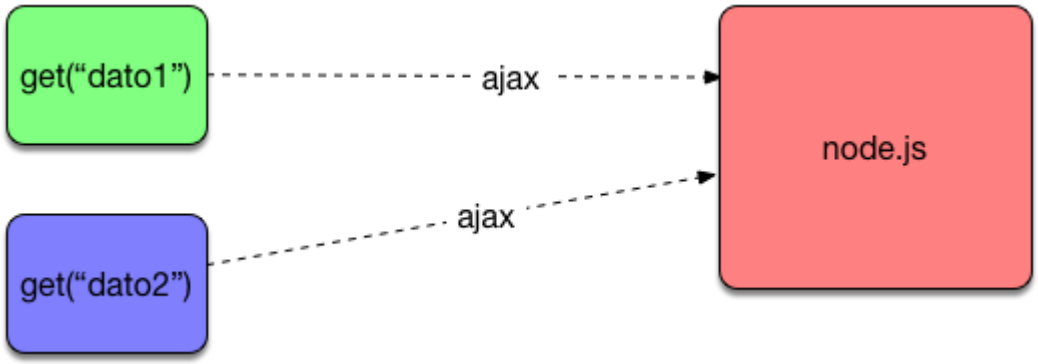
El concepto de JavaScript Deferred Objects siempre es difícil de entender en un primer momento. Sin embargo es algo que tenemos que conocer ya que sino nuestro código será mucho menos reutilizable. Vamos a explicar este concepto tan interesante a través de un grupo de ejemplos.

Peticiones Ajax

Supongamos que tenemos dos urls en nuestro servidor sobre las cuales realizamos peticiones Ajax .



En este caso el código lo voy a construir utilizando el framework express de javascript.

```
var express = require('express');
var app = express();


app.use(express.static(__dirname+'/publica'));
app.get('/dato1', function (req, res) {
setTimeout(function() {
```

```
res.send('hola1');
},2000);
});
app.get('/dato2', function (req, res) {
setTimeout(function() {
res.send('hola2');
},2000);
});

app.listen(3000, function () {
console.log('Example app listening on port 3000!');
});
```

La peculiaridad que tienen estas urls “dato1” y “dato2” es que tardan 2 segundos en devolvernos la información que necesitamos. En principio esto no es problemático ya que podemos realizar dos peticiones ajax con jQuery y obtener el texto “hola1” y “hola2”.

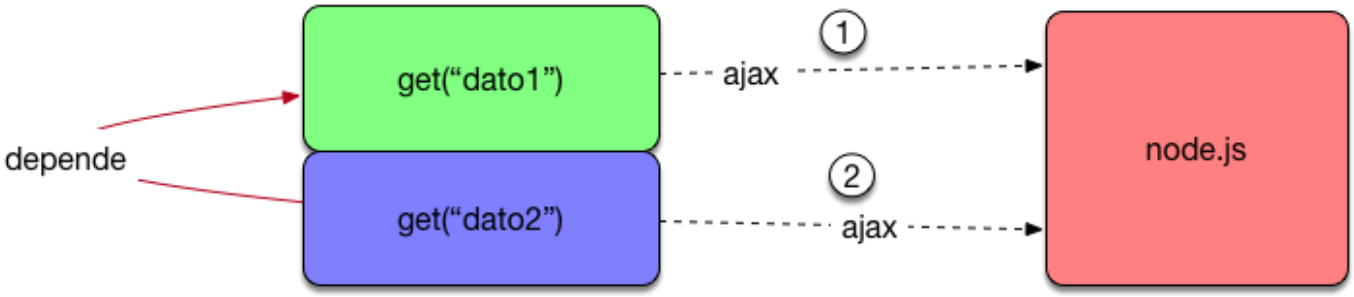
```
$(document).ready(function() {

$.get("dato1",function(datos) {

console.log(datos);

});
$.get("dato2",function(datos) {

console.log(datos);

});
```

```
});
```

Los problemas comienzan cuando queremos por ejemplo crear un array con los datos de ambas peticiones e imprimir el resultado por la consola.

```
  $(document).ready(function() {

    $.get("dato1",function(datos1) {


      $.get("dato2",function(datos2) {

          var lista=[];
          lista.push(datos1);
          lista.push(datos2);
          console.log(lista);
    });

});

  });
```

Este código ya es más problemático porque obliga a acoplar las dos peticiones que eran independientes en un principio y ademas tarda un total de 4segundos ya que hasta que no termina la primera petición no comienza la segunda.

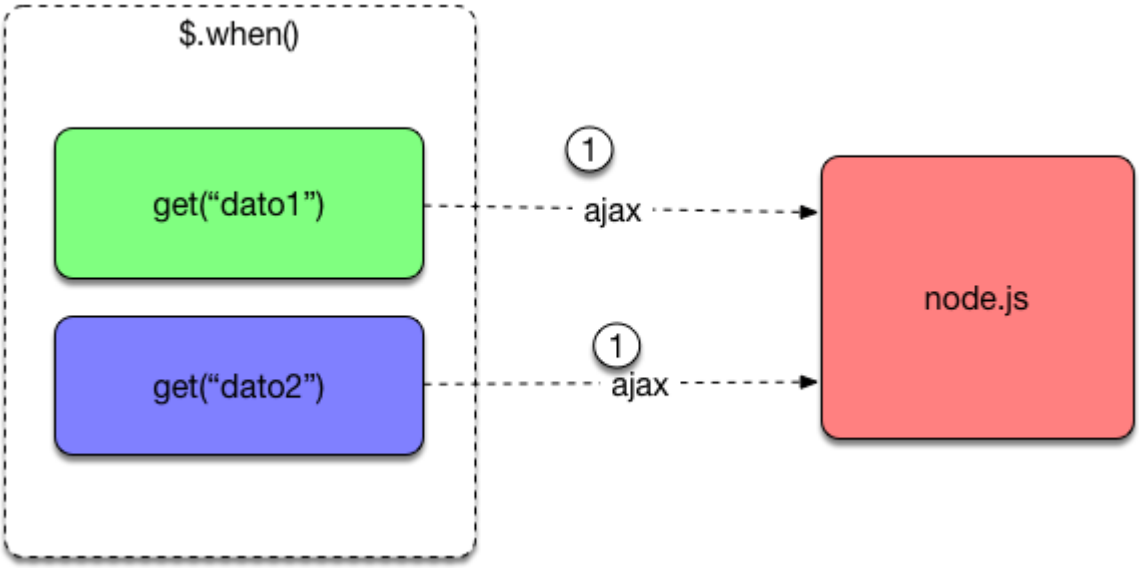


JavaScript Promises

Normalmente este problema se solventa utilizando promesas y el nuevo código podría ser similar a :

```
$(document).ready(function() {

$.when($.get("dato1"),$.get("dato2")).then(function(datos1,datos2){

      var lista=[];
      lista.push(datos1[0]);
      lista.push(datos2[0]);
      console.log(lista)
    });
```

Esto nos permite realizar las peticiones de forma simultanea y no acoplar una a otra .



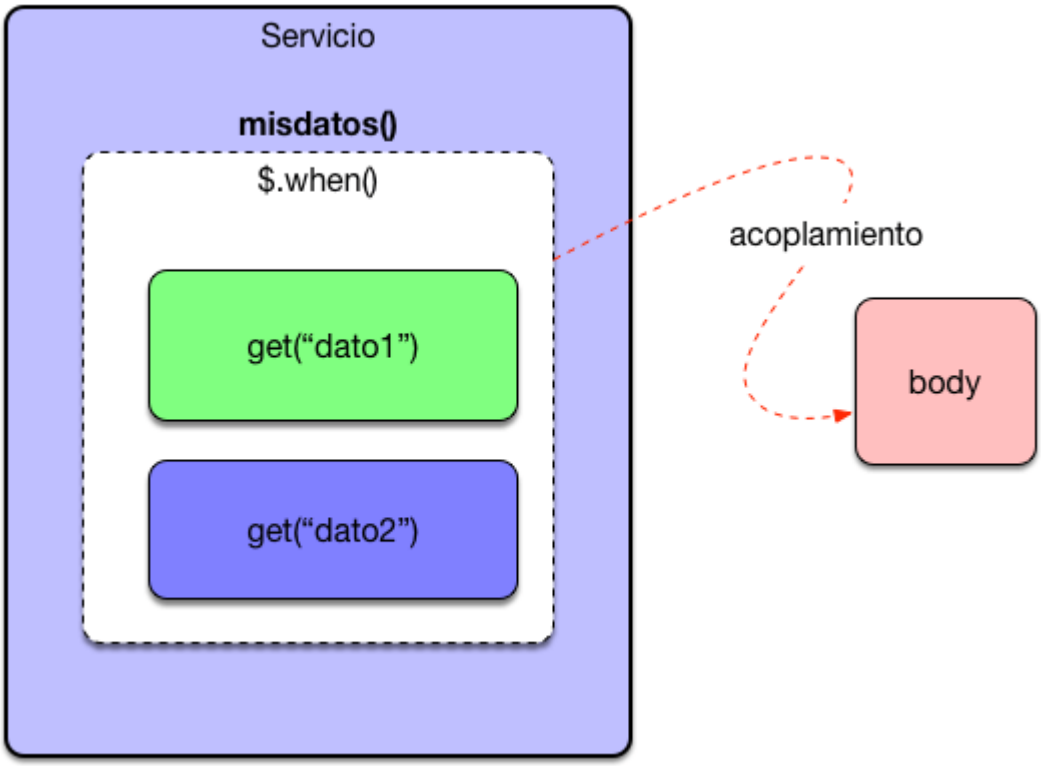
El resultado lo imprimimos en la consola:

Muchas veces da la sensación de que este es un código bastante correcto ya que hemos hecho uso de promesas . Sin embargo tiene sus limitaciones. Imaginemos que vamos a usarle en una clase de servicio para imprimir la información que nos llega vía ajax en nuestra página. El código que tendríamos que construir sería algo así:

```
function Servicio() {

    this.misdatos=function() {
```

```
$.when($.get("dato1"),$.get("dato2")).then(function(datos1,datos2){

          var lista=[];
          lista.push(datos1[0]);
          lista.push(datos2[0]);
          lista.forEach(function(item) {

            $("body").append(item);
          })
        });

      }
    }


  $(document).ready(function() {

      var objeto= new Servicio();
      objeto.misdatos();

  });
```

Esto imprimirá los datos en la página html. El problema es que acoplaremos el servicio a una parte muy concreta de la capa de presentación ,en este caso el body.



El problema es que podríamos haberlo enlazado también a una tabla o a una lista lo cual eliminaría cualquier posibilidad de reutilizar el código. ¿Cómo podemos solventar este problema de programación asíncrona?

JavaScript Deferred Objects , la solución

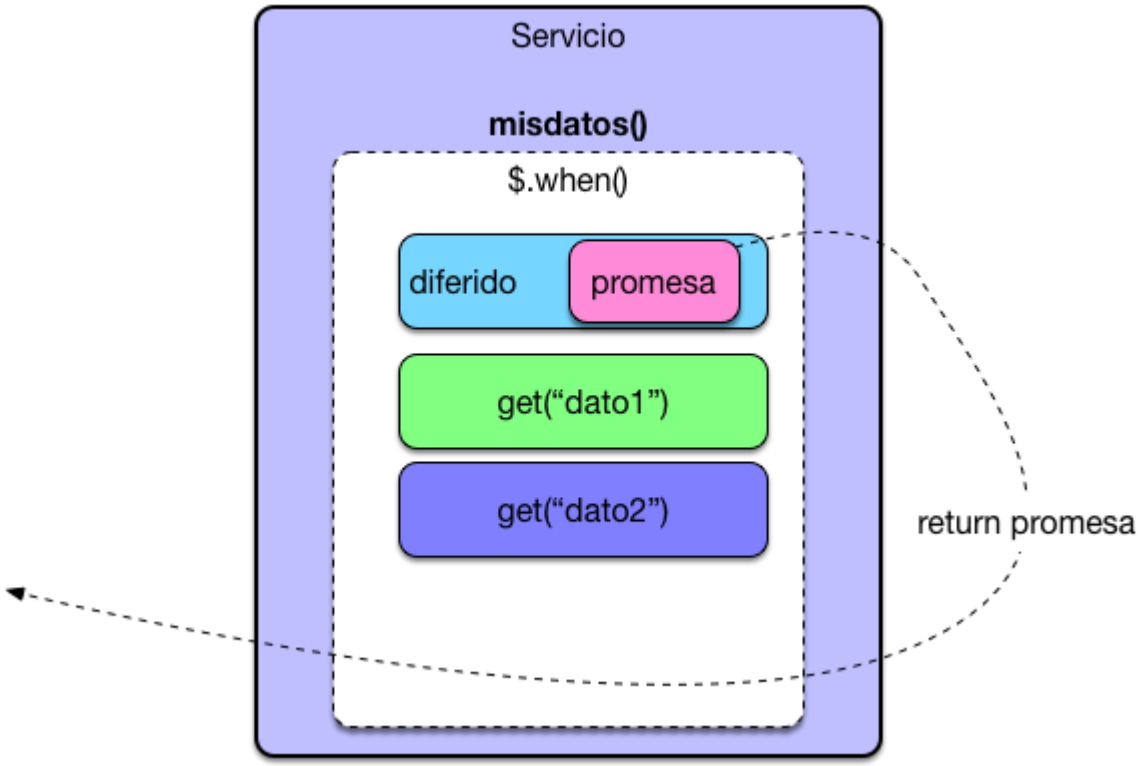
Para solventar este problema tenemos que construir un objeto diferido , en nuestro caso vamos a utilizar jQuery pero se trata de un concepto general que todos los frameworks modernos soportan. Vamos a ver el código:

```
function Servicio() {
```

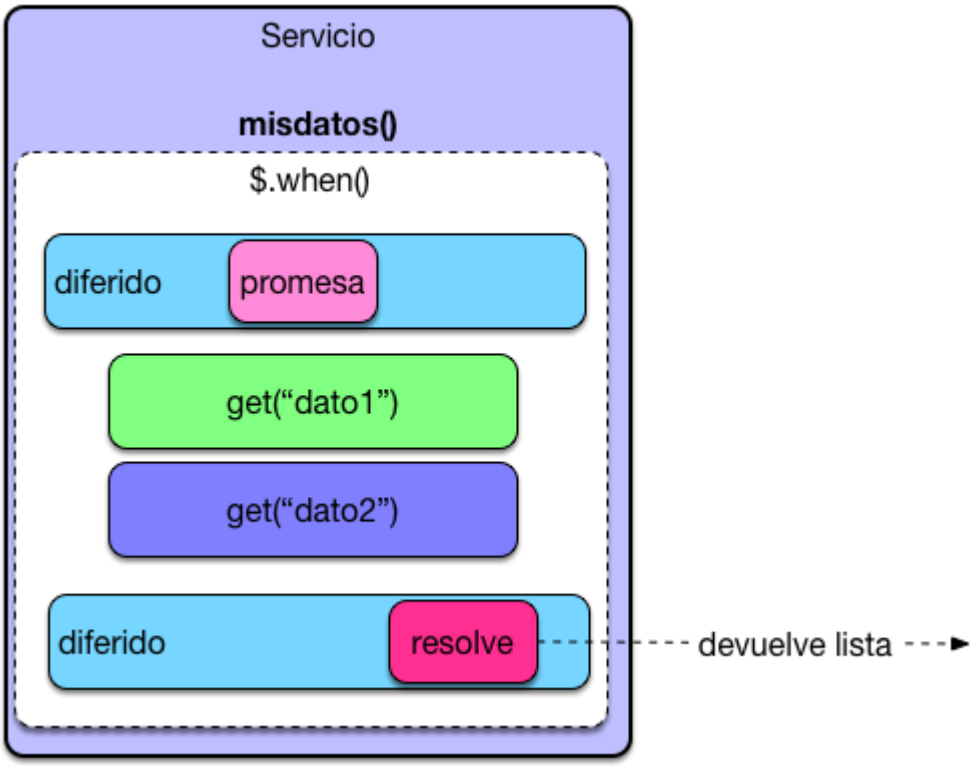
```
      this.misdatos=function() {

        var diferido= $.Deferred();
$.when($.get("dato1"),$.get("dato2")).then(function(datos1,datos2){

          var lista=[];
          lista.push(datos1[0]);
          lista.push(datos2[0]);

          diferido.resolve(lista);


        });

        return diferido.promise();

      }
    }


  $(document).ready(function() {

      var objeto= new Servicio();
      objeto.misdatos().then(function(datos) {

        $("body").append(datos);

      });
```

Este código en un principio cuesta entenderle. Lo primero que hacemos es crear un objeto

diferido dentro de nuestro servicio. Este objeto contiene en su interior una promesa que es lo que nosotros retornamos de forma inmediata mientras esperamos que las dos peticiones ajax se ejecuten .



Al terminar ambas peticiones invocamos al método resolve del objeto diferido pasándole la lista de elementos que acabamos de construir . Al resolver el objeto se ejecuta la promesa y con el método then accedemos desde la capa de presentación a los datos.



De esta forma conseguimos desacoplar la capa de servicio de la capa de presentación en una situación de programación asíncrona. Acabamos de usar JavaScript Deferred Objects para solventar un problema complejo.

Otros artículos relacionados:

1. jQuery Promise y AJAX
2. jQuery $.ajax ,$.get, $.post
3. RxJS y la programación reactiva.
4. El objeto diferido en jQuery