JavaScript 2015 (y X). Promesas.

Javascript 2015 cuenta con una nueva clase llamada **Promise**]. Es un bonito nombre, que suena interesante, y que, como vamos a ver en seguida, aporta una funcionalidad interesante.
Con esto terminamos esta serie sobre las novedades de JavaScript en su versión de 2015. Como hemos visto, salvo algunos deshonrosos ejemplos, la mayoría son útiles y nos aportan facilidades que se echaban de menos. Esperemos que, en sucesivas revisiones, subsanen las carencias o defectos que nosotros, los miembros de la comunidad de desarrolladores, aún hemos encontrado.
Al menos, estas promesas sí se cumplen. No como las de algunos jefes, o las de aquella vecina que nos usaba de pagafantas en el instituto. :lol:

QUÉ SON LAS PROMESAS
Los objertos de la clase Promise son una manera de encapsular procesos en segundo plano (es decir, en el caso más típico, consumo de datos por Ajax), de manera que se pueda emular una llamada síncrona utilizando una llamada asíncrona.
Vamos por partes. Empecemos hablando del problema. Imagina que tienes una llamada Ajax por post que recupera un dataset, digamos de un PHP externo que, a su vez, obtiene los resultados de una base de datos. Un esquema típico sería el siguiente:

```
var listaDePersonas = "";
$.post({
   url: "listar_personas.php",
   data: {'anualidad':'2015'},
   function (resultado){
      listaDePersonas = resultado;
   }
});
```

Si no estás habituado al uso de $.post(), pensarás que, al final de este fragmento de código, tienes una lista de personas que constituyen un dataset en la variable listaDePersonas. Sin embargo, cuando compruebes el contenido de la variable, encontrarás que está vacía, y no puedes usarla porque no tiene el dataset que esperabas.
Si eres avezado en estas situaciones, ya sabes por qué la variable está vacía. Si no lo eres, déjame que te haga una reflexión. La función de jQuery $.post() es una llamada ajax asíncrona. Por lo tanto, para cuando compruebas el valor de listaDePersonas, el proceso en segundo plano no ha concluido, sino que aún se está ejecutando, así que la variable está vacía.
La solución tradicional viene siendo emplear el método $.ajax() en lugar de $.post(), estableciendo el valor de async en false, lo que hace que la llamada actúe de forma síncrona. Por lo tanto, si pides una comprobación de la variable, esta no se llevará a cabo hasta que la llamada Ajax haya concluido, con lo que ya tienes los resultados y todo va bien. El problema es que establecer async en false es una técnica que los fabricantes de jQuery han marcado como deprecada y, aunque aún estará en funcionamiento bastante tiempo, porque hay mucho código escrito que la emplea, antes o después desaparecerá y dejará de funcionar. Necesitamos otra solución.
El objeto de la clase Promise nos la ofrece. Como te he comentado, nos permite encapsular $.post() de tal modo que, aunque sea un proceso asíncrono, cuando la ejecución del script continúa nos da el resultado correcto, cómo si el proceso hubiera sido síncrono, pero sin infringir una deprecación del fabricante de jQuery.
Vamos a ver cómo actúa. Observa el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
 <meta charset="UTF-8">
 <title>JS 6</title>
</head>
<body>
 <div id="capaDeResultados"></div>
 <script language="javascript" src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
 <script language="javascript">
  /* Se crea un objeto de la clase Promise. Recibe, cómo argumento, una función callback,
  es decir, que se ejecutará cuando el objeto tenga un resultado. Este resultado puede ser
  uno de dos: o el código interno ha funcionado, y todo ha salido bien, o el código ha
  fallado. La función callback recibe, a su vez, dos parámetros. El primero corresponde al
  caso de que todo ha salido bien. El segundo corresponde al caso de que se haya producido
  un fallo. Estos parámetros se conocen, genéricamente, como resolve (el primero) y como
  reject (el segundo). Normalmente, se suelen denominar con esos nombres. Auí hemos empleado
  los nombres correcto y fallo, para que veas, si miras otros ejemplos en Internet, que sólo
  son nombres, y que puedes ponerlos como consideres adecuado. */
  var pruebaPromesa = new Promise(function respuesta(correcto, fallo){
   /* Dentro del cuerpo del objeto Promise tenemos definido un proceso que se ejecuta en
   segundo plano, por Ajax. Este recupera de una tabla de personas, aquellas que nacieron en 2015.
   Da igual, sólo es un ejemplo.  */
   var listaDePersonas = $.post({
    url: "listar_personas.php",
    data: {'anualidad':'2015'},
    function (resultado){
     return resultado;
    }
   });

   /* Aquí comprobamos si la variable tiene el dataset esperado o está aún vacía.
   Según tenga o no contenido invocamos a una de dos funciones, con los mismos
   nombres que se usaron en la creación del objeto de la clase Promise, en la línea 20.
   Observa que para el caso de resolve, pasamos como argumento la variable con la que
   estamos trabajando. Para el caso de reject, pasamos como argumento una función llamada
   Error, que es propia de Promise, y que recibe un texto de error.*/
   if (listaDePersonas > ""){
    correcto(listaDePersonas);
   } else {
    fallo(Error("No se ha cargado"));
   }
  });

  /* Hasta ahora se ha creado el objeto Promise, pero aún no hemos hecho nada con él.
  Realmente, lo que dispara su funcionamiento es el método .then().
  Este método recibe, como argumentos, dos funciones.
  La primera, se ejecutará si el objeto Promise ha invocado a la función resolve (correcto, en nuestro ejemplo).
  La segunda se ejecuta si el proceso descrito en Promise falló y el objeto ha invocado a la función reject
  (fallo, en nuestro ejemplo). */
  pruebaPromesa.then(function(resultadoDePromesa){
   console.log("Lista de personas:" + resultadoDePromesa);
```

```
  }, function (error){
   console.log(error);
  });
 </script>
</body>
</html>
```

Este código está ampliamente comentado en cada paso importante, para que puedas cotejarlo con su funcionamiento, y con lo que hemos comentado más arriba. Recuerda que tienes que tener abierta la consola JavaScript de tu navegador, para ver los resultados. Cómo este código se complementa con un PHP que obtiene el dataset y lo devuelve en JSON, y con una base de datos, te lo he dejado todo empaquetado en este enlace, para que puedas probarlo y ver como funciona.