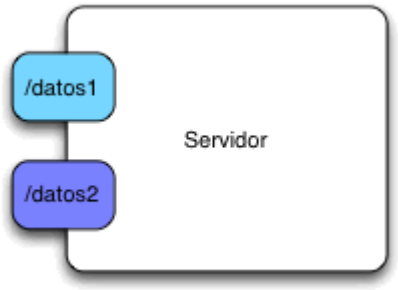
jQuery Promise es uno de los conceptos que cuesta más entender cuanto de trabajamos con Ajax. Sin embargo sino entendemos este concepto será muy difícil desarrollar aplicaciones Ajax que sean flexibles y reutilizables. Vamos a mostrar un ejemplo que nos ayude a entender mejor las cosas . Supongamos que disponemos de un servidor que nos devuelve datos a través de dos urls /datos1 y /datos2.



Si queremos acceder a los datos simplemente realizaremos dos peticiones peticiones Ajax .

```
<!DOCTYPE html>
<html>
<head>

 <meta charset="utf-8" />
 <script type="text/javascript" src="js/jquery-1.11.3.min.js">
 </script>
 <script type="text/javascript">

 $(document).ready(function() {


 $("#boton").click(function() {
```

```
 var peticion1=$.get("http://localhost:8080/datos1",function(datos1) {

 console.log(datos1)
 });

 var peticion2=$.get("http://localhost:8080/datos2",function(datos2) {

 console.log(datos2)
 });
 });

 });
 </script>
 <title>Promesas</title>
</head>

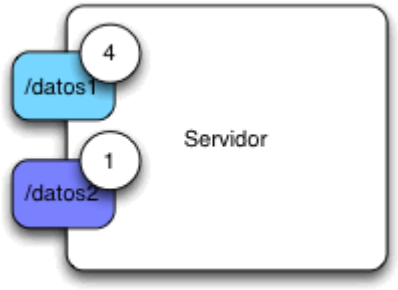
<body>
 <input type="button" id="boton" value="pedirAjax"/>
</body>

</html>
```

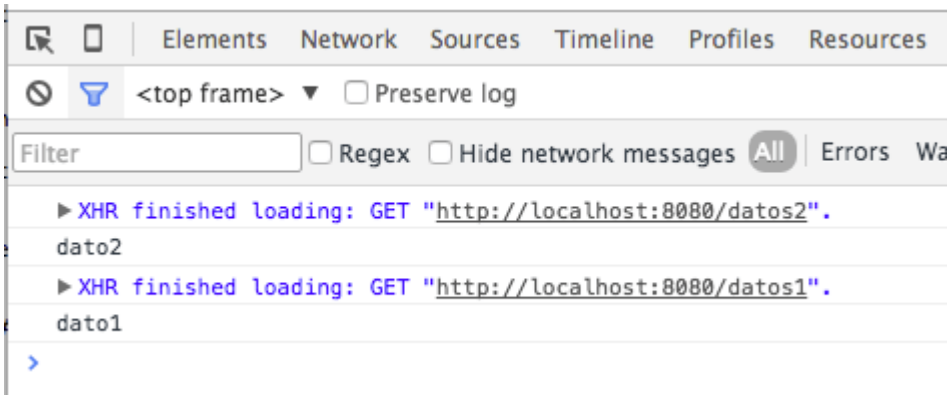
El resultado será que en la consola se imprimirán los datos :



Ahora bien que pasaría si el tiempo de espera para la petición de dato1 fuera mucho más elevado que para dato2.



El resultado sería justamente el contrario:



El problema es que queremos imprimir primero dato1 que dato2 y tenemos dos peticiones asíncronas independientes. La primera solución que se suele usar es anidar las peticiones Ajax:

```
var peticion1=$.get("http://localhost:8080/datos1",function(datos1) {

console.log(datos1);

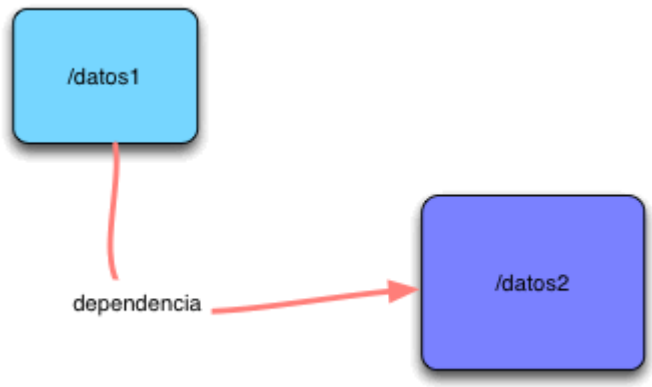
var peticion2=$.get("http://localhost:8080/datos2",function(datos2) {
```

```
console.log(datos2);
});
});
});
```

Los datos saldrán de forma correcta:

```
▶ XHR finished loading: GET "http://localhost:8080/datos1".
dato1
▶ XHR finished loading: GET "http://localhost:8080/datos2".
dato2
>
```

El problema es que hemos generado una dependencia entre la petición de /datos1 y la petición de /datos2 , algo que no debería existir y que elimina la posibilidad de reutilizar la petición de /datos1 ya que siempre se ejecutara /datos2.



Utilizando jQuery Promise

El concepto de jQuery Promise viene a solventar este problema. Una promesa es un objeto que se ejecutará en un futuro. Todas las peticiones Ajax son Promesas. El concepto de promesa pertenece al mundo de la programación asíncrona y no es exclusivo de jQuery.

Para acceder al concepto de promesa con el código de jQuery deberemos realizar la siguiente modificación:

```
$("#boton").click(function() {

var promesa1=$.get("http://localhost:8080/datos1");
var promesa2=$.get("http://localhost:8080/datos2");

console.log(promesa1);
console.log(promesa2);

});
```

Ya disponemos de dos promesas , que como muestra la consola son Objetos de Javascript que incorporan una funcionalidad compleja.



Sin embargo las operaciones básicas no son complejas de abordar. En este caso podemos usar la función $.when de jQuery que se encarga de controlar la ejecución de varias promesas y cuando todas hayan finalizado imprime la información requerida en el orden solicitado.

```
$("#boton").click(function() {

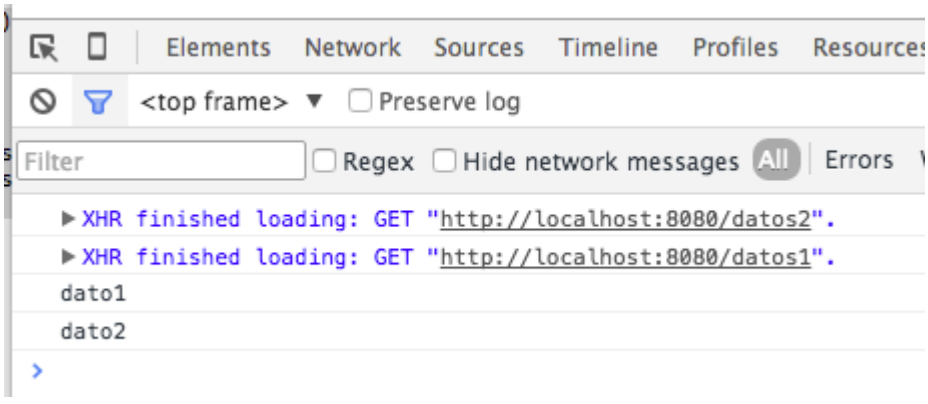
var promesa1=$.get("http://localhost:8080/datos1");
var promesa2=$.get("http://localhost:8080/datos2");

$.when (promesa1, promesa2).done(function (dato1,dato2) {

console.log(dato1[0]);
console.log(dato2[0]);
})

});
```

El resultado saldrá en el orden solicitado:



Hemos conseguido desacoplar ambas peticiones Ajax e imprimir los resultados en el orden deseado. Las promesas son imprescindibles a la hora de desarrollar en el entorno de Javascript

Otros artículos relacionados : jQuery Ajax , jQuery On/Off , jQuery Promises API