Que es DOM?

DOM significa Document Object Model (Modelo de Objetos del Documento, en español) y es una estructura de árbol que representa el contenido de una página web. Básicamente, el DOM es la representación en memoria del código HTML que forma una página web.

Cada elemento HTML se representa como un objeto en el DOM, lo que permite acceder a sus propiedades y modificar su contenido, estilo y comportamiento mediante JavaScript o frameworks como jQuery. Además, el DOM también permite la interacción entre el usuario y la página web, ya que es el medio para manejar eventos como clics de mouse, ingreso de texto, etc.

El DOM es creado automáticamente por el navegador web cuando carga una página web y se puede acceder a él mediante el objeto global document en JavaScript. Con jQuery, se puede seleccionar y manipular los elementos del DOM de forma fácil y rápida, lo que facilita la creación de páginas web interactivas y dinámicas.

Jquery para DOM

Vamos a hablar sobre jQuery y su relación con el DOM. Como saben, el DOM es una estructura de árbol que representa el contenido de una página web, y jQuery es una librería de JavaScript que simplifica el manejo del DOM y proporciona una sintaxis más amigable para seleccionar y manipular los elementos del DOM.

En jQuery, la selección de elementos del DOM se realiza mediante el símbolo $, seguido de un selector que identifica los elementos que se quieren seleccionar. Por ejemplo, si quisieramos seleccionar todos los elementos de la clase "boton", podríamos hacerlo de la siguiente forma:

```
$('.boton')
```

Este selector nos devolvería un objeto jQuery que contiene todos los elementos que tengan la clase "boton". A partir de este objeto, podemos utilizar métodos de jQuery para modificar el contenido, el estilo y el comportamiento de estos elementos. Por ejemplo, si quisieramos cambiar el color de fondo de todos los botones a rojo, podríamos hacerlo de la siguiente forma:

```
$('.boton').css('background-color', 'red');
```

El método css() nos permite modificar el estilo CSS de los elementos seleccionados, en este caso cambiando el valor de la propiedad background-color a "red".

Otro método muy utilizado en jQuery es on(), que nos permite agregar eventos a los elementos seleccionados. Por ejemplo, si quisieramos agregar un evento "click" a todos los botones de la clase "boton", podríamos hacerlo de la siguiente forma:

```
$('.boton').on('click', function() {
    alert('¡Hiciste clic en un botón!');
  });
```

Este código agrega un evento "click" a todos los elementos seleccionados por el selector $('.boton'), y cuando se hace clic en alguno de estos elementos, se muestra una alerta en la pantalla con el mensaje "¡Hiciste clic en un botón!".

Varios ejemplos manipulando DOM

1. Agregar o quitar clases a elementos del DOM: Podemos utilizar el método addClass() o removeClass() para agregar o quitar clases CSS a los elementos del DOM seleccionados por un selector. Por ejemplo, si tenemos un botón con la clase "boton" y queremos agregar la clase "rojo" cuando se hace clic en él, podemos hacer lo siguiente:

```
$('.boton').on('click', function() {
    $(this).addClass('rojo');
  });
```

2. Cambiar el contenido de un elemento del DOM: Podemos utilizar el método html() para cambiar el contenido HTML de un elemento seleccionado. Por ejemplo, si tenemos un párrafo con la clase "parrafo" y queremos cambiar su contenido por "Hola mundo!", podemos hacer lo siguiente:

```
$('.parrafo').html('Hola mundo!');
```

3. Ocultar o mostrar elementos del DOM: Podemos utilizar el método hide() o show() para ocultar o mostrar elementos seleccionados por un selector. Por ejemplo, si tenemos una imagen con la clase "imagen" y queremos ocultarla cuando se hace clic en un botón, podemos hacer lo siguiente:

```
$('.boton').on('click', function() {
    $('.imagen').hide();
  });
```

4. Agregar o quitar elementos del DOM: Podemos utilizar los métodos append() o remove() para agregar o quitar elementos del DOM. Por ejemplo, si tenemos un div con la clase "contenedor" y queremos agregar un botón dentro de él, podemos hacer lo siguiente:

```
$('.contenedor').append('<button>Enviar</button>');
```

Y si queremos quitar un botón dentro del mismo div, podemos hacer lo siguiente:

```
$('.contenedor button').remove();
```

Estos son solo algunos ejemplos de lo que podemos hacer con jQuery para manipular el DOM de una página web. Con práctica y creatividad, podemos crear páginas web interactivas y dinámicas utilizando esta poderosa herramienta.

Que es AJAX

AJAX (Asynchronous JavaScript and XML) es una técnica de programación que permite actualizar partes de una página web sin necesidad de recargarla completa. En otras palabras, con AJAX podemos enviar y recibir datos desde el servidor sin interrumpir la experiencia del usuario.

La técnica se basa en utilizar JavaScript para realizar peticiones asíncronas al servidor, lo que significa que la página web puede seguir siendo utilizada

mientras se espera la respuesta del servidor. Una vez que la respuesta es recibida, se puede actualizar el contenido de la página web sin tener que recargarla completa.

Aunque el nombre AJAX incluye "XML", en realidad se pueden utilizar otros formatos de datos para enviar y recibir información, como JSON.

En resumen, AJAX permite crear páginas web más rápidas y dinámicas, ya que evita la necesidad de recargar la página completa cada vez que se necesita actualizar una parte de ella. Es una técnica muy utilizada en aplicaciones web modernas, como redes sociales, tiendas en línea y herramientas de productividad.

AJAX y jquery

jQuery es una biblioteca de JavaScript que incluye herramientas para facilitar el uso de AJAX. Una de las principales funciones de jQuery es proporcionar una manera más sencilla de enviar y recibir datos utilizando AJAX, mediante el uso de los métodos $.ajax(), $.get(), $.post(), entre otros.

Un ejemplo práctico de cómo utilizar jQuery y AJAX sería el siguiente:

Supongamos que tenemos una página web con un formulario para enviar comentarios a un servidor. Al enviar el formulario, queremos que el comentario se envíe al servidor sin tener que recargar la página completa. Para hacer esto, podemos utilizar AJAX con jQuery.

Primero, debemos agregar un manejador de eventos para el formulario, que se active cuando se envíe el formulario:

```
$('#formulario').submit(function(event) {
    // Aquí se realizará la petición AJAX
    event.preventDefault(); // Evita que se recargue la página al enviar el formulario
  });
```

Dentro de la función del manejador de eventos, podemos utilizar el método $.post() de jQuery para enviar los datos del formulario al servidor. Por ejemplo:

```
$.post('servidor.php', $('#formulario').serialize(), function(respuesta) {
    // Aquí se procesa la respuesta del servidor
  });
```

En este caso, servidor.php es la URL del archivo PHP que procesará la petición AJAX, $('#formulario').serialize() es un método de jQuery que convierte los datos

del formulario en una cadena de consulta que puede ser enviada al servidor, y la función de callback es opcional y se ejecutará cuando se reciba la respuesta del servidor.

Con esta técnica, podemos enviar y recibir datos sin tener que recargar la página completa, lo que proporciona una experiencia de usuario más fluida y rápida.

Consultando una API con AJAX y jquery

Para hacer consultas a una API utilizando AJAX y jQuery, se pueden utilizar los métodos $.ajax() o $.get(), dependiendo de la forma en que la API esté diseñada para recibir las consultas.

Supongamos que queremos hacer una consulta a la API de GitHub para obtener información sobre un usuario específico. En este caso, la API de GitHub está diseñada para recibir consultas HTTP GET, por lo que utilizaremos el método $.get() de jQuery para realizar la consulta.

Primero, debemos crear una función que se ejecute cuando se haga clic en un botón en la página, por ejemplo:

```
$('#boton-consultar').click(function() {
    var usuario = $('#input-usuario').val(); // Obtener el nombre de usuario desde un campo de entrada
    var url = 'https://api.github.com/users/' + usuario; // Construir la URL de la consulta
    $.get(url, function(respuesta) {
      // Aquí se procesa la respuesta de la API
    });
  });
```

En este caso, $('#input-usuario').val() es una forma de obtener el valor del campo de entrada donde se ingresa el nombre de usuario, y var url = 'https://api.github.com/users/' + usuario; construye la URL de la consulta, que en este caso incluye el nombre de usuario ingresado por el usuario.

El método $.get() realiza la consulta a la API utilizando la URL que construimos, y la función de callback se ejecuta cuando se recibe la respuesta de la API. Dentro de la función de callback, podemos procesar la respuesta de la API de la manera que necesitemos, por ejemplo:

```
$.get(url, function(respuesta) {
    $('#nombre-usuario').text(respuesta.name); // Actualizar un elemento en la página con el nombre del usuario
    $('#avatar-usuario').attr('src', respuesta.avatar_url); // Actualizar la imagen del usuario en la página
  });
```

En este caso, utilizamos jQuery para actualizar dos elementos en la página web: uno con el nombre del usuario ($('#nombre-usuario').text(respuesta.name)), y otro con la imagen de perfil ($('#avatar-usuario').attr('src', respuesta.avatar_url)). La respuesta de la API de GitHub incluye los datos que necesitamos para actualizar estos elementos de la página web.

Pintando elementos de una API en mi web

Para mostrar los elementos de una API que se ha consumido en jQuery en una página web, se puede utilizar la función de callback del método $.get() para actualizar el contenido de un elemento HTML en la página con los datos de la respuesta de la API.

Supongamos que hemos hecho una consulta a la API de GitHub utilizando el método $.get(), y queremos mostrar los nombres de los repositorios del usuario en un elemento HTML <ul> en la página web. Podemos hacer lo siguiente:

```
$('#boton-consultar').click(function () {
    var usuario = $('#input-usuario').val();
    var url = 'https://api.github.com/users/' + usuario + '/repos';
    $.get(url, function (respuesta) {
        var repositorios = respuesta.map(function (repo) {
            return '<li>' + repo.name + '</li>';
        }).join('');
        $('#lista-repos').html(repositorios);
    });
});
```

En este ejemplo, la URL de la consulta incluye la ruta /repos al final para obtener los repositorios del usuario especificado. La función de callback recibe la respuesta de la API, y utiliza el método map() de JavaScript para crear un array de elementos HTML <li> que contienen el nombre de cada repositorio. Luego, utiliza el método join() para unir los elementos HTML en una cadena de texto. Por último, utiliza el método html() de jQuery para actualizar el contenido del elemento <ul> con la cadena de texto que contiene la lista de repositorios.

En la página web, se podría tener un elemento HTML como este:

```html
<ul id="lista-repos"></ul>
```

Al hacer clic en el botón de consulta, se realizará la consulta a la API de GitHub, y el contenido del elemento <ul> se actualizará con la lista de repositorios del usuario especificado.

Consumiendo API de clima

A continuación dejaré a mano un ejemplo práctico de cómo consumir los datos de clima de una API y reflejarlo en la web.

```html
<div id="weather"></div>
```

```javascript
$(document).ready(function() {
    var api_key = 'TU_API_KEY'; // Reemplaza TU_API_KEY por tu propia clave de API
    var city = 'Buenos Aires'; // Reemplaza Buenos Aires por la ciudad de la que quieres obtener los datos de clima

    var url = 'https://api.openweathermap.org/data/2.5/weather?q=' + city + '&appid=' + api_key;

    $.ajax({
      url: url,
      method: 'GET',
      success: function(response) {
        var temp_c = response.main.temp - 273.15;
        var temp_f = temp_c * 9/5 + 32;
        var description = response.weather[0].description;
        var city_name = response.name;

        var weather_html = `
          <h2>Clima en ${city_name}</h2>
          <p>Temperatura: ${temp_c.toFixed(2)} °C / ${temp_f.toFixed(2)} °F</p>
          <p>Descripción: ${description}</p>
        `;

        $('#weather').html(weather_html);
      },
      error: function(err) {
        console.log(err);
        $('#weather').html('<p>No se pudo obtener los datos de clima.</p>');
      }
    });
  });
```

En este ejemplo, se utiliza la API de OpenWeatherMap para obtener los datos de clima de una ciudad específica. La clave de API se define en la variable api_key, y la ciudad se define en la variable city. Se construye la URL de la API utilizando estas variables, y luego se utiliza el método $.ajax() para realizar la solicitud GET a la API.

La función de éxito de la solicitud se ejecuta cuando se reciben los datos de la API, y se utiliza la respuesta para crear una cadena de HTML que contiene los datos de clima de la ciudad. La cadena de HTML se inserta en el elemento con el id weather utilizando el método html() de jQuery.

En caso de que ocurra un error en la solicitud, se muestra un mensaje de error en el elemento #weather.

Recuerda que para que esto funcione correctamente, debes reemplazar TU_API_KEY con tu propia clave de API de OpenWeatherMap, y elegir la ciudad de la que deseas obtener los datos de clima.