jQuery y AJAX

Introducción

AJAX significa Asynchronous JavaScript and XML. Esta tecnología nos permite comunicarnos con un servicio web sin tener que recargar la página. A pesar de que su nombre lo dice, XML no es requerido para usar AJAX, de hecho, se utiliza JSON más seguido.

jQuery es un framework Javascript, un framework es un producto que sirve como base para la programación avanzada de aplicaciones, que aporta una serie de funciones o códigos para realizar tareas habituales.

Por decirlo de otra manera, framework son unas librerías de código que contienen procesos o rutinas ya listos para usar. Los programadores utilizan los frameworks para no tener que desarrollar ellos mismos las tareas más básicas, puesto que en el propio framework ya hay implementaciones que están probadas, funcionan y no se necesitan volver a programar.

Con jQuery se obtiene ayuda en la creación de interfaces de usuario, efectos dinámicos, aplicaciones que hacen uso de Ajax, entre otras ventajas.

Para hacer uso de jQuery acceder a la página de jQuery (www.jquery.com) y descargar la última versión del framework.

En el mismo directorio donde has colocado el archivo js (por ejemplo: jquery-1.3.2.min.js), coloca un archivo html con el siguiente código:

```
<html>
   <head>
      <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
      <script>

       </script>
   </head>
<body>
<a href="www.google.com.ar">Busqueda Google</a>
</body>
</html>
```

Esto provocara que el servidor web interprete que se utilizara la sintaxis jQuery dentro de funciones javascript.

Conceptos clave Javascript

jQuery se encuentra escrito en JavaScript, un lenguaje de programación muy rico y expresivo.

Operadores

Operadores básicos

Los operadores básicos permiten manipular valores.

Concatenación

```
var foo = 'hello';

var bar = 'world';

console.log(foo + ' ' + bar); // la consola de depuración muestra 'hello world'
```

Multiplicación y división

```
2 * 3; 2 / 3;
```

Incrementación y decrementación

```
var i = 1;

var j = ++i;  // incrementación previa:  j es igual a 2; i es igual a 2

var k = i++;  // incrementación posterior: k es igual a 2; i es igual a 3
```

Operaciones con números y cadenas de caracteres

En JavaScript, las operaciones con números y cadenas de caracteres (en inglés *strings*) pueden ocasionar resultados no esperados.

Suma vs. concatenación

```
var foo = 1;

var bar = '2';

console.log(foo + bar);  // error: La consola de depuración muestra 12
```

Forzar a una cadena de caracteres actuar como un número

```
var foo = 1;

var bar = '2';

// el constructor 'Number' obliga a la cadena comportarse como un número

console.log(foo + Number(bar));  // la consola de depuración muestra 3
```

El constructor `Number`, cuando es llamado como una función (como se muestra en el ejemplo) obliga a su argumento a comportarse como un número. También es posible utilizar el operador de *suma unaria*, entregando el mismo resultado:

Forzar a una cadena de caracteres actuar como un número (utilizando el operador de suma unaria)

```
console.log(foo + +bar);
```

Operadores lógicos

Los operadores lógicos permiten evaluar una serie de operandos utilizando operaciones AND y OR.

Operadores lógicos AND y OR

```
var foo = 1;

var bar = 0;

var baz = 2;


foo || bar;  // devuelve 1, el cual es verdadero (true)

bar || foo;  // devuelve 1, el cual es verdadero (true)


foo && bar;  // devuelve 0, el cual es falso (false)

foo && baz;  // devuelve 2, el cual es verdadero (true)

baz && foo;  // devuelve 1, el cual es verdadero (true)
```

El operador `||` (OR lógico) devuelve el valor del primer operando, si éste es verdadero; caso contrario devuelve el segundo operando. Si ambos operandos son falsos devuelve falso (`false`). El operador `&&`(AND lógico) devuelve el valor del primer operando si éste es falso; caso contrario devuelve el segundo operando. Cuando ambos valores son verdaderos devuelve verdadero (`true`), sino devuelve falso.

Puede consultar la sección *"Elementos Verdaderos y Falsos"* para más detalles sobre que valores se evalúan como `true` y cuales se evalúan como `false`.

NOTA

Puede que a veces note que algunos desarrolladores utilizan esta lógica en flujos de control en lugar de utilizar la declaración `if`. Por ejemplo:

```
// realizar algo con foo si foo es verdadero

foo && doSomething(foo);


// establecer bar igual a baz si baz es verdadero;

// caso contrario, establecer a bar igual al

// valor de createBar()

var bar = baz || createBar();
```

Este estilo de declaración es muy elegante y conciso; pero puede ser difícil para leer. Por eso se explícita, para reconocerlo cuando este leyendo código. Sin embargo su utilización no es recomendable a menos que esté cómodo con el concepto y su comportamiento.

Operadores de comparación

Los operadores de comparación permiten comprobar si determinados valores son equivalentes o idénticos.

Operadores de Comparación

```
var foo = 1;

var bar = 0;

var baz = '1';

var bim = 2;


foo == bar; // devuelve falso (false)

foo != bar; // devuelve verdadero (true)

foo == baz; // devuelve verdadero (true); tenga cuidado


foo === baz; // devuelve falso (false)

foo !== baz; // devuelve verdadero (true)

foo === parseInt(baz); // devuelve verdadero (true)


foo > bim;   // devuelve falso (false)

bim > baz;   // devuelve verdadero (true)

foo <= baz;  // devuelve verdadero (true)
```

Sintaxis básica

Comprensión de declaraciones, nombres de variables, espacios en blanco, y otras sintaxis básicas de JavaScript.

Declaración simple de variable

```
var foo = 'hello world';
```

Los espacios en blanco no tienen valor fuera de las comillas

```
var foo =         'hello world';
```

Los paréntesis indican prioridad

```
2 * 3 + 5;    // es igual a 11; la multiplicación ocurre primero

2 * (3 + 5);  // es igual a 16; por lo paréntesis, la suma ocurre primero
```

La tabulación mejora la lectura del código, pero no posee ningún significado especial

```
var foo = function() {

    console.log('hello');

};
```

Código condicional

A veces se desea ejecutar un bloque de código bajo ciertas condiciones. Las estructuras de control de flujo — a través de la utilización de las declaraciones `if` y `else` permiten hacerlo.

Control del flujo

```
var foo = true;

var bar = false;

if (bar) {

    // este código nunca se ejecutará

    console.log('hello!');

}

if (bar) {

    // este código no se ejecutará

} else {

    if (foo) {

        // este código se ejecutará

    } else {

        // este código se ejecutará si foo y bar son falsos (false)

    }

}
```

NOTA

En una línea singular, cuando se escribe una declaración `if`, las llaves no son estrictamente necesarias; sin embargo es recomendable su utilización, ya que hace que el código sea mucho más legible.

Debe tener en cuenta de no definir funciones con el mismo nombre múltiples veces dentro de declaraciones `if`/`else`, ya que puede obtener resultados no esperados.

Elementos verdaderos y falsos

Para controlar el flujo adecuadamente, es importante entender qué tipos de valores son "verdaderos" y cuales "falsos". A veces, algunos valores pueden parecer una cosa pero al final terminan siendo otra.

Valores que devuelven verdadero (true)

```
'0';          // una cadena de texto cuyo valor sea 0

'any string'; // cualquier cadena

[];           // un array vacío

{};           // un objeto vacío

1;            // cualquier número distinto a cero
```

Valores que devuelven falso (false)

```
0;

'';        // una cadena vacía

NaN;       // la variable JavaScript "not-a-number" (No es un número)

null;      // un valor nulo

undefined; // tenga cuidado -- indefinido (undefined) puede ser redefinido
```

Variables condicionales utilizando el operador ternario

A veces se desea establecer el valor de una variable dependiendo de cierta condición. Para hacerlo se puede utilizar una declaración `if`/`else`, sin embargo en muchos casos es más conveniente utilizar el operador ternario. [Definición: El *operador ternario* evalúa una condición; si la condición es verdadera, devuelve cierto valor, caso contrario devuelve un valor diferente.]

El operador ternario

```
// establecer a foo igual a 1 si bar es verdadero;

// caso contrario, establecer a foo igual a 0

var foo = bar ? 1 : 0;
```

El operador ternario puede ser utilizado sin devolver un valor a la variable, sin embargo este uso generalmente es desaprobado.

Declaración `switch`

En lugar de utilizar una serie de declaraciones `if`/`else`/`else if`/`else`, a veces puede ser útil la utilización de la declaración `switch`. [Definición: La declaración `Switch` evalúa el valor de una variable o expresión, y ejecuta diferentes bloques de código dependiendo de ese valor.]

Una declaración switch

```
switch (foo) {

    case 'bar':

        alert('el valor es bar');

    break;

    case 'baz':

        alert('el valor es baz');

    break;

    default:

        alert('de forma predeterminada se ejecutará este código');

    break;

}
```

Las declaraciones `switch` son poco utilizadas en JavaScript, debido a que el mismo comportamiento es posible obtenerlo creando un objeto, el cual posee más potencial ya que es posible reutilizarlo, usarlo para realizar pruebas, etc. Por ejemplo:

```
var stuffToDo = {

    'bar' : function() {

        alert('el valor es bar');

    },



    'baz' : function() {

        alert('el valor es baz');

    },



    'default' : function() {
```

```
        alert('de forma predeterminada se ejecutará este código');
    }
};


if (stuffToDo[foo]) {
    stuffToDo[foo]();
} else {
    stuffToDo['default']();
}
```

Bucles

Los bucles (en inglés *loops*) permiten ejecutar un bloque de código un determinado número de veces.

```
// muestra en la consola 'intento 0', 'intento 1', ..., 'intento 4'
for (var i=0; i<5; i++) {
    console.log('intento ' + i);
}
```

*Note que en el ejemplo se utiliza la palabra `var` antes de la variable `i`, esto hace que dicha variable quede dentro del "alcance" (en inglés *scope*) del bucle. Más adelante en este capítulo se examinará en profundidad el concepto de alcance.*

Bucles utilizando for

Un bucle utilizando `for` se compone de cuatro estados y posee la siguiente estructura:

```
for ([expresiónInicial]; [condición]; [incrementoDeLaExpresión])
    [cuerpo]
```

El estado *expresiónInicial* es ejecutado una sola vez, antes que el bucle comience. Éste otorga la oportunidad de preparar o declarar variables.

El estado *condición* es ejecutado antes de cada repetición, y retorna un valor que decide si el bucle debe continuar ejecutándose o no. Si el estado condicional evalúa un valor falso el bucle se detiene.

El estado *incrementoDeLaExpresión* es ejecutado al final de cada repetición y otorga la oportunidad de cambiar el estado de importantes variables. Por lo general, este estado implica la incrementación o decrementación de un contador.

El *cuerpo* es el código a ejecutar en cada repetición del bucle.

Un típico bucle utilizando for

```
for (var i = 0, limit = 100; i < limit; i++) {

    // Este bloque de código será ejecutado 100 veces

    console.log('Currently at ' + i);

    // Nota: el último registro que se mostrará

    // en la consola será "Actualmente en 99"

}
```

Bucles utilizando `while`

Un bucle utilizando `while` es similar a una declaración condicional `if`, excepto que el cuerpo va a continuar ejecutándose hasta que la condición a evaluar sea falsa.

```
while ([condición]) [cuerpo]
```

Un típico bucle utilizando while

```
var i = 0;

while (i < 100) {

    // Este bloque de código se ejecutará 100 veces

    console.log('Actualmente en ' + i);

    i++; // incrementa la variable i

}
```

Puede notar que en el ejemplo se incrementa el contador dentro del cuerpo del bucle, pero también es posible combinar la condición y la incrementación, como se muestra a continuación:

Bucle utilizando while con la combinación de la condición y la incrementación

```
var i = -1;

while (++i < 100) {

    // Este bloque de código se ejecutará 100 veces

    console.log('Actualmente en ' + i);

}
```

Se comienza en `-1` y luego se utiliza la incrementación previa (`++i`).

Bucles utilizando `do-while`

Este bucle es exactamente igual que el bucle utilizando `while` excepto que el cuerpo es ejecutado al menos una vez antes que la condición sea evaluada.

```
do [cuerpo] while ([condición])
```

Un bucle utilizando do-while

```
do {

    // Incluso cuando la condición sea falsa

    // el cuerpo del bucle se ejecutará al menos una vez.

    alert('Hello');

} while (false);
```

Este tipo de bucles son bastantes atípicos ya que en pocas ocasiones de necesita un bucle que se ejecute al menos una vez. De cualquier forma debe estar al tanto de ellos.

Break y Continue

Usualmente, el fin de la ejecución de un bucle resultará cuando la condición no siga evaluando un valor verdadero, sin embargo también es posible parar un bucle utilizando la declaración `break` dentro del cuerpo.

Detener un bucle con break

```
for (var i = 0; i < 10; i++) {

    if (something) {

        break;

    }

}
```

También puede suceder que quiera continuar con el bucle sin tener que ejecutar más sentencias del cuerpo del mismo bucle. Esto puede realizarse utilizando la declaración `continue`.

Saltar a la siguiente iteración de un bucle

```
for (var i = 0; i < 10; i++) {

    if (something) {

        continue;

    }



    // La siguiente declaración será ejecutada
```

```
    // si la condición 'something' no se cumple

    console.log('Hello');

}
```

Palabras reservadas

JavaScript posee un número de *palabras reservadas*, o palabras que son especiales dentro del mismo lenguaje. Debe utilizar estas palabras cuando las necesite para su uso específico.

- `abstract`
- `boolean`
- `break`
- `byte`
- `case`
- `catch`
- `char`
- `class`
- `const`
- `continue`
- `debugger`
- `default`
- `delete`
- `do`
- `double`
- `else`
- `enum`
- `export`
- `extends`
- `final`
- `finally`
- `float`

- `for`
- `function`
- `goto`
- `if`
- `implements`
- `import`
- `in`
- `instanceof`
- `int`
- `interface`
- `long`
- `native`
- `new`
- `package`
- `private`
- `protected`
- `public`
- `return`
- `short`
- `static`
- `super`
- `switch`
- `synchronized`
- `this`
- `throw`
- `throws`
- `transient`

- `try`
- `typeof`
- `var`
- `void`
- `volatile`
- `while`
- `with`

Arrays

Los arrays (en inglés *arrays*) son listas de valores con índice-cero (en inglés *zero-index*), es decir, que el primer elemento del array está en el índice `0`. éstos son una forma práctica de almacenar un conjunto de datos relacionados (como cadenas de caracteres), aunque en realidad, un array puede incluir múltiples tipos de datos, incluso otros arrays.

Un array simple

```
var myArray = [ 'hello', 'world' ];
```

Acceder a los ítems del array a través de su índice

```
var myArray = [ 'hello', 'world', 'foo', 'bar' ];

console.log(myArray[3]);   // muestra en la consola 'bar'
```

Obtener la cantidad de ítems del array

```
var myArray = [ 'hello', 'world' ];

console.log(myArray.length);   // muestra en la consola 2
```

Cambiar el valor de un ítem de un array

```
var myArray = [ 'hello', 'world' ];

myArray[1] = 'changed';
```

Como se muestra en el ejemplo *Cambiar el valor de un ítem de un array* es posible cambiar el valor de un ítem de un array, sin embargo, por lo general, no es aconsejable.

Añadir elementos a un array

```
var myArray = [ 'hello', 'world' ];

myArray.push('new');
```

Trabajar con arrays

```
var myArray = [ 'h', 'e', 'l', 'l', 'o' ];
```

```
var myString = myArray.join('');   // 'hello'

var mySplit = myString.split('');  // [ 'h', 'e', 'l', 'l', 'o' ]
```

Objetos

Los objetos son elementos que pueden contener cero o más conjuntos de pares de nombres claves y valores asociados a dicho objeto. Los nombres claves pueden ser cualquier palabra o número válido. El valor puede ser cualquier tipo de valor: un número, una cadena, un array, una función, incluso otro objeto.

[Definición: Cuando uno de los valores de un objeto es una función, ésta es nombrada como un *método* del objeto.] De lo contrario, se los llama *propiedades*.

Curiosamente, en JavaScript, casi todo es un objeto — arrays, funciones, números, incluso cadenas — y todos poseen propiedades y métodos.

Creación de un "objeto literal"

```
var myObject = {

    sayHello : function() {

        console.log('hello');

    },

    myName : 'Rebecca'

};


// se llama al método sayHello, el cual muestra en la consola 'hello'

myObject.sayHello();


// se llama a la propiedad myName, la cual muestra en la consola 'Rebecca'

console.log(myObject.myName);
```

NOTA

Notar que cuando se crean objetos literales, el nombre de la propiedad puede ser cualquier identificador JavaScript, una cadena de caracteres (encerrada entre comillas) o un número:

```
var myObject = {

    validIdentifier: 123,

    'some string': 456,

    99999: 789

};
```

Funciones

Las funciones contienen bloques de código que se ejecutaran repetidamente. A las mismas se le pueden pasar argumentos, y opcionalmente la función puede devolver un valor.

Las funciones pueden ser creadas de varias formas:

Declaración de una función

```
function foo() { /* hacer algo */ }
```

Declaración de una función nombrada

```
var foo = function() { /* hacer algo */ }
```

Es preferible el método de función nombrada debido a algunas profundas razones técnicas. Igualmente, es probable encontrar a los dos métodos cuando se revise código JavaScript.

Utilización de funciones

Una función simple

```
var greet = function(person, greeting) {

    var text = greeting + ', ' + person;

    console.log(text);

};


greet('Rebecca', 'Hello');  // muestra en la consola 'Hello, Rebecca'
```

Una función que devuelve un valor

```
var greet = function(person, greeting) {

    var text = greeting + ', ' + person;

    return text;

};


console.log(greet('Rebecca','hello'));
```

Una función que devuelve otra función

```
// la función devuelve 'Hello, Rebecca',

// la cual se muestra en la consola
```

```
var greet = function(person, greeting) {

    var text = greeting + ', ' + person;

    return function() { console.log(text); };

};


var greeting = greet('Rebecca', 'Hello');

greeting();  // se muestra en la consola 'Hello, Rebecca'
```

Funciones anónimas autoejecutables

Un patrón común en JavaScript son las funciones anónimas autoejecutables. Este patrón consiste en crear una expresión de función e inmediatamente ejecutarla. El mismo es muy útil para casos en que no se desea intervenir espacios de nombres globales, debido a que ninguna variable declarada dentro de la función es visible desde afuera.

Función anónima autoejecutable

```
(function(){

    var foo = 'Hello world';

})();

console.log(foo);   // indefinido (undefined)
```

Funciones como argumentos

En JavaScript, las funciones son *ciudadanos de primera clase* — pueden ser asignadas a variables o pasadas a otras funciones como argumentos. En jQuery, pasar funciones como argumentos es una práctica muy común.

Pasar una función anónima como un argumento

```
var myFn = function(fn) {

    var result = fn();

    console.log(result);

};


myFn(function() { return 'hello world'; }); // muestra en la consola 'hello world'
```

Pasar una función nombrada como un argumento

```
var myFn = function(fn) {
```

```
    var result = fn();

    console.log(result);

};


var myOtherFn = function() {

    return 'hello world';

};

myFn(myOtherFn);   // muestra en la consola 'hello world'
```

La palabra clave this

En JavaScript, así como en la mayoría de los lenguajes de programación orientados a objetos, this es una palabra clave especial que hace referencia al objeto en donde el método está siendo invocado. El valor de `this` es determinado utilizando una serie de simples pasos:

1. Si la función es invocada utilizando Function.call o Function.apply, this tendrá el valor del primer argumento pasado al método. Si el argumento es nulo (`null`) o indefinido (`undefined`), this hará referencia el objeto global (el objeto `window`);

2. Si la función a invocar es creada utilizando Function.bind, `this` será el primer argumento que es pasado a la función en el momento en que se la crea;

3. Si la función es invocada como un método de un objeto, `this` referenciará a dicho objeto;

4. De lo contrario, si la función es invocada como una función independiente, no unida a algún objeto, this referenciará al objeto global.

Una función invocada utilizando Function.call

```
var myObject = {

    sayHello : function() {

        console.log('Hola, mi nombre es ' + this.myName);

    },

    myName : 'Rebecca'

};


var secondObject = {

    myName : 'Colin'
```

```
};


myObject.sayHello();                  // registra 'Hola, mi nombre es Rebecca'
myObject.sayHello.call(secondObject); // registra 'Hola, mi nombre es Colin'
```

Una función creada utilizando Function.bind

```
var myName = 'el objeto global',


    sayHello = function () {
        console.log('Hola, mi nombre es ' + this.myName);
    },
    myObject = {
        myName : 'Rebecca'
    };


var myObjectHello = sayHello.bind(myObject);


sayHello();       // registra 'Hola, mi nombre es el objeto global'
myObjectHello();  // registra 'Hola, mi nombre es Rebecca'
```

Una función vinculada a un objeto

```
var myName = 'el objeto global',
    sayHello = function() {
        console.log('Hola, mi nombre es ' + this.myName);
    },
    myObject = {
        myName : 'Rebecca'
    },
    secondObject = {
        myName : 'Colin'
    };
```

```
myObject.sayHello = sayHello;

secondObject.sayHello = sayHello;


sayHello();               // registra 'Hola, mi nombre es el objeto global'

myObject.sayHello();      // registra 'Hola, mi nombre es Rebecca'

secondObject.sayHello();  // registra 'Hola, mi nombre es Colin'
```

En algunas oportunidades, cuando se invoca una función que se encuentra dentro de un espacio de nombres (en inglés *namespace*) amplio, puede ser una tentación guardar la referencia a la función actual en una variable más corta y accesible. Sin embargo, es importante no realizarlo en instancias de métodos, ya que puede llevar a la ejecución de código incorrecto. Por ejemplo:

```
var myNamespace = {

    myObject : {

        sayHello : function() {

            console.log('Hola, mi nombre es ' + this.myName);

        },

        myName : 'Rebecca'

    }

};


var hello = myNamespace.myObject.sayHello;

hello();  // registra 'Hola, mi nombre es undefined'
```

Para que no ocurran estos errores, es necesario hacer referencia al objeto en donde el método es invocado:

```
var myNamespace = {

    myObject : {

        sayHello : function() {

            console.log('Hola, mi nombre es ' + this.myName);

        },

        myName : 'Rebecca'
```

```
    }
};


var obj = myNamespace.myObject;

obj.sayHello();  // registra 'Hola, mi nombre es Rebecca'
```

Alcance

El "alcance" (en inglés *scope*) se refiere a las variables que están disponibles en un bloque de código en un tiempo determinado. La falta de comprensión de este concepto puede llevar a una frustrante experiencia de depuración.

Cuando una variable es declarada dentro de una función utilizando la palabra clave `var`, ésta únicamente está disponible para el código dentro de la función — todo el código fuera de dicha función no puede acceder a la variable. Por otro lado, las funciones definidas dentro de la función podrán acceder a la variable declarada.

Las variables que son declaradas dentro de la función sin la palabra clave `var` no quedan dentro del ámbito de la misma función — JavaScript buscará el lugar en donde la variable fue previamente declarada, y en caso de no haber sido declarada, es definida dentro del alcance global, lo cual puede ocasionar consecuencias inesperadas.

Funciones tienen acceso a variables definidas dentro del mismo alcance

```
var foo = 'hello';

var sayHello = function() {

    console.log(foo);

};

sayHello();         // muestra en la consola 'hello'

console.log(foo);   // también muestra en la consola 'hello'
```

El código de afuera no tiene acceso a la variable definida dentro de la función

```
var sayHello = function() {

    var foo = 'hello';

    console.log(foo);

};


sayHello();         // muestra en la consola 'hello'

console.log(foo);   // no muestra nada en la consola
```

Variables con nombres iguales pero valores diferentes pueden existir en diferentes alcances

```
var foo = 'world';

var sayHello = function() {

    var foo = 'hello';

    console.log(foo);

};

sayHello();         // muestra en la consola 'hello'

console.log(foo);   // muestra en la consola 'world'
```

Las funciones pueden "ver" los cambios en las variables antes de que la función sea definida

```
var myFunction = function() {

    var foo = 'hello';

    var myFn = function() {

        console.log(foo);

    };


    foo = 'world';

    return myFn;

};


var f = myFunction();

f();  // registra 'world' -- error
```

Alcance

```
// una función anónima autoejecutable

(function() {

    var baz = 1;

    var bim = function() { alert(baz); };

    bar = function() { alert(baz); };

})();
```

```
console.log(baz);  // La consola no muestra nada, ya que baz
                   // esta definida dentro del alcance de la función anónima


bar();  // bar esta definido fuera de la función anónima
        // ya que fue declarada sin la palabra clave var; además,
        // como fue definida dentro del mismo alcance que baz,
        // se puede consultar el valor de baz a pesar que
        // ésta este definida dentro del alcance de la función anónima


bim();  // bim no esta definida para ser accesible fuera de la función anónima,
        // por lo cual se mostrará un error
```

Conceptos Básicos de jQuery

$(document).ready()

No es posible interactuar de forma segura con el contenido de una página hasta que el documento no se encuentre preparado para su manipulación. jQuery permite detectar dicho estado a través de la declaración `$(document).ready()` de forma tal que el bloque se ejecutará sólo una vez que la página este disponible.

El bloque $(document).ready()

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

    console.log('el documento está preparado');

});
```

Existe una forma abreviada para `$(document).ready()` la cual podrá encontrar algunas veces; sin embargo, es recomendable no utilizarla en caso que este escribiendo código para gente que no conoce jQuery.

Forma abreviada para $(document).ready()

```
$(function() {

    console.log('el documento está preparado');

});
```

Además es posible pasarle a `$(document).ready()` una función nombrada en lugar de una anónima:

Pasar una función nombrada en lugar de una función anónima

```
function readyFn() {

    // código a ejecutar cuando el documento este listo

}

$(document).ready(readyFn);
```

Selección de elementos

El concepto más básico de jQuery es el de *"seleccionar algunos elementos y realizar acciones con ellos"*. La biblioteca soporta gran parte de los selectores CSS3 y varios más no estandarizados.

A continuación se muestran algunas técnicas comunes para la selección de elementos:

Selección de elementos en base a su `id`

```
$('#myId'); // notar que los IDs deben ser únicos por página
```

Selección de elementos en base al nombre de clase

```
$('div.myClass'); // si se especifica el tipo de elemento,

                  // se mejora el rendimiento de la selección
```

Selección de elementos por su atributo

```
$('input[name=first_name]'); // tenga cuidado, que puede ser muy lento
```

Selección de elementos en forma de selector CSS

```
$('#contents ul.people li');
```

Pseudo-selectores

```
// selecciona el primer elemento <a> con la clase 'external'

$('a.external:first');


// selecciona todos los elementos <tr> impares de una tabla

$('tr:odd');


// selecciona todos los elementos del tipo input dentro del formulario #myForm

$('#myForm :input');


// selecciona todos los divs visibles
```

```
$('div:visible');


// selecciona todos los divs excepto los tres primeros

$('div:gt(2)');

// selecciona todos los divs actualmente animados

$('div:animated');
```

NOTA

Cuando se utilizan los pseudo-selectores `:visible` y `:hidden`, jQuery comprueba la visibilidad actual del elemento pero no si éste posee asignados los estilos CSS `visibility` o `display` — en otras palabras, verifica si el alto y ancho físico del elemento es mayor a cero. Sin embargo, esta comprobación no funciona con los elementos `<tr>`; en este caso, jQuery comprueba si se está aplicando el estilo `display` y va a considerar al elemento como oculto si posee asignado el valor `none`. Además, los elementos que aún no fueron añadidos al DOM serán tratados como ocultos, incluso si tienen aplicados estilos indicando que deben ser visibles (En la sección Manipulación de este manual, se explica como crear y añadir elementos al DOM).

Como referencia, este es el fragmento de código que utiliza jQuery para determinar cuando un elemento es visible o no. Se incorporaron los comentarios para que quede más claro su entendimiento:

```
jQuery.expr.filters.hidden = function( elem ) {

    var width = elem.offsetWidth, height = elem.offsetHeight,

        skip = elem.nodeName.toLowerCase() === "tr";


    // ¿el elemento posee alto 0, ancho 0 y no es un <tr>?

    return width === 0 && height === 0 && !skip ?


        // entonces debe estar oculto (hidden)

        true :


        // pero si posee ancho y alto y no es un <tr>

        width > 0 && height > 0 && !skip ?


            // entonces debe estar visible
```

```
            false :


            // si nos encontramos aquí, es porque el elemento posee ancho

            // y alto, pero además es un <tr>,

            // entonces se verifica el valor del estilo display

            // aplicado a través de CSS

            // para decidir si está oculto o no

            jQuery.curCSS(elem, "display") === "none";

};



jQuery.expr.filters.visible = function( elem ) {

    return !jQuery.expr.filters.hidden( elem );

};
```

Elección de selectores

La elección de buenos selectores es un punto importante cuando se desea mejorar el rendimiento del código. Una pequeña especificidad — por ejemplo, incluir el tipo de elemento (como `div`) cuando se realiza una selección por el nombre de clase — puede ayudar bastante. Por eso, es recomendable darle algunas "pistas" a jQuery sobre en que lugar del documento puede encontrar lo que desea seleccionar. Por otro lado, demasiada especificidad puede ser perjudicial. Un selector como `#miTabla thead tr th.especial` es un exceso, lo mejor sería utilizar `#miTabla th.especial`.

jQuery ofrece muchos selectores basados en atributos, que permiten realizar selecciones basadas en el contenido de los atributos utilizando simplificaciones de expresiones regulares.

```
// encontrar todos los <a> cuyo atributo rel terminan en "thinger"

$("a[rel$='thinger']");
```

Estos tipos de selectores pueden resultar útiles pero también ser muy lentos. Cuando sea posible, es recomendable realizar la selección utilizando IDs, nombres de clases y nombres de etiquetas.

Si desea conocer más sobre este asunto, Paul Irish realizó una gran presentación sobre mejoras de rendimiento en JavaScript (en ingles), la cual posee varias diapositivas centradas en selectores.

Comprobar selecciones

Una vez realizada la selección de los elementos, querrá conocer si dicha selección entregó algún resultado. Para ello, pueda que escriba algo así:

```
if ($('div.foo')) { ... }
```

Sin embargo esta forma no funcionará. Cuando se realiza una selección utilizando `$()`, siempre es devuelto un objeto, y si se lo evalúa, éste siempre devolverá true. Incluso si la selección no contiene ningún elemento, el código dentro del bloque `if` se ejecutará.

En lugar de utilizar el código mostrado, lo que se debe hacer es preguntar por la cantidad de elementos que posee la selección que se ejecutó. Esto es posible realizarlo utilizando la propiedad JavaScript `length`. Si la respuesta es `0`, la condición evaluará falso, caso contrario (más de `0` elementos), la condición será verdadera.

Evaluar si una selección posee elementos

```
if ($('div.foo').length) { ... }
```

Guardar selecciones

Cada vez que se hace una selección, una gran cantidad de código es ejecutado. jQuery no guarda el resultado por si solo, por lo tanto, si va a realizar una selección que luego se hará de nuevo, deberá salvar la selección en una variable.

Guardar selecciones en una variable

```
var $divs = $('div');
```

NOTA

En el ejemplo "Guardar selecciones en una variable", la variable comienza con el signo de dólar. Contrariamente a otros lenguajes de programación, en JavaScript este signo no posee ningún significado especial — es solamente otro carácter. Sin embargo aquí se utilizará para indicar que dicha variable posee un objeto jQuery. Esta práctica — una especie de Notación Húngara — es solo una convención y no es obligatoria.

Una vez que la selección es guardada en la variable, se la puede utilizar en conjunto con los métodos de jQuery y el resultado será igual que utilizando la selección original.

NOTA

La selección obtiene sólo los elementos que están en la página cuando se realizó dicha acción. Si luego se añaden elementos al documento, será necesario repetir la selección o añadir los elementos nuevos a la selección guardada en la variable. En otras palabras, las selecciones guardadas no se actualizan *mágicamente* cuando el DOM se modifica.

Refinamiento y filtrado de selecciones

A veces, puede obtener una selección que contiene más de lo que necesita; en este caso, es necesario refinar dicha selección. jQuery ofrece varios métodos para poder obtener exactamente lo que desea.

Refinamiento de selecciones

```
$('div.foo').has('p');          // el elemento div.foo contiene elementos <p>
```

```
$('h1').not('.bar');            // el elemento h1 no posse la clase 'bar'

$('ul li').filter('.current');  // un item de una lista desordenada

                                // que posse la clase 'current'

$('ul li').first();             // el primer item de una lista desordenada

$('ul li').eq(5);               // el sexto item de una lista desordenada
```

Selección de elementos de un formulario

jQuery ofrece varios pseudo-selectores que ayudan a encontrar elementos dentro de los formularios, éstos son especialmente útiles ya que dependiendo de los estados de cada elemento o su tipo, puede ser difícil distinguirlos utilizando selectores CSS estándar.

- `:button` Selecciona elementos `<button>` y con el atributo `type='button'`
- `:checkbox` Selecciona elementos `<input>` con el atributo `type='checkbox'`
- `:checked` Selecciona elementos `<input>` del tipo checkbox seleccionados
- `:disabled` Selecciona elementos del formulario que están deshabitados
- `:enabled` Selecciona elementos del formulario que están habilitados
- `:file` Selecciona elementos `<input>` con el atributo `type='file'`
- `:image` Selecciona elementos `<input>` con el atributo `type='image'`
- `:input` Selecciona elementos `<input>`, `<textarea>` y `<select>`
- `:password` Selecciona elementos `<input>` con el atributo `type='password'`
- `:radio` Selecciona elementos `<input>` con el atributo `type='radio'`
- `:reset` Selecciona elementos `<input>` con el atributo `type='reset'`
- `:selected` Selecciona elementos `<options>` que están seleccionados
- `:submit` Selecciona elementos `<input>` con el atributo `type='submit'`
- `:text` Selecciona elementos `<input>` con el atributo `type='text'`

Utilizando pseudo-selectores en elementos de formularios

```
// obtiene todos los elementos inputs dentro del formulario #myForm

$('#myForm :input');
```

Trabajar con selecciones

Una vez realizada la selección de los elementos, es posible utilizarlos en conjunto con diferentes métodos. éstos, generalmente, son de dos tipos: obtenedores (en inglés *getters*) y establecedores (en inglés *setters*). Los métodos obtenedores devuelven una propiedad del elemento seleccionado; mientras que los métodos establecedores fijan una propiedad a todos los elementos seleccionados.

Encadenamiento

Si en una selección se realiza una llamada a un método, y éste devuelve un objeto jQuery, es posible seguir un "encadenado" de métodos en el objeto.

Encadenamiento

```
$('#content').find('h3').eq(2).html('nuevo texto para el tercer elemento h3');
```

Por otro lado, si se está escribiendo un encadenamiento de métodos que incluyen muchos pasos, es posible escribirlos línea por línea, haciendo que el código luzca más agradable para leer.

Formateo de código encadenado

```
$('#content')

    .find('h3')

    .eq(2)

    .html('nuevo texto para el tercer elemento h3');
```

Si desea volver a la selección original en el medio del encadenado, jQuery ofrece el método `$.fn.end`para poder hacerlo.

Restablecer la selección original utilizando el método $.fn.end

```
$('#content')

    .find('h3')

    .eq(2)

        .html('nuevo texto para el tercer elemento h3')

    .end() // reestablece la selección a todos los elementos h3 en #content

    .eq(0)

        .html('nuevo texto para el primer elemento h3');
```

NOTA

El encadenamiento es muy poderoso y es una característica que muchas bibliotecas JavaScript han adoptado desde que jQuery se hizo popular. Sin embargo, debe ser utilizado con cuidado. Un encadenamiento de métodos extensivo pueden hacer un código extremadamente difícil de

modificar y depurar. No existe una regla que indique que tan largo o corto debe ser el encadenado — pero es recomendable que tenga en cuenta este consejo.

Obtenedores (getters) y establecedores (setters)

jQuery "sobrecarga" sus métodos, en otras palabras, el método para establecer un valor posee el mismo nombre que el método para obtener un valor. Cuando un método es utilizado para establecer un valor, es llamado método establecedor (en inglés *setter*). En cambio, cuando un método es utilizado para obtener (o leer) un valor, es llamado obtenedor (en inglés *getter*).

El método $.fn.html utilizado como establecedor

```
$('h1').html('hello world');
```

El método html utilizado como obtenedor

```
$('h1').html();
```

Los métodos establecedores devuelven un objeto jQuery, permitiendo continuar con la llamada de más métodos en la misma selección, mientras que los métodos obtenedores devuelven el valor por el cual se consultó, pero no permiten seguir llamando a más métodos en dicho valor.

CSS, estilos y dimensiones

jQuery incluye una manera útil de obtener y establecer propiedades CSS a los elementos.

NOTA

Las propiedades CSS que incluyen como separador un guión del medio, en JavaScript deben ser transformadas a su estilo CamelCase. Por ejemplo, cuando se la utiliza como propiedad de un método, el estilo CSS `font-size` deberá ser expresado como `fontSize`. Sin embargo, esta regla no es aplicada cuando se pasa el nombre de la propiedad CSS al método `$.fn.css` — en este caso, los dos formatos (en CamelCase o con el guión del medio) funcionarán.

Obtener propiedades CSS

```
$('h1').css('fontSize'); // devuelve una cadena de caracteres como "19px"

$('h1').css('font-size'); // también funciona
```

Establecer propiedades CSS

```
// establece una propiedad individual CSS

$('h1').css('fontSize', '100px');



// establece múltiples propiedades CSS

$('h1').css({ 'fontSize' : '100px', 'color' : 'red' });
```

Notar que el estilo del argumento utilizado en la segunda línea del ejemplo — es un objeto que contiene múltiples propiedades. Esta es una forma común de pasar múltiples argumentos a una función, y muchos métodos establecedores de la biblioteca aceptan objetos para fijar varias propiedades de una sola vez.

Utilizar clases para aplicar estilos CSS

Para obtener valores de los estilos aplicados a un elemento, el método `$.fn.css` es muy útil, sin embargo, su utilización como método establecedor se debe evitar (ya que, para aplicar estilos a un elemento, se puede hacer directamente desde CSS). En su lugar, lo ideal, es escribir reglas CSS que se apliquen a clases que describan los diferentes estados visuales de los elementos y luego cambiar la clase del elemento para aplicar el estilo que se desea mostrar.

Trabajar con clases

```
var $h1 = $('h1');

$h1.addClass('big');

$h1.removeClass('big');

$h1.toggleClass('big');


if ($h1.hasClass('big')) { ... }
```

Las clases también pueden ser útiles para guardar información del estado de un elemento, por ejemplo, para indicar que un elemento fue seleccionado.

Dimensiones

jQuery ofrece una variedad de métodos para obtener y modificar valores de dimensiones y posición de un elemento.

El código mostrado en el ejemplo "Métodos básicos sobre Dimensiones" es solo un breve resumen de las funcionalidades relaciones a dimensiones en jQuery.

Métodos básicos sobre Dimensiones

```
$('h1').width('50px');  // establece el ancho de todos los elementos H1

$('h1').width();        // obtiene el ancho del primer elemento H1


$('h1').height('50px'); // establece el alto de todos los elementos H1

$('h1').height();       // obtiene el alto del primer elemento H1


// devuelve un objeto conteniendo información sobre la posición
```

```
// del primer elemento relativo al "offset" (posición) de su elemento padre

$('h1').position();
```

Atributos

Los atributos de los elementos HTML que conforman una aplicación pueden contener información útil, por eso es importante poder establecer y obtener esa información.

El método `$.fn.attr` actúa tanto como método establecedor como obtenedor. Además, al igual que el método `$.fn.css`, cuando se lo utiliza como método establecedor, puede aceptar un conjunto de palabra clave-valor o un objeto conteniendo más conjuntos.

Establecer atributos

```
$('a').attr('href', 'allMyHrefsAreTheSameNow.html');

$('a').attr({

    'title' : 'all titles are the same too',

    'href' : 'somethingNew.html'

});
```

En el ejemplo, el objeto pasado como argumento está escrito en varias líneas. Como se explicó anteriormente, los espacios en blanco no importan en JavaScript, por lo cual, es libre de utilizarlos para hacer el código más legible. En entornos de producción, se pueden utilizar herramientas de minificación, los cuales quitan los espacios en blanco (entre otras cosas) y comprimen el archivo final.

Obtener atributos

```
$('a').attr('href');  // devuelve el atributo href perteneciente

                      // al primer elemento <a> del documento
```

Recorrer el DOM

Una vez obtenida la selección, es posible encontrar otros elementos utilizando a la misma selección.

NOTA

Debe ser cuidadoso en recorrer largas distancias en un documento — recorridos complejos obligan que la estructura del documento sea siempre la misma, algo que es difícil de garantizar. Uno o dos pasos para el recorrido esta bien, pero generalmente hay que evitar atravesar desde un contenedor a otro.

Moverse a través del DOM utilizando métodos de recorrido

```
// seleccionar el inmediato y próximo elemento <p> con respecto a H1

$('h1').next('p');
```

```
// seleccionar el elemento contenedor a un div visible

$('div:visible').parent();


// seleccionar el elemento <form> más cercano a un input

$('input[name=first_name]').closest('form');

// seleccionar todos los elementos hijos de #myList

$('#myList').children();

// seleccionar todos los items hermanos del elemento <li>

$('li.selected').siblings();
```

También es posible interactuar con la selección utilizando el método `$.fn.each`. Dicho método interactúa con todos los elementos obtenidos en la selección y ejecuta una función por cada uno. La función recibe como argumento el índice del elemento actual y al mismo elemento. De forma predeterminada, dentro de la función, se puede hacer referencia al elemento DOM a través de la declaración `this`.

Interactuar en una selección

```
$('#myList li').each(function(idx, el) {

    console.log(

        'El elemento ' + idx +

        'contiene el siguiente HTML: ' +

        $(el).html()

); });
```

Manipulación de elementos

Una vez realizada la selección de los elementos que desea utilizar, *la diversión comienza*. Es posible cambiar, mover, eliminar y duplicar elementos. También crear nuevos a través de una sintaxis simple.

Obtener y establecer información en elementos

Existen muchas formas por las cuales se puede modificar un elemento. Entre las tareas más comunes están las de cambiar el HTML interno o algún atributo del mismo. Para este tipo de tareas, jQuery ofrece métodos simples, funcionales en todos los navegadores modernos. Incluso es posible obtener información sobre los elementos utilizando los mismos métodos pero en su forma de método obtenedor.

NOTA

Realizar cambios en los elementos, es un trabajo trivial, pero hay debe recordar que el cambio afectará a todos los elementos en la selección, por lo que, si desea modificar un sólo elemento, tiene que estar seguro de especificarlo en la selección antes de llamar al método establecedor.

NOTA

Cuando los métodos actúan como obtenedores, por lo general, solamente trabajan con el primer elemento de la selección. Además no devuelven un objeto jQuery, por lo cual no es posible encadenar más métodos en el mismo. Una excepción es el método `$.fn.text`, el cual permite obtener el texto de los elementos de la selección.

- `$.fn.html` Obtiene o establece el contenido HTML de un elemento.
- `$.fn.text` Obtiene o establece el contenido en texto del elemento; en caso se pasarle como argumento código HTML, este es despojado.
- `$.fn.attr` Obtiene o establece el valor de un determinado atributo.
- `$.fn.width` Obtiene o establece el ancho en pixeles del primer elemento de la selección como un entero.
- `$.fn.height` Obtiene o establece el alto en pixeles del primer elemento de la selección como un entero.
- `$.fn.position` Obtiene un objeto con información sobre la posición del primer elemento de la selección, relativo al primer elemento padre posicionado. Este método es solo obtenedor.
- `$.fn.val` Obtiene o establece el valor (*value*) en elementos de formularios.

Cambiar el HTML de un elemento

```
$('#myDiv p:first')

    .html('Nuevo <strong>primer</strong> párrafo');
```

Mover, copiar y eliminar elementos

Existen varias maneras para mover elementos a través del DOM; las cuales se pueden separar en dos enfoques:

- Querer colocar el/los elementos seleccionados de forma relativa a otro elemento
- Querer colocar un elemento relativo a el/los elementos seleccionados.

Por ejemplo, jQuery provee los métodos `$.fn.insertAfter` y `$.fn.after`. El método `$.fn.insertAfter`coloca a el/los elementos seleccionados después del elemento que se haya pasado como argumento; mientras que el método `$.fn.after` coloca al elemento pasado como argumento después del elemento seleccionado. Otros métodos también siguen este patrón: `$.fn.insertBefore` y `$.fn.before`; `$.fn.appendTo` y `$.fn.append`; y `$.fn.prependTo` y `$.fn.prepend`.

La utilización de uno u otro método dependerá de los elementos que tenga seleccionados y el tipo de referencia que se quiera guardar con respecto al elemento que se esta moviendo.

Mover elementos utilizando diferentes enfoques

```
// hacer que el primer item de la lista sea el último

var $li = $('#myList li:first').appendTo('#myList');



// otro enfoque para el mismo problema

$('#myList').append($('#myList li:first'));



// debe tener en cuenta que no hay forma de acceder a la

// lista de items que se ha movido, ya que devuelve la lista en sí
```

Clonar elementos

Cuando se utiliza un método como `$.fn.appendTo`, lo que se está haciendo es mover al elemento; pero a veces en lugar de eso, se necesita mover un duplicado del mismo elemento. En este caso, es posible utilizar el método `$.fn.clone`.

Obtener una copia del elemento

```
// copiar el primer elemento de la lista y moverlo al final de la misma

$('#myList li:first').clone().appendTo('#myList');
```

NOTASi se necesita copiar información y eventos relacionados al elemento, se debe pasar `true` como argumento de `$.fn.clone`.

Eliminar elementos

Existen dos formas de eliminar elementos de una página: Utilizando `$.fn.remove` o `$.fn.detach`. Cuando desee eliminar de forma permanente al elemento, utilize el método `$.fn.remove`. Por otro lado, el método `$.fn.detach` también elimina el elemento, pero mantiene la información y eventos asociados al mismo, siendo útil en el caso que necesite reinsertar el elemento en el documento.

NOTA

El método `$.fn.detach` es muy útil cuando se esta manipulando de forma severa un elemento, ya que es posible eliminar al elemento, trabajarlo en el código y luego restaurarlo en la página nuevamente. Esta forma tiene como beneficio no tocar el DOM mientras se está modificando la información y eventos del elemento.

Por otro lado, si se desea mantener al elemento pero se necesita eliminar su contenido, es posible utiliza el método `$.fn.empty`, el cual "vaciará" el contenido HTML del elemento.

Crear nuevos elementos

jQuery provee una forma fácil y elegante para crear nuevos elementos a través del mismo método `$()`que se utiliza para realizar selecciones.

Crear nuevos elementos

```
$('<p>Un nuevo párrafo</p>');

$('<li class="new">nuevo item de la lista</li>');
```

Crear un nuevo elemento con atributos utilizando un objeto

```
$('<a/>', {

    html : 'Un <strong>nuevo</strong> enlace',

    'class' : 'new',

    href : 'foo.html'

});
```

Note que en el objeto que se pasa como argumento, la propiedad `class` está entre comillas, mientras que la propiedad `href` y `html` no lo están. Por lo general, los nombres de propiedades no deben estar entre comillas, excepto en el caso que se utilice como nombre una palabra reservada (como es el caso de `class`).

Cuando se crea un elemento, éste no es añadido inmediatamente a la página, sino que se debe hacerlo en conjunto con un método.

Crear un nuevo elemento en la página

```
var $myNewElement = $('<p>Nuevo elemento</p>');

$myNewElement.appendTo('#content');



// eliminará al elemento <p> existente en #content

$myNewElement.insertAfter('ul:last');



// clonar al elemento <p> para tener las dos versiones

$('ul').last().after($myNewElement.clone());
```

Estrictamente hablando, no es necesario guardar al elemento creado en una variable — es posible llamar al método para añadir el elemento directamente después de `$()`. Sin embargo, la mayoría de las veces se deseará hacer referencia al elemento añadido, por lo cual, si se guarda en una variable no es necesario seleccionarlo después.

Crear y añadir al mismo tiempo un elemento a la página

```
$('ul').append('<li>item de la lista</li>');
```

NOTA

La sintaxis para añadir nuevos elementos a la página es muy fácil de utilizar, pero es tentador olvidar que hay un costo enorme de rendimiento al agregar elementos al DOM de forma repetida. Si esta añadiendo muchos elementos al mismo contenedor, en lugar de añadir cada elemento uno por vez, lo mejor es concatenar todo el HTML en una única cadena de caracteres para luego anexarla al contenedor. Una posible solución es utilizar un array que posea todos los elementos, luego reunirlos utilizando `join` y finalmente anexarla.

```
var myItems = [], $myList = $('#myList');



for (var i=0; i<100; i++) {

    myItems.push('<li>item ' + i + '</li>');

}



$myList.append(myItems.join(''));
```

Manipulación de atributos

Las capacidades para la manipulación de atributos que ofrece la biblioteca son extensos. La realización de cambios básicos son simples, sin embargo el método `$.fn.attr` permite manipulaciones más complejas.

Manipular un simple atributo

```
$('#myDiv a:first').attr('href', 'newDestination.html');
```

Manipular múltiples atributos

```
$('#myDiv a:first').attr({

    href : 'newDestination.html',

    rel : 'super-special'

});
```

Utilizar una función para determinar el valor del nuevo atributo

```
$('#myDiv a:first').attr({

    rel : 'super-special',

    href : function(idx, href) {

        return '/new/' + href;

    }

});
```

```
$('#myDiv a:first').attr('href', function(idx, href) {

    return '/new/' + href;

});
```

$ vs $()

Hasta ahora, se ha tratado completamente con métodos que se llaman desde el objeto jQuery. Por ejemplo:

```
$('h1').remove();
```

Dichos métodos son parte del espacio de nombres (en inglés *namespace*) `$.fn`, o del prototipo (en inglés *prototype*) de jQuery, y son considerados como métodos del objeto jQuery.

Sin embargo, existen métodos que son parte del espacio de nombres de `$` y se consideran como métodos del núcleo de jQuery.

Estas distinciones pueden ser bastantes confusas para usuarios nuevos. Para evitar la confusión, debe recordar estos dos puntos:

- Los métodos utilizados en selecciones se encuentran dentro del espacio de nombres `$.fn`, y automáticamente reciben y devuelven una selección en sí.
- Métodos en el espacio de nombres `$` son generalmente métodos para diferentes utilidades, no trabajan con selecciones, no se les pasa ningún argumento y el valor que devuelven puede variar.

Existen algunos casos en donde métodos del objeto y del núcleo poseen los mismos nombres, como sucede con `$.each` y `$.fn.each`. En estos casos, debe ser cuidadoso de leer bien la documentación para saber que objeto utilizar correctamente.

Métodos útiles

jQuery ofrece varios métodos utilitarios dentro del espacio de nombres `$`.

`$.trim` Remueve los espacios en blanco del principio y final.

```
$.trim('    varios espacios en blanco   ');

// devuelve 'varios espacios en blanco'
```

`$.each` Interactúa en arrays y objetos.

```
$.each([ 'foo', 'bar', 'baz' ], function(idx, val) {

    console.log('elemento ' + idx + 'es ' + val);

});
```

```
$.each({ foo : 'bar', baz : 'bim' }, function(k, v) {

    console.log(k + ' : ' + v);

});
```

NOTA

Como se dijo antes, existe un método llamado `$.fn.each`, el cual interactúa en una selección de elementos.

`$.inArray` Devuelve el índice de un valor en un array, o `-1` si el valor no se encuentra en el array.

```
var myArray = [ 1, 2, 3, 5 ];

if ($.inArray(4, myArray) !== -1) {

    console.log('valor encontrado');

}
```

`$.extend` Cambia la propiedades del primer objeto utilizando las propiedades de los subsecuentes objetos.

```
var firstObject = { foo : 'bar', a : 'b' };

var secondObject = { foo : 'baz' };


var newObject = $.extend(firstObject, secondObject);

console.log(firstObject.foo); // 'baz'

console.log(newObject.foo);   // 'baz'
```

Si no se desea cambiar las propiedades de ninguno de los objetos que se utilizan en `$.extend`, se debe incluir un objeto vacío como primer argumento.

```
var firstObject = { foo : 'bar', a : 'b' };

var secondObject = { foo : 'baz' };


var newObject = $.extend({}, firstObject, secondObject);

console.log(firstObject.foo); // 'bar'

console.log(newObject.foo);   // 'baz'
```

`$.proxy` Devuelve una función que siempre se ejecutará en el alcance (*scope*) provisto — en otras palabras, establece el significado de `this` (incluido dentro de la función) como el segundo argumento.

```
var myFunction = function() { console.log(this); };

var myObject = { foo : 'bar' };
```

```
myFunction(); // devuelve el objeto window


var myProxyFunction = $.proxy(myFunction, myObject);

myProxyFunction(); // devuelve el objeto myObject
```

Si se posee un objeto con métodos, es posible pasar dicho objeto y el nombre de un método para devolver una función que siempre se ejecuta en el alcance de dicho objeto.

```
var myObject = {

    myFn : function() {

        console.log(this);

    }

};


$('#foo').click(myObject.myFn); // registra el elemento DOM #foo

$('#foo').click($.proxy(myObject, 'myFn')); // registra myObject
```

Comprobación de tipos

Como se mencionó en el capítulo *"Conceptos Básicos de JavaScript"*, jQuery ofrece varios métodos útiles para determinar el tipo de un valor específico.

Comprobar el tipo de un determinado valor

```
var myValue = [1, 2, 3];


// Utilizar el operador typeof de JavaScript para comprobar tipos primitivos

typeof myValue == 'string'; // falso (false)

typeof myValue == 'number'; // falso (false)

typeof myValue == 'undefined'; // falso (false)

typeof myValue == 'boolean'; // falso (false)


// Utilizar el operador de igualdad estricta para comprobar
```

```
// alores nulos (null)

myValue === null; // falso (false)


// Utilizar los métodos jQuery para comprobar tipos no primitivos

jQuery.isFunction(myValue); // falso (false)

jQuery.isPlainObject(myValue); // falso (false)

jQuery.isArray(myValue); // verdadero (true)
```

El método data

A menudo encontrará que existe información acerca de un elemento que necesita guardar. En JavaScript es posible hacerlo añadiendo propiedades al DOM del elemento, pero esta práctica conlleva enfrentarse a consumos excesivos de memoria (en inglés *memory leaks*) en algunos navegadores. jQuery ofrece una manera sencilla para poder guardar información relacionada a un elemento, y la misma biblioteca se ocupa de manejar los problemas que pueden surgir por falta de memoria.

Guardar y recuperar información relacionada a un elemento

```
$('#myDiv').data('keyName', { foo : 'bar' });

$('#myDiv').data('keyName'); // { foo : 'bar' }
```

A través del método `$.fn.data` es posible guardar cualquier tipo de información sobre un elemento, y es difícil exagerar la importancia de este concepto cuando se está desarrollando una aplicación compleja.

Por ejemplo, si desea establecer una relación entre el ítem de una lista y el div que hay dentro de este ítem, es posible hacerlo cada vez que se interactúa con el ítem, pero una mejor solución es hacerlo una sola vez, guardando un puntero al div utilizando el método `$.fn.data`:

Establecer una relación entre elementos utilizando el método $.fn.data

```
$('#myList li').each(function() {

    var $li = $(this), $div = $li.find('div.content');

    $li.data('contentDiv', $div);

});


// luego, no se debe volver a buscar al div;

// es posible leerlo desde la información asociada al item de la lista
```

```
var $firstLi = $('#myList li:first');

$firstLi.data('contentDiv').html('nuevo contenido');
```

Además es posible pasarle al método un objeto conteniendo uno o más pares de conjuntos palabra clave-valor.

Vincular eventos a elementos

jQuery ofrece métodos para la mayoría de los eventos — entre ellos `$.fn.click`, `$.fn.focus`, `$.fn.blur`, `$.fn.change`, etc. Estos últimos son formas reducidas del método `$.fn.bind` de jQuery. El método `bind`es útil para vincular (en inglés *binding*) la misma función de controlador a múltiples eventos, para cuando se desea proveer información al controlador de evento, cuando se está trabajando con eventos personalizados o cuando se desea pasar un objeto a múltiples eventos y controladores.

Vincular un evento utilizando un método reducido

```
$('p').click(function() {

    console.log('click');

});
```

Vincular un evento utilizando el método $.fn.bind method

```
$('p').bind('click', function() {

    console.log('click');

});
```

Vincular un evento utilizando el método $.fn.bind con información asociada

```
$('input').bind(

    'click change',  // es posible incluir múltiples eventos al elemento

    { foo : 'bar' }, // se debe pasar la información asociada como argumento

    function(eventObject) {

        console.log(eventObject.type, eventObject.data);

        // registra el tipo de evento y la información asociada { foo : 'bar' }

} );
```

Vincular eventos para ejecutar una vez

A veces puede necesitar que un controlador particular se ejecute solo una vez — y después de eso, necesite que ninguno más se ejecute, o que se ejecute otro diferente. Para este propósito jQuery provee el método `$.fn.one`.

Cambiar controladores utilizando el método $.fn.one

```
$('p').one('click', function() {

    console.log('Se clickeó al elemento por primera vez');

    $(this).click(function() { console.log('Se ha clickeado nuevamente'); });

});
```

El método `$.fn.one` es útil para situaciones en que necesita ejecutar cierto código la primera vez que ocurre un evento en un elemento, pero no en los eventos sucesivos.

Desvincular eventos

Para desvincular (en ingles *unbind*) un controlador de evento, puede utilizar el método `$.fn.unbind`pasándole el tipo de evento a desconectar. Si se pasó como adjunto al evento una función nombrada, es posible aislar la desconexión de dicha función pasándola como segundo argumento.

Desvincular todos los controladores del evento click en una selección

```
$('p').unbind('click');
```

Desvincular un controlador particular del evento click

```
var foo = function() { console.log('foo'); };

var bar = function() { console.log('bar'); };



$('p').bind('click', foo).bind('click', bar);

$('p').unbind('click', bar); // foo esta atado aún al evento click
```

Espacios de nombres para eventos

Cuando se esta desarrollando aplicaciones complejas o extensiones de jQuery, puede ser útil utilizar espacios de nombres para los eventos, y de esta forma evitar que se desvinculen eventos cuando no lo desea.

Asignar espacios de nombres a eventos

```
$('p').bind('click.myNamespace', function() { /* ... */ });

$('p').unbind('click.myNamespace');
```

```
// desvincula todos los eventos con el espacio de nombre 'myNamespace'

$('p').unbind('.myNamespace');
```

Vinculación de múltiples eventos

Muy a menudo, elementos en una aplicación estarán vinculados a múltiples eventos, cada uno con una función diferente. En estos casos, es posible pasar un objeto dentro de `$.fn.bind` con uno o más pares de nombres claves/valores. Cada nombre clave será el nombre del evento mientras que cada valor será la función a ejecutar cuando ocurra el evento.

Vincular múltiples eventos a un elemento

```
$('p').bind({

    'click': function() { console.log('clickeado'); },

    'mouseover': function() { console.log('sobrepasado'); }

});
```

El objeto del evento

Como se menciona en la introducción, la función controladora de eventos recibe un objeto del evento, el cual contiene varios métodos y propiedades. El objeto es comúnmente utilizado para prevenir la acción predeterminada del evento a través del método `preventDefault`. Sin embargo, también contiene varias propiedades y métodos útiles:

- `pageX`, `pageY` La posición del puntero del ratón en el momento que el evento ocurrió, relativo a las zonas superiores e izquierda de la página.
- `type` El tipo de evento (por ejemplo `click`).
- `which` El botón o tecla presionada.
- `data` Alguna información pasada cuando el evento es ejecutado.
- `target` El elemento DOM que inicializó el evento.
- `preventDefault()` Cancela la acción predeterminada del evento (por ejemplo: seguir un enlace).
- `stopPropagation()` Detiene la propagación del evento sobre otros elementos.

Por otro lado, la función controladora también tiene acceso al elemento DOM que inicializó el evento a través de la palabra clave `this`. Para convertir a dicho elemento DOM en un objeto jQuery (y poder utilizar los métodos de la biblioteca) es necesario escribir `$(this)`, como se muestra a continuación:

```
var $this = $(this);
```

Cancelar que al hacer click en un enlace, éste se siga

```
$('a').click(function(e) {

    var $this = $(this);

    if ($this.attr('href').match('evil')) {

        e.preventDefault();

        $this.addClass('evil');

    }

});
```

Efectos incorporados en la biblioteca

Los efectos más utilizados ya vienen incorporados dentro de la biblioteca en forma de métodos:

- `$.fn.show` Muestra el elemento seleccionado.
- `$.fn.hide` Oculta el elemento seleccionado.
- `$.fn.fadeIn` De forma animada, cambia la opacidad del elemento seleccionado al 100 %.
- `$.fn.fadeOut` De forma animada, cambia la opacidad del elemento seleccionado al `0`
- `$.fn.slideDown` Muestra el elemento seleccionado con un movimiento de deslizamiento vertical.
- `$.fn.slideUp` Oculta el elemento seleccionado con un movimiento de deslizamiento vertical.
- `$.fn.slideToggle` Muestra o oculta el elemento seleccionado con un movimiento de deslizamiento vertical, dependiendo si actualmente el elemento está visible o no.

Uso básico de un efecto incorporado

```
$('h1').show();
```

Cambiar la duración de los efectos

Con la excepción de `$.fn.show` y `$.fn.hide`, todos los métodos tienen una duración predeterminada de la animación en 400 milisegundos. Este valor es posible cambiarlo.

Configurar la duración de un efecto

```
$('h1').fadeIn(300);      // desvanecimiento en 300ms

$('h1').fadeOut('slow');  // utilizar una definición de velocidad interna
```

jQuery.fx.speeds

jQuery posee un objeto en `jQuery.fx.speeds` el cual contiene la velocidad predeterminada para la duración de un efecto, así como también los valores para las definiciones *slow* y *fast*.

```
speeds: {

    slow: 600,

    fast: 200,

    // velocidad predeterminada

    _default: 400

}
```

Por lo tanto, es posible sobrescribir o añadir nuevos valores al objeto. Por ejemplo, puede que quiera cambiar el valor predeterminado del efecto o añadir una velocidad personalizada.

Añadir velocidades personalizadas a jQuery.fx.speeds

```
jQuery.fx.speeds.muyRapido = 100;

jQuery.fx.speeds.muyLento = 2000;
```

Realizar una acción cuando un efecto fue ejecutado

A menudo, querrá ejecutar una acción una vez que la animación haya terminado — ya que si ejecuta la acción antes que la animación haya acabado, puede llegar a alterar la calidad del efecto o afectar a los elementos que forman parte de la misma. [Definición: Las funciones de devolución de llamada (en inglés *callback* functions) proveen una forma para ejecutar código una vez que un evento haya terminado.]

En este caso, el evento que responderá a la función será la conclusión de la animación. Dentro de la función de devolución, la palabra clave `this` hace referencia al elemento en donde el efecto fue ejecutado y al igual que sucede con los eventos, es posible transformarlo a un objeto jQuery utilizando `$(this)`.

Ejecutar cierto código cuando una animación haya concluido

```
$('div.old').fadeOut(300, function() { $(this).remove(); });
```

Note que si la selección no retorna ningún elemento, la función nunca se ejecutará. Este problema lo puede resolver comprobando si la selección devuelve algún elemento; y en caso que no lo haga, ejecutar la función de devolución inmediatamente.

Ejecutar una función de devolución incluso si no hay elementos para animar

```
var $thing = $('#nonexistent');



var cb = function() {
```

```
    console.log('realizado');
};


if ($thing.length) {
    $thing.fadeIn(300, cb);
} else {
    cb();
}
```

Efectos personalizados con $.fn.animate

Es posible realizar animaciones en propiedades CSS utilizando el método `$.fn.animate`. Dicho método permite realizar una animación estableciendo valores a propiedades CSS o cambiando sus valores actuales.

Efectos personalizados con $.fn.animate

```
$('div.funtimes').animate(
    {
        left : "+=50",
        opacity : 0.25
    },
    300, // duration

    // función de devolución de llamada
    function() { console.log('realizado');
});
```

NOTA

Las propiedades relacionadas al color no pueden ser animadas utilizando el método `$.fn.animate`, pero es posible hacerlo a través de la extensión color plugin. Más adelante en el libro de discutirá la utilización de extensiones.

Easing

[Definición: El concepto de *Easing* describe la manera en que un efecto ocurre — es decir, si la velocidad durante la animación es constante o no.] jQuery incluye solamente dos métodos de easing: `swing` y `linear`. Si desea transiciones más naturales en las animaciones, existen varias extensiones que lo permiten.

A partir de la versión 1.4 de la biblioteca, es posible establecer el tipo de transición por cada propiedad utilizando el método `$.fn.animate`.

Transición de easing por cada propiedad

```
$('div.funtimes').animate(

    {

        left : [ "+=50", "swing" ],

        opacity : [ 0.25, "linear" ]

    },

    300

);
```

Control de los efectos

jQuery provee varias herramientas para el manejo de animaciones.

- `$.fn.stop` Detiene las animaciones que se están ejecutando en el elemento seleccionado.
- `$.fn.delay` Espera un tiempo determinado antes de ejecutar la próxima animación.

```
$('h1').show(300).delay(1000).hide(300);
```

`jQuery.fx.off` Si el valor es verdadero (`true`), no existirán transiciones para las animaciones; y a los elementos se le establecerá el estado final de la animación. Este método puede ser especialmente útil cuando se esta trabajando con navegadores antiguos.

Conceptos clave AJAX

La utilización correcta de los métodos Ajax requiere primero la comprensión de algunos conceptos clave.

GET vs. POST

Los dos métodos HTTP más comunes para enviar una petición a un servidor son GET y POST. Es importante entender la utilización de cada uno.

El método GET debe ser utilizado para operaciones no-destructivas — es decir, operaciones en donde se esta *obteniendo* datos del servidor, pero no modificando. Por ejemplo, una consulta a un servicio de búsqueda podría ser una petición GET. Por otro lado, las solicitudes GET pueden ser almacenadas en la cache del navegador, pudiendo conducir a un comportamiento impredecible si no se lo espera. Generalmente, la información enviada al servidor, es enviada en una cadena de datos (en inglés *query string*).

El método POST debe ser utilizado para operaciones destructivas — es decir, operaciones en donde se está incorporando información al servidor. Por ejemplo, cuando un usuario guarda un artículo en un blog, esta acción debería utilizar POST. Por otro lado, este tipo de método no se guarda en la cache del navegador. Además, una cadena de consulta puede ser parte de la URL, pero la información tiende a ser enviada de forma separada.

Tipos de datos

Generalmente, jQuery necesita algunas instrucciones sobre el tipo de información que se espera recibir cuando se realiza una petición Ajax. En algunos casos, el tipo de dato es especificado por el nombre del método, pero en otros casos se lo debe detallar como parte de la configuración del método:

- `text` Para el transporte de cadenas de caracteres simples.
- `html` Para el transporte de bloques de código HTML que serán ubicados en la página.
- `script` Para añadir un nuevo script con código JavaScript a la página.
- `json` Para transportar información en formato JSON, el cual puede incluir cadenas de caracteres, arrays y objetos.

Es recomendable utilizar los mecanismos que posea el lenguaje del lado de servidor para la generación de información en JSON.

- `jsonp` Para transportar información JSON de un dominio a otro.
- `xml` Para transportar información en formato XML.

A pesar de los diferentes tipos de datos de que se puede utilizar, es recomendable utilizar el formato JSON, ya que éste es muy flexible, permitiendo por ejemplo, enviar al mismo tiempo información plana y HTML.

Asincronismo

Debido a que, de forma predeterminada, las llamadas Ajax son asíncronas, la respuesta del servidor no esta disponible de forma inmediata. Por ejemplo, el siguiente código no debería funcionar:

```
var response;

$.get('foo.php', function(r) { response = r; });

console.log(response); // indefinido (undefined)
```

En su lugar, es necesario especificar una función de devolución de llamada; dicha función se ejecutará cuando la petición se haya realizado de forma correcta ya que es en ese momento cuando la respuesta del servidor esta lista.

```
$.get('foo.php', function(response) { console.log(response); });
```

Ajax y Firebug

Firebug (o el inspector WebKit que viene incluido en Chrome o Safari) son herramientas imprescindibles para trabajar con peticiones Ajax, ya que es posible observarlas desde la pestaña Consola de Firebug (o yendo a Recursos > Panel XHR desde el inspector de Webkit) y revisar los detalles de dichas peticiones. Si algo esta fallando cuando trabaja con Ajax, este es el primer lugar en donde debe dirigirse para saber cuál es el problema.

JSON I - ¿QUÉ ES Y PARA QUÉ SIRVE JSON?

JSON (*JavaScript Object Notation*) es un formato para el intercambios de datos, básicamente JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos. **JSON** nació como una **alternativa a XML**. Una de las mayores ventajas que tiene el uso de JSON es que puede ser leído por **cualquier lenguaje de programación**. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías.

Veamos un sencillo ejemplo de JSON:

Imaginemos que tenemos una frutería y que queremos obtener el nombre y la cantidad de fruta y verdura que tenemos. En un principio vamos a suponer que tenemos lo siguiente:

- Fruta:

- 10 manzanas
- 20 Peras
- 30 Naranjas

- Verduras

- 80 lechugas
- 15 tomates
- 50 pepinos

Para empezar, nos tenemos que **familiarizar con la sintaxis de Json**:

JSON Nombre/Par de Valores

Para asignar a un nombre un valor debemos usar los dos puntos '**:**' este separador es el equivalente al igual ('=') de cualquier lenguaje.

```
"Nombre" : "Teoria del Big Bang"
```

Valores Json

Los tipos de valores que podemos encontrar en Json son los siguientes:

- Un **número** (entero o float)
- Un **string** (entre comillas simples)
- Un **booleano** (true o false)
- Un **array** (entre corchetes [])
- Un **objeto** (entre llaves {})
- **Null**

Objetos JSON

Los objetos JSON se identifican entre llaves, un objeto puede ser en nuestro caso una fruta o una verdura.

```
{ "NombreFruta":"Manzana" , "Cantidad":20 }
```

Arrays JSON

En un Json puedes incluir **arrays**, para ellos el contenido del array debe ir entre corchetes []:

```
{
"Frutas": [
{ "NombreFruta":"Manzana" , "cantidad":10 },
{ "NombreFruta":"Pera" , "cantidad":20 },
{ "NombreFruta":"Naranja" , "cantidad":30 }
]
}
```

Una vez explicado el funcionamiento de la sintaxis JSON, vamos a aplicar nuestro ejemplo de la frutería.

```
{"Fruteria":
 [
  {"Fruta":
   [
    {"Nombre":"Manzana","Cantidad":10},
    {"Nombre":"Pera","Cantidad":20},
    {"Nombre":"Naranja","Cantidad":30}
   ]
```

```
 },
 {"Verdura":
  [
   {"Nombre":"Lechuga","Cantidad":80},
   {"Nombre":"Tomate","Cantidad":15},
   {"Nombre":"Pepino","Cantidad":50}
  ]
 }
]
}
```

Como podemos observar, hemos creado un objeto llamado *frutería* y, dentro de ese objeto hemos almacenado un **array de dos elementos**. El primer elemento del array contiene un objeto llamado *fruta* y el segundo elemento del array contiene otro objeto llamado *verdura*. Estos objetos a su vez **contienen un array** cuyo contenido es el nombre y la cantidad de cada fruta o verdura.

Imaginemos que nos gustaría saber la cantidad de manzanas que tenemos. El path de este array sería el siguiente:

```
Path: json['Fruteria'][0]['Fruta'][0]['Cantidad']
```

Observamos que la cantidad de manzanas se almacena dentro del primer elemento del array que contiene el objeto *Frutería*, y a su vez dentro del primer elemento del array que contiene el objeto *Fruta*.

Métodos Ajax de jQuery

Como se indicó anteriormente, jQuery posee varios métodos para trabajar con Ajax. Sin embargo, todos están basados en el método `$.ajax`, por lo tanto, su comprensión es obligatoria. A continuación se abarcará dicho método y luego se indicará un breve resumen sobre los demás métodos.

Generalmente, es preferible utilizar el método `$.ajax` en lugar de los otros, ya que ofrece más características y su configuración es muy comprensible.

$.ajax

El método `$.ajax` es configurado a través de un objeto, el cual contiene todas las instrucciones que necesita jQuery para completar la petición. Dicho método es particularmente útil debido a que ofrece la posibilidad de especificar acciones en caso que la petición haya fallado o no. Además, al estar configurado a través de un objeto, es posible definir sus propiedades de forma separada, haciendo que sea más fácil la reutilización del código. Puede visitar api.jquery.com/jQuery.ajax para consultar la documentación sobre las opciones disponibles en el método.

Utilizar el método $.ajax

```
$.ajax({
    // la URL para la petición
    url : 'post.php',

    // la información a enviar
    // (también es posible utilizar una cadena de datos)
    data : { id : 123 },

    // especifica si será una petición POST o GET
    type : 'GET',

    // el tipo de información que se espera de respuesta
    dataType : 'json',

    // código a ejecutar si la petición es satisfactoria;
    // la respuesta es pasada como argumento a la función
    success : function(json) {
        $('<h1/>').text(json.title).appendTo('body');
        $('<div class="content"/>')
            .html(json.html).appendTo('body');
    },

    // código a ejecutar si la petición falla;
    // son pasados como argumentos a la función
    // el objeto de la petición en crudo y código de estatus de la petición
    error : function(xhr, status) {
        alert('Disculpe, existió un problema');
    },
```

```
    // código a ejecutar sin importar si la petición falló o no

    complete : function(xhr, status) {

        alert('Petición realizada');

    }

});
```

NOTA

Una aclaración sobre el parámetro `dataType`: Si el servidor devuelve información que es diferente al formato especificado, el código fallará, y la razón de porque lo hace no siempre quedará clara debido a que la respuesta HTTP no mostrará ningún tipo de error. Cuando esté trabajando con peticiones Ajax, debe estar seguro que el servidor esta enviando el tipo de información que esta solicitando y verifique que la cabecera `Content-type` es exacta al tipo de dato. Por ejemplo, para información en formato JSON, la cabecera Content-type debería ser `application/json`.

Opciones del método $.ajax

El método `$.ajax` posee muchas opciones de configuración, y es justamente esta característica la que hace que sea un método muy útil. Para una lista completa de las opciones disponibles, puede consultar api.jquery.com/jQuery.ajax; a continuación se muestran las más comunes:

`async`

Establece si la petición será asíncrona o no. De forma predeterminada el valor es true. Debe tener en cuenta que si la opción se establece en `false`, la petición bloqueará la ejecución de otros códigos hasta que dicha petición haya finalizado.

`cache`

Establece si la petición será guardada en la cache del navegador. De forma predeterminada es true para todos los `dataType` excepto para `script` y `jsonp`. Cuando posee el valor false, se agrega una cadena de caracteres anti-cache al final de la URL de la petición.

`complete`

Establece una función de devolución de llamada que se ejecuta cuando la petición esta completa, aunque haya fallado o no. La función recibe como argumentos el objeto de la petición en crudo y el código de estatus de la misma petición.

`context`

Establece el alcance en que la/las funciones de devolución de llamada se ejecutaran (por ejemplo, define el significado de this dentro de las funciones). De manera predeterminada `this` hace referencia al objeto originalmente pasado al método `$.ajax`.

`data`

Establece la información que se enviará al servidor. Esta puede ser tanto un objeto como una cadena de datos (por ejemplo `foo=bar&baz=bim`)

`dataType`

Establece el tipo de información que se espera recibir como respuesta del servidor. Si no se especifica ningún valor, de forma predeterminada, jQuery revisa el tipo de MIME que posee la respuesta.

`error`

Establece una función de devolución de llamada a ejecutar si resulta algún error en la petición. Dicha función recibe como argumentos el objeto de la petición en crudo y el código de estatus de la misma petición.

`jsonp`

Establece el nombre de la función de devolución de llamada a enviar cuando se realiza una petición JSONP. De forma predeterminada el nombre es *callback*

`success`

Establece una función a ejecutar si la petición a sido satisfactoria. Dicha función recibe como argumentos la información de la petición (convertida a objeto JavaScript en el caso que `dataType` sea JSON), el estatus de la misma y el objeto de la petición en crudo.

`timeout`

Establece un tiempo en milisegundos para considerar a una petición como fallada. traditional Si su valor es `true`, se utiliza el estilo de serialización de datos utilizado antes de jQuery 1.4. Para más detalles puede visitar api.jquery.com/jQuery.param.

`type`

De forma predeterminada su valor es `GET`. Otros tipos de peticiones también pueden ser utilizadas (como `PUT` y `DELETE`), sin embargo pueden no estar soportados por todos los navegadores.

`url`

Establece la URL en donde se realiza la petición. La opción `url` es obligatoria para el método `$.ajax`;

Métodos convenientes

En caso que no quiera utilizar el método `$.ajax`, y no necesite los controladores de errores, existen otros métodos más convenientes para realizar peticiones Ajax (aunque, como se indicó antes, estos están basados el método `$.ajax` con valores pre-establecidos de configuración).

Los métodos que provee la biblioteca son:

- `$.get` Realiza una petición GET a una URL provista.
- `$.post` Realiza una petición POST a una URL provista.
- `$.getScript` Añade un script a la página.
- `$.getJSON` Realiza una petición GET a una URL provista y espera que un dato JSON sea devuelto.

Los métodos deben tener los siguientes argumentos, en orden:

- `url` La URL en donde se realizará la petición. Su valor es obligatorio.
- `data` La información que se enviará al servidor. Su valor es opcional y puede ser tanto un objeto como una cadena de datos (como `foo=bar&baz=bim`).Nota: esta opción no es valida para el método `$.getScript`.
- `success callback` Una función opcional que se ejecuta en caso que petición haya sido satisfactoria. Dicha función recibe como argumentos la información de la petición y el objeto en bruto de dicha petición.
- `data type` El tipo de dato que se espera recibir desde el servidor. Su valor es opcional. Nota: esta opción es solo aplicable para métodos en que no está especificado el tipo de dato en el nombre del mismo método.

Utilizar métodos convenientes para peticiones Ajax

```
// obtiene texto plano o html

$.get('/users.php', { userId : 1234 }, function(resp) {

    console.log(resp);

});


// añade un script a la página y luego ejecuta la función especificada

$.getScript('/static/js/myScript.js', function() {

    functionFromMyScript();

});


// obtiene información en formato JSON desde el servidor
```

```
$.getJSON('/details.php', function(resp) {

    $.each(resp, function(k, v) {

        console.log(k + ' : ' + v);

}); });
```

$.fn.load

El método `$.fn.load` es el único que se puede llamar desde una selección. Dicho método obtiene el código HTML de una URL y rellena a los elementos seleccionados con la información obtenida. En conjunto con la URL, es posible especificar opcionalmente un selector, el cual obtendrá el código especificado en dicha selección.

Utilizar el método $.fn.load para rellenar un elemento

```
$('#newContent').load('/foo.html');
```

Utilizar el método $.fn.load para rellenar un elemento basado en un selector

```
$('#newContent').load('/foo.html #myDiv h1:first', function(html) {

  alert('Contenido actualizado');

});
```

Ajax y formularios

Las capacidades de jQuery con Ajax pueden ser especialmente útiles para el trabajo con formularios. Por ejemplo, la extensión jQuery Form Plugin es una extensión para añadir capacidades Ajax a formularios. Existen dos métodos que debe conocer para cuando este realizando este tipo de trabajos: `$.fn.serialize` y `$.fn.serializeArray`.

Transformar información de un formulario a una cadena de datos

```
$('#myForm').serialize();
```

Crear un array de objetos conteniendo información de un formulario

```
$('#myForm').serializeArray();



// crea una estructura como esta:

[

    { name : 'field1', value : 123 },

    { name : 'field2', value : 'hello world' }

]
```

Eventos Ajax

A menudo, querrá ejecutar una función cuando una petición haya comenzado o terminado, como por ejemplo, mostrar o ocultar un indicador. En lugar de definir estas funciones dentro de cada petición, jQuery provee la posibilidad de vincular eventos Ajax a elementos seleccionados. Para una lista completa de eventos Ajax, puede consultar docs.jquery.com/Ajax_Events.

Mostrar/Ocultar un indicador utilizando Eventos Ajax

```
$('#loading_indicator')

    .ajaxStart(function() { $(this).show(); })

    .ajaxStop(function() { $(this).hide(); });
```