules for which parts of the site a crawler can access:

```
User-Agent: foobot
Allow: /example/page/
Disallow: /example/page/disallowed.gif

User-Agent: bazbot
Disallow: /example/page.html
```

Crawlers are meant to **check for their own user-agent string** in the file (always placed in the webroot) and follow the rules. 爬虫程序旨在检查文件中自己的用户代理字符串（始终放置在 webroot 中）并遵循规则。
Robot.txt – issue

- Doesn't actually enforce access restrictions. 实际上并不强制执行访问限制
- Need real security mechanisms to do that (e.g., authentication). 需要真正的安全机制来做到这一点（例如，身份验证）。
- By listing site components, makes content findable. 通过列出网站组件，使内容易于查找

More of an 'honour system' to enable **good bots** to **respect the wishes of website owners**. 更多的是一个"荣誉系统"，使好的机器人能够尊重网站所有者的意愿。
wget is a good bot.

More wget

elinks danluu.com
In the labs you'll *practice recursive* downloading and true web mirroring using wget. 练习*递归*下载真正的的网络镜像
Key use of this: make a **personal offline backup** of a website you like.
其主要用途：对您喜欢的网站进行个人离线备份。

Other limits on crawling(爬虫的其他限制)

You have a **generally unrestricted right** to access public web content. This doesn't mean you can do anything you want with it. 您通常拥有不受限制的访问公共网络内容的权利。但是这并不意味着您可以用它做任何您想做的事情。

- **copyright limits** on republishing content 重新发布内容的**版权限制**
- **ethical limits**, especially for authenticated content 道德限制，特别是对于经过身份验证的内容

The means（方式） of accessing content can also be important – aggressive downloading of large sites can pose a **burden on servers**. Servers sometimes respond by blacklisting clients.
Generally 'polite' to introduce small delays between requests, even if not asked for.
访问内容的方式也很重要——**大量**下载大型网站可能会**给服务器带来负担**。服务器有时会通过将客户端列入**黑名单**来做出响应。通常，即使没有要求，也会"礼貌"地在请求之间引入较小的延迟。 || API endpoints designed for automated access may impose different rate limits. 专为自动访问而设计的 API 端点可能会施加不同的速率限制。

What else can we do with it?（有关爬虫我们还能做些其他什么）

Webpages often present **structured information** which **we would like to** work with.
However, **page structures can be complex**, and are very site-specific.
Need a system for accessing page content **programmatically.**
Javascript has methods for this within the browser, but often we want our **code to run on its own.**
网页通常呈现我们想要使用的结构化信息。然而，页面结构可能很复杂，并且非常特定于站

点。所以需要**一个**以**编程**方式访问页面内容的系统。已知 Javascript 在浏览器中具有实现此目的的方法，但通常我们希望我们的**代码能够独立运行**。

BeautifulSoup（美丽汤）

Python library for extracting data from HTML files. 这是用于从 HTML 文件中提取数据的 Python 库。
- Not a HTML parser itself, but **can use many parsers**. 本身不是 HTML 解析器，但可以使用许多解析器
- Provides a **flexible interface** for navigating, searching and altering HTML programmatically. 提供灵活的界面，用于以编程方式导航、搜索和更改 HTML

Current version is BeautifulSoup 4 (bs4). 当前美丽汤版本

HN demonstrates（**演示**）

```
from bs4 import BeautifulSoup

filename = "news.html"
handle = open(filename, 'r')
text = handle.read()
soup = BeautifulSoup(text)
```

Python ‘s Dicts & Lists

Dicts

```
d = {}
d['this'] = 'that'
d['other'] = 'value'
d = {'a': 1, 'b': 2}
```

Lists

```
l = ['a']
l.append('b')
l.append('c')
```

Cryptography

An introduction to OpenSSL, PGP and Let’s Encrypt.