theme : "white" transition: "zoom" highlightTheme: "darkula" customTheme : "lola_theme"

Tema 6:

JavaScript Avanzado

EI1042 - Tecnologías y Aplicaciones Web

EI1036- Tecnologías Web para los Sistemas de Información

(2018/2019)

Professora: Dra. Dolores Mª Llidó Escrivá

Universitat Jaume I.

Índice

El elemento SCRIPT

Atributos:

- **src** : URI del recurso con los programas a ser cargados.
- **type:** por defecto **text/javascript**.
- **charset** : Por defecto utf-8. La codificación de caracteres.
- **async:** Boolean. **true** indica que el fichero del src debe ser cargados asincrónicamente.
- **defer:** Booleano. **true** el programa será recuperado en paralelo al procesamiento del documento y evaluado sólo cuando se haya completado el procesamiento del documento.

--

Script

Los scripts sin atributo async o defer, son interpretados y ejecutados inmediatamente, antes de que el navegador continúe procesando la página, lo que puede ralentizar la carga de la página.

Ejemplo Async + Template

http://piruletas.cloudaccess.host/teoria/T6/cargaTemplateAsync.html

```
<script type="text/javascript"
src="listarTemplateAsync.js" async defer >
</script>
```

Función Callback

Un callback es un tipo de función que se pasa como parametro a otra función por referencia y se ejecutada desde una subrutina.

```
function ejemplo2(fn) {
  var nombre = "Pepe";
  fn(nombre);
}

ejemplo2(function(nom) {
  console.log("hola " + nom);
});  // "hola Pepe"
```

Ejemplos:

```
x.addEventListener ("Evento", funcion_Callback, Boolean*)

setTimeout(funcion_Callback[, retraso]);
```

setTimeout: ejecuta la función callback tras los segundos indicados en retraso.

--

Problemas paso valor/referencia:

```
var one = function() { mike.showName(); };
var two = mike.showName;
var three = mike.showName();
var four = (function() { mike.showName(); })();
```

¿Qué diferencia hay entre las distintas asingnaciones?

```
setTimeout(mike.showName(), 5000); //Error

setTimeout(function(){mike.showName();},5000);//correcto
```

Función Flecha => (arrow functions)

Forma de definir funciones anónimas más cortas.

`([param] [, param]) => { instrucciones }`

`param => expresión` (Un parámetro)

```
(function(quien){alert("hola" + quien)})("mundo");
(quien => alert("hola" + quien))("mundo");

["mundo"].map(quien => alert("hola" + quien))
```

El método *array.map()* crea un array como resultado de aplicar función que se pasa por parámetro a cada elemento del array del método. Es una función *callback*.

--

Ejemplo:

```
var elementos = [ "Hidrógeno","Helio","Litio"];
a = elementos.map(function(elemento){
      return elemento.length;
});

b = elementos.map((elemento) => {
  return elemento.length;
});

c = elementos.map( elementos => elementos.length );
console.log(a);
```

Closure

- En JavaScript, el ámbito de una variable se define por su ubicación dentro del código fuente y las funciones anidadas tienen acceso a las variables declaradas en su ámbito externo.

- Un **closure** permite que una función dentro de otra función contenedora pueda hacer referencia a las variables después de que la función contenedora ha terminado de ejecutarse. Recuerda el entorno en el que se ha creado.
- REGLA:Toma el último valor de la variable de la función contenedora.

--

Cuestiones:

¿Qué funciones se han definido?

¿Dónde se ha definido nombre?

¿Qué diferencia hay entre los 2 alerts?

¿Hay algún error de ejecución?

```
function init() {
  var nombre= "Mozilla";
  function displayName() {alert(nombre);}
    displayName();
  }
init();
alert(nombre);
```

--

Ejemplos Closure

```
function llamaotra1(A){
  console.log(A);
  return "1";
 }
function llamaotra2(A){
  console.log(eval(A));
  return "2";//ejecuta funcion
}
function funcionllamada(B){
  console.log(B);
  return 0;
}

llamaotra1(funcionllamada("Estoy Aqui1"));
llamaotra1('funcionllamada("Estoy Aqui2")');
llamaotra2('funcionllamada("Estoy Aqui3")');
```

¿Hay algún callback? ¿Que aparecerá en la pantalla? EjemploClosure1

--

Problema Closures

```
<p id="help">Helpful notes will appear here</p>
<p>E-mail: <input type="text" id="email" name="email"></p>
<p>Name: <input type="text" id="name" name="name"></p>
<p>Age: <input type="text" id="age" name="age"></p>
```

```
function showHelp(help) {
  document.getElementById('help').innerHTML = help;
}

function setupHelp() {
var helpText = [
    {'id': 'email', 'help': 'Dirección de correo electrónico'},
    {'id': 'name',  'help': 'Nombre completo'},
    {'id': 'age',   'help': 'Edad (debes tener más de 16 años)'} ];

for (var i = 0 ; i < helpText.length; i++) {
   var item = helpText[i];
   document.getElementById(item.id).addEventListener
    ("focus",function() { showHelp(item.help)});}
}

setupHelp();
```

http://piruletas.cloudaccess.host/teoria/T6/closureProblem.html

--

Solución con función flecha

```
helpText.forEach(item => document.getElementById(item.id).addEventListener
    ("focus",
    function() { showHelp(item.help);}
     ));
```

--

Cuestiones ejemplo anterior:

¿Qué valor muestra en la ayuda al pulsar sobre *Name*?

¿Por qué?

¿Hay algún callback?

Solución: usar let en el bucle.

http://piruletas.cloudaccess.host/teoria/T6/closureProblemSolv.html

Modelo Asíncrono

Asincronía: acción que no tiene lugar en total correspondencia temporal con otra acción. (Fuente: Wikipedia).

¿Ejemplo de asincronía?

Promesas- Promise

- Patrón de diseño para controlar la ejecución de un determinado cómputo del cual no sabemos cómo, ni cuándo se nos va a devolver un determinado valor.
- Una promesa es un objeto que por medio de una máquina de estados podamos controlar cuándo un valor está disponible o no.
- Los métodos de las promesas devuelven promesas, permitiendo que las promesas se puedan encadenar.
- EmacScript 7

--

Métodos Promesas

promise = new Promise(function(resolve[, reject]) {});

- promise.resolve (Obligatorio). Método que se ejecuta para indicar que la promesa se completó correctamente.
- promise.reject (Opcional). Método que se ejecuta para indicar que la promesa se rechazó con un error.

Para encadenar promesas en caso de éxito o fracaso tenemos:

- promise.then(onFulfilled,onRejected) : Método para indicar que hacer cuando una promesa devuelve el valor deseado. **Resolve**
- promise.catch(onRejected): Método para indicar que hacer cuando una promesa devuelve el valor no deseado. **Reject**

--

Promise.resolve()

Retorna un objeto *Promise* que es resuelto con el parámetro:

- si el valor es una promesa, esa promesa es devuelta;
- si el valor es un *thenable* (si tiene un método "then"), el valor devuelto le seguirá a ese thenable, adoptando su estado;
- de otro modo la promise devuelta estará completada con el valor.

```
var p1 = new Promise(function(resolve, reject) {
  resolve('456');
});

p1.then(function(value) {
  console.log(value);
  // expected output: 456
});
```

```
var promise1 = Promise.resolve(123);

promise1.then(function(value) {
  console.log(value);
  // expected output: 123
});
```

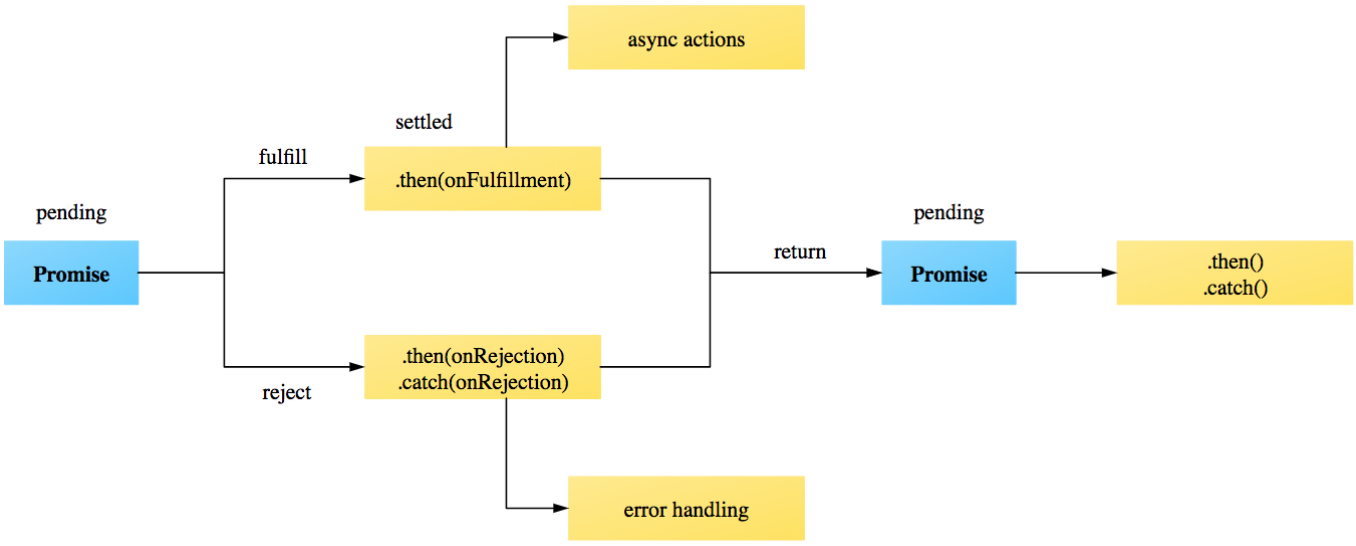
--

Estados de una Promesa

Una promesa puede encontrarse en 4 estados:

- **fulfilled (cumplida)** : la acción relacionada con la promesa se completa con éxito.
- **rejected (rechazada)** : la acción relacionada con la promesa no se completa con éxito.
- **pending (pendiente)** : aún no está completa ni se rechaza.
- **settled (finalizada)** : se completa o se rechaza.

--



--

Ejemplo

```
function isDinnerTime() {
    return new Promise(function(resolve, reject) {
        setTimeout(function () {
            const now = new Date();
            if (now.getHours() >= 22) {
                resolve('yes');
            } else {
                reject('no');
            }
        }, 50);
    });
}

isDinnerTime()
    .then(data => console.log('success:'+data));
    .catch(data => console.log('error'+data));
```

--

¿Qué problemas tienen las promesas?

- Notación inteligente
- Anidamientos

Funciones asíncronas: Async / Await

- Te permite escribir un código basado en promesas como si fuese síncrono, pero sin bloquear el hilo principal.
- Hacen a tu código asíncrono menos "inteligente" y más legible.
- Las funciones asíncronas siempre devuelven una promesa.
- EmacScript 7.
- Con **async** señalamos que la función es asíncrona, debe devolver una promesa.
- Una función async puede contener una expresión await, la cual pausa la ejecución de la función asíncrona y espera la resolución de la Promise pasada y, a continuación, reanuda la ejecución de la función async y devuelve el valor resuelto.

--

Ejemplo:

```
function isDinnerTime() {
    return new Promise(function(resolve, reject) {
        setTimeout(function () {
            const now = new Date();
```

```
            if (now.getHours() >= 12) {
                resolve('yes');
            } else {
                reject('no');
            }
        }, 50);
    });
}

async function hello() {
    a = await isDinnerTime();
    console.log("success: " + a);
}

hello();
```

--

HTTP Asíncrono

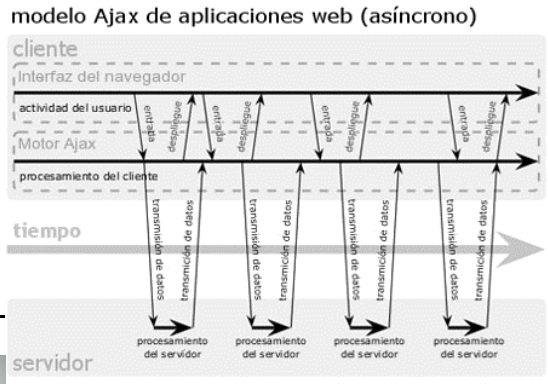
Soluciones en Javascript:

- API XMLHttpRequest: Eventos. Motor AJAX
- API FETCH API: Promesas.

AJAX: API XMLHttpRequest

- Permite realizar múltiples peticiones mediante una comunicación cliente/servidor Asincrona
- En el modelo C/S el usuario debe esperar a que se recargue la página completa cada vez que hace una petición al servidor.

- Con AJAX el usuario puede seguir trabajando con la información de que disponía, mientras el navegador carga los recursos solicitados asincronamente.



--

AJAX: Jesse James Garrett

- El término AJAX se presentó en 2005 por primera vez en el artículo "Ajax: A New Approach to Web Applications http://adaptivepath.org/ideas/ajax-ne w-approach-web-applications/ "
- En realidad, el término AJAX es un acrónimo de **Asynchronous JavaScript + XML** , que se puede traducir como "JavaScript asíncrono + XML".

--

Interacción

- AJAX mejora la interacción del usuario con la aplicación, evitando las recargas constantes de la página.
- En el **cliente** se requiere un motor del Ajax que establece las peticiones con el servidor de forma asíncrona,(API XMLHttpRequest).
- En el **servidor** no se requiere nada en particular, simplemente debe devolver el recurso que se le solicita.

--

API XMLHttpRequest: Métodos

```
new XMLHttpRequest();
```

- .open(método,URL,asincrono,usuario,clave)

  (Métodos: get,post,put - Asincrono: true(default)/false)

- .send(datos)

  (datos: Vacía método get.)

- .setRequestHeader()
- .getResponseHeader()
- .abort()

--

Ejemplo: Petición URL sincrona

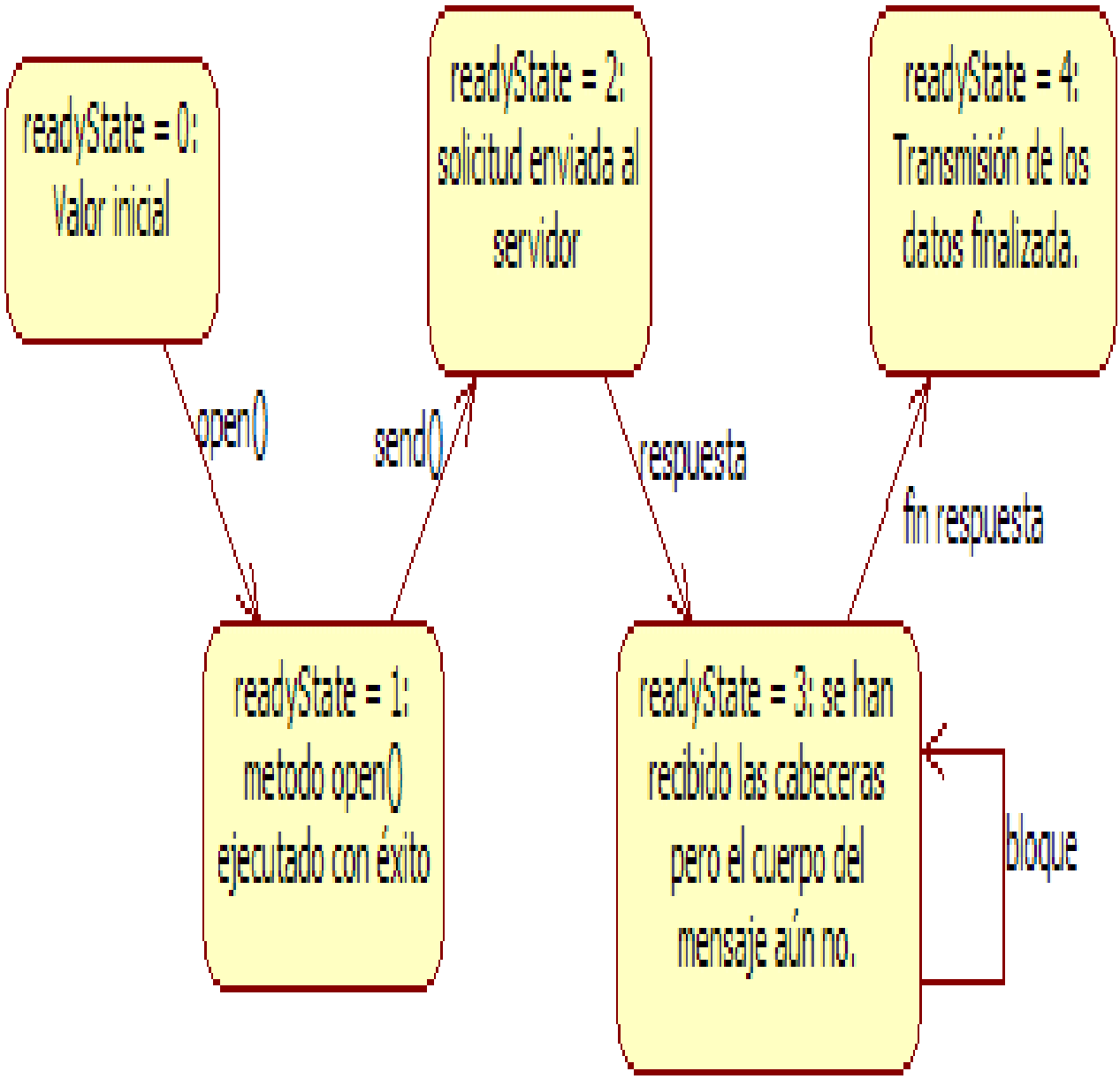
```
var Ajax1=new XMLHttpRequest();
Ajax1.open(“GET”,” http://www.example.com”);
Ajax1.send();
document.getElementByID("central").innerHTML=Ajax1.responseText;
```

- ¿Hay algun problema si es síncrono?
- ¿Y si es asíncrono?

--

Estados del motor de Ajax

API XMLHttpRequest



--

API XMLHttpRequest: propiedades

- .status (respuesta del estado del servidor)
- .onreadystatechange (handler del cambio estado)
- .responseXML: Respuesta formato XML
- .responseText: Respuesta formato HTML
- .timeout: Permite indicar el timeout para que se active evento asociado.
- .withCredentials: Asignamos true/false para permitir CORS
- .upload:** devuelve el objeto **XMLHttpRequestUpload** (envio imágenes)

--

Ejemplo: Carga hiperenlaces de forma asíncrona

http://piruletas.cloudaccess.host/teoria/T6/AsincroAjax.html

```
function cargaAjax(src_url, lugar){
var Ajax1=new XMLHttpRequest();
Ajax1.addEventListener('readystatechange', function()
{ if (this.readyState === 4 )
  { if (this.status< 400 )
    {lugar.innerHTML =Ajax1.responseText;}}});
Ajax1.open("GET",src_url);
Ajax1.send();
}

function ready()
 {enlace=document.querySelector("nav a");
 src_url=enlace.getAttribute("href");
 enlace.addEventListener("click",function (event)
  { event.preventDefault();
   cargaAjax(event.target.src,event.target.parentElement);
});}

document.addEventListener("DOMContentLoaded",
         function(){ready()});
```

--

Cuestiones AJAX

- ¿Por qué está vacío el método sent?
- ¿Cómo sabemos que se ha enviado la respuesta completa?
- ¿Cómo indicamos si queremos que se realice síncrono o asíncrono?
- ¿Cómo sabemos si se ha enviado la petición completa con POST?

--

Objeto XMLHttpRequestUpload

- Permite controlar las subidas de información HTTP asincrono al servidor.
- XMLHttpRequest.upload devuelve un objeto XMLHttpRequestUpload.

Eventos del objetho XMLHttpRequest y XMLHttpRequestUpload :

- loadstart

- progress
- abort
- error
- load
- timeout
- loadend
- readystatechange

--

Ejemplo envío JSON con JavaScript

http://piruletas.cloudaccess.host/teoria/T6/ASincroUpload.html

```
function EnviaAjax(url_src,updateNode,data){
    var Ajax1=new XMLHttpRequest();
    Ajax1.addEventListener("progress", function (event)
{updateProgress(event)});
    Ajax1.addEventListener("load", function (event)
                {transferComplete(event,updateNode,url_src)});
    Ajax1.addEventListener("error", transferFailed);
    Ajax1.addEventListener("abort", transferCanceled);
    Ajax1.upload.addEventListener("error", sendFailed);
    Ajax1.open("POST",url_src);
    //Ajax1.setRequestHeader('Content-Type', 'application/json')
    Ajax1.send(data);
}
}
```

FETCH API

- Objeto global para recolectar/buscar recursos devolviendo una promesa.
- Ofrece una definición genérica de los objetos Request y Response.
- API más simple y más limpio que XHTMLRequest.
- No envia ni recibe ninguna **cookie**.
- Utiliza las promesas.

fetch(RequestInfo input[,RequestInit init]);

Objeto **fetch** permite solicitar el recurso solicitado con input(URL/Request) y se retorna la promesa Response.

--

Fetch GET

```
fetch('./api/some.json')
  .then(
    function(response) {
      if (response.status !== 200) {
        console.log('Looks like there was a problem. Status Code: ' +
          response.status);
        return;
      }

      // Examine the text in the response
      response.json().then(function(data) {
        console.log(data);
      });
    }
  )
  .catch(function(err) {
    console.log('Fetch Error :', err);
  });
```

--

Input

- una cadena con la **URL** .
- o un objeto **Request** con los datos para realizar le patición.

Objeto Request

Objeto que contiene los datos para la solicitud de un recurso.

request = new Request(input [, init])

- Representa una solicitud de recursos.
- Se pasa como parámetro a fetch.

--

Init

Es un parámetro opcional, que es un Objeto JSON con las siguientes propiedades:

- Cadena **method**: request HTTP method:Por defecto GET.
- Objeto **headers**: request HTTP headers.
- Objeto **body**: request HTTP body.
- Cadena **mode**: cors, no-cors, same-origin, navigate.

--

Objeto Headers

- **Headers** : Objeto que Representa los encabezados de la respuesta/solicitud, lo que le permite consultar y tomar diferentes acciones en función de los resultados.

```
var myHeaders = new Headers();
myHeaders.append('Content-Type', 'text/xml');
myHeaders.get('Content-Type') // should return 'text/xml'
```

--

Objeto Body

Objeto proporciona métodos relacionados con el contenido de la respuesta/solicitud, lo que le permite declarar cuál es su tipo y cómo debe manejarse.

```
body: 'foo=bar&lorem=ipsum' ```

--

# Response

- Representa la respuesta a una solicitud.
- Es  la promesa que devuelve fetch.
- Devolverá un error sólo cuando hay un error de red. No codigo error HTTP como 404  o 500.

Métodos:

 - response.ok :   true (false) si el estado esta entre  200-299.
 - response.status:  Codigo HTTP de respuesta.

Tipos de respuesta:

 - response.arrayBuffer(): El objeto ArrayBuffer se usa para representar un buffer genérico, de datos binarios crudos (raw) con una longitud específica
 - response.blob():Un objeto Blob representa un objeto tipo fichero de datos planos inmutables
 - response.json()
 - response.text()


--

# Fetch: Request  Get

```js
var myHeaders = new Headers();
```
```

```
var myInit = { method: 'GET',
               headers: myHeaders,
               mode: 'cors',
               cache: 'default' };

var myRequest = new Request('flowers.jpg', myInit);

fetch(myRequest)
.then(function(response) {
 return response.blob();
})
.then(function(myBlob) {
 var objectURL = URL.createObjectURL(myBlob);
 myImage.src = objectURL;
});
```

--

Ejemplo: Carga una imagen

Ejemplo: http://piruletas.cloudaccess.host/teoria/T6/ImageSend.html

```
async function asyncCall() {
 var myImage = document.querySelector('#mi_imagen');
 const response = await
fetch('https://upload.wikimedia.org/wikipedia/commons/7/77/Delete_key1.jpg
');
 const imgblob = await response.blob()
 var objectURL = URL.createObjectURL(imgblob);
 myImage.src = objectURL;
}
asyncCall();
```

--

Ejempo Template + Fetch + JSON

```
document.querySelector('#listar').addEventListener("click", function
(event) {
        event.preventDefault();
        var myRequest = new Request(event.target.href, myInit);
        fetch(myRequest)
                .then(function (response) {
                        if (response.status == 200) return
response.json();
                        else throw new Error('Something went wrong on api
server!');
```

```
                })
                .then(function (response) {
                        console.debug(response);
                        llamarTemplate(response.plantilla,
response.datos);
                        // ...
                })
                .catch(function (error) {
                        console.error(error);
                });
});
```

FORMDATA Element

Los parámetros de un formulario se pueden recoger con el objeto FormData.

```
var formData = new FormData(form)
var formElement = new FormdData
(document.getElementById("myFormElement"));
```

--

Ejemplo:http://piruletas.cloudaccess.host/teoria/T6/ImageSend.html

```
# fetch POST
async function enviaForm(evento) {
   try {
      evento.preventDefault();
      let url = evento.target.getAttribute("action")
      let data = new FormData(evento.target);
      let init = {
         url: url,
         method: 'post',
         body: data
      };
      let request0 = new Request(init);
      const response = await fetch(request0);
      if (!response.ok) {
         throw Error(response.statusText);
      }
      const result = await response.text();
      console.log('Correcto devuelvo:', result);
   } catch (error) {
      console.log(error);
   }

}
```

```
if (document.forms.length > 0) {
   document.forms[0].addEventListener("submit", function (event) {
      enviaForm(event);
   })
}
```

Enlaces de Interés

- http://www.w3.org/TR/XMLHttpRequest/
- https://developer.mozilla.org/es/docs/XMLHttpRequest/Using_XMLHttpRequest
- https://fetch.spec.whatwg.org/#
- https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Utilizando_Fetch
- http://www.etnassoft.com/2016/10/10/estudiando-la-nueva-api-fetch-la-evolucion-natural-de-xhr-en-el-nuevo-javascript/
- https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Funciones/Arrow_functions