Para Aplicaciones Web Modernas – Parte II

Fernando Saez
saezfernando@Gmail.com

JavaScript Functions 4 Ways

```
// Function Declaration
function square(x) {
  return x * x;
}
// Function Expression
const square = function(x) {
  return x * x;
}
// Arrow Function Expression
const square = (x) => {
   return x * x;
}
// Concise Arrow Function Expression
const square = x => x * x;
```

Contexto de Ejecución

El contexto de ejecución (EC) se define como el entorno en el que se ejecuta el código JavaScript. Por entorno, nos referimos al valor de this, variables, objetos y funciones a las que el código JavaScript tiene acceso en un momento determinado.

1. Contexto de ejecución global (GEC)
2. Contexto de ejecución funcional (FEC)
3. Eval: Contexto de ejecución dentro de la función eval.

El motor de JavaScript crea un contexto de ejecución en las siguientes dos etapas:

1. Fase de creación
2. Fase de ejecución

This

En el Contexto de ejecución global (GEC) esto se refiere al objeto global, que es el objeto Windows en el browser y Global en Node.

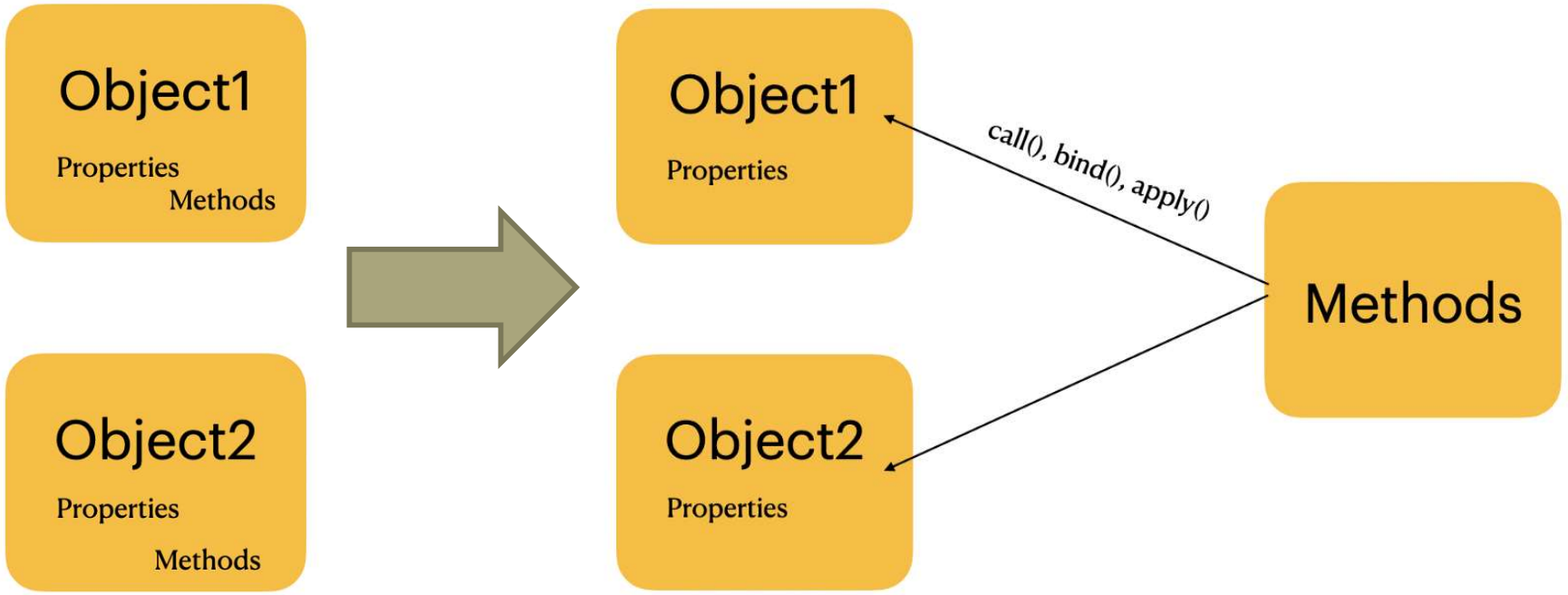
```
var occupation = "Frontend Developer";

function addOne(x) {
    console.log(x + 1)
}
```

```
window.occupation = "Frontend Developer";
window.addOne = (x) => {
console.log(x + 1)
};
```

En el caso del contexto de ejecución funcional (FEC), no se crea el objeto this. Más bien, obtiene acceso al entorno en el que está definido.

En los objetos, la palabra clave this no apunta al GEC, sino al objeto en sí mismo.

Call, Apply y Bind



1

```
var obj = { num: 2 };

function add(a){
  return this.num + a;
}
```

2

```
add.call(obj, 3);
```

Call, Apply y Bind

1

```
var obj = { num: 2 };

function add(a, b){
  return this.num + a + b;
}

console.log(add.call(obj, 3, 5));
```

2

```
console.log(add.apply(obj, [3, 5]));
```

3

```
const func = add.bind(obj, 3, 5);
func(); // Returns 10
```

Closure

- Inner function tienen acceso a las variables locales de la outer function

Outer function

```
function iniciar() {
  var nombre = "Mozilla";  // La variable nombre es una variable local.
  function mostrarNombre() {    //mostrarNombre() es una función interna,
                                 // una clausura.
    alert(nombre);               // Usa una variable declarada en la
                                 //función externa.
  }
  mostrarNombre();
}
iniciar();
```

- Una closure es la combinación de una función y el lexical scoping en el cual la función se declaro.

Ejercicio 1

```
function foo(){
  let value=10;
  return () => {console.log(value++);}
}

var bar1 = foo();
var bar2 = foo();

bar1(); // ??
bar1(); // ??
bar2(); // ??
bar2(); // ??
```

Clousure:
Un ejemplo práctico:
Retornar funciones-parametrizadas

```
function creaSumador(x) {
  return function(y) {
    return x + y;
  };
}

var suma5 = creaSumador(5);
var suma10 = creaSumador(10);

console.log(suma5(2));  // muestra 7
console.log(suma10(2)); // muestra 12
```

Closure:
Un ejemplo práctico: Crear contexto privado y optimizar (1)

```
var meses = [   "Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio",
"Julio",  "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"];

function getMes (n) {
        if (n < 1 || n>12) throw new RangeError("Rango incorrecto");
        return meses[n - 1];
}

console.log(getMes(3)); // Marzo
```

Closure:
Un ejemplo práctico: Crear contexto privado y optimizar (2)

```
function getMes (n) {
  var meses = [
    "Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio",
    "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"];

  if (n < 1 || n>12) throw new RangeError("Rango incorrecto");
  return meses[n - 1];
}

console.log(getMes(9)); // Septiembre
```

Closure:
Un ejemplo práctico: Crear contexto privado y optimizer (3)

```
var getMes = (function () {

var meses = [ "Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio",
   "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"];

  return function inner(n) {
   if (n < 1 || n>12) throw new RangeError("Rango incorrecto");
   return meses[n - 1];
 };
})();

console.log(getMes(12)); // Diciembre
```

IIFE Immediately-invoked Function Expression

```
(function () {
    statements
})();
```

1

```
(function () {
    var nombre = "Barry";
})();
Console.log(nombre) // throws "Uncaught
ReferenceError: aName is not defined“
```

2

```
var resultado = (function () {
    var saludo = “Hola";
    return saludo;
})();
Console.log(result); // "Barry"
```

Clases

```
class Documento {
  constructor(titulo, autor, esPublicado) {
    this.titulo = titulo;
    this.autor = autor;
    this.esPublicado = esPublicado;    }
  publicar(){
    this.esPublicado = true;
  }
}

class Libro extends Documento{
  constructor(titulo, autor, topico){
    super(titulo, autor, true);
    this.topico = topico;
  }
}
```

Clases

```
class MyObject {
    constructor(param1, param2) {
    let atributo1 = param1; // atributo privado
    this.atributo2 = param2; // atributo público
    this.metodo1 = function(…) { // método
                                        público.
      // cuerpo del método
   }
     let metodo2 = function(…) { // método
                                        privado
      // cuerpo del método
  }
}
var obj = new MyObject(x,y); //crea instancia
Obj.atributo1 //Error
```

Métodos estáticos

```
class Punto {
  constructor ( x , y ){
    this.x = x;
    this.y = y;
  }
  static distancia ( a , b) {
    const dx = a.x - b.x;
    const dy = a.y - b.y;
    return Math.sqrt ( dx * dx + dy * dy );
  }
}
const p1 = new Punto(5, 5);
const p2 = new Punto(10, 10);
console.log (Punto.distancia(p1, p2)); //
 7.071067811865475
```

Módulos (Exportar e Importar)

```
// exportar un modulo en el archive lib/greetings.js
module "utils" {
    export function greeting(name){
        console.log("Hi! " + name);
    }
}
```

```
// importa la function greeting desde el modulo utils
import { greeting } from "utils";
var app = {
     welcome: function(){
          greeting("Mike");
} }
```

Módulos

```
// ES6 // lib/math.js
export function mult(a, b){
  return a*b;
}
export const PI = 3.141593;
export default function(a, b){
  return a + b;
}
//Podemos exportar todo lo que necesitemos en una única
//línea al final del archivo
//export { mult, PI}
```

```
Import defaultmember {mult, PI} from "./math.js"
```

Módulos

```
// existen varias formas de importar un módulo
import defaultMember from "module-name";
import * as name from "module-name";
import { member } from "module-name";
import { member as alias } from "module-name";
import { member1 , member2 } from "module-name";
import { member1 , member2 as alias2 , [...] } from
"module-name";
import defaultMember, { member [ , [...] ] } from "module-
name";
import defaultMember, * as name from "module-name";
import "module-name";
```

Módulos (Importar en el navegador)

```
<script type="module" src="main.js"></script>
```

1) Los módulos solo se ejecutan una vez, incluso si se les ha hecho referencia en varias etiquetas <script>.

2) las características del módulo se importan al alcance de un solo script — no están disponibles en el alcance global.

3) Import y export están disponibles solo para módulos, no podemos utilizarlos en scripts standars

4) Agregan algunos mecanismos de seguridad extra

Iterables

Un objeto es **iterable** si define cómo se itera.

Tipos integrados iterables: Array, Map, Set y TypedArray.

for...of.

Ver Iteradores y generadores
https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Iterators_and_Generators
Ver TypedArray
https://developer.mozilla.org/es/docs/Web/JavaScript/Vectores_tipados

for .. of

```
//ES5
var numbers = [1,2,3,4,5];
numbers.forEach(function(value) {
    console.log(value);
}); //1, 2, 3, 4, 5
```

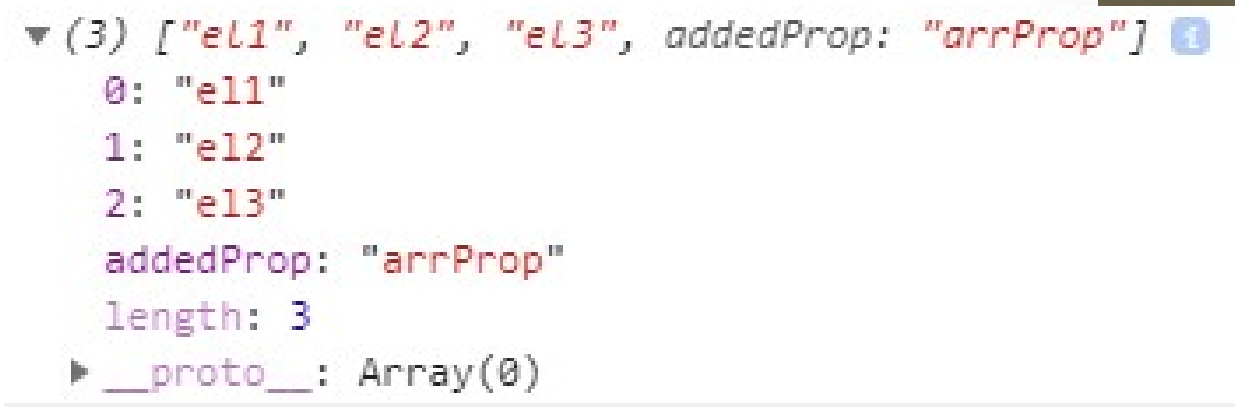
```
//ES6
var numbers = [1,2,3,4,5];
for(let item of numbers){
    console.log(item);
}; //1, 2, 3, 4, 5
```

Diferencias entre for .. in y for .. of

for..in itera sobre todas las propiedades enumerables de un objeto.

for..of itera sobre los valores de un objeto iterable. (arrays, strings, map, set)

```
let arr = ['el1', 'el2', 'el3'];
arr.addedProp = 'arrProp';

for (let elKey in arr) {
  console.log(elKey); // 0, 1, 2, addedProp
}

for (let elValue of arr) {
  console.log(elValue) // el1, el2, el3
}
```



Map

```
//ES6
let map = new Map();
map.set('foo', 123);
let user = {userId: 1}; //object
map.set(user, 'Alex');
map.get('foo'); //123
map.get(user); //Alex
map.size; //2
map.has('foo'); //true
map.delete('foo'); //true
map.has('foo'); //false
```

size

clear()
forEach()
get()
has()
keys()
set()
values()

Map

Un objeto Map puede iterar sobre sus elementos en orden de inserción.

Un bucle for..of devolverá un array de [clave, valor] en cada iteración.

```
map = new Map([['user1','Alex'], ['user2', 'Vicky'], ['user3', 'Enrique']]);
```

Diferencias entre objetos y mapas

Set

Los sets son conjuntos de elementos no repetidos, que pueden ser tanto objetos, como valores primitivos.

```
let set = new Set();
set.add('foo');
set.add({bar:’baz’});
set.size  //2
for(let item of set){
   console.log(item);
}
//"foo"
//{bar:’baz’}
```

size

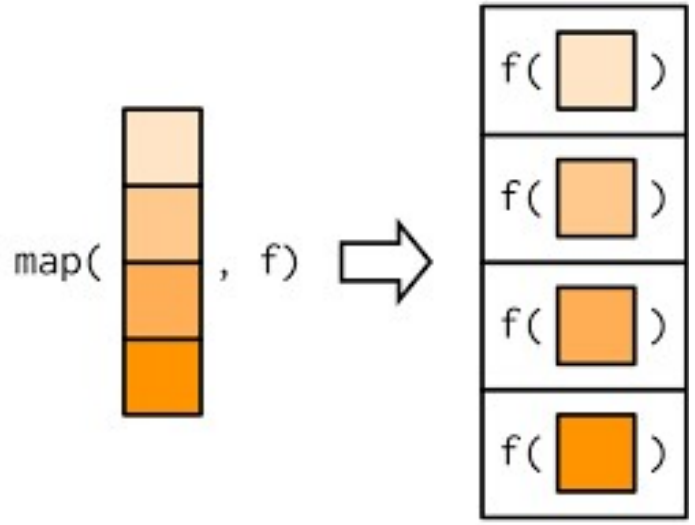
clear()
forEach()
get()
has()
keys()
add()
values()

Un poco de programación funcional

Las técnicas funcionales pueden ayudarte a escribir código más declarativo.

- map
- filter
- reduce
- find
- forEach

Un poco de programación funcional



map

Cuando llamas a map() en un array, este ejecuta una función en cada elemento dentro de él, retornando un nuevo array con los valores que la función retorna.

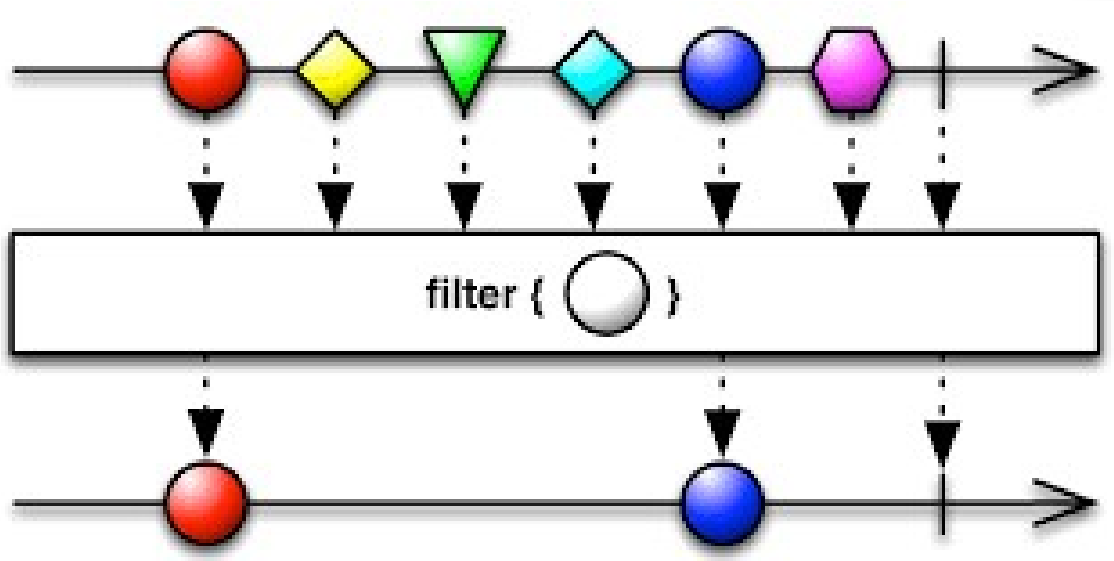
```
var myArray = [10, 20, 30];
var newArray = myArray.map(number => number + 1);
console.log(newArray);
// [11, 21, 31]
```

Map() - Ejemplo

```
const reg = /\d{3}/g;
const str = "Java323Scr995ip4894545t";
const nuevaStr = str.replace(reg, "");
console.log(nuevaStr);

const arr = [
  "fer555nan123do",
  "hola534 789que ta983l",
  "c532om453o estas234!!",
];
arr.map((item) => item.replace(reg, "")).forEach((item) =>
console.log(item));
```

Un poco de programación funcional

filter()



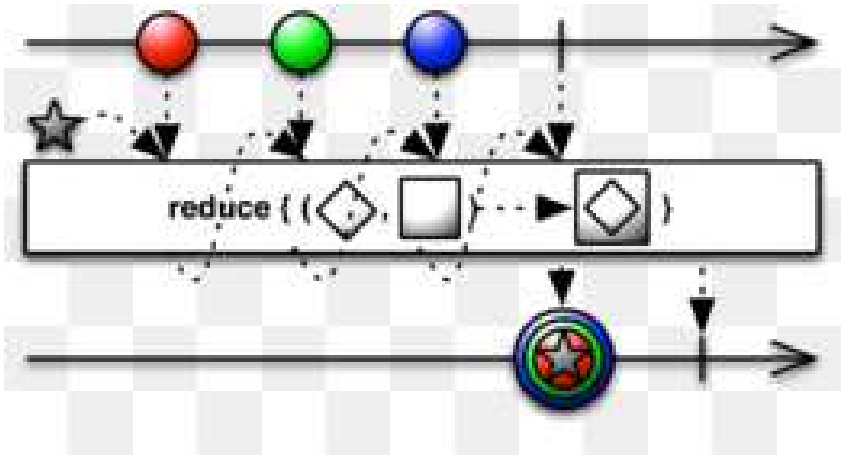

```
var myArray = [10, 20, 30, 40];
var filteredValues = myArray.filter(number => number > 20);
filteredValues
// [30, 40]
```

filter() - Ejemplo

```
let usuarios = [
  {id: 1, name: “Jose“, isAdmin:true},
  {id: 2, name: “Ana“, isAdmin:false},
  {id: 3, name: “Juan“, isAdmin:true}
];

// Retorna array con los 2 primeros usuarios
let usuarios1y2 = usuarios.filter(item => item.id < 3);
alert(usuarios1y2.length); // 2
//Retorna los usuarios administradores
Let usuariosAdmin = usuarios.filter(item=> item.isAdmin)
alert(usuariosAdmin.length); // 2
```

Un poco de programación funcional

reduce()




```
var myArray = [10, 20, 30];
var total = myArray.reduce((accumulator,actual) => {
  return accumulator + actual;
});
total; // 60
```

reduce() - Ejemplo

```
let usuarios = [
  { name: "Jose", job: "Data Analyst", country: "AR" },
  { name: "juan", job: "Developer", country: "US" },
  { name: "Ana", job: "Developer", country: "US" },
  { name: "Karen", job: "Software Eng", country: "CA" },
  { name: "Jonas", job: "QA", country: "CA" },
  { name: "Ale", job: "Designer", country: "AR" },
];
let usuariosAgrupadosPorPais = usuarios.reduce((acumuladorGrupo,
usuario) => {
  let newkey = usuario["country"];
  if (!acumuladorGrupo[newkey]) acumuladorGrupo[newkey] = [];
  acumuladorGrupo[newkey].push(usuario);
  return acumuladorGrupo;
}, []);
```

Un poco de programación funcional

reduce()

```
var myArray = [10, 20, 30];
var objectCreatedFromArray =
myArray.reduce((accumulator, number, index, array) => {
    accumulator[`number${index}`] = number;
    return accumulator;
}, {});
objectCreatedFromArray;
// {number0: 10, number1: 20, number2: 30}
```

find()

```
const array1 = [5, 12, 8, 130, 44];
const found = array1.find(element => element > 10);
console.log(found);
// output: 12

const foundIndex = array1.findIndex(element => element > 10);
console.log(found);
// output: 1
```

forEach()

```
const array1 = ['a', 'b', 'c'];

array1.forEach(element => console.log(element));

// output: "a"
// output: "b"
// output: "c"
```

Fernando Saez
saezfernando@Gmail.com

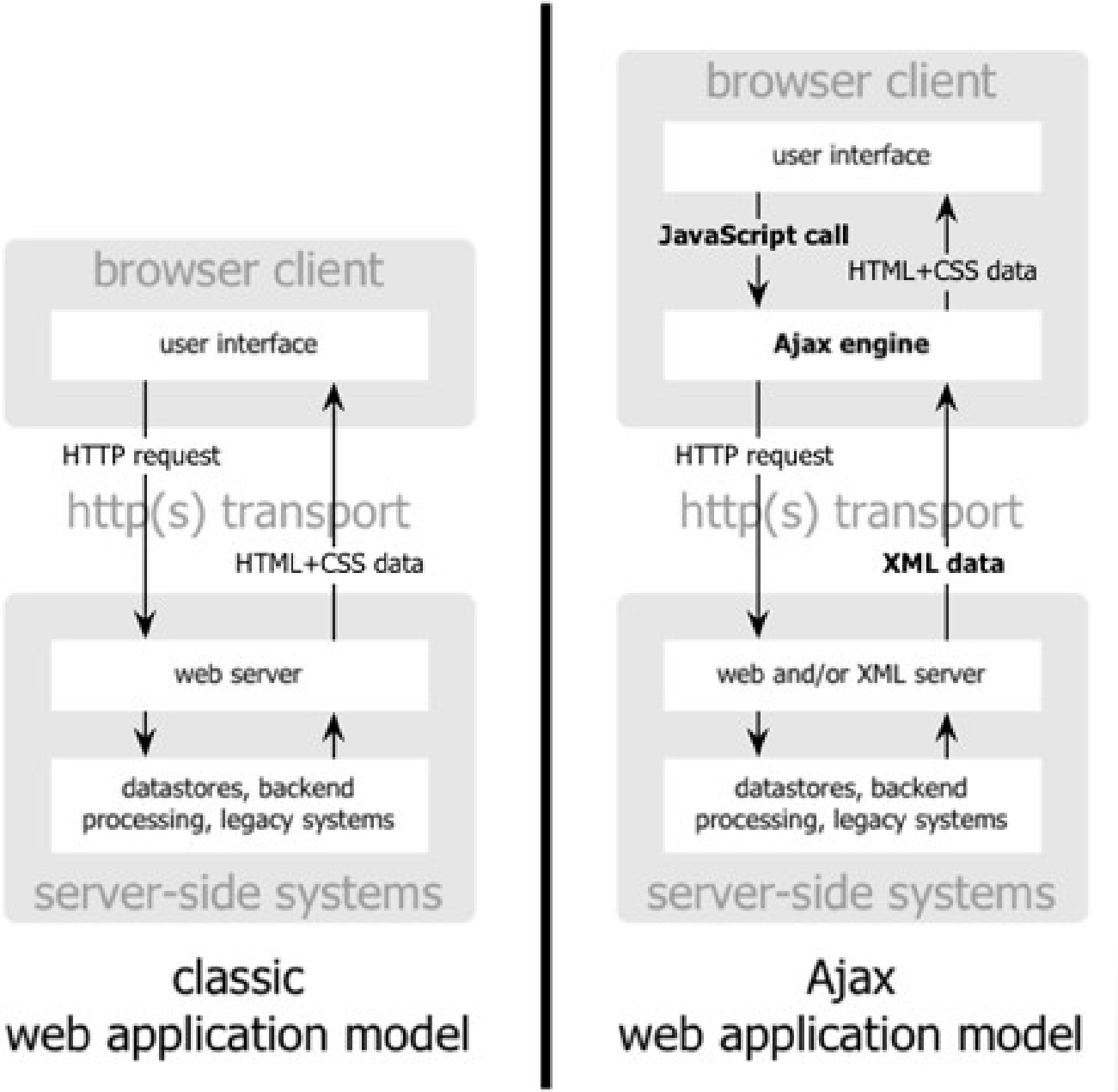
AJAX: Asynchronous JavaScript & XML

- Técnica de desarrollo Web para crear aplicaciones Web Interactivas
- Las aplicaciones se ejecutan en el navegador del usuario y se comunican asíncronamente con el servidor en background.

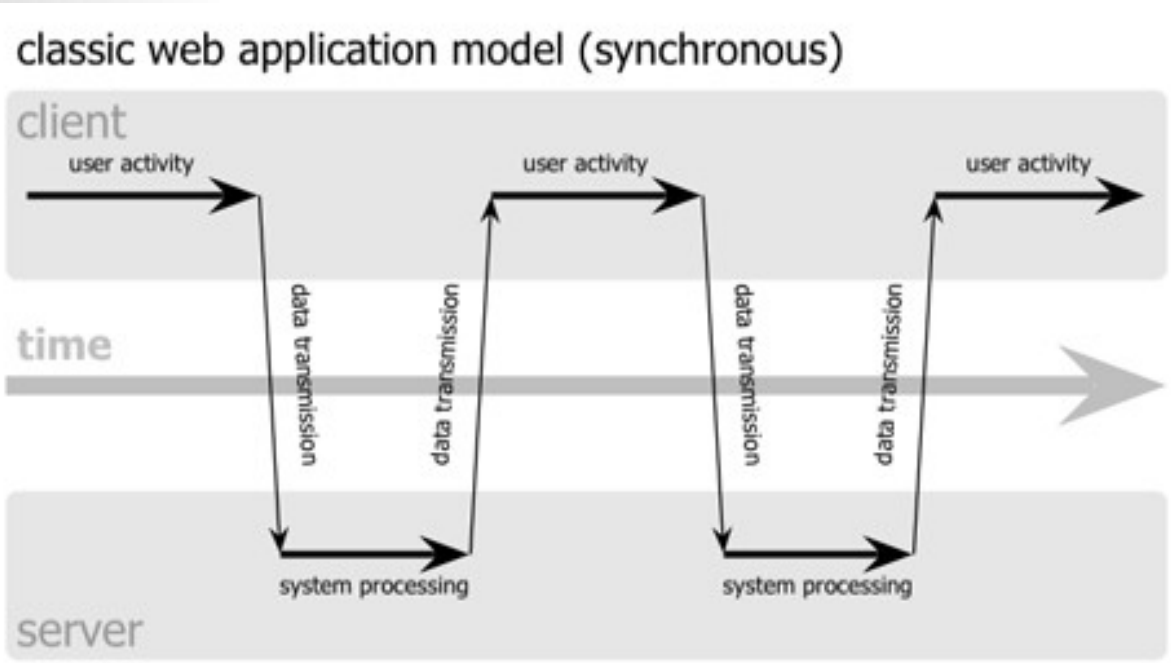
Ajax engloba varias tecnologías

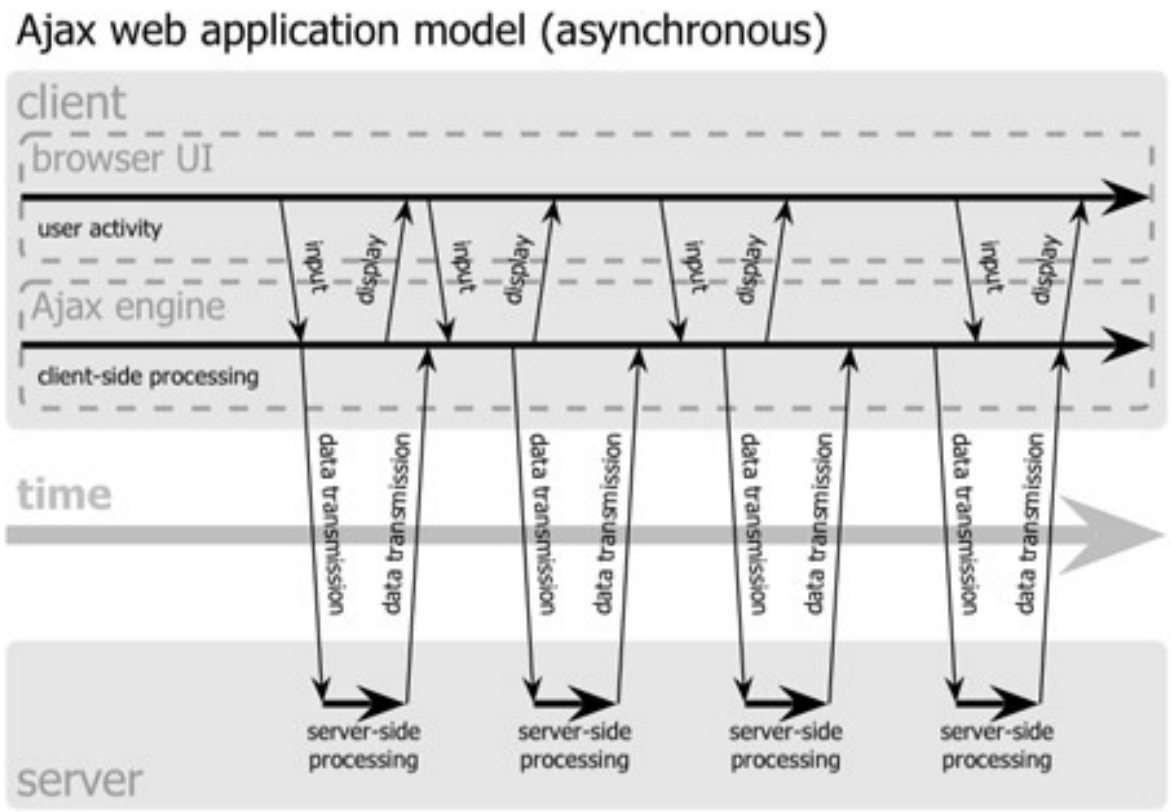
- HTML y CSS para la presentación
- DOM para mostrar e interactuar con la información dinámica
- Intercambio de datos usando XML y XSLT (también JSON)
- XMLHttpRequest para el intercambio de datos asíncronamente.
- Javascript para unir todo

AJAX vs Apps Web Tradicionales



Síncrono vs. Asíncrono





Hacer un request de una URL

- Objeto xmlHttpRequest

1. Crear la URL

   url = “http://server/personas”

2. Crear el objeto XMLHttpRequest

   objXMLHttp=new XMLHttpRequest()

3. Indicar al browser el nombre de la función manejadora de la respuesta.

   objXMLHttp.onreadystatechange = function() { … }

4. Realizar la petición

   objXMLHttp.open("GET", url, true);

   objXMLHttp.send();

XMLHttpRequest

En el Cliente

```
onclick="makeRequest()">
…
function makeRequest(){
xmlHttp = new XMLHttpRequest();
var url="servertime.php"
xmlHttp.onreadystatechange=actualizarDocumento;
   xmlHttp.open("GET",url,true)
   xmlHttp.send(null)
}
function actualizarDocumento(){
let data = xmlHttp.responseXML;
var nodo =
data.getElementsByTagName('time').item(0);
alert(nodo.firstChild.data);
}
```

```
// En el servidor servertime.php

<?
   // a time document
   header("Content-type: text/xml");
   print("<time>");
   print( date('Y-m-d'));
   print("</time>");
?>
```

Retorna

```
<?xml version="1.0" ?>
<time>
    2021-03-18.
</time>
```

JSON

```
//Si el servidor retorna un JSON
{
   "firstName": "John",
   "lastName": "Smith",
   "age": 25,
   "address": {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": 10021 },
      "phoneNumbers": [
          {
            "type": "home",
            "number": "212 555-1234“
          },
          {
            "type": "fax",
            "number": "646 555-4567“
          }
     ]
}
```

```
//Leer JSON en el cliente
var my_JSON_object = {};
var http_request = new XMLHttpRequest();
http_request.open("GET", url, true);
http_request.onreadystatechange = function ()
{
      var done = 4, ok = 200;
      if (http_request.readyState == done &&
http_request.status == ok) {
        my_JSON_object =
JSON.parse(http_request.responseText);
      }
};
http_request.send(null);
```

```
//Actualiza DOM con valores
My_JSON_object.firstname=“John”;
My_JSON_object.lastname=“Smith”;
My_JSON_object.age=25;
…
```

jQuery/AJAX

```
$.ajax({
    type: "POST",
    url: "example.php",
    data: "name=John&location=Boston“
 }).done( function(msg)  {
     alert( "Data Saved: " + msg );
 }).fail( function( xmlHttpRequest, statusText, errorThrown ) {
     alert(
            "Your form submission failed.\n\n"
              + "XML Http Request: " + JSON.stringify( xmlHttpRequest )
              + ",\nStatus Text: " + statusText
              + ",\nError Thrown: " + errorThrown );
 });
```

Métodos jQuery AJAX

```
<script
    src="https://code.jquery.com/jquery-3.6.0.js"
    integrity="sha256-H+K7U5CnXl1h5ywQfKtSj8PCmoN9aaq30gDh27Xc0jk="
    crossorigin="anonymous"
  ></script>
// Cargar el contenido de una página HTML en un elemento
$("div#noticias").load("noticias.html");

// Descargar un script cliente desde el servidor y ejecutarlo
$.getScript("/ruta/hasta/miScript.js");

// Petición GET al servidor con parámetros y función de respuesta
// La petición POST es idéntica, salvo que se cambia $.get por $.post
$.get("/ruta/hasta/el/scriptDelServidor.php",{
        idProducto: "AX00342", cantidad: "3" },
 function(data){    alert("Se ha añadido al carrito: " + data);   });
```

Métodos jQuery AJAX

- Jquery: Librería poderosa de Javascript
  - Simplificar tareas comunes de javascript
  - Acceder a “partes” de una página
    - Usando CSS o XPath
  - Modificar la apariencia de una página
  - Generar fácilmente peq. animaciones
  - Soporte simplificado para Ajax
  - Validación de formularios mas sencilla

Fetch

- Provee un método global fetch() para obtener recursos de forma asíncrona por http.

- Fetch proporciona una alternativa mejor a XMLHttpRequest

```
fetch('http://example.com/movies.json')
  .then(response => response.json())
  .then(data => console.log(data));
```

API Fetch

```
fetch('flores.jpg').then(function(response) {
  if(response.ok) {
    response.blob().then(function(miBlob) {
      var objectURL = URL.createObjectURL(miBlob);
      miImagen.src = objectURL;
    });
  } else {
    console.log('Respuesta de red OK pero respuesta HTTP 404');
  }
})
.catch(function(error) {
  console.log('Hubo un problema con la petición Fetch:' + error.message);
});
```

Axios

- Axios es un cliente HTTP basado en promesas para node.js y el navegador.
- En servidor $ `npm install` axios
- En cliente `<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>`

```
axios('http://example.com/movies.json’)
    .then(response => console.log(response.data));
```

API Axios

```
axios.post('/user/12345',
  {
    firstName: ‘Alejandro',
    lastName: ‘Magno'
  }
}).then(function (response) {
    console.log(response);
  });
```

API Axios

```
axios({
    method: ‘post',
    url: ‘/user’
    data : {nombre:’jose’, edad: 34},
    headers: {'X-Requested-With': 'XMLHttpRequest’}
    params: {ID:12345}
  }
}).then(function (response) {
    console.log(response);
  });
```