Fonaments de jQuery

Rebecca Murphey

PID_00235944

Temps de lectura i comprensió: **11 hores**

UOC
Universitat Oberta de Catalunya

Índex

1. Benvingut

jQuery s'està convertint ràpidament en una eina que tot desenvolupador d'interfícies web hauria de conèixer. El propòsit d'aquest mòdul és fer un resum de la biblioteca, de tal manera que, quan s'hagi acabat de llegir, s'estigui capacitat per fer tasques bàsiques utilitzant jQuery i es tingui una base sòlida per a continuar l'aprenentatge.

La modalitat de treball és la següent: en primer lloc, es dedicarà temps a comprendre un concepte per tal de fer després un exercici que hi estigui relacionat. Alguns dels exercicis poden arribar a ser trivials, mentre que altres no ho són tant. L'objectiu és aprendre a resoldre de manera fàcil el que normalment es resoldria amb jQuery. Les solucions a tots els exercicis estan incloses en el mateix material d'aprenentatge.

1.1. Obtenir el material d'aprenentatge

El material d'aprenentatge i el codi font dels exemples que s'utilitzen en el mòdul estan allotjats en un repositori de Github. Des d'allà es pot descarregar un arxiu .zip o .tar amb el codi per a utilitzar en un servidor web.

Git

Si soleu utilitzar Git, és aconsellable clonar o modificar el repositori.

1.2. Programari

Per a treballar amb els continguts del mòdul, faran falta les eines següents:

- navegador web Firefox;
- l'extensió Firebug, per Firefox;
- un editor de textos plans (com Notepad++/Sublime Text 2 para Windows, gedit /Kate para Linux o TextMate per a Mac OS X);
- per a les seccions dedicades a Ajax: un servidor local (com WAMP o MAMP) o un client FTP/SSH (com FileZilla) per a accedir a un servidor remot.

1.3. Afegir JavaScript a una pàgina

Hi ha dues maneres d'inserir codi JavaScript dins d'una pàgina: escrivint codi en la pàgina (en anglès, *inline*) o per mitjà d'un arxiu extern utilitzant l'etiqueta *script*. L'ordre en què s'inclou el codi és important: un codi que depèn d'un altre s'ha d'incloure després del que referència. Per exemple, si la funció B depèn de A, l'ordre ha de ser A, B i no B, A.

Per a millorar el rendiment de la pàgina, el codi JavaScript ha de ser inclòs al final de l'HTML. A més, quan es treballa en un ambient de producció amb múltiples arxius JavaScript, aquests han de ser combinats en un sol arxiu.

Exemple de codi JavaScript en línia

```
<script>
console.log('hello');
</script>
```

Exemple d'inclusió d'un arxiu extern JavaScript

```
<script src='/js/jquery.js'></script>
```

1.4. Depuració del codi JavaScript

La utilització d'una eina de depuració és essencial per a treballar amb Java-Script. Firefox proveeix d'un depurador a través de l'extensió Firebug, mentre que Safari i Chrome ja en porten un d'integrat.

Cada depurador ofereix:

- un editor multilínia per a experimentar amb JavaScript;
- un inspector per a revisar el codi generat a la pàgina;
- un visualitzador de xarxa o recursos, per a examinar les peticions que s'efectuen.

Quan s'està escrivint codi JavaScript, es pot utilitzar algun dels mètodes següents per a enviar missatges a la consola del depurador:

- `console.log()` per a enviar i registrar missatges generals;
- `console.dir()` per a registrar un objecte i visualitzar-ne les propietats;
- `console.warn()` per a registrar missatges d'alerta;
- `console.error()` per a registrar missatges d'error.

Hi ha altres mètodes per a utilitzar des de la consola, però poden variar segons el navegador. La consola, a més, proporciona la possibilitat d'establir punts d'interrupció i observar expressions en el codi amb la finalitat de facilitar-ne la depuració.

1.5. Exercicis

La majoria dels apartats conclouen amb un o més exercicis. En alguns, es podrà treballar directament amb Firebug; en altres, s'haurà d'escriure codi JavaScript després d'incloure la biblioteca jQuery en el document.

Tot i així, per a completar determinats exercicis, caldrà consultar la documentació oficial de jQuery. Aprendre a trobar respostes és una part important del procés d'aprenentatge.

Aquests són alguns suggeriments per a fer front als problemes:

- En primer lloc, cal assegurar-se que s'entén bé el problema que s'està tractant de resoldre.

- Després, cal esbrinar a quins elements s'haurà d'accedir amb la finalitat de resoldre el problema, i determinar com accedir-hi. Es pot utilitzar Firebug per a verificar que s'està obtenint el resultat esperat.

- Finalment, cal esbrinar què cal fer amb aquests elements per a resoldre el problema. Pot ser útil, abans de començar, escriure comentaris en què s'expliqui el que es farà.

No tingueu por de cometre errors. No intenteu tampoc escriure el vostre codi de manera perfecta al primer intent. Cometre errors i experimentar amb solucions és part del procés d'aprenentatge i us ajudarà a ser un millor desenvolupador.

Vegeu també

En la carpeta `/exercicis/solucions` podeu trobar exemples de solucions als exercicis del mòdul.

1.6. Convencions utilitzades en el mòdul

Hi ha una sèrie de convencions utilitzades en el mòdul.

Els mètodes que poden ser cridats des de l'objecte jQuery es referenciaran com a `$.fn.nomDelMetode`. Els mètodes que hi ha en l'espai de noms (en anglès, *namespace*) de jQuery, però que no poden ser cridats des de l'objecte jQuery, seran referenciats com a `$.nomDelMetode`. Si no ho acabeu d'entendre, no us amoïneu, a mesura que aneu progressant en el mòdul us resultarà més clar.

1.7. Notes de la traducció

- Com que el material té com a finalitat l'aprenentatge i l'ensenyament, està traduït al català.

- Molts conceptes tècnics s'anomenen en la versió traduïda al català. No obstant això, per a tenir la referència original, també s'explica com es diu en anglès.

- Els exemples d'exercicis i les seves solucions no estan completament traduïts. Això és a causa del fet que, quan estigueu treballant en un projecte real, el codi que trobareu en altres llocs probablement estigui en anglès. Tot i així, s'han traduït els comentaris incorporats en els codis d'exemples i alguns textos particulars per a facilitar-ne la comprensió.

1.8. Material de referència

Hi ha una gran quantitat d'articles que s'ocupen d'algun aspecte de jQuery. Alguns són excel·lents, però altres, francament, són erronis. Quan llegiu un article sobre jQuery, cal que us assegureu que inclou la mateixa versió de la biblioteca que esteu utilitzant, i resistiu la temptació de copiar i enganxar el codi (preneu-vos un temps per a poder entendre'l).

A continuació, trobareu una llista d'una sèrie de recursos excel·lents per a utilitzar durant l'aprenentatge. El més important de tots és el codi font de jQuery, que conté (en el format sense comprimir) una documentació completa a través de comentaris. La biblioteca no és una caixa negra –la seva comprensió creixerà exponencialment si la reviseu de tant en tant– i és molt recomanable que la deseu en els favorits del vostre navegador per tal de tenir-la com a guia de referència.

- El codi font de jQuery
- Documentació de jQuery
- Fòrum de jQuery
- Favorits a Delicious
- Canal IRC #jquery a Freenode

2. Conceptes bàsics de JavaScript

2.1. Introducció

jQuery està escrit en JavaScript, un llenguatge de programació molt ric i expressiu.

Aquest apartat està orientat a persones sense experiència en el llenguatge, i inclou conceptes bàsics i problemes freqüents que es poden presentar en treballar-hi. D'altra banda, la matèria pot ser beneficiosa per als qui utilitzin altres llenguatges de programació per a entendre les peculiaritats de JavaScript.

Lectura recomanada

Si esteu interessats a aprendre el llenguatge més en profunditat, podeu llegir el llibre *JavaScript: The Good Parts*, escrit per Douglas Crockford.

2.2. Sintaxi bàsica

Comprensió de declaracions, noms de variables, espais en blanc i altres sintaxis bàsiques de JavaScript.

Declaració simple de variable

```
var foo = 'hola món';
```

Els espais en blanc no tenen valor fora de les cometes

```
var foo = 'hola món';
```

Els parèntesis indiquen prioritat

```
2 * 3 + 5;// és igual a 11, la multiplicació té lloc primer
2 * (3 + 5);  // és igual a 16, pels parèntesis, la suma té lloc primer
```

La tabulació millora la lectura del codi, però no té cap significat especial

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

2.3. Operadors

2.3.1. Operadors bàsics

Els operadors bàsics permeten manipular valors.

Concatenació

```
var foo = 'hola';
var bar = 'món';

console.log(foo + ' ' + bar); // la consola de depuració mostra 'hola món'
```

Multiplicació i divisió

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

Incrementació i decrementació

```
var i = 1;

var j = ++i;  // incrementació prèvia:  j és igual a 2; i és igual a 2
var k = i++;  // incrementació posterior: k és igual a 2; i és igual a 3
```

2.3.2. Operacions amb nombres i cadenes de caràcters

En JavaScript, les operacions amb nombres i cadenes de caràcters (en anglès, *strings*) poden ocasionar resultats no esperats.

Suma enfront de concatenació.

```
var foo = 1;
var bar = '2';

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

Forçar una cadena de caràcters a actuar com un nombre

```
var foo = 1;
var bar = '2';

// el constructor 'Number' obliga a la cadena a comportar-se com un nombre
console.log(foo + Number(bar));  // la consola de depuració mostra 3
```

El constructor *Number*, quan és cridat com una funció (com es mostra en l'exemple), obliga el seu argument a comportar-se com un nombre. També es pot utilitzar l'operador de *suma unària*, que dóna el mateix resultat:

Forçar una cadena de caràcters a actuar com un nombre (utilitzant l'operador de suma unària)

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

2.3.3. Operadors lògics

Els operadors lògics permeten avaluar una sèrie d'operands utilitzant operacions AND i OR.

Operadors lògics AND i OR

```
var foo = 1;
var bar = 0;
var baz = 2;

foo || bar;   // retorna 1, el qual és veritable (true)
bar || foo;   // retorna 1, el qual és veritable (true)

foo && bar;   // retorna 0, el qual és fals (false)
foo && baz;   // retorna 2, el qual és veritable (true)
baz && foo;   // retorna 1, el qual és veritable (true)
```

L'operador `||` (OR lògic) retorna el valor del primer operand, si aquest és veritable; en cas contrari, retorna el segon operand. Si tots dos operands són falsos, retorna fals (*false*). L'operador `&&` (AND lògic) retorna el valor del primer operand si aquest és fals; en cas contrari, retorna el segon operand. Quan tots dos valors són veritables, retorna veritable (*true*); sinó, retorna fals.

Vegeu també

Es pot consultar la secció `Elements veritables i falsos` per a obtenir més detalls sobre quins valors s'avaluen com a `true` i quins s'avaluen com a `false`.

Nota

Pot ser que de vegades observeu que alguns desenvolupadors utilitzen aquesta lògica en fluxos de control en lloc d'utilitzar la declaració `if`. Per exemple:

```
// fer alguna cosa amb foo si foo és veritable
foo && doSomething(foo);

// establir bar igual a baz si baz és veritable;
// cas contrari, establir a bar igual al
// valor de createBar()
var bar = baz || createBar();
```

Aquest estil de declaració és molt elegant i concís, però pot ser difícil per a llegir (sobretot per a principiants). Per això s'explicita, per a reconèixer-lo, quan s'estigui llegint codi. No obstant això, la seva utilització no és recomanable, llevat que s'estigui còmode amb el concepte i el seu comportament.

2.3.4. Operadors de comparació

Els operadors de comparació permeten comprovar si determinats valors són equivalents o idèntics.

Operadors de comparació

```
var foo = 1;
var bar = 0;
var baz = '1';
var bim = 2;

foo == bar;   // retorna fals (false)
foo != bar;   // retorna veritable (true)
foo == baz;   // retorna veritable (true); vagi amb compte

foo === baz; // retorna fals (false)
foo !== baz; // retorna veritable (true)
foo === parseInt(baz);   // retorna veritable (true)

foo > bim;// retorna fals (false)
bim > baz;// retorna veritable (true)
foo <= baz;   // retorna veritable (true)
```

2.4. Codi condicional

De vegades, es vol executar un bloc de codi sota certes condicions. Les estructures de control de flux, per mitjà de la utilització de les declaracions `if` i `else`, permeten fer-ho.

Nota

Quan s'escriu una declaració `if`, les claus no són estrictament necessàries. Tanmateix, és recomanable la seva utilització, ja que fa que el codi sigui molt més llegible.

Control del flux

```
var foo = true;
var bar = false;

if (bar) {
   // aquest codi no s'executarà mai
   console.log('hola!');
}

if (bar) {
   // aquest codi no s'executarà
} else {
   if (foo) {
      // aquest codi s'executarà
   } else {
      // aquest codi s'executarà si foo i bar són falsos (false)
   }
}
```

Cal tenir en compte no definir funcions amb el mateix nom múltiples vegades dins de declaracions `if`/`else`, ja que es poden obtenir resultats no esperats.

2.4.1. Elements veritables i falsos

Per a controlar adequadament el flux, és important entendre quins tipus de valors són «veritables» i quins «falsos». De vegades, alguns valors poden semblar una cosa, però al final acaben sent-ne una altra.

Valors que retornen `veritable (true)`

```
'0';
'any string'; // qualsevol cadena
[];  // un vector buit
{};  // un objecte buit
1;   // qualsevol nombre diferent a zero
```

Valors que retornen `fals (false)`

```
0;
'';  // una cadena buida
NaN; // la variable Javascript "not-a-number" (No és un nombre)
null; // un valor nul
undefined;  // cal anar amb compte -- indefinit (undefined) pot ser redefinit
```

2.4.2. Variables condicionals utilitzant l'operador ternari

De vegades, es vol establir el valor d'una variable depenent d'una certa condició. Per a fer-ho, es pot utilitzar una declaració `if`/`else`; no obstant això, en molts casos és més convenient utilitzar l'operador ternari.

L'operador ternari avalua una condició; si la condició és veritable, retorna cert valor, i en cas contrari retorna un valor diferent.

L'operador ternari

```
// establir a foo igual a 1 si bar és veritable;
// cas contrari, establir a foo igual a 0
var foo = bar ? 1 : 0;
```

L'operador ternari es pot utilitzar sense retornar un valor a la variable, encara que aquest ús generalment és desaprovat.

2.4.3. Declaració *switch*

En lloc d'utilitzar una sèrie de declaracions *if/else/else if/else*, de vegades pot ser útil la utilització de la declaració `switch`.

La declaració `switch` avalua el valor d'una variable o expressió, i executa diferents blocs de codi depenent d'aquest valor.

Una declaració *switch*

```
switch (foo) {

   case 'bar':
      alert('el valor és bar');
   break;

   case 'baz':
      alert('el valor és baz');
   break;

   default:
      alert('de forma predeterminada s'executarà aquest codi');
   break;

}
```

Les declaracions `switch` són poc utilitzades en JavaScript, ja que es pot obtenir el mateix comportament creant un objecte, que té més potencial perquè es pot reutilitzar, usar per a fer proves, etc. Per exemple:

```
var stuffToDo = {
   'bar' : function() {
      alert('el valor és bar');
   },

   'baz' : function() {
      alert('el valor és baz');
   },

   'default' : function() {
      alert('de forma predeterminada s'executarà aquest codi');
   }
};

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

Més endavant, es tractarà el concepte d'objectes.

2.5. Bucles

Els bucles (en anglès, *loops*) permeten executar un bloc de codi un determinat nombre de vegades.

Bucles

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

Observeu que en l'exemple s'utilitza la paraula `var` abans de la variable *i*; això fa que aquesta variable quedi dins de l'«àmbit» (en anglès, *scope*) del bucle.

Vegeu també

Més endavant, en aquest mòdul, s'examinarà en profunditat el concepte d'àmbit.

2.5.1. Bucles utilitzant *for*

Un bucle utilitzant `for` es compon de quatre estats i té l'estructura següent:

```
for ([expressióInicial]; [condició]; [incrementDeLaExpressió])
   [cos]
```

L'estat *expressióInicial* és executat una sola vegada, abans que el bucle comenci. Aquest atorga l'oportunitat de preparar o declarar variables.

L'estat *condició* és executat abans de cada repetició, i retorna un valor que decideix si el bucle s'ha de continuar executant o no. Si l'estat condicional avalua un valor fals, el bucle s'atura.

L'estat *incrementDeLaExpressió* és executat al final de cada repetició i atorga l'oportunitat de canviar l'estat d'importants variables. En general, aquest estat implica la incrementació o decrementació d'un comptador.

El *cos* és el codi a executar en cada repetició del bucle.

Un típic bucle utilitzant `for`

```
for (var i = 0, limit = 100; i < limit; i++) {
   // Aquest bloc de codi serà executat 100 vegades
   console.log('Actualment en ' + i);
   // Nota: l'últim registre que es mostrarà
   // en la consola serà "Actualment en 99"
}
```

2.5.2. Bucles utilitzant *while*

Un bucle utilitzant `while` és similar a una declaració condicional `if`, excepte que el cos es continuarà executant fins que la condició a avaluar sigui falsa.

```
while ([condició]) [cos]
```

Un típic bucle utilitzant `while`

```
var i = 0;
while (i < 100) {
   // Aquest bloc de codi s'executarà 100 vegades
   console.log('Actualment en ' + i);
   i++; // incrementa la variable i
}
```

Observeu que en l'exemple s'incrementa el comptador dins el cos del bucle, però també es poden combinar la condició i la incrementació, com es mostra a continuació:

Bucle utilitzant `while` amb la combinació de la condició i la incrementació

```
var i = -1;
while (++i < 100) {
   // Aquest bloc de codi s'executarà 100 vegades
   console.log('Actualment en ' + i);
}
```

Es comença en `-1` i després s'utilitza la incrementació prèvia (`++i`).

2.5.3. Bucles utilitzant *do-while*

Aquest bucle és exactament igual que el bucle utilitzant `while`, excepte que el cos és executat almenys una vegada abans que la condició sigui avaluada.

```
do [cos] while ([condició])
```

Un bucle utilitzant `do-while`

```
do {

   // Fins i tot quan la condició sigui falsa
   // el cos del bucle s'executarà almenys una vegada.

   alert('Hola');

```

```
} while (false);
```

Aquest tipus de bucles són força atípics, ja que poques vegades cal un bucle que s'executi almenys una vegada. En tot cas, cal estar-ne al cas.

2.5.4. *Break* i *continue*

Usualment, la fi de l'execució d'un bucle resultarà quan la condició no continuï avaluant un valor veritable; no obstant això, també es pot aturar un bucle utilitzant la declaració `break` dins del cos.

Aturar un bucle amb break

```
for (var i = 0; i < 10; i++) {
   if (something) {
      break;
   }
}
```

També pot passar que es vulgui continuar amb el bucle sense haver d'executar més sentències del cos del mateix bucle. Això es pot fer utilitzant la declaració `continue`.

Saltar a la següent iteració d'un bucle

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

   if (something) {
      continue;
   }

   // La següent declaració serà executada
   // si la condició 'something' no es compleix
   console.log('Hola');

}
```

2.6. Paraules reservades

JavaScript posseeix un nombre de «paraules reservades» o paraules que són especials dins del mateix llenguatge. Aquestes paraules s'han de fer servir quan es necessitin per al seu ús específic:

- `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`

2.7. Vectors

Els vectors (en català també anomenats matrius, i en anglès, *arrays*) són llistes de valors amb índex zero (en anglès, *zero-index*), és a dir, que el primer element del vector està en l'índex 0. Són una manera pràctica d'emmagatzemar un conjunt de dades relacionades (com ara cadenes de caràcters), encara que en realitat un vector pot incloure múltiples tipus de dades, fins i tot altres vectors.

Un vector simple

```
var myArray = [ 'hola', 'món' ];
```

Accedir als ítems del vector a través del seu índex

```
var myArray = [ 'hola', 'món', 'foo', 'bar' ];
console.log(myArray[3]);   // mostra en la consola 'bar'
```

Obtenir la quantitat d'ítems del vector

```
var myArray = [ 'hola', 'món' ];
console.log(myArray.length);   // mostra en la consola 2
```

Canviar el valor d'un ítem d'un vector

```
var myArray = [ 'hola', 'món' ];
myArray[1] = 'changed';
```

Com es mostra en l'exemple «Canviar el valor d'un ítem d'un vector», es pot canviar el valor d'un ítem d'un vector, encara que en general no és aconsellable.

Afegir elements a un vector

```
var myArray = [ 'hola', 'món' ];
myArray.push('new');
```

Treballar amb vectors

```
var myArray = [ 'h', 'o', 'l', 'a' ];
var myString = myArray.join('');   // 'hola'
```

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

2.8. Objectes

Els objectes són elements que poden contenir zero o més conjunts de parells de noms clau i valors associats a aquest objecte. Els noms clau poden ser qualsevol paraula o nombre vàlid. El valor pot ser qualsevol tipus de valor: un nombre, una cadena, un vector, una funció, fins i tot un altre objecte.

Quan un dels valors d'un objecte és una funció, aquesta és anomenada com un *mètode* de l'objecte. En cas contrari, s'anomenen *propietats*.

Curiosament, en JavaScript, gairebé tot és un objecte –vectors, funcions, nombres, fins i tot cadenes– i tots tenen propietats i mètodes.

Creació d'un «objecte literal»

```
var myObject = {
   sayHello: function() {
      console.log('hola');
   },

   myName: 'Rebecca'
};

myObject.sayHello();   // es crida el mètode sayHello,
         // el qual mostra en la consola 'hola'

console.log(myObject.myName);// es crida la propiedad myName,
         // la qual mostra en la consola 'Rebecca'
```

Nota

Quan es creen objectes literals, el nom de la propietat pot ser qualsevol identificador JavaScript, una cadena de caràcters (tancada entre cometes) o un nombre:

```
var myObject = {
   validIdentifier: 123,
   'some string': 456,
   99999: 789
};
```

Lectura recomanada

Els objectes literals poden ser molt útils per a l'organització del codi. Per a més informació, podeu llegir l'article (en anglès) «Using objects to organize your code» de Rebecca Murphey.

2.9. Funcions

Les funcions contenen blocs de codi que s'executaran repetidament. A aquestes se'ls poden passar arguments i, opcionalment, la funció pot retornar un valor.

Les funcions es poden crear de diverses maneres:

Declaració d'una funció

```
function foo() { /* fer alguna cosa */ }
```

Declaració d'una funció anomenada

```
var foo = function() { /* fer alguna cosa */ }
```

És preferible el mètode de funció anomenada a causa d'algunes profundes raons tècniques. Igualment, és probable trobar els dos mètodes quan es revisi codi JavaScript.

2.9.1. Utilització de funcions

Una funció simple

```
var greet = function(person, greeting) {
   var text = greeting + ', ' + person;
   console.log(text);
};



greet('Rebecca', 'Hola');  // mostra en la consola 'Hola, Rebecca'
```

Una funció que retorna un valor

```
var greet = function(person, greeting) {
   var text = greeting + ', ' + person;
   return text;
};

console.log(greet('Rebecca','Hola'));   // la funció retorna 'Hola, Rebecca',
                                        // la qual es mostra en la consola
```

Una funció que retorna una altra funció

```
var greet = function(person, greeting) {
   var text = greeting + ', ' + person;
   return function() { console.log(text); };
```

```
};



var greeting = greet('Rebecca', 'Hola');
greeting();  // es mostra en la consola 'Hola, Rebecca'
```

2.9.2. Funcions anònimes autoexecutables

Un patró comú en JavaScript són les funcions anònimes autoexecutables. Aquest patró consisteix a crear una expressió de funció i immediatament executar-la. És molt útil per a casos en què no es vol intervenir espais de noms globals, a causa del fet que cap variable declarada dins de la funció és visible des de fora.

Funció anònima autoexecutable

```
(function(){
   var foo = 'Hola món';
})();



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

2.9.3. Funcions com a arguments

En JavaScript, les funcions són «ciutadans de primera classe» (poden ser assignades a variables o passades a altres funcions com a arguments). En jQuery, passar funcions com a arguments és una pràctica molt comuna.

Passar una funció anònima com un argument

```
var myFn = function(fn) {
   var result = fn();
   console.log(result);
};

myFn(function() { return 'hola món'; });   // mostra en la consola 'hola món'
```

Passar una funció anomenada com un argument

```
var myFn = function(fn) {
   var result = fn();
   console.log(result);
};

var myOtherFn = function() {
      return 'hola món';
```

```
};

myFn(myOtherFn);   // mostra en la consola 'hola món'
```

2.10. Determinació del tipus de variable

JavaScript ofereix una manera de poder comprovar el «tipus» (en anglès, *type*) d'una variable. No obstant això, el resultat pot ser confús; per exemple, el tipus d'un vector és «object».

Per això, és una pràctica comuna utilitzar l'operador `typeof` quan es tracta de determinar el tipus d'un valor específic.

Determinar el tipus en diferents variables

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

var myObject = {
   foo : 'bar'
};

var myArray = [ 'a', 'b', 'c' ];

var myString = 'hola';

var myNumber = 3;

typeof myFunction;   // retorna 'function'
typeof myObject; // retorna 'object'
typeof myArray;  // retorna 'object' -- cal anar amb compte
typeof myString; // retorna 'string'
typeof myNumber; // retorna 'number'

typeof null; // retorna 'object' -- cal anar amb compte



if (myArray.push && myArray.slice && myArray.join) {
   // probablement sigui un vector
   // (aquest estil s'anomena, en anglès, "duck typing")
}

if (Object.prototype.toString.call(myArray) === '[object Array]') {
   // definitivament és un vector;
   // aquesta és considerada la forma més robusta
   // de determinar si un valor és un vector.
```

```
}
```

jQuery ofereix mètodes per a ajudar a determinar el tipus d'un determinat valor. Aquests mètodes es veuran més endavant.

2.11. La paraula clau *this*

En JavaScript, com en la majoria dels llenguatges de programació orientats a objectes, `this` és una paraula clau especial que fa referència a l'objecte on el mètode està sent invocat. El valor de `this` és determinat utilitzant una sèrie de simples passos:

- Si la funció és invocada utilitzant Function.call o Function.apply, `this` tindrà el valor del primer argument passat al mètode. Si l'argument és nul (*null*) o indefinit (*undefined*), `this` farà referència a l'objecte global (l'objecte `window`).

- Si la funció que es vol invocar és creada utilitzant Function.bind, `this` serà el primer argument que és passat a la funció en el moment en què es crea.

- Si la funció és invocada com un mètode d'un objecte, `this` referenciarà aquest objecte.

- En cas contrari, si la funció és invocada com una funció independent, no unida a algun objecte, `this` referenciarà l'objecte global.

Una funció invocada utilitzant *Function.call*

```
var myObject = {
   sayHello : function() {
      console.log('Hola, el meu nom és ' + this.myName);
   },

   myName : 'Rebecca'
};

var secondObject = {
   myName : 'Colin'
};

myObject.sayHello();  // registra 'Hola, el meu nom és Rebecca'
myObject.sayHello.call(secondObject); // registra 'Hola, el meu nom és Colin'
```

Una funció creada utilitzant Function.bind

```
var myName = 'l'objecte global',
```

```
   sayHello = function () {
      console.log('Hola, el meu nom és ' + this.myName);
   },

   myObject = {
      myName : 'Rebecca'
   };

var myObjectHello = sayHello.bind(myObject);

sayHello();   // registra 'Hola, el meu nom és l'objecte global'
myObjectHello();  // registra 'Hola, el meu nom és Rebecca'
```

Una funció vinculada a un objecte

```
var myName = 'l'objecte global',

   sayHello = function() {
      console.log('Hola, el meu nom és ' + this.myName);
   },

   myObject = {
      myName : 'Rebecca'
   },

   secondObject = {
   myName : 'Colin'
   };

myObject.sayHello = sayHello;
secondObject.sayHello = sayHello;

sayHello();   // registra 'Hola, el meu nom és l'objecte global'
myObject.sayHello();  // registra 'Hola, el meu nom és Rebecca'
secondObject.sayHello();  // registra 'Hola, el meu nom és Colin'
```

Nota

En algunes oportunitats, quan s'invoca una funció que és dins d'un espai de noms (en anglès, *namespace*) ampli, pot ser una temptació desar la referència a la funció actual en una variable més curta i accessible. No obstant això, és important no fer-ho en instàncies de mètodes, ja que pot portar a l'execució de codi incorrecte. Per exemple:

```
var myNamespace = {
   myObject: {
      sayHello: function() {
         console.log('Hola, el meu nom és ' + this.myName);
      },

      myName: 'Rebecca'
   }
};
```

```
var hello = myNamespace.myObject.sayHello;

hello();  // registra 'Hola, el meu nom és undefined'
```

Perquè no es donin aquests errors, cal fer referència a l'objecte on el mètode és invocat:

```
var myNamespace = {
   myObject : {
      sayHello : function() {
         console.log('Hola, el meu nom és ' + this.myName);
      },

      myName : 'Rebecca'
   }
};

var obj = myNamespace.myObject;

obj.sayHello();  // registra 'Hola, el meu nom és Rebecca'
```

2.12. Àmbit

L'«àmbit» (en anglès, *scope*) es refereix a les variables que estan disponibles en un bloc de codi en un temps determinat. La falta de comprensió d'aquest concepte pot portar a una frustrant experiència de depuració.

Quan una variable és declarada dins d'una funció utilitzant la paraula clau `var`, aquesta únicament està disponible per al codi dins de la funció: tot el codi fora d'aquesta funció no pot accedir a la variable. D'altra banda, les funcions definides *dins de la* funció *podran* accedir a la variable declarada.

Les variables que són declarades dins de la funció sense la paraula clau `var` no queden dins de l'àmbit de la mateixa funció; JavaScript buscarà el lloc on la variable va ser prèviament declarada, i en cas de no haver estat declarada, és definida dins de l'àmbit global, la qual cosa pot ocasionar conseqüències inesperades.

Les funcions tenen accés a variables definides dins del mateix àmbit

```
var foo = 'hola';

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

sayHello(); // mostra en la consola 'hola'
console.log(foo);   // també mostra en la consola 'hola'
```

El codi de fora no té accés a la variable definida dins de la funció

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

sayHello(); // mostra en la consola 'hola'
console.log(foo);   // no mostra res en la consola
```

Variables amb noms iguals però valors diferents poden existir en diferents àmbits

```
var foo = 'mundo';

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

sayHello(); // mostra en la consola 'hola'
console.log(foo);   // mostra en la consola 'món'
```

Les funcions poden «veure» els canvis en les variables abans que la funció sigui definida

```
var myFunction = function() {
   var foo = 'hola';

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

   foo = 'món';

   return myFn;
};

var f = myFunction();
f();  // registra 'món' -- error
```

Àmbit

```
// una funció anònima autoexecutable
(function() {
   var baz = 1;
   var bim = function() { alert(baz); };
```

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

console.log(baz);  // La consola no mostra res, ja que baz
         // està definida dins l'àmbit de la funció anònima

bar();  // bar està definit fora de la funció anònima
         // ja que va ser declarada sense la paraula clau var; a més,
         // com que va ser definida dins el mateix àmbit que baz,
         // es pot consultar el valor de baz malgrat que
         // aquesta estigui definida dins l'àmbit de la funció anònima

bim();  // bim no està definida per a ser accessible fora de la funció anònima,
         // per la qual cosa es mostrarà un error
```

2.13. Clausures

Les clausures (en anglès, *closures*) són una extensió del concepte d'àmbit (*scope*); funcions que tenen accés a les variables que estan disponibles dins de l'àmbit on es va crear la funció. Si aquest concepte és confús, no cal preocupar-se: s'entén millor per mitjà d'exemples.

En el següent exemple es mostra la manera com funcions hi tenen accés per canviar el valor de les variables. El mateix comportament succeeix en funcions creades dins de bucles; la funció «observa» el canvi en la variable, fins i tot després que la funció sigui definida, i en resulta que en tots els clics aparegui una finestra d'alerta que mostra el valor 5.

Com establir el valor de `i` ?

```
/* Això no es comporta com es desitja; */
/* cada clic mostrarà una finestra d'alerta amb el valor 5 */
for (var i=0; i<5; i++) {
   $('<p>fer click</p>').appendTo('body').click(function() {
      alert(i);
   });
}
```

Establir el valor de `i` utilitzant una clausura

```
/* solució: "clausurar" el valor d'i dins de createFuntion  */
var createFunction = function(i) {
   return function() {
      alert(i);
   };
};
```

```
for (var i = 0; i < 5; i++) {
   $('<p>hacer click</p>').appendTo('body').click(createFunction(i));
}
```

Les clausures també es poden utilitzar per a resoldre problemes amb la paraula clau `this`, la qual és única en cada àmbit.

Utilitzar una clausura per accedir simultàniament a instàncies d'objectes interns i externs.

```
var outerObj = {
   myName: 'extern',
   outerFunction: function() {

      // proveeix d'una referència al mateix objecte outerObj
      // per a utilitzar dins d'innerFunction
      var self = this;

      var innerObj = {
         myName: 'intern',
         innerFunction: function() {
            console.log(self.myName, this.myName); // registra 'extern intern'
         }
      };

      innerObj.innerFunction();

      console.log(this.myName); // registra 'extern'
   }
};

outerObj.outerFunction();
```

Aquest mecanisme pot ser útil quan es treballi amb funcions de devolució de crides (en anglès, *callbacks*). No obstant això, en aquests casos, és preferible utilitzar Function.bind, ja que s'evitarà qualsevol sobrecàrrega associada amb l'àmbit (*scope*).

3. Conceptes bàsics de jQuery

3.1. $(document).ready()

No es pot interactuar de manera segura amb el contingut d'una pàgina fins que el document no està preparat per a la seva manipulació. jQuery permet detectar aquest estat a través de la declaració `$(document).ready()` de tal manera que el bloc s'executarà només quan la pàgina estigui disponible.

El bloc $(document).ready()

```
$(document).ready(function() {
   console.log('el document està preparat');
});
```

Hi ha una forma abreujada per a `$(document).ready()` que podreu trobar algunes vegades; no obstant això, és recomanable no utilitzar-la en cas que s'estigui escrivint codi per a gent que no coneix jQuery.

Forma abreujada per a $(document).ready()

```
$(function() {
   console.log('el document està preparat');
});
```

A més, es pot passar-li `a $(document).ready()` una funció anomenada en lloc d'una d'anònima.

Passar una funció anomenada en lloc d'una funció anònima

```
function readyFn() {
   // codi a executar quan el document estigui llest
}


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

3.2. Selecció d'elements

El concepte més bàsic de jQuery és el de «seleccionar alguns elements i fer accions amb ells». La biblioteca suporta gran part dels selectors CSS3 i diversos més no estandarditzats.

Enllaç d'interès

A http://api.jquery.com/category/selectors/ es pot trobar una referència completa sobre els selectors de la biblioteca.

A continuació, es mostren algunes tècniques comunes per a la selecció d'elements:

Selecció d'elements a partir del seu ID

```
$('#myId'); // els ID han de ser únics per pàgina
```

Selecció d'elements a partir del nom de classe

```
$('div.myClass'); // si s'hi especifica el tipus d'element,
         // es millora el rendiment de la selecció
```

Selecció d'elements pel seu atribut

```
$('input[name=first_name]'); // compte, que pot ser molt lent
```

Selecció d'elements en forma de selector CSS

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

Pseudoselectors

```
$('a.external:first');  // selecciona el primer element <a>
         // amb la classe 'external'
$('tr:odd');// selecciona tots els elements <tr>
         // imparells d'una taula
$('#myForm :input');// selecciona tots els elements del tipus input
         // dins del formulari #myForm
$('div:visible');   // selecciona tots els divs visibles
$('div:gt(2)'); // selecciona tots els divs excepte els tres primers
$('div:animated');  // selecciona tots els divs actualment animats
```

Vegeu també

En el subapartat «Manipulació» d'aquest mòdul s'explica com crear i afegir elements al DOM.

Nota

Quan s'utilitzen els pseudoselectors `:visible` i `:hidden`, jQuery comprova la visibilitat actual de l'element, però no si aquest té assignats els estils CSS `visibility` o `display`; en altres paraules, verifica si *l'alt i ample físic de l'element* és superior a zero. No obstant això, aquesta comprovació no funciona amb els elements `<tr>`; en aquest cas, jQuery comprova si s'està aplicant l'estil `display` i considerarà l'element com a ocult si té assignat el valor `none`. A més, els elements que encara no van ser afegits al DOM seran tractats com a ocults, fins i tot si tenen aplicats estils indicant que han de ser visibles.

Com a referència, aquest és el fragment de codi que utilitza jQuery per a determinar quan un element és visible o no. S'hi han incorporat els comentaris perquè s'entengui millor:

```
jQuery.expr.filters.hidden = function( elem ) {
   var width = elem.offsetWidth, height = elem.offsetHeight,
      skip = elem.nodeName.toLowerCase() === "tr";

   // l'element posseeix alt 0, ample 0 i no és un <tr>?
   return width === 0 && height === 0 && !skip ?

      // llavors ha d'estar ocult (hidden)
      true :

      // pero si posseeix ample i alt
      // i no és un <tr>
      width > 0 && height > 0 && !skip ?

         // llavors ha d'estar visible
         false :

         // si ens trobem aquí, és perquè l'element posseeix ample
         // i alt, però a més és un <tr>,
         // llavors es verifica el valor de l'estil display
         // aplicat a través de CSS
         // per decidir si està ocult o no
         jQuery.curCSS(elem, "display") === "none";
};

jQuery.expr.filters.visible = function( elem ) {
   return !jQuery.expr.filters.hidden( elem );
};
```

Elecció de selectors

L'elecció de bons selectors és un punt important quan es vol millorar el rendiment del codi. Una petita especificitat –per exemple, incloure-hi el tipus d'element (com `div`) quan es fa una selecció pel nom de classe– pot ajudar bastant. Per això, és recomanable donar algunes «pistes» a jQuery sobre en quin lloc del document pot trobar el que es vol seleccionar. D'altra banda, massa especificitat pot ser perjudicial. Un selector com `#laMevaTaula thead tr th.especial` és un excés, el millor seria utilitzar `#laMevaTaula th.especial`.

jQuery ofereix molts selectors basats en atributs, que permeten fer seleccions basades en el contingut dels atributs utilitzant simplificacions d'expressions regulars.

```
// trobar tots els <a> l'atribut rel dels quals acabin en "thinger"
$("a[rel$='thinger']");
```

Aquests tipus de selectors poden ser útils, però també molt lents. Quan sigui possible, és recomanable fer la selecció utilitzant ID, noms de classes i noms d'etiquetes.

Enllaç d'interès

Si es vol conèixer més sobre aquesta qüestió, Paul Irish va fer una gran presentació sobre millores de rendiment en JavaScript (en anglès), la qual inclou diverses diapositives centrades en selectors.

3.2.1. Comprovar seleccions

Una vegada feta la selecció dels elements, voldreu saber si aquesta selecció va donar algun resultat. Per a això, pot ser que escriviu alguna cosa així:

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

No obstant això, aquesta forma no funcionarà. Quan es fa una selecció utilitzant `$()`, sempre és retornat un objecte, i si s'avalua, aquest sempre retornarà `true`. Fins i tot si la selecció no conté cap element, el codi dins del bloc `if` s'executarà.

En lloc d'utilitzar el codi mostrat, el que cal fer és preguntar per la quantitat d'elements que posseeix la selecció que s'executarà. Això es pot fer utilitzant la propietat JavaScript `length`. Si la resposta és 0, la condició avaluarà fals; en cas contrari (més de 0 elements), la condició serà veritable.

Avaluar si una selecció posseeix elements

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

3.2.2. Desar seleccions

Cada vegada que es fa una selecció, s'executa una gran quantitat de codi. jQuery no desa el resultat per si sol; per tant, si s'ha de fer una selecció que després es farà de nou, s'ha de salvar la selecció en una variable.

Desar seleccions en una variable

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

Nota

En l'exemple «Desar seleccions en una variable», la variable comença amb el signe de dòlar. Contràriament a altres llenguatges de programació, en JavaScript aquest signe no posseeix cap significat especial; és només un altre caràcter. No obstant això, aquí s'utilitzarà per a indicar que aquesta variable posseeix un objecte jQuery. Aquesta pràctica –una mena de notació hongaresa– és tan sols una convenció i no és obligatòria.

Una vegada que es desa la selecció en la variable, es pot utilitzar en conjunt amb els mètodes de jQuery, i el resultat serà igual que utilitzant la selecció original.

Nota

La selecció obté només els elements que hi ha a la pàgina quan es fa aquesta acció. Si després s'afegeixen elements al document, cal repetir la selecció o afegir els elements nous a la selecció desada en la variable. En altres paraules, les seleccions desades no s'actualitzen «màgicament» quan el DOM es modifica.

3.2.3. Refinament i filtratge de seleccions

De vegades, es pot obtenir una selecció que conté més del que es necessita; en aquest cas, cal refinar aquesta selecció. jQuery ofereix diversos mètodes per a poder obtenir exactament el que es vol.

Refinament de seleccions

```
$('div.foo').has('p');  // l'element div.foo conté elements <p>
$('h1').not('.bar');// l'element h1 no posseeix la classe 'bar'
$('ul li').filter('.current');  // un ítem d'una llista desordenada
         // que posseeix la classe 'current'
$('ul li').first(); // el primer ítem d'una llista desordenada
$('ul li').eq(5);   // el sisè ítem d'una llista desordenada
```

3.2.4. Selecció d'elements d'un formulari

jQuery ofereix diversos pseudoselectors que ajuden a trobar elements dins dels formularis; aquests són especialment útils, ja que, segons els estats de cada element o el seu tipus, pot ser difícil distingir-los utilitzant selectors CSS estàndard.

- **:button**. Selecciona elements `<button>` i amb l'atribut `type='button'`
- **:checkbox**. Selecciona elements `<input>` amb l'atribut `type='checkbox'`
- **:checked**. Selecciona elements `<input>` del tipus `checkbox` seleccionats
- **:disabled**. Selecciona elements del formulari que estan deshabitats
- **:enabled**. Selecciona elements del formulari que estan habilitats
- **:file**. Selecciona elements `<input>` amb l'atribut `type='file'`
- **:image**. Selecciona elements `<input>` amb l'atribut `type='image'`
- **:input**. Selecciona elements `<input>`, `<textarea>` i `<select>`
- **:password**. Selecciona elements `<input>` amb l'atribut `type='password'`

- **:radio**. Selecciona elements `<input>` amb l'atribut `type='radio'`
- **:reset**. Selecciona elements `<input>` amb l'atribut `type='reset'`
- **:selected**. Selecciona elements `<options>` que estan seleccionats
- **:submit**. Selecciona elements `<input>` amb l'atribut `type='submit'`
- **:text**. Selecciona elements `<input>` amb l'atribut `type='text'`

Utilitzant pseudoselectors en elements de formularis

```
$('#myForm :input'); // obté tots els elements inputs
         // dins el formulari #myForm
```

3.3. Treballar amb seleccions

Una vegada feta la selecció dels elements, es poden utilitzar en conjunt amb diferents mètodes. Aquests, generalment, són de dos tipus: obtenidors (en anglès, *getters*) i establidors (en anglès, *setters*). Els mètodes obtenidors retornen una propietat de l'element seleccionat, mentre que els mètodes establidors fixen una propietat a tots els elements seleccionats.

3.3.1. Encadenament

Si en una selecció es fa una crida a un mètode, i aquest retorna un objecte jQuery, es pot seguir un «encadenament» de mètodes en l'objecte.

Encadenament

```
$('#content').find('h3').eq(2).html('nou text per al tercer element h3');
```

D'altra banda, si s'està escrivint un encadenament de mètodes que inclouen molts passos, es poden escriure línia per línia, i fer que el codi sigui més agradable de llegir.

Format de codi encadenat

```
$('#content')
   .find('h3')
   .eq(2)
   .html('nou text per al tercer element h3');
```

Si es vol tornar a la selecció original en el mig de l'encadenat, jQuery ofereix el mètode `$.fn.end` per a poder fer-ho.

Restablir la selecció original utilitzant el mètode `$.fn.end`

```
$('#content')
   .find('h3')
   .eq(2)
      .html('nou text per al tercer element h3')
   .end() // reestableix la selecció a tots els elements h3 en #content
   .eq(0)
      .html('nou text per al tercer element h3');
```

Nota

L'encadenament és molt poderós i és una característica que moltes biblioteques JavaScript han adoptat des que jQuery es va fer popular. No obstant això, s'ha de fer servir amb compte. Un encadenament de mètodes extensiu pot fer un codi extremament difícil de modificar i depurar. No hi ha una regla que indiqui com de llarg o curt ha de ser l'encadenat, però és recomanable tenir en compte aquest consell.

3.3.2. Obtenidors (*getters*) i establidors (*setters*)

jQuery «sobrecarrega» els seus mètodes; en altres paraules, el mètode per a establir un valor posseeix el mateix nom que el mètode per a obtenir un valor. Quan un mètode és utilitzat per a establir un valor, és anomenat mètode establidor (en anglès, *setter*). En canvi, quan un mètode és utilitzat per a obtenir (o llegir) un valor, és anomenat obtenidor (en anglès, *getter*).

El mètode `$.fn.html` utilitzat com a establidor

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

El mètode html utilitzat com a obtenidor

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

Els mètodes establidors retornen un objecte jQuery, que permet continuar amb la crida de més mètodes en la mateixa selecció, mentre que els mètodes obtenidors retornen el valor pel qual es va consultar, però no permeten continuar cridant més mètodes en aquest valor.

3.4. CSS, estils i dimensions

jQuery inclou una manera útil d'obtenir i establir propietats CSS als elements.

Nota

Les propietats CSS que inclouen com a separador un guió del mig, en JavaScript s'han de transformar al seu estil *CamelCase*. Per exemple, quan s'utilitza com a propietat d'un mètode, l'estil CSS `font-size` s'ha d'expressar com a `fontSize`. No obstant això, aquesta regla no s'aplica quan es passa el nom de la propietat CSS al mètode `$.fn.css`; en aquest cas, els dos formats (en *CamelCase* o amb el guió del mig) funcionaran.

Obtenir propietats CSS

```
$('h1').css('fontSize'); // retorna una cadena de caràcters com "19px"
$('h1').css('font-size'); // també funciona
```

Establir propietats CSS

```
$('h1').css('fontSize', '100px'); // estableix una propietat individual CSS
$('h1').css({
   'fontSize' : '100px',
   'color' : 'red'
}); // estableix múltiples propietats CSS
```

Observeu que l'estil de l'argument utilitzat en la segona línia de l'exemple és un objecte que conté múltiples propietats. Aquesta és una manera comuna de passar múltiples arguments a una funció, i molts mètodes establidors de la biblioteca accepten objectes per a fixar diverses propietats d'una sola vegada.

A partir de la versió 1.6 de la biblioteca, utilitzant `$.fn.css` també es poden establir valors relatius en les propietats CSS d'un element determinat:

Establir valors CSS relatius

```
$('h1').css({
   'fontSize' : '+=15px', // suma 15px a la mida original de l'element
   'paddingTop' : '+=20px' // suma 20px al padding superior original de l'element
});
```

3.4.1. Utilitzar classes per a aplicar estils CSS

Per a obtenir valors dels estils aplicats a un element, el mètode `$.fn.css` és molt útil, encara que la seva utilització com a mètode establidor s'ha d'evitar (ja que, per a aplicar estils a un element, es pot fer directament des de CSS). En el seu lloc, l'ideal és escriure regles CSS que s'apliquin a classes que descriguin els diferents estats visuals dels elements, i després canviar la classe de l'element per tal d'aplicar-hi l'estil que es vol mostrar.

Treballar amb classes

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

$h1.addClass('big');
$h1.removeClass('big');
$h1.toggleClass('big');

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

Les classes també poden ser útils per a desar informació de l'estat d'un element; per exemple, per indicar que un element va ser seleccionat.

3.4.2. Dimensions

jQuery ofereix una varietat de mètodes per a obtenir i modificar valors de dimensions i posició d'un element.

Mètodes bàsics sobre dimensions

```
$('h1').width('50px');   // estableix l'amplada de tots els elements H1
$('h1').width(); // obté l'amplada del primer element H1

$('h1').height('50px');  // estableix l'altura de tots els elements H1
$('h1').height();// obté l'altura del primer element H1

$('h1').position();  // retorna un objecte que conté
         // informació sobre la posició
         // del primer element relatiu a
         // l'"offset" (posició) del seu element pare
```

Enllaç d'interès

El codi mostrat en l'exemple «Mètodes bàsics sobre dimensions» és tan sols un breu resum de les funcionalitats relacionades amb dimensions en jQuery; per a un detall complet, es pot consultar http://api.jquery.com/category/dimensions/.

3.5. Atributs

Els atributs dels elements HTML que conformen una aplicació poden contenir informació útil, per això és important poder establir i obtenir aquesta informació.

El mètode `$.fn.attr` actua tant com a mètode establidor com obtenidor. A més, igual que el mètode `$.fn.css`, quan s'utilitza com a mètode establidor, pot acceptar un conjunt de paraula clau-valor o un objecte que conté més conjunts.

Establir atributs

```
$('a').attr('href', 'allMyHrefsAreTheSameNow.html');
$('a').attr({
   'title' : 'all titles are the same too',
   'href' : 'somethingNew.html'
});
```

En l'exemple, l'objecte passat com a argument està escrit en diverses línies. Com s'ha explicat anteriorment, els espais en blanc no importen en JavaScript, per la qual cosa es poden utilitzar per a fer el codi més llegible. En entorns de producció, es poden utilitzar eines de minificació, els quals treuen els espais en blanc (entre altres coses) i comprimeixen l'arxiu final.

Obtenir atributs

```
$('a').attr('href');  // retorna l'atribut href que pertany
         // al primer element <a> del document
```

3.6. Recórrer el DOM

Una vegada obtinguda la selecció, es poden trobar altres elements utilitzant la mateixa selecció.

Nota

Cal ser acurat a l'hora de recórrer llargues distàncies en un document: recorreguts complexos obliguen que l'estructura del document sigui sempre la mateixa, la qual cosa és difícil de garantir. Un o dos passos per al recorregut estan bé, però generalment cal evitar travessar d'un contenidor a un altre.

Enllaços d'interès

A http://api.jquery.com/category/traversing/es pot trobar una documentació completa sobre els mètodes de recorregut de DOM (en anglès, *traversing*) de què disposa jQuery.

Moure's a través del DOM utilitzant mètodes de recorregut

```
$('h1').next('p');  // seleccionar l'immediat i pròxim
         // element <p> respecte de H1
$('div:visible').parent();  // seleccionar l'element contenidor
         // a un div visible
$('input[name=first_name]').closest('form');  // seleccionar l'element
         // <form> més proper a un input
$('#myList').children();// seleccionar tots els elements
         // hijos de #myList
$('li.selected').siblings();// seleccionar tots els ítems
         // germans de l'element <li>
```

També es pot interactuar amb la selecció utilitzant el mètode `$.fn.each`. Aquest mètode interactua amb tots els elements obtinguts en la selecció i executa una funció per cadascun. La funció rep com a argument l'índex de l'element actual i el mateix element. De manera predeterminada, dins de la funció, es pot fer referència a l'element DOM per mitjà de la declaració `this`.

Interactuar en una selecció

```
$('#myList li').each(function(idx, el) {
   console.log(
      'L'element ' + idx +
      'conté el següent HTML: ' +
      $(el).html()
   );
```

```
});
```

3.7. Manipulació d'elements

Una vegada feta la selecció dels elements que es volen utilitzar, «comença la diversió». Es pot canviar, moure, remoure i duplicar elements. També crear-ne de nous mitjançant una sintaxi simple.

Enllaç d'interès

La documentació completa sobre els mètodes de manipulació es troba en la secció `Manipulation`: http://api.jquery.com/category/manipulation/.

3.7.1. Obtenir i establir informació en elements

Hi ha moltes maneres de modificar un element. Entre les tasques més comunes, hi ha la de canviar l'HTML intern o algun atribut d'aquest. Per a aquest tipus de tasques, jQuery ofereix mètodes simples, funcionals en tots els navegadors moderns. Fins i tot es pot obtenir informació sobre els elements utilitzant els mateixos mètodes però en la seva forma de mètode obtenidor.

Nota

Quan els mètodes actuen com a obtenidors, en general, només treballen amb el primer element de la selecció. A més, no retornen un objecte jQuery, per la qual cosa no s'hi poden encadenar més mètodes. Una excepció és el mètode `$.fn.text`, que permet obtenir el text dels elements de la selecció.

Nota

Fer canvis en els elements és un treball trivial, però cal recordar que el canvi afectarà tots els elements en la selecció, per la qual cosa, si es vol modificar un sol element, cal estar segur d'especificar-ho en la selecció abans de cridar el mètode establidor.

- **$.fn.html**. Obté o estableix el contingut HTML d'un element.
- **$.fn.text**. Obté o estableix el contingut en text de l'element; en cas de passar-li com a argument codi HTML, aquest n'és desposseït.
- **$.fn.attr**. Obté o estableix el valor d'un determinat atribut.
- **$.fn.width**. Obté o estableix l'amplada en píxels del primer element de la selecció com un enter.
- **$.fn.height**. Obté o estableix l'altura en píxels del primer element de la selecció com un enter.
- **$.fn.position**. Obté un objecte amb informació sobre la posició del primer element de la selecció, relatiu al primer element pare posicionat. *Aquest mètode només és obtenidor.*
- **$.fn.val**. Obté o estableix el valor (*value*) en elements de formularis.

Canviar l'HTML d'un element

```
$('#myDiv p:first')
   .html('Nou <strong>primer</strong> paràgraf');
```

3.7.2. Moure, copiar i remoure elements

Hi ha diverses maneres de moure elements a través del DOM, que es poden separar en dos enfocaments:

- voler col·locar el/s element/s seleccionats de forma relativa a un altre element;

- voler col·locar un element relatiu al/s element/s seleccionats.

Per exemple, jQuery proveeix els mètodes `$.fn.insertAfter` i `$.fn.after`. El mètode `$.fn.insertAfter` col·loca el/s element/s seleccionats després de l'element que s'hagi passat com a argument; mentre que el mètode `$.fn.after` col·loca l'element passat com a argument després de l'element seleccionat. Altres mètodes també segueixen aquest patró: `$.fn.insertBefore` i `$.fn.before`; `$.fn.appendTo` i `$.fn.append`; i `$.fn.prependTo` i `$.fn.prepend`.

La utilització d'un o altre mètode dependrà dels elements que es tinguin seleccionats i el tipus de referència que es desar guardar pel que fa a l'element que s'està movent.

Moure elements utilitzant diferents enfocaments

```
// fer que el primer ítem de la llista sigui l'últim
var $li = $('#myList li:first').appendTo('#myList');

// un altre enfocament per al mateix problema
$('#myList').append($('#myList li:first'));

// ha de tenir en compte que no hi ha manera d'accedir a la
// llista d'items que s'ha mogut, ja que retorna
// la llista en si
```

Clonar elements

Quan s'utilitza un mètode com `$.fn.appendTo`, el que s'està fent és moure l'element; però de vegades, en lloc d'això, cal moure un duplicat del mateix element. En aquest cas, es pot utilitzar el mètode `$.fn.clone`.

Obtenir una còpia de l'element

```
// copiar el primer element de la llista i moure'l al final d'aquesta
$('#myList li:first').clone().appendTo('#myList');
```

Nota

Si es necessita copiar informació i esdeveniments relacionats amb l'element, s'ha de passar `true` com a argument de `$.fn.clone`.

Remoure elements

Hi ha dues maneres de remoure elements d'una pàgina: utilitzant `$.fn.remove` o `$.fn.detach`. Quan es vol remoure de manera permanent l'element, cal fer servir el mètode `$.fn.remove`. D'altra banda, el mètode `$.fn.detach` també remou l'element, però manté la informació i els esdeveniments que hi estan associats, i és útil en cas que calgui reinserir l'element en el document.

Nota

El mètode `$.fn.detach` és molt útil quan s'està manipulant de manera severa un element, ja que es pot eliminar l'element, treballar-lo en el codi i després restaurar-lo a la pàgina novament. Aquesta forma té com a benefici no tocar el DOM mentre s'està modificant la informació i els esdeveniments de l'element.

D'altra banda, si es vol mantenir l'element però cal eliminar-ne el contingut, es pot utilitzar el mètode `$.fn.empty`, que «buidarà» el contingut HTML de l'element.

3.7.3. Crear nous elements

jQuery proporciona una manera fàcil i elegant de crear nous elements a través del mateix mètode `$()` que s'utilitza per a fer seleccions.

Crear nous elements

```
$('<p>Un nou paràgraf</p>');
$('<li class="new">nou ítem de la llista</li>');
```

Crear un nou element amb atributs utilitzant un objecte

```
$('<a/>', {
   html : 'Un <strong>nou</strong> enllaç',
   'class' : 'new',
   href : 'foo.html'
});
```

Observeu que en l'objecte que es passa com a argument la propietat *class* està entre cometes, mentre que la propietat *href* i *html* no ho estan. En general, els noms de propietats no han d'estar entre cometes, excepte en cas que s'utilitzi com a nom una paraula reservada (com és el cas de *class*).

Quan es crea un element, no és afegit immediatament a la pàgina, sinó que s'ha de fer en conjunt amb un mètode.

Crear un nou element a la pàgina

```
var $myNewElement = $('<p>Nou element</p>');
```

```
$myNewElement.appendTo('#content');


$myNewElement.insertAfter('ul:last'); // eliminarà l'element <p>
         // existent en #content
$('ul').last().after($myNewElement.clone());  // clonar l'element <p>
         // per tenir les dues versions
```

Estrictament parlant, no cal desar l'element creat en una variable; es pot cridar el mètode i afegir l'element directament després de `$()`. No obstant això, la majoria de les vegades es voldrà fer referència a l'element afegit, per la qual cosa, si es desa en una variable, no cal seleccionar-lo després.

Crear i afegir al mateix temps un element a la pàgina

```
$('ul').append('<li>ítem de la llista</li>');
```

Nota

La sintaxi per a afegir nous elements a la pàgina és molt fàcil d'utilitzar, però és temptador oblidar que hi ha un cost enorme de rendiment en agregar elements al DOM de manera repetida. Si esteu afegint molts elements al mateix contenidor, en lloc d'afegir cada element un per vegada, el millor és concatenar tot l'HTML en una única cadena de caràcters i després annexar-la al contenidor. Una possible solució és utilitzar un vector que posseeixi tots els elements, després reunir-los utilitzant `join` i finalment annexar-la.

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

for (var i=0; i<100; i++) {
   myItems.push('<li>item ' + i + '</li>');
}

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

3.7.4. Manipulació d'atributs

Les capacitats per a la manipulació d'atributs que ofereix la biblioteca són extenses. La realització de canvis bàsics són simples; no obstant això, el mètode `$.fn.attr` permet manipulacions més complexes.

Manipular un simple atribut

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

Manipular múltiples atributs

```
$('#myDiv a:first').attr({
   href : 'newDestination.html',
   rel : 'super-special'
});
```

Utilitzar una funció per a determinar el valor del nou atribut

```
$('#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;
});
```

3.8. Exercicis

3.8.1. Seleccions

Obriu l'arxiu `/ejercicios/index.html` en el navegador. Feu l'exercici utilitzant l'arxiu `/ejercicios/js/sandbox.js` o treballeu directament amb Firebug per complir els punts següents:

1) Seleccioneu tots els elements `div` que tinguin la classe «module».

2) Especifiqueu tres seleccions que puguin seleccionar el tercer ítem de la llista desordenada #myList. Quin és el millor per utilitzar? Per què?

3) Seleccioneu l'element `label` de l'element `input` utilitzant un selector d'atribut.

4) Esbrineu quants elements a la pàgina estan ocults (ajuda: `.length`).

5) Esbrineu quantes imatges a la pàgina posseeixen l'atribut `alt`.

6) Seleccioneu totes les files imparelles del cos de la taula.

3.8.2. Recórrer el DOM

Obriu l'arxiu `/ejercicios/index.html` en el navegador. Feu l'exercici utilitzant l'arxiu `/ejercicios/js/sandbox.js` o treballeu directament amb Firebug per complir els punts següents:

1) Seleccioneu totes les imatges en la pàgina; registreu en la consola l'atribut `alt` de cada imatge.

2) Seleccioneu l'element `input`, després aneu cap al formulari i afegiu-hi una classe.

3) Seleccioneu l'ítem que té la classe «current» dins la llista #myList i removeu aquesta classe en l'element; després, afegiu la classe «current» a l'ítem següent de la llista.

4) Seleccioneu l'element `select` dins de #specials; després aneu al botó `submit`.

5) Seleccioneu el primer ítem de la llista en l'element #slideshow; afegiu-hi la classe «current» i després afegiu la classe «disabled» als elements germans.

3.8.3. Manipulació

Obriu l'arxiu `/ejercicios/index.html` en el navegador. Feu l'exercici utilitzant l'arxiu `/ejercicios/js/sandbox.js` o treballeu directament amb Firebug per complir els punts següents:

1) Afegiu cinc nous ítems al final de la llista desordenada #myList. Ajuda:

```
for (var i = 0; i<5; i++) { ... }
```

2) Removeu els ítems imparells de la llista.

3) Afegiu un altre element `h2` i un altre paràgraf a l'últim `div.module`.

4) Afegiu una altra opció a l'element `select`; doneu-li a l'opció afegida el valor «Wednesday».

5) Afegiu un nou `div.module` a la pàgina després de l'últim; després afegiu-hi una còpia d'una de les imatges existents dins del nou `div`.

4. El nucli de jQuery

4.1. $ enfront de $()

Fins ara, s'ha tractat completament amb mètodes que es criden des de l'objecte jQuery. Per exemple:

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

Aquests mètodes són part de l'espai de noms (en anglès, *namespace*) `$.fn`, o del prototip (en anglès, *prototype*) de jQuery, i es consideren mètodes de l'objecte jQuery.

No obstant això, hi ha mètodes que són part de l'espai de noms de $ i es consideren mètodes del nucli de jQuery.

Aquestes distincions poden ser bastants confuses per a usuaris nous. Per a evitar la confusió, cal recordar aquests dos punts:

- els mètodes utilitzats en seleccions es troben dins de l'espai de noms `$.fn`, i automàticament reben i retornen una selecció en si;

- mètodes en l'espai de noms `$` són generalment mètodes per a diferents utilitats, no treballen amb seleccions, no se'ls passa cap argument i el valor que retornen pot variar.

Hi ha alguns casos en què mètodes de l'objecte i del nucli tenen els mateixos noms, com passa amb `$.each` i `$.fn.each`. En aquests casos, cal ser curosos a l'hora de llegir bé la documentació per a saber quin objecte utilitzar correctament.

4.2. Mètodes utilitaris

jQuery ofereix diversos mètodes utilitaris dins de l'espai de noms `$`. Aquests mètodes són de gran ajuda per a dur a terme tasques rutinàries de programació. A continuació es mostren alguns exemples; per a una documentació completa sobre ells, visiteuhttp://api.jquery.com/category/utilities/:

1) $.trim. Remou els espais en blanc del principi i del final.

```
$.trim('varios espacios en blanco   ');
// retorna 'diversos espais en blanc'
```

2) $.each. Interactua en vectors i objectes.

```
$.each([ 'foo', 'bar', 'baz' ], function(idx, val) {
   console.log('element ' + idx + 'es ' + val);
});

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

Nota

Com s'ha dit més amunt, hi ha un mètode anomenat `$.fn.each`, que interactua en una selecció d'elements.

3) $.inArray. Retorna l'índex d'un valor en un vector, o -1 si el valor no es troba en el vector.

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

if ($.inArray(4, myArray) !== -1) {
   console.log('valor trobat');
}
```

4) $.extend. Canvia les propietats del primer objecte utilitzant les propietats dels objectes subsegüents.

```
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 es volen canviar les propietats de cap dels objectes que s'utilitzen en `$.extend`, s'hi ha d'incloure un objecte buit com a primer argument.

```
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'
```

5) $.proxy. Retorna una funció que sempre s'executarà en l'àmbit (*scope*) proveït; en altres paraules, estableix el significat de *this* (inclòs dins de la funció) com el segon argument.

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

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

myFunction(); // retorna l'objecte window

var myProxyFunction = $.proxy(myFunction, myObject);
myProxyFunction(); // retorna l'objecte myObject
```

Si es posseeix un objecte amb mètodes, es pot passar aquest objecte i el nom d'un mètode per a retornar una funció que sempre s'executa en l'àmbit d'aquest objecte.

```
var myObject = {
   myFn : function() {
      console.log(this);
   }
};

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

4.3. Comprovació de tipus

Com s'ha dit en l'apartat 2, jQuery ofereix diversos mètodes útils per a determinar el tipus d'un valor específic.

Comprovar el tipus d'un determinat valor

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

// Utilitzar l'operador typeof de JavaScript per comprovar tipus primitius
typeof myValue == 'string'; // fals (false)
typeof myValue == 'number'; // fals (false)
typeof myValue == 'undefined'; // fals (false)
typeof myValue == 'boolean'; // fals (false)

// Utilitzar l'operador d'igualtat estricta per comprovar valors nuls (null)
myValue === null; // fals (false)

// Utilitzar els mètodes jQuery per comprovar tipus no primitius
jQuery.isFunction(myValue); // fals (false)
jQuery.isPlainObject(myValue); // fals (false)
jQuery.isArray(myValue); // veritable (true)
jQuery.isNumeric(16); // veritable (true). No disponible en versions inferiors a jQuery 1.7
```

4.4. El mètode data

Sovint hi ha informació sobre un element que cal desar. En JavaScript es pot fer afegint propietats al DOM de l'element, però aquesta pràctica comporta enfrontar-se a pèrdues de memòria (en anglès, *memory leaks*) en alguns navegadors. jQuery ofereix una manera senzilla de desar informació relacionada amb un element, i la mateixa biblioteca s'ocupa de manejar els problemes que poden sorgir per falta de memòria.

Desar i recuperar informació relacionada amb un element

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

Per mitjà del mètode `$.fn.data` es pot desar qualsevol tipus d'informació sobre un element. És difícil exagerar la importància d'aquest concepte quan s'està desenvolupant una aplicació complexa.

Per exemple, si es vol establir una relació entre l'ítem d'una llista i el *div* que hi ha dins d'aquest ítem, es pot fer cada vegada que s'interactua amb l'ítem, però una solució millor és fer-ho una sola vegada, desant un punter al *div* utilitzant el mètode `$.fn.data`.

Establir una relació entre elements utilitzant el mètode `$.fn.data`

```
$('#myList li').each(function() {
   var $li = $(this), $div = $li.find('div.content');
   $li.data('contentDiv', $div);
});

// per tant, no s'ha de tornar a buscar el div;
// és possible llegir-ho des de la informació associada a l'ítem de la llista
var $firstLi = $('#myList li:first');
$firstLi.data('contentDiv').html('nou contingut');
```

A més, es pot passar al mètode un objecte que contingui un o més parells de conjunts paraula clau – valor.

A partir de la versió 1.5 de la biblioteca, jQuery permet utilitzar el mètode `$.fn.data` per a obtenir la informació associada a un element que posseeixi l'atribut HTML5 `data-*`:

Elements amb l'atribut `data-*`

```
<a id='foo' data-foo='baz' href='#'>Foo</a>
```

```
<a id='foobar' data-foo-bar='fol' href='#'>Foo Bar</a>
```

Obtenir els valors associats als atributs `data-*` amb `$.fn.data`

```
// obté el valor de l'atribut data-foo
// utilitzant el mètode $.fn.data
console.log($('#foo').data('foo')); // registra 'baz'

// obté el valor del segons element
console.log($('#foobar').data('fooBar')); // registra 'fol'
```

Enllaç d'interès

Per a més informació sobre l'atribut HTML5 `data-*` visitad https://www.w3.org/TR/html5/dom.html#embedding-custom-non-visible-data-with-the-data-*-attributes.

Nota

A partir de la versió 1.6 de la biblioteca, per a obtenir el valor de l'atribut `data-foo-bar` del segon element, l'argument en `$.fn.data` s'ha de passar en estil *CamelCase*.

4.5. Detecció de navegadors i característiques

Més enllà del fet que jQuery elimini la majoria de les peculiaritats de JavaScript entre cada navegador, de vegades cal executar codi en un navegador específic.

Per a aquest tipus de situacions, jQuery ofereix l'objecte `$.support` i `$.browser` (aquest últim en desús).

Enllaç d'interès

Es pot trobar documentació completa sobre els objectes `$.support` i `$.browser` a http://api.jquery.com/jquery.support/ i http://api.jquery.com/jquery.browser/.

L'objectiu de `$.support` és determinar quines característiques suporta el navegador web.

L'objecte `$.browser` permet detectar el tipus de navegador i la seva versió. Aquest objecte està en desús (encara que a curt termini no està planificada la seva eliminació del nucli de la biblioteca) i es recomana utilitzar l'objecte `$.support` per a aquests propòsits.

4.6. Evitar conflictes amb altres biblioteques JavaScript

Si s'està utilitzant jQuery conjuntament amb altres biblioteques JavaScript, que també utilitzen la variable `$`, es poden arribar a donar una sèrie d'errors. Per a poder solucionar-los, cal posar jQuery en el mode «no-conflicte». Això s'ha de fer immediatament després que jQuery es carregui a la pàgina i abans del codi que s'executarà.

Quan es posa jQuery en mode «no-conflicte», la biblioteca ofereix l'opció d'assignar un nom per a reemplaçar la variable `$`.

Posar jQuery en mode «no-conflicte»

```
<script src="prototype.js"></script> // la biblioteca prototype
        // també utilitza $
<script src="jquery.js"></script>// es carrega jquery
```

```
          // en la pàgina
<script>var $j = jQuery.noConflict();</script>   // s'inicialitza
          // el mode "no-conflicte"
```

També es pot continuar utilitzant `$` contenint el codi en una funció anònima autoexecutable. Aquest és un patró estàndard per a la creació d'extensions per a la biblioteca, ja que `$` queda tancada dins l'àmbit de la mateixa funció anònima.

Utilitzar $ dins d'una funció anònima autoexecutable

```
<script src="prototype.js"></script>
<script src="jquery.js"></script>
<script>
jQuery.noConflict();

(function($) {
   // el codi va aquí, i es pot utilitzar $
})(jQuery);
</script>
```

5. Esdeveniments

5.1. Introducció

jQuery proveeix mètodes per a associar controladors d'esdeveniments (en anglès, *event handlers*) a selectors. Quan un esdeveniment ocorre, la funció proveïda és executada. Dins de la funció, la paraula clau `this` fa referència a l'element en què l'esdeveniment ocorre.

Enllaç d'interès

Per a més detalls sobre els esdeveniments en jQuery, es pot consultar http://api.jquery.com/category/events/.

La funció del controlador d'esdeveniments pot rebre un objecte. Aquest objecte es pot utilitzar per a determinar la naturalesa de l'esdeveniment o, per exemple, prevenir el comportament predeterminat d'aquest.

Enllaç d'interès

Per a més detall sobre l'objecte de l'esdeveniment, visiteu http://api.jquery.com/category/events/event-object/.

5.2. Vincular esdeveniments a elements

jQuery ofereix mètodes per a la majoria dels esdeveniments, entre aquests `$.fn.clic`, `$.fn.focus`, `$.fn.blur`, `$.fn.change`, etc. Aquests últims són formes reduïdes del mètode `$.fn.on` de jQuery (`$.fn.bind` en versions anteriors a jQuery 1.7). El mètode `$.fn.on` és útil per a vincular (en anglès, *binding*) la mateixa funció de controlador a múltiples esdeveniments, per a quan es vol proveir informació al controlador d'esdeveniment, quan s'està treballant amb esdeveniments personalitzats o quan es vol passar un objecte a múltiples esdeveniments i controladors.

Vincular un esdeveniment utilitzant un mètode reduït

```
$('p').click(function() {
   console.log('click');
});
```

Vincular un esdeveniment utilitzant el mètode `$.fn.on`

```
$('p').on('click', function() {
   console.log('click');
});
```

Vincular un esdeveniment utilitzant el mètode `$.fn.on` amb informació associada

```
$('input').on(
   'click blur',  // es poden vincular múltiples esdeveniments a l'element
   { foo : 'bar' }, // s'ha de passar la informació associada com a argument
```

```
    function(eventObject) {
       console.log(eventObject.type, eventObject.data);
       // registra el tipus d'esdeveniment i la informació associada { foo : 'bar' }
    }
);
```

5.2.1. Vincular esdeveniments per a executar una vegada

De vegades, es pot necessitar que un controlador particular s'executi només una vegada, i després d'això es necessiti que no s'executi cap més, o que s'executi un altre de diferent. Per a aquest propòsit, jQuery proveeix el mètode `$.fn.one`.

Canviar controladors utilitzant el mètode `$.fn.one`

```
$('p').one('click', function() {
   console.log('Es va clicar l'element per primera vegada');
   $(this).click(function() { console.log('S'ha clicat de nou'); });
});
```

El mètode `$.fn.one` és útil per a situacions en què cal executar un cert codi la primera vegada que ocorre un esdeveniment en un element, però no en els esdeveniments successius.

5.2.2. Desvincular esdeveniments

Per a desvincular (en anglès, *unbind*) un controlador d'esdeveniment, es pot utilitzar el mètode `$.fn.off` i passant-li el tipus d'esdeveniment que es vol desconnectar. Si es va passar com a adjunt a l'esdeveniment una funció anomenada, es pot aïllar la desconnexió d'aquesta funció passant-la com a segon argument.

Desvincular tots els controladors de l'esdeveniment clic en una selecció

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

Desvincular un controlador particular de l'esdeveniment clic

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

$('p').on('click', foo).on('click', bar);
$('p').off('click', bar); // foo està lligat encara a l'esdeveniment click
```

5.2.3. Espais de noms per a esdeveniments

Quan s'estan desenvolupant aplicacions complexes o extensions de jQuery, pot ser útil utilitzar espais de noms per als esdeveniments i, d'aquesta manera, evitar que es desvinculin esdeveniments quan no es desitja.

Assignar espais de noms a esdeveniments

```
$('p').on('click.myNamespace', function() { /* ... */ });
$('p').off('click.myNamespace');
$('p').off('.myNamespace'); // desvincula tots els esdeveniments amb
         // l'espai de nom 'myNamespace'
```

5.2.4. Vinculació de múltiples esdeveniments

Molt sovint, elements en una aplicació estaran vinculats a múltiples esdeveniments, cadascun amb una funció diferent. En aquests casos, es pot passar un objecte dins de `$.fn.on` amb un o més parells de noms clau / valors. Cada clau serà el nom de l'esdeveniment, mentre que cada valor serà la funció a executar quan ocorri l'esdeveniment.

Vincular múltiples esdeveniments a un element

```
$('p').on({
   'click': function() {
      console.log('clicat');
   },
   'mouseover': function() {
      console.log('sobrepassat');
   }
});
```

5.3. L'objecte de l'esdeveniment

Com s'esmenta en la introducció, la funció controladora d'esdeveniments rep un objecte de l'esdeveniment, el qual conté diversos mètodes i propietats. L'objecte és comunament utilitzat per a prevenir l'acció predeterminada de l'esdeveniment a través del mètode *preventDefault*. No obstant això, també conté diverses propietats i mètodes útils:

- **pageX**, **pageY**. La posició del punter del ratolí al moment que l'esdeveniment va ocórrer, relatiu a les zones superiors i esquerra de la pàgina.

- **type**. El tipus d'esdeveniment (per exemple «clic»).

- **which**. El botó o tecla que es prem.
- **data**. Alguna informació passada quan l'esdeveniment és executat.
- **target**. L'element DOM que va inicialitzar l'esdeveniment.
- **preventDefault()**. Cancel·la l'acció predeterminada de l'esdeveniment (per exemple: seguir un enllaç).
- **stopPropagation()**. Atura la propagació de l'esdeveniment sobre altres elements.

D'altra banda, la funció controladora també té accés a l'element DOM que va inicialitzar l'esdeveniment a través de la paraula clau `this`. Per a convertir aquest element DOM en un objecte jQuery (i poder utilitzar els mètodes de la biblioteca), cal escriure `$(this)`, com es mostra a continuació:

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

Cancel·lar que en fer clic en un enllaç, aquest se segueixi

```
$('a').click(function(e) {
   var $this = $(this);
   if ($this.attr('href').match('evil')) {
      e.preventDefault();
      $this.addClass('evil');
   }
});
```

5.4. Execució automàtica de controladors d'esdeveniments

A través del mètode `$.fn.trigger`, jQuery proporciona una manera de disparar controladors d'esdeveniments sobre algun element sense requerir l'acció de l'usuari. Si bé aquest mètode té els seus usos, no s'hauria d'utilitzar per a cridar simplement una funció que pugui ser executada amb un clic de l'usuari. En el seu lloc, s'hauria de desar la funció que es necessita cridar en una variable, i després passar el nom de la variable quan es fa el vincle (*binding*). D'aquesta manera, es pot cridar la funció quan es vol en lloc d'executar `$.fn.trigger`.

Disparar un controlador d'esdeveniments de la manera correcta

```
var foo = function(e) {
   if (e) {
      console.log(e);
   } else {
      console.log('aquesta execució no prové d'un  esdeveniment');
   }
```

```
};



$('p').click(foo);


foo(); // en lloc de fer $('p').trigger('click')
```

5.5. Incrementar el rendiment amb la delegació d'esdeveniments

Quan treballeu amb jQuery, afegireu freqüentment nous elements a la pàgina, i quan ho feu, necessitareu vincular esdeveniments a aquests elements. En lloc de repetir la tasca cada vegada que s'hi afegeix un element, es pot utilitzar la delegació d'esdeveniments per a fer-ho. Amb ella, es pot enllaçar un esdeveniment a un element contenidor, i després, quan l'esdeveniment ocorri, es pot veure en quin element succeeix.

La delegació d'esdeveniments presenta alguns beneficis, fins i tot si no es té pensat afegir més elements a la pàgina. El temps requerit per a enllaçar controladors d'esdeveniments a centenars d'elements no és un treball trivial; si es posseeix un gran conjunt d'elements, s'hauria de considerar utilitzar la delegació d'esdeveniments a un element contenidor.

Nota

A partir de la versió 1.4.2, es va introduir `$.fn.delegate`; no obstant això, a partir de la versió 1.7 és preferible utilitzar l'esdeveniment `$.fn.on` per a la delegació d'esdeveniments.

Delegar un esdeveniment utilitzant `$.fn.on`

```
$('#myUnorderedList').on('click', 'li', function(e) {
   var $myListItem = $(this);
   // ...
});
```

Delegar un esdeveniment utilitzant `$.fn.delegate`

```
$('#myUnorderedList').delegate('li', 'click', function(e) {
   var $myListItem = $(this);
   // ...
});
```

5.5.1. Desvincular esdeveniments delegats

Si necessiteu remoure esdeveniments delegats, no ho podeu fer simplement desvinculant-los. Per a això, feu servir el mètode `$.fn.off` per a esdeveniments connectats amb `$.fn.on`, i `$.fn.undelegate` per a esdeveniments connectats amb `$.fn.delegate`. Igual que quan es fa un vincle, opcionalment, es pot passar el nom d'una funció vinculada.

Desvincular esdeveniments delegats

```
$('#myUnorderedList').off('click', 'li');
```

```
$('#myUnorderedList').undelegate('li', 'click');
```

5.6. Funcions auxiliars d'esdeveniments

jQuery ofereix dues funcions auxiliars per a treballar amb esdeveniments:

5.6.1. $.fn.hover

El mètode `$.fn.hover` permet passar una o dues funcions que s'executaran quan els esdeveniments `mouseenter` i `mouseleave` ocorrin en l'element seleccionat. Si es passa una sola funció, aquesta s'executarà en tots dos esdeveniments; en canvi, si es passen dues, la primera s'executarà quan ocorri l'esdeveniment `mouseenter`, mentre que la segona s'executarà quan ocorri `mouseleave`.

Nota

A partir de la versió 1.4 de jQuery, el mètode requereix obligatòriament dues funcions.

La funció auxiliar hover

```
$('#menu li').hover(function() {
   $(this).toggleClass('hover');
});
```

5.6.2. $.fn.toggle

Igual que el mètode anterior, `$.fn.toggle` rep dues o més funcions; cada vegada que un esdeveniment ocorre, s'executarà la funció següent en la llista. Generalment, `$.fn.toggle` és utilitzada amb només dues funcions. En cas que s'utilitzin més de dues funcions, cal anar amb compte, ja que es pot dificultar la depuració del codi.

La funció auxiliar toggle

```
$('p.expander').toggle(
   function() {
      $(this).prev().addClass('open');
   },
   function() {
      $(this).prev().removeClass('open');
   }
);
```

5.7. Exercicis

5.7.1. Crear un «suggeriment» per a una caixa d'ingrés de text

Obriu l'arxiu `/ejercicios/index.html` en el navegador. Feu l'exercici utilitzant l'arxiu `/ejercicios/js/inputhint.js` o treballeu directament amb Firebug. La tasca que s'ha de fer és utilitzar el text de l'element *label* i aplicar un «suggeriment» en la caixa d'ingrés de text. Els passos que s'han de seguir són els següents:

1) Establiu el valor de l'element *input* igual al valor de l'element *label*.

2) Afegiu la classe *hint* a l'element *input*.

3) Removeu l'element *label*.

4) Vinculeu un esdeveniment *focus* en *l'input* per a remoure el text de suggeriment i la classe *hint*.

5) Vinculeu un esdeveniment *blur* en *l'input* per a restaurar el text de suggeriment i la classe *hint* en cas que no s'hagi ingressat algun text.

Quines altres consideracions cal considerar si es vol aplicar aquesta funcionalitat a un lloc real?

5.7.2. Afegir una navegació per pestanyes

Obriu l'arxiu `/ejercicios/index.html` en el navegador. Feu l'exercici utilitzant l'arxiu `/ejercicios/js/tabs.js` o treballeu directament amb Firebug. La tasca que s'ha de fer és crear una navegació mitjançant pestanyes per als dos elements *div.module*. Els passos que s'han de seguir són els següents:

1) Oculteu tots els elements *div.module*.

2) Creeu una llista desordenada abans del primer *div.module* per a utilitzar com a pestanyes.

3) Interactueu amb cada *div* utilitzant `$.fn.each`. Per cadascun, utilitzeu el text de l'element *h2* com el text per a l'ítem de la llista desordenada.

4) Vinculeu un esdeveniment clic a cada ítem de la llista de manera que:

a) mostreu el *div* corresponent i oculteu l'altre;

b) afegiu la classe *current* a l'ítem seleccionat;

c) removeu la classe *current* de l'altre ítem de la llista;

5) finalment, mostreu la primera pestanya.

6. Efectes

6.1. Introducció

Amb jQuery, agregar efectes a una pàgina és molt fàcil. Aquests efectes tenen una configuració predeterminada, però també és possible proveir-los paràmetres personalitzats. A més, es poden crear animacions particulars establint valors de propietats CSS.

Per a una documentació completa sobre els diferents tipus d'efectes, podeu visitar la secció `effects`: http://api.jquery.com/category/effects/.

6.2. Efectes incorporats en la biblioteca

Els efectes més utilitzats ja vénen incorporats dins de la biblioteca en forma de mètodes:

- **$.fn.xou**. Mostra l'element seleccionat.
- **$.fn.hide**. Oculta l'element seleccionat.
- **$.fn.fadeIn**. De manera animada, canvia l'opacitat de l'element seleccionat al 100%.
- **$.fn.fadeOut**. De manera animada, canvia l'opacitat de l'element seleccionat al 0.
- **$.fn.slideDown**. Mostra l'element seleccionat amb un moviment de desplaçament vertical.
- **$.fn.slideUp**. Oculta l'element seleccionat amb un moviment de desplaçament vertical.
- **$.fn.slideToggle**. Mostra o oculta l'element seleccionat amb un moviment de desplaçament vertical, depenent de si actualment l'element està visible o no.

Ús bàsic d'un efecte incorporat

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

6.2.1. Canviar la durada dels efectes

A excepció de `$.fn.xou` i `$.fn.hide`, tots els mètodes tenen una durada predeterminada de l'animació en 400 ms. Aquest valor es pot canviar.

Configurar la durada d'un efecte

```
$('h1').fadeIn(300);  // esvaniment en 300ms
$('h1').fadeOut('slow');  // utilitzar una definició de velocitat interna
```

jQuery.fx.speeds

jQuery posseeix un objecte en `jquery.fx.speeds` que conté la velocitat predeterminada per a la durada d'un efecte, i també els valors per a les definicions *slow* i *fast*.

```
speeds: {
   slow: 600,
   fast: 200,
   // velocitat predeterminada
   _default: 400
}
```

Per tant, es poden sobreescriure o afegir nous valors a l'objecte. Per exemple, pot ser que es vulgui canviar el valor predeterminat de l'efecte o afegir-hi una velocitat personalitzada.

Afegir velocitats personalitzades a `jQuery.fx.speeds`

```
jQuery.fx.speeds.muyRapido = 100;
jQuery.fx.speeds.muyLento = 2000;
```

6.2.2. Fer una acció quan s'ha executat un efecte

Sovint, es voldrà executar una acció una vegada que l'animació hagi acabat, ja que si s'executa l'acció abans que l'animació hagi acabat, es pot arribar a alterar la qualitat de l'efecte o afectar els elements que en formen part.

Les funcions de devolució de crida (en anglès, *callback functions*) proporcionen una manera d'executar codi una vegada que un esdeveniment hagi acabat.

En aquest cas, l'esdeveniment que respondrà a la funció serà la conclusió de l'animació. Dins de la funció de devolució, la paraula clau `this` fa referència a l'element on l'efecte va ser executat i, igual que passa amb els esdeveniments, es pot transformar en un objecte jQuery utilitzant `$(this)`.

Executar cert codi quan una animació ha conclòs

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

Observeu que si la selecció no retorna cap element, la funció mai no s'executarà. Aquest problema es pot resoldre comprovant si la selecció retorna algun element; i en cas que no ho faci, cal executar la funció de devolució immediatament.

Executar una funció de devolució fins i tot si no hi ha elements per a animar

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

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

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

6.3. Efectes personalitzats amb $.fn.animate

Es poden fer animacions en propietats CSS utilitzant el mètode `$.fn.animate`. Aquest mètode permet fer una animació establint valors a propietats CSS o canviant-ne els valors actuals.

Efectes personalitzats amb `$.fn.animate`

```
$('div.funtimes').animate(
   {
      left : "+=50",
      opacity : 0.25
   },
   300, // duration
   function() { console.log('realizado'); // funció de devolució de crida
});
```

Nota

Les propietats relacionades amb el color no poden ser animades utilitzant el mètode `$.fn.animate`, però es pot fer a través de l'extensió color plugin. Més endavant en el mòdul, es discutirà la utilització d'extensions.

6.3.1. *Easing*

El concepte d'*easing* descriu la manera com un efecte ocorre, és a dir, si la velocitat durant l'animació és constant o no.

jQuery inclou només dos mètodes d'easing: *swing* i *linear*. Si es colen transicions més naturals en les animacions, hi ha diverses extensions que ho permeten.

A partir de la versió 1.4 de la biblioteca, es pot establir el tipus de transició per cada propietat utilitzant el mètode `$.fn.animate`.

Transició d'*easing* per cada propietat

```
$('div.funtimes').animate(
   {
      left : [ "+=50", "swing" ],
      opacity : [ 0.25, "linear" ]
   },
   300
);
```

Enllaç d'interès

Per a més detalls sobre les opcions d'*easing*, consulteu http://api.jquery.com/animate/.

6.4. Control dels efectes

jQuery proveeix de diverses eines per al maneig d'animacions:

- **$.fn.stop**. Atura les animacions que s'estan executant en l'element seleccionat.

- **$.fn.delay**. Espera un temps determinat abans d'executar la pròxima animació.

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

- **jQuery.fx.off**. Si el valor és veritable (*true*), no hi haurà transicions per a les animacions, i als elements se'ls establirà l'estat final de l'animació. Aquest

mètode pot ser especialment útil quan s'està treballant amb navegadors antics.

6.5. Exercicis

6.5.1. Mostrar text ocult

Obriu l'arxiu `/ejercicios/index.html` en el navegador. Feu l'exercici utilitzant l'arxiu `/ejercicios/js/blog.js`. La tasca és afegir alguna interactivitat a la secció blog de la pàgina:

- en fer clic en algun dels titulars del *div #blog*, s'ha de mostrar el paràgraf corresponent amb un efecte de desplaçament;

- en fer clic en un altre titular, s'ha d'ocultar el paràgraf mostrat amb un efecte de desplaçament i mostrar novament el paràgraf corresponent també amb un efecte de desplaçament. Ajuda: no oblideu de fer servir el selector `:visible`.

6.5.2. Crear un menú desplegable

Obriu l'arxiu `/ejercicios/index.html` en el navegador. Feu l'exercici utilitzant l'arxiu `/ejercicios/js/navigation.js`. La tasca és poder desplegar els ítems del menú superior de la pàgina:

- en passar el punter del ratolí per sobre d'un ítem del menú, s'ha de mostrar el seu submenú en cas que existeixi;

- com que no està més al damunt d'un ítem, el submenú s'ha d'ocultar.

Per poder fer-ho, utilitzeu el mètode `$.fn.hover` per a afegir o remoure una classe en el submenú per a poder controlar si ha d'estar ocult o visible (l'arxiu `/ejercicios/css/styles.css` inclou una classe *hover* per a aquest propòsit).

6.5.3. Crear un *slideshow*

Obriu l'arxiu `/ejercicios/index.html` en el navegador. Feu l'exercici utilitzant l'arxiu `/ejercicios/js/slideshow.js`. La tasca és afegir un *slideshow* a la pàgina amb JavaScript.

1) Moveu l'element *#slideshow* a la part superior de la pàgina.

2) Escriviu un codi que permeti mostrar els ítems de manera cíclica, mostrant un ítem durant uns segons, després ocultant-lo amb un efecte *fade out* i mostrant el següent amb un efecte **fade in*.

3) Una vegada s'arriba a l'últim ítem de la llista, cal començar de nou amb el primer.

Per a un desafiament més gran, feu una àrea de navegació per sota de l'*slideshow* que mostri quantes imatges hi ha i en quina es troba (ajuda: *$.fn.prevAll`* pot ser útil).

7. Ajax

7.1. Introducció

El mètode *XMLHttpRequest* (XHR) permet als navegadors comunicar-se amb el servidor sense la necessitat de recarregar la pàgina. Aquest mètode, també conegut com a Ajax (*Asynchronous JavaScript and XML*), permet la creació d'aplicacions riques en interactivitat.

Les peticions Ajax són executades pel codi JavaScript, el qual envia una petició a una URL i, quan rep una resposta, es pot executar una funció de devolució. Aquesta rep com a argument la resposta del servidor i fa alguna cosa amb ella. Com que la resposta és asíncrona, la resta del codi de l'aplicació es continua executant, per la qual cosa és imperatiu que una funció de devolució sigui executada per manejar la resposta.

A través de diversos mètodes, jQuery proveeix suport per a Ajax, i permet abstreure les diferències que hi pot haver entre navegadors. Els mètodes en qüestió són `$.get()`, `$.getScript()`, `$.getJSON()`, `$.post()` i `$().load()`.

Malgrat que la definició d'Ajax posseeix la paraula *XML*, la majoria de les aplicacions no utilitzen aquest format per a transportar dades, sinó que en el seu lloc s'utilitza HTML pla o informació en format JSON (*JavaScript Object Notation*).

En general, Ajax no treballa a través de dominis diferents. No obstant això, hi ha excepcions, com els serveis que proveeixen d'informació en format JSONP (*JSON with Padding*), que permeten una funcionalitat limitada a través de diferents dominis.

7.2. Conceptes clau

La utilització correcta dels mètodes Ajax requereix primer la comprensió d'alguns conceptes clau.

7.2.1. GET enfront de POST

Els dos mètodes HTTP més comuns per a enviar una petició a un servidor són GET i POST. És important entendre la utilització de cadascun.

El mètode GET s'ha d'utilitzar per a operacions no destructives, és a dir, operacions en què s'estan «obtenint» dades del servidor, però no modificant. Per exemple, una consulta a un servei de cerca podria ser una petició GET. D'altra

banda, les sol·licituds GET es poden emmagatzemar a la memòria cau (*caché*) del navegador, i pot conduir a un comportament impredictible si no s'espera. Generalment, la informació enviada al servidor s'envia en una cadena de dades (en anglès, *query string*).

El mètode POST s'ha d'utilitzar per a operacions destructives, és a dir, operacions en què s'està incorporant informació al servidor. Per exemple, quan un usuari desa un article en un blog, aquesta acció hauria d'utilitzar POST. D'altra banda, aquest tipus de mètode no es desa a la memòria cau del navegador. A més, una cadena de dades pot ser part de la URL, però la informació tendeix a ser enviada de manera separada.

7.2.2. Tipus de dades

Generalment, jQuery necessita algunes instruccions sobre el tipus d'informació que s'espera rebre quan es fa una petició Ajax. En alguns casos, el tipus de dada és especificat pel nom del mètode, però en altres casos s'ha de detallar com a part de la configuració del mètode:

- **text**. Per al transport de cadenes de caràcters simples.
- **html**. Per al transport de blocs de codi HTML que seran situats en la pàgina.
- **script**. Per a afegir un nou *script* amb codi JavaScript a la pàgina.
- **json**. Per a transportar informació en format JSON, que pot incloure cadenes de caràcters, vectors i objectes.

Nota

A partir de la versió 1.4 de la biblioteca, si la informació JSON no està formatada correctament, la petició podria fallar.

Enllaç d'interès

Visitad http://json.org per a obtenir detalls sobre un format de dades correcte en JSON.

És recomanable utilitzar els mecanismes que posseeixi el llenguatge del costat de servidor per a la generació d'informació en format JSON.

- **jsonp**. Per a transportar informació JSON d'un domini a un altre.
- **xml**. Per a transportar informació en format XML.

Malgrat els diferents tipus de dades que es poden utilitzar, és recomanable fer servir el format JSON, ja que és molt flexible, i permet per exemple enviar al mateix temps informació plana i HTML.

7.2.3. Asincronisme

A causa del fet que, de manera predeterminada, les crides Ajax són asíncrones, la resposta del servidor no està disponible de manera immediata. Per exemple, el codi següent no hauria de funcionar:

```
var response;
$.get('foo.php', function(r) { response = r; });
console.log(response); // indefinit (undefined)
```

En el seu lloc, cal especificar una funció de devolució de crida; aquesta funció s'executarà quan la petició s'hagi fet de manera correcta, ja que és en aquest moment quan la resposta del servidor està llesta.

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

7.2.4. Polítiques del mateix origen i JSONP

En general, les peticions Ajax estan limitades a utilitzar el mateix protocol (*http* o *https*), el mateix port i el mateix domini d'origen. Aquesta limitació no s'aplica als *scripts* carregats a través del mètode Ajax de jQuery.

L'altra excepció és quan es fa una petició que rebrà una resposta en format JSONP. En aquest cas, el proveïdor de la resposta ha de respondre la petició amb un `script` que es pot carregar utilitzant l'etiqueta `<script>`, i evitar així la limitació de fer peticions des del mateix domini. Aquesta resposta contindrà la informació sol·licitada, continguda en una funció.

7.2.5. Ajax i Firebug

Firebug (o l'inspector WebKit que ve inclòs en Chrome o Safari) són eines imprescindibles per a treballar amb peticions Ajax, ja que es poden observar des de la pestanya Consola de Firebug (o anant a Recursos > Panell XHR des de l'inspector de Webkit) i revisar els detalls d'aquestes peticions. Si alguna cosa està fallant quan es treballa amb Ajax, aquest és el primer lloc on anar per a saber quin és el problema.

7.3. Mètodes Ajax de jQuery

Com s'ha indicat anteriorment, jQuery posseeix diversos mètodes per a treballar amb Ajax. No obstant això, tots estan basats en el mètode `$.ajax`; per tant, la seva comprensió és obligatòria. A continuació, es tractarà aquest mètode i després es farà un breu resum dels altres mètodes.

Generalment, és preferible utilitzar el mètode $.ajax en lloc dels altres, ja que ofereix més característiques i la seva configuració és molt comprensible.

7.3.1. $.ajax

El mètode `$.ajax` es configura a través d'un objecte, el qual conté totes les instruccions que necessita jQuery per a completar la petició. Aquest mètode és particularment útil, ja que ofereix la possibilitat d'especificar accions en cas que la petició hagi fallat o no. A més, com que està configurat a través d'un objecte, se'n poden definir les propietats de manera separada, i això fa que sigui més fàcil la reutilització del codi. Podeu visitar http://api.jquery.com/jquery.ajax/ per a consultar la documentació sobre les opcions disponibles en el mètode.

Utilitzar el mètode $.ajax

```
$.ajax({
   // la URL per a la petició
   url : 'post.php',

   // la informació que s'ha d'enviar
   // (també es pot utilitzar una cadena de dades)
   data : { id : 123 },

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

   // el tipus d'informació que s'espera de resposta
   dataType : 'json',

   // codi que s'ha d'executar si la petició és satisfactòria;
   // la resposta és passada com a argument a la funció
   success : function(json) {
      $('<h1/>').text(json.title).appendTo('body');
      $('<div class="content"/>')
         .html(json.html).appendTo('body');
   },

   // codi que s'ha d'executar si la petició falla;
   // són passats com a arguments a la funció
   // l'objecte jqXHR (extensió de XMLHttpRequest), un text amb l'estatus
   // de la petició i un text amb la descripció de l'error que hagi donat el servidor
   error : function(jqXHR, status, error) {
      alert('Disculpeu, hi ha un problema');
   },

   // codi que s'ha d'executar sense que importi si la petició va fallar o no
   complete : function(jqXHR, status) {
      alert('Petició realitzada');
   }
```

```
});
```

Nota

Un aclariment sobre el paràmetre `dataType`. Si el servidor retorna informació que és diferent del format especificat, el codi fallarà, i la raó de per què ho fa no sempre quedarà clara a causa que la resposta HTTP no mostrarà cap tipus d'error. Quan estigueu treballant amb peticions Ajax, us heu d'assegurar que el servidor està enviant el tipus d'informació que esteu sol·licitant i verificar que la capçalera `Content-type` és exacta al tipus de dada. Per exemple, per a informació en format JSON, la capçalera `Content-type` hauria de ser `application/json`.

Opcions del mètode $.ajax

El mètode `$.ajax` posseeix moltes opcions de configuració, i és justament aquesta característica la que fa que sigui un mètode molt útil. A continuació es mostren les més comunes:

Enllaç d'interès

Per a una llista completa de les opcions disponibles, podeu consultar http://api.jquery.com/jquery.ajax/.

- **async**. Estableix si la petició serà asíncrona o no. De manera predeterminada el valor és `true`. Heu de tenir en compte que, si l'opció s'estableix en `false`, la petició bloquejarà l'execució d'altres codis fins que aquesta petició hagi finalitzat.

- **cache**. Estableix si la petició es desarà en la memòria cau (*caché)* del navegador. De manera predeterminada és `true` per a tots els *dataType* excepte per a *script* i *jsonp*. Quan posseeix el valor `false`, s'agrega una cadena de caràcters *anticaché* al final de la URL de la petició.

- **complete**. Estableix una funció de devolució de crida que s'executa quan la petició està completa, encara que hagi fallat o no. La funció rep com a arguments l'objecte `jqXHR` (en versions anteriors o iguals a jQuery 1.4, rep en el seu lloc l'objecte de la petició en cru `XMLHTTPRequest`) i un text en què s'especifica l'estatus de la mateixa petició (`success`, `notmodified`, `error`, `timeout`, `abort`, o `parsererror`).

- **context**. Estableix l'àmbit en què la funció o les funcions de devolució de crida s'executaran (per exemple, defineix el significat de `this` dins de les funcions). De manera predeterminada `this` fa referència a l'objecte originàriament passat al mètode `$.ajax`.

- **data**. Estableix la informació que s'enviarà al servidor. Aquesta pot ser tant un objecte com una cadena de dades (per exemple,`foo=bar&baz=bim`).

- **dataType**. Estableix el tipus d'informació que s'espera rebre com a resposta del servidor. Si no s'hi especifica cap valor, de manera predeterminada jQuery revisa el tipus de *MIME* que posseeix la resposta.

- **error**. Estableix una funció de devolució de crida a executar si resulta algun error en la petició. Aquesta funció rep com a arguments l'objecte `jqXHR` (en versions anteriors o iguals a jQuery 1.4, rep en el seu lloc l'objecte de la petició en cru `XMLHTTPRequest`), un text en què s'especifica l'estatus

de la mateixa petició (`timeout`, `error`, `abort`, o `parsererror`) i un text amb la descripció de l'error que hagi enviat el servidor (per exemple, `Not Found` o `Internal Server Error`).

- **jsonp**. Estableix el nom de la funció de devolució de crida que s'ha d'enviar quan es fa una petició *JSONP*. De manera predeterminada el nom és *callback*.

- **success**. Estableix una funció a executar si la petició ha estat satisfactòria. Aquesta funció rep com a arguments l'objecte `jqXHR` (en versions anteriors o iguals a jQuery 1.4, rep en el seu lloc l'objecte de la petició en cru `XMLHTTPRequest`), un text en què s'especifica l'estatus de la mateixa petició, la informació de la petició (convertida a objecte JavaScript en cas que *dataType* sigui *JSON*) i l'estatus d'aquesta.

- **timeout**. Estableix un temps en mil·lisegons per a considerar una petició com a fallada.

- **traditional**. Si el seu valor és *true*, s'utilitza l'estil de serialització de dades utilitzat abans de jQuery 1.4. Per a obtenir-ne més detalls, podeu visitar http://api.jquery.com/jquery.param/.

- **type**. De manera predeterminada el seu valor és «GET». Es poden utilitzar també altres tipus de peticions (com PUT i DELETE); no obstant això, poden no estar suportats per tots els navegadors.

- **url**. Estableix la URL on es fa la petició.

L'opció `url` és obligatòria per al mètode `$.ajax`;

Nota

A partir de la versió 1.5 de jQuery, les opcions `beforeSend`, `success`, `error` i `complete` reben, com un dels seus arguments, l'objecte `jqXHR`, que és una extensió de l'objecte natiu `XMLHTTPRequest`. L'objecte `jqXHR` posseeix una sèrie de mètodes i propietats que permeten modificar o obtenir informació particular de la petició que s'ha de fer, com per exemple sobreescriure el tipus de *MIME* que posseeix la resposta que s'espera per part del servidor.

Enllaç d'interès

Per a obtenir informació sobre l'objecte `jqXHR` podeu consultar http://api.jquery.com/jquery.ajax/#jqXHR.

Nota

A partir de la versió 1.5 de jQuery, les opcions `success`, `error` i `complete` poden rebre un vector amb diverses funcions de devolució, les quals seran executades en torns.

7.3.2. Mètodes convenients

En cas que no es vulgui utilitzar el mètode `$.ajax`, i no es necessitin els controladors d'errors, hi ha altres mètodes més convenients per a fer peticions Ajax (encara que, com s'ha indicat més amunt, aquests estan basats en el mètode `$.ajax` amb valors preestablerts de configuració).

Els mètodes de què proveeix la biblioteca són:

- **$.get**. Efectua una petició GET a una URL proveïda.

- **$.post**. Efectua una petició POST a una URL proveïda.

- **$.getScript**. Afegeix un *script* a la pàgina.

- **$.getJSON**. Efectua una petició GET a una URL proveïda i espera que una dada JSON sigui retornada.

Els mètodes han de tenir els arguments següents, en ordre:

- **url**. La URL on es farà la petició. El seu valor és obligatori.

- **data**. La informació que s'enviarà al servidor. El seu valor és opcional i pot ser tant un objecte com una cadena de dades (com `foo=bar&baz=bim`). Aquesta opció no és vàlida per al mètode `$.getScript`.

- **success callback**. Una funció opcional que s'executa en cas que la petició hagi estat satisfactòria. Aquesta funció rep com a arguments la informació de la petició i l'objecte en brut d'aquesta petició.

- **data type**. El tipus de dada que s'espera rebre des del servidor. El seu valor és opcional. Aquesta opció només és aplicable per a mètodes en què no està especificat el tipus de dada en el nom del mateix mètode.

Utilitzar mètodes convenients per a peticions Ajax

```
// obté text pla o html
$.get('/users.php', { userId : 1234 }, function(resp) {
   console.log(resp);
});

// afegeix un script a la pàgina i després executa la funció especificada
$.getScript('/static/js/myScript.js', function() {
   functionFromMyScript();
});

// obté informació en format JSON des del servidor
$.getJSON('/details.php', function(resp) {
   $.each(resp, function(k, v) {
      console.log(k + ' : ' + v);
   });
});
```

7.3.3. $.fn.load

El mètode `$.fn.load` és l'únic que es pot cridar des d'una selecció. Aquest mètode obté el codi HTML d'una URL i omple els elements seleccionats amb la informació obtinguda. En conjunt amb la URL, es pot especificar opcionalment un selector, el qual obtindrà el codi especificat en aquesta selecció.

Utilitzar el mètode `$.fn.load` per a omplir un element

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

Utilitzar el mètode `$.fn.load` per a omplir un element basat en un selector

```
$('#newContent').load('/foo.html #myDiv h1:first', function(html) {
   alert('Contingut actualitzat');
});
```

7.4. Ajax i formularis

Les capacitats de jQuery amb Ajax poden ser especialment útils per al treball amb formularis. Per exemple, l'extensió jQuery Form Plugin és una extensió per a afegir capacitats Ajax a formularis. Hi ha dos mètodes que cal conèixer a l'hora de fer aquest tipus de treballs: `$.fn.serialize` i `$.fn.serializeArray`.

Transformar informació d'un formulari a una cadena de dades

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

Crear un vector d'objectes contenint informació d'un formulari

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

// crea una estructura com aquesta:
[
   { name : 'field1', value : 123 },
   { name : 'field2', value : 'hello world' }
]
```

7.5. Treballar amb JSONP

En els últims temps, la introducció de JSONP ha permès la creació d'aplicacions híbrides de continguts. Molts llocs importants ofereixen JSONP com a servei d'informació, al qual s'accedeix a través d'una API (en anglès, *application pro-*

gramming interface) predefinida. Un servei particular que permet obtenir informació en format JSONP és Yahoo! Query Language, el qual s'utilitza a continuació per a obtenir, per exemple, notícies sobre gats.

Utilitzar YQL i JSONP

```
$.ajax({
   url : 'http://query.yahooapis.com/v1/public/yql',

   // s'agrega com a paràmetre el nom de la funció de devolució,
   // segons que s'especifica en el servei de YQL
   jsonp : 'callback',

   // se li indica a jQuery que s'espera informació en format JSONP
   dataType : 'jsonp',

   // se li indica al servei de YQL quina és la informació
   // que es vol i que es vol en format JSON
   data : {
      q : 'select title,abstract,url from search.news where query="cat"',
      format : 'json'
   },

   // s'executa una funció, ja que és satisfactòria la petició
   success : function(response) {
      console.log(response);
   }
});
```

jQuery s'encarrega de solucionar tots els aspectes complexos de la petició JSONP. L'únic que cal fer és especificar el nom de la funció de devolució (en aquest cas, *callback*, segons el que especifica YQL) i el resultat final serà com una petició Ajax normal.

7.6. Esdeveniments Ajax

Sovint, voldreu executar una funció quan una petició hagi començat o acabat, com per exemple, mostrar o ocultar un indicador. En lloc de definir aquestes funcions dins de cada petició, jQuery proveeix de la possibilitat de vincular esdeveniments Ajax a elements seleccionats.

Enllaç d'interès

Per a una llista completa d'esdeveniments Ajax, es pot consultar http://docs.jquery.com/ajax_events.

Mostrar/ocultar un indicador utilitzant esdeveniments Ajax

```
$('#loading_indicator')
   .ajaxStart(function() { $(this).show(); })
   .ajaxStop(function() { $(this).hide(); });
```

7.7. Exercicis

7.7.1. Carregar contingut extern

Obriu l'arxiu `/ejercicios/index.html` en el navegador. Feu l'exercici utilitzant l'arxiu `/ejercicios/js/load.js`. La tasca és carregar el contingut d'un article de blog quan l'usuari faci clic en el títol de l'ítem.

1) Creeu un element *div* després de cada títol d'article de blog i desar una referència cap a ells en l'element de títol utilitzant `$.fn.data`.

2) Vinculeu un esdeveniment clic al títol, el qual utilitzarà el mètode `$.fn.load` per carregar en cada *div* creat el contingut apropiat des de l'arxiu `/ejercicios/data/blog.html`. No oblideu deshabilitar el comportament predeterminat de l'esdeveniment clic.

Cada títol d'article de blog en `index.html` inclou un enllaç cap a l'article. Caldrà aprofitar l'atribut *href* de cada enllaç per a obtenir el contingut propi de blog.html. Una vegada obtingut el valor de l'atribut, es pot utilitzar la següent forma per a processar la informació i convertir-la en un selector per a utilitzar en conjunt amb `$.fn.load`:

```
var href = 'blog.html#post1';
var tempArray = href.split('#');
var id = '#' + tempArray[1];
```

Recordeu utilitzar `console.log` per a assegurar-vos que esteu fent el correcte.

7.7.2. Carregar contingut utilitzant JSON

Obriu l'arxiu `/ejercicios/index.html` en el navegador. Feu l'exercici utilitzant l'arxiu `/ejercicios/js/specials.js`. La tasca és mostrar els detalls de l'usuari per a un dia determinat quan se selecciona des de la llista desplegable.

1) Afegiu un element *div* després del formulari que es troba dins de l'element *#specials*; aquest serà el lloc en què es col·locarà la informació que es vol obtenir.

2) Vinculeu l'esdeveniment *change* en l'element *select*; quan es fa un canvi en la selecció, s'ha d'enviar una petició Ajax a `/ejercicios/data/specials.json`.

3) Quan la petició retorna una resposta, cal utilitzar el valor seleccionat en el *select* (ajuda: `$.fn.val`) per a buscar la informació corresponent en la resposta JSON.

4) Afegiu algun HTML amb la informació obtinguda en *el div* creat anteriorment.

5) Finalment, removeu el botó *submit* del formulari.

Cada vegada que la selecció canvia, es fa una petició Ajax. Com canviaria el codi per a fer només una petició i desar la informació per tal d'aprofitar-la quan es torna a canviar l'opció seleccionada?

8. Extensions

8.1. Què és una extensió?

Una extensió de jQuery és simplement un nou mètode que s'utilitzarà per a estendre el prototip (*prototype*) de l'objecte jQuery. Quan s'estén el prototip, tots els objectes jQuery hereten els mètodes afegits. Per tant, quan es fa una crida `jQuery()`, es crea un nou objecte jQuery amb tots els mètodes heretats.

L'objectiu d'una extensió és fer una acció utilitzant una col·lecció d'elements, de la mateixa manera que ho fan, per exemple, els mètodes `fadeOut` o `addClass` de la biblioteca.

Podeu fer les vostres pròpies extensions i utilitzar-les de manera privada en el vostre projecte, o també les podeu publicar perquè altres persones en treguin profit.

8.2. Crear una extensió bàsica

El codi per a fer una extensió bàsica és el següent:

```
(function($){
   $.fn.myNewPlugin = function() {
      return this.each(function(){
         // fer alguna cosa
      });
   };
})(jQuery);
```

L'extensió del prototip de l'objecte jQuery ocorre en la línia següent:

```
$.fn.myNewPlugin = function() { //...
```

La qual és tancada en una funció autoexecutable:

```
(function($){
   //...
})(jQuery);
```

Aquesta té l'avantatge de crear un àmbit «privat», i permet utilitzar el signe dòlar sense preocupar-se que una altra biblioteca també estigui utilitzant aquest signe.

Ara com ara, internament l'extensió queda:

```
$.fn.myNewPlugin = function() {
   return this.each(function(){
      // fer alguna cosa
   });
};
```

Dins d'ella, la paraula clau `this` fa referència a l'objecte jQuery, en què l'extensió és anomenada.

```
var somejQueryObject = $('#something');

$.fn.myNewPlugin = function() {
   alert(this === somejQueryObject);
};

somejQueryObject.myNewPlugin(); // mostra una alerta amb 'true'
```

L'objecte jQuery, normalment, contindrà referències a diversos elements DOM; és per això que sovint s'hi refereix com una col·lecció.

Per a interactuar amb la col·lecció d'elements, cal fer un bucle, el qual s'aconsegueix fàcilment amb el mètode `each()`:

```
$.fn.myNewPlugin = function() {
   return this.each(function(){

   });
};
```

Igual que altres mètodes, `each()` retorna un objecte jQuery, i permet utilitzar l'encadenament de mètodes (`$(...).css().attr()...`). Per a no trencar aquesta convenció, l'extensió que s'haurà de crear haurà de retornar l'objecte `this`, per a permetre seguir amb l'encadenament. A continuació, se'n mostra un petit exemple:

```
(function($){
   $.fn.showLinkLocation = function() {
      return this.filter('a').each(function(){
         $(this).append(
            ' (' + $(this).attr('href') + ')'
         );
      });
   };
})(jQuery);
```

```
// Exemple d'utilització:
$('a').showLinkLocation();
```

L'extensió modificarà tots els enllaços dins de la col·lecció d'elements i hi afegirà el valor del seu atribut `href` entre parèntesis.

```
<!-- Abans que l'extensió sigui crida: -->
<a href="page.html">Foo</a>

<!-- Després que l'extensió sigui crida: -->
<a href="page.html">Foo (page.html)</a>
```

També es pot optimitzar l'extensió:

```
(function($){
   $.fn.showLinkLocation = function() {
      return this.filter('a').append(function(){
         return ' (' + this.href + ')';
      });
   };
})(jQuery);
```

El mètode `append` permet especificar una funció de devolució de crida, i el valor retornat determinarà què és el que s'afegirà a cada element. Observeu també que no s'utilitza el mètode `attr` a causa del fet que l'API nativa del DOM permet un accés fàcil a la propietat `href`.

A continuació, es mostra un altre exemple d'extensió. En aquest cas, no es requereix fer un bucle en cada element, ja que es delega la funcionalitat directament en un altre mètode jQuery:

```
(function($){
   $.fn.fadeInAndAddClass = function(duration, className) {
      return this.fadeIn(duration, function(){
         $(this).addClass(className);
      });
   };
})(jQuery);

// Exemple d'utilització:
$('a').fadeInAndAddClass(400, 'finishedFading');
```

8.3. Trobar i avaluar extensions

Un dels aspectes més populars de jQuery és la diversitat d'extensions que hi ha.

No obstant això, la qualitat entre extensions pot variar enormement. Moltes són intensivament provades i ben mantingudes, però altres són creades de manera precipitada i després ignorades, sense seguir bones pràctiques.

Google és la millor eina per a trobar extensions (encara que l'equip de jQuery estigui treballant per a millorar el seu repositori d'extensions). Una vegada trobada l'extensió, potser voldreu consultar la llista de correus de jQuery o el canal IRC #jquery per a obtenir l'opinió d'altres persones sobre aquesta extensió.

Cal assegurar-se que l'extensió està ben documentada i que s'hi ofereixen exemples de la seva utilització. També cala anar amb compte amb les extensions que fan més del que necessiteu; aquestes poden arribar a sobrecarregar la vostra pàgina.

Lectura recomanada

Per a obtenir més consells sobre como detectar una extensió mediocre, podeu llegir l'article (en anglès) «Signs of a poorly written jQuery plugin» de Remy Sharp.

Una vegada seleccionada l'extensió, s'haurà d'afegir a la vostra pàgina. Primer, descarregueu l'extensió, descomprimiu-la (si cal) i moveu-la a la carpeta de la vostra aplicació. Finalment, inseriu-la utilitzant l'element *script* (després de la inclusió de jQuery).

8.4. Escriure extensions

De vegades, es vol fer una funcionalitat disponible en tot el codi, per exemple, un mètode que pugui ser cridat des d'una selecció que faci una sèrie d'operacions.

La majoria de les extensions són mètodes creats dins de l'espai de noms `$.fn`. jQuery garanteix que un mètode anomenat sobre l'objecte jQuery sigui capaç d'accedir a aquest objecte a través de `this`. En contrapartida, l'extensió ha de garantir retornar el mateix objecte rebut (tret que s'hi especifiqui el contrari).

A continuació, se'n mostra un exemple:

Crear una extensió per a afegir i remoure una classe en un element en succeir l'esdeveniment *hover*

```
// definició de l'extensió
(function($){
   $.fn.hoverClass = function(c) {
      return this.hover(
         function() { $(this).toggleClass(c); }
      );
   };
})(jQuery);

// utilitzar l'extensió
$('li').hoverClass('hover');
```

Lectura recomanada

Per a obtenir més informació sobre el desenvolupament d'extensions, podeu consultar l'article (en anglès) «A Plugin Development Pattern» de Mike Alsup. En aquest article, es desenvolupa una extensió anomenada `$.fn.hilight`, la qual proveeix de suport per a l'extensió metadata (en cas que hi sigui present) i proveeix d'un mètode descentralitzat per a establir opcions globals o d'instàncies de l'extensió.

El patró de desenvolupament d'extensions per a jQuery explicat per Mike Alsup

```
//
// crear una clausura
//
(function($) {
   //
   // definició de l'extensió
   //
   $.fn.hilight = function(options) {
      debug(this);
      // generació de les opcions principals abans d'interactuar
      var opts = $.extend({}, $.fn.hilight.defaults, options);
      // s'interactua i es formata cada element
      return this.each(function() {
         $this = $(this);
         // generació de les opcions específiques de cada element
         var o = $.meta ? $.extend({}, opts, $this.data()) : opts;
         // actualització dels estils de cada element
         $this.css({
            backgroundColor: o.background,
            color: o.foreground
         });
         var markup = $this.html();
         // es crida la funció de formatació
         markup = $.fn.hilight.format(markup);
         $this.html(markup);
      });
   };
   //
   // funció privada per fer depuració
   //
   function debug($obj) {
      if (window.console && window.console.log)
         window.console.log('hilight selection count: ' + $obj.size());
   };
   //
   // definir i exposar la funció de formatació
   //
   $.fn.hilight.format = function(txt) {
      return '<strong>' + txt + '</strong>';
```

```
    };
    //
    // opcions predeterminades
    //
    $.fn.hilight.defaults = {
       foreground: 'red',
       background: 'yellow'
    };
//
// fi de la clausura
//
})(jQuery);
```

8.5. Escriure extensions amb manteniment d'estat utilitzant Widget Factory de jQuery UI

Nota

Aquest subapartat està basat, amb permís de l'autor, en l'article «Building Stateful jQuery Plugins» de Scott Gonzalez.

Mentre que la majoria de les extensions per a jQuery són sense manteniment d'estat (en anglès, *stateless*), és a dir, extensions que s'executen només sobre un element, sent aquesta la seva única interacció, hi ha un gran conjunt de funcionalitats que no s'aprofiten en el patró bàsic amb què es desenvolupen les extensions.

Amb la finalitat d'omplir aquest buit, jQuery UI (jQuery User Interface) ha implementat un sistema més avançat d'extensions. Aquest sistema permet manejar estats i admet múltiples funcions per a ser exposades en una única extensió. És anomenat Widget Factory i forma part de la versió 1.8 de jQuery UI a través de `jQuery.widget`, encara que també es pot utilitzar sense dependre de jQuery UI.

Per a demostrar les capacitats de Widget Factory, es crearà una extensió que tindrà com a funcionalitat ser una barra de progrés.

Ara com ara, l'extensió només permetrà establir el valor de la barra de progrés una sola vegada. Això es farà cridant a `jQuery.widget` amb dos paràmetres: el nom de l'extensió a crear i un objecte literal que contindrà les funcions suportades per l'extensió. Quan l'extensió és anomenada, es crea una instància d'aquesta i totes les funcions s'executaran en el context d'aquesta instància.

Hi ha dues diferències importants en comparació amb una extensió estàndard de jQuery: en primer lloc, el context és un objecte, no un element Dg. En segon lloc, el context sempre és un únic objecte, mai una col·lecció.

Una simple extensió amb manteniment d'estat utilitzant Widget Factory de jQuery UI

```
$.widget("nmk.progressbar", {
   _create: function() {
```

```
        var progress = this.options.value + "%";
        this.element
            .addClass("progressbar")
            .text(progress);
    }
});
```

El nom de l'extensió ha de contenir un espai de noms; en aquest cas, s'utilitza `nmk`. Els espais de noms tenen una limitació d'un sol nivell de profunditat, és a dir, que per exemple no es pot utilitzar `nmk.foo`. Com es pot veure en l'exemple, Widget Factory proveeix de dues propietats per a ser utilitzades. La primera, `this.element`, és un objecte jQuery que conté exactament un element. En cas que l'extensió sigui executada en més d'un element, es crearà una instància separada de l'extensió per a cada element i cadascuna tindrà el seu propi `this.element`. La segona propietat, `this.options`, és un conjunt de parells clau/valor amb totes les opcions de l'extensió. Aquestes opcions es poden passar a l'extensió, com es mostra a continuació.

Nota

Quan estigueu fent les vostres pròpies extensions, és recomanable utilitzar el vostre propi espai de noms, ja que deixa clar d'on prové l'extensió i si és part d'una col·lecció més gran. D'altra banda, l'espai de noms `ui` està reservat per a les extensions oficials de jQuery UI.

Passar opcions al Widget

```
$("<div></div>")
    .appendTo( "body" )
    .progressbar({ value: 20 });
```

Quan es crida `jQuery.widget` s'estén a jQuery afegint el mètode a `jquery.fn` (de la mateixa manera que quan es crea una extensió estàndard). El nom de la funció que s'afegeix està basat en el nom que es passa a `jQuery.widget`, sense l'espai de noms (en aquest cas, el nom serà `jquery.fn.progressbar`).

Com es mostra a continuació, es poden especificar valors predeterminats per a qualsevol opció. Aquests valors s'haurien de basar en la utilització més comuna de l'extensió.

Establir opcions predeterminades per a un widget

```
$.widget("nmk.progressbar", {
    // opcions predeterminades
    options: {
        value: 0
    },

    _create: function() {
        var progress = this.options.value + "%";
        this.element
            .addClass( "progressbar" )
            .text( progress );
```

```
    }
});
```

8.5.1. Afegir mètodes a un widget

Ara que es pot inicialitzar l'extensió, cal afegir-hi l'habilitat de fer accions a través de mètodes definits en l'extensió. Per a definir un mètode en l'extensió, cal incloure la funció en l'objecte literal que es passa a `jQuery.widget`. També es poden definir mètodes «privats» anteposant un guió baix al nom de la funció.

Crear mètodes en el widget

```
$.widget("nmk.progressbar", {
   options: {
      value: 0
   },

   _create: function() {
      var progress = this.options.value + "%";
      this.element
         .addClass("progressbar")
         .text(progress);
   },

   // crear un mètode públic
   value: function(value) {
      // no es passa cap valor, llavors actua com a mètode obtenidor
      if (value === undefined) {
         return this.options.value;
      // es passa un valor, llavors actua com a mètode establidor
      } else {
         this.options.value = this._constrain(value);
         var progress = this.options.value + "%";
         this.element.text(progress);
      }
   },

   // crear un mètode privat
   _constrain: function(value) {
      if (value > 100) {
         value = 100;
      }
      if (value < 0) {
         value = 0;
      }
      return value;
   }
```

```
});
```

Per cridar un mètode en una instància de l'extensió, s'ha de passar el nom d'aquest mètode a l'extensió. En cas que es cridi un mètode que accepta paràmetres, aquests s'han de passar després del nom del mètode.

Cridar mètodes en una instància d'extensió

```
var bar = $("<div></div>")
   .appendTo("body")
   .progressbar({ value: 20 });

// obté el valor actual
alert(bar.progressbar("value"));

// actualitza el valor
bar.progressbar("value", 50);

// obté novament el valor
alert(bar.progressbar("value"));
```

Nota

Executar mètodes passant el nom del mètode a la mateixa funció jQuery que s'utilitza per a inicialitzar l'extensió pot semblar estrany; no obstant això, es fa així per prevenir la «contaminació» de l'espai de noms de jQuery i mantenir, al mateix temps, la capacitat de cridar mètodes en cadena.

8.5.2. Treballar amb les opcions del widget

Un dels mètodes disponibles automàticament per a l'extensió és `option`. Aquest mètode permet obtenir i establir opcions després de la inicialització i funciona exactament igual que els mètodes *attr* i *css* de jQuery: passant únicament un nom com a argument el mètode funciona com a obtenidor, mentre que passant un o més conjunts de noms i valors, el mètode funciona com a establidor. Quan és utilitzat com a mètode obtenidor, l'extensió retornarà el valor actual de l'opció corresponent al nom passat com a argument. D'altra banda, quan és utilitzat com un mètode establidor, el mètode `_setOption` de l'extensió serà cridat per cada opció que es vol establir.

Respondre quan una opció és establerta

```
$.widget("nmk.progressbar", {
   options: {
      value: 0
   },

   _create: function() {
      this.element.addClass("progressbar");
```

```
        this._update();
    },

    _setOption: function(key, value) {
        this.options[key] = value;
        this._update();
    },

    _update: function() {
        var progress = this.options.value + "%";
        this.element.text(progress);
    }
});
```

8.5.3. Afegir funcions de devolució de crida

Una de les maneres més fàcils d'estendre una extensió és afegir funcions de devolució de crida perquè, d'aquesta manera, l'usuari pugui reaccionar quan l'estat de l'extensió canviï. A continuació, es mostrarà com afegir una funció de devolució de crida a l'extensió creada per a indicar quan la barra de progrés arribi al 100%. El mètode `_trigger` obté tres paràmetres: el nom de la funció de devolució, l'objecte d'esdeveniment natiu que inicialitza la funció de devolució, i un conjunt d'informació rellevant de l'esdeveniment. El nom de la funció de devolució és l'únic paràmetre obligatori, però els altres poden ser molt útils si l'usuari vol implementar funcionalitats personalitzades.

Proveir funcions de devolució de crida

```
$.widget("nmk.progressbar", {
    options: {
        value: 0
    },

    _create: function() {
        this.element.addClass("progressbar");
        this._update();
    },

    _setOption: function(key, value) {
        this.options[key] = value;
        this._update();
    },

    _update: function() {
        var progress = this.options.value + "%";
        this.element.text(progress);
        if (this.options.value == 100) {
```

```
            this._trigger("complete", null, { value: 100 });
        }
    }
});
```

Les funcions de devolució són essencialment només opcions addicionals, per la qual cosa, es poden establir com qualsevol altra opció. Cada vegada que s'executa una funció de devolució, s'activa també un esdeveniment corresponent. El tipus d'esdeveniment es determina mitjançant la concatenació del nom de l'extensió i el nom de la funció de devolució. Aquesta funció i esdeveniment reben dos mateixos paràmetres: un objecte d'esdeveniment i un conjunt d'informació rellevant de l'esdeveniment.

Si l'extensió tindrà alguna funcionalitat que podrà ser cancel·lada per l'usuari, la millor manera de fer-ho és creant funcions de devolució cancel·lables. L'usuari podrà cancel·lar una funció de devolució o l'esdeveniment que hi estigui associat de la mateixa manera que es cancel·la qualsevol esdeveniment natiu: cridant `event.preventDefault()` o utilitzant `return false`.

Vincular esdeveniments del widget

```
var bar = $("<div></div>")
    .appendTo("body")
    .progressbar({
        complete: function(event, data) {
            alert( "Funció de devolució" );
        }
    })
    .on("progressbarcomplete", function(event, data) {
        alert("El valor de la barra de progrés és " + data.value);
    });

bar.progressbar("option", "value", 100);
```

En profunditat: Widget Factory

Quan es crida `jQuery.widget`, aquesta crea una funció constructora per a l'extensió i estableix l'objecte literal que es passa com el prototip per a totes les instàncies de l'extensió. Totes les funcionalitats que automàticament s'afegeixen a l'extensió provenen del prototip base del widget, el qual és definit com `jQuery.Widget.prototype`. Quan es crea una instància de l'extensió, es desa en l'element DOM original utilitzant `jQuery.data`, amb el nom de l'extensió com a paraula clau.

A causa del fet que la instància de l'extensió està directament vinculada a l'element DOM, es pot accedir a la instància de l'extensió de manera directa. Això permet cridar mètodes directament en la instància de l'extensió en lloc de passar el nom del mètode com una cadena de caràcters, i donar la possibilitat d'accedir a les propietats de l'extensió.

```
var bar = $("<div></div>")
   .appendTo("body")
   .progressbar()
   .data("progressbar" );

// cridar un mètode directament en la instància de l'extensió
bar.option("value", 50);

// accedir a propietats en la instància de l'extensió
alert(bar.options.value);
```

Un dels beneficis més importants de tenir un constructor i un prototip per a una extensió és la facilitat d'estendre l'extensió. El fet d'afegir o canviar mètodes en el prototip de l'extensió també permet modificar-los en totes les instàncies de l'extensió. Per exemple, si volem afegir un mètode a l'extensió de barra de progrés per a permetre restablir el progrés a 0%, es pot fer afegint aquest mètode al prototip i, automàticament, estarà disponible per ser cridada des de qualsevol instància de l'extensió.

```
$.nmk.progressbar.prototype.reset = function() {
   this._setOption("value", 0);
};
```

8.5.4. Neteja

En alguns casos, tindrà sentit permetre als usuaris aplicar i desaplicar l'extensió. Això es pot fer a través del mètode `destroy`. Amb aquest mètode, es pot desfer tot el que s'ha fet amb l'extensió. També aquest és cridat automàticament si l'element vinculat a l'extensió és eliminat del DOM (per la qual cosa també es pot utilitzar per a la «recol·lecció d'escombraries»). El mètode `destroy` predeterminat remou el vincle entre l'element DOM i la instància de l'extensió.

Afegir un mètode destroy al widget

```
$.widget( "nmk.progressbar", {
   options: {
      value: 0
   },

   _create: function() {
```

```
        this.element.addClass("progressbar");
        this._update();
    },

    _setOption: function(key, value) {
        this.options[key] = value;
        this._update();
    },

    _update: function() {
        var progress = this.options.value + "%";
        this.element.text(progress);
        if (this.options.value == 100 ) {
            this._trigger("complete", null, { value: 100 });
        }
    },

    destroy: function() {
        this.element
            .removeClass("progressbar")
            .text("");

        // crida la funció base destroy
        $.Widget.prototype.destroy.call(this);
    }
});
```

8.5.5. Conclusió

La utilització de Widget Factory és només una manera de crear extensions amb manteniment d'estat. Es poden utilitzar alguns models diferents, i cadascun té els seus avantatges i desavantatges. Widget Factory resol molts problemes comuns i millora significativament la productivitat i la reutilització de codi.

8.6. Exercicis

8.6.1. Fer una taula ordenable

Aquest exercici consisteix a identificar, descarregar i implementar una extensió que permeti ordenar la taula existent a la pàgina index.html. Quan estigui llest, totes les columnes de la taula han de poder ser ordenables.

8.6.2. Escriure una extensió per a canviar el color del fons en les taules

Obriu l'arxiu `/ejercicios/index.html` en el navegador. Feu l'exercici utilitzant l'arxiu `/ejercicios/js/stripe.js`. La tasca és escriure una extensió anomenada «stripe», que podrà ser cridada des de qualsevol element *table* i haurà de canviar el color de fons de les files imparells en el cos de la taula. El color podrà ser especificat com a paràmetre de l'extensió.

```
$('#myTable').stripe('#cccccc');
```

No oblideu retornar la taula perquè es puguin encadenar altres mètodes després de la crida a l'extensió.

9. Millors pràctiques per a augmentar el rendiment

Aquest apartat cobreix nombroses millors pràctiques de JavaScript i jQuery, sense un ordre en particular. Moltes d'aquestes pràctiques estan basades en la presentació «jQuery Anti-Patterns for Performance» (en anglès) de Paul Irish.

9.1. Desar la longitud en bucles

En un bucle, no cal accedir a la longitud d'un vector cada vegada que s'avalua la condició; aquest valor es pot desar prèviament en una variable.

```
var myLength = myArray.length;

for (var i = 0; i < myLength; i++) {
   // do stuff
}
```

9.2. Afegir nou contingut per fora d'un bucle

Si heu d'inserir molts elements en el DOM, feu-ho d'una sola vegada, no una per vegada.

```
// malament
$.each(myArray, function(i, item) {
   var newListItem = '<li>' + item + '</li>';
   $('#ballers').append(newListItem);
});

// millor: feu això
var frag = document.createDocumentFragment();

$.each(myArray, function(i, item) {
   var newListItem = '<li>' + item + '</li>';
   frag.appendChild(newListItem);
});
$('#ballers')[0].appendChild(frag);

// o això:
var myHtml = '';

$.each(myArray, function(i, item) {
   myHtml += '<li>' + item + '</li>';
});
```

```
$('#ballers').html(myHtml);
```

9.3. No us repetiu

No us repetiu. Feu les coses una vegada i només una; en cas contrari, ho estareu fent malament.

```
// MALAMENT
if ($eventfade.data('currently') != 'showing') {
   $eventfade.stop();
}

if ($eventhover.data('currently') != 'showing') {
   $eventhover.stop();
}

if ($spans.data('currently') != 'showing') {
   $spans.stop();
}

// BÉ
var $elems = [$eventfade, $eventhover, $spans];
$.each($elems, function(i,elem) {
   if (elem.data('currently') != 'showing') {
      elem.stop();
   }
});
```

9.4. Compte amb les funcions anònimes

No és aconsellable utilitzar en gran manera les funcions anònimes. Aquestes són difícils de depurar, mantenir, provar o reutilitzar. En el seu lloc, utilitzeu un objecte literal per a organitzar i anomenar els seus controladors i funcions de devolució de crida.

```
// MALAMENT
$(document).ready(function() {
   $('#magic').click(function(e) {
      $('#yayeffects').slideUp(function() {
         // ...
      });
   });

   $('#happiness').load(url + ' #unicorns', function() {
      // ...
   });
});
```

```
// MILLOR
var PI = {
   onReady : function() {
      $('#magic').click(PI.candyMtn);
      $('#happiness').load(PI.url + ' #unicorns', PI.unicornCb);
   },

   candyMtn : function(e) {
      $('#yayeffects').slideUp(PI.slideCb);
   },

   slideCb : function() { ... },

   unicornCb : function() { ... }
   };

$(document).ready(PI.onReady);
```

9.5. Optimització de selectors

L'optimització de selectors és menys important del que acostumava a ser, a causa de la implementació en alguns navegadors de `document.querySelectorAll()`, passant la càrrega de jQuery cap al navegador. No obstant això, cal tenir en compte alguns consells.

9.5.1. Selectors basats en ID

Sempre és millor començar les seleccions amb un ID.

```
// ràpid
$('#container div.robotarm');

// súper-ràpid
$('#container').find('div.robotarm');
```

L'exemple que utilitza `$.fn.find` és més ràpid a causa del fet que la primera selecció utilitza el motor de selecció intern Sizzle, mentre que la selecció efectuada únicament per ID utilitza `document.getElementById()`, el qual és extremament ràpid perquè és una funció nativa del navegador.

9.5.2. Especificitat

Intenteu ser específics per al costat dret de la selecció i menys específics per a l'esquerre.

```
// no optimitzat
```

```
$('div.data .gonzalez');

// optimitzat
$('.data td.gonzalez');
```

Feu servir, en la mesura que pugueu, `etiqueta.classe` al costat dret de la selecció, i solament `etiqueta` o `.classe` a la part esquerra.

```
$('.data table.attendees td.gonzalez');

// molt millor: eliminar la part mitjana ha de ser possible
$('.data td.gonzalez');
```

La segona selecció té millor rendiment, ja que travessa menys capes per a buscar l'element.

9.5.3. Evitar el selector universal

Seleccions en què s'especifica de manera implícita o explícita una selecció universal poden ser molt lentes.

```
$('.buttons > *');  // molt lent
$('.buttons').children();  // molt millor

$('.gender :radio');   // selecció universal implícita
$('.gender *:radio');  // mateixa forma, però de forma explícita
$('.gender input:radio');  // molt millor
```

9.6. Utilitzar la delegació d'esdeveniments

La delegació d'esdeveniments permet vincular un controlador d'esdeveniment a un element contenidor (per exemple, una llista desordenada) en lloc de múltiples elements continguts (per exemple, els ítems d'una llista). jQuery fa fàcil aquest treball a través de `$.fn.live` i `$.fn.delegate`. Mentre sigui possible, és recomanable utilitzar `$.fn.delegate` en lloc de `$.fn.live`, ja que elimina la necessitat d'una selecció i el seu context explícit redueix la càrrega en aproximadament un 80%.

A més, la delegació d'esdeveniments permet afegir nous elements contenidors a la pàgina sense haver de tornar a vincular els seus controladors d'esdeveniments.

```
// malament (si hi ha molts ítems en la llista)
$('li.trigger').click(handlerFn);

// millor: delegació d'esdeveniments amb $.fn.live
$('li.trigger').live('click', handlerFn);
```

```
// molt millor: delegació d'esdeveniments amb $.fn.delegate
// permet especificar un context de manera fàcil
$('#myList').delegate('li.trigger', 'click', handlerFn);
```

9.7. Separar elements per a treballar amb ells

Mentre sigui possible, cal evitar la manipulació del DOM. Per ajudar a aquest propòsit, a partir de la versió 1.4, jQuery introdueix `$.fn.detach`, que permet treballar elements de manera separada del DOM i després inserir-los.

```
var $table = $('#myTable');
var $parent = $table.parent();

$table.detach();
// ... s'afegeixen moltes cel·les a la taula
$parent.append(table);
```

9.8. Utilitzar estils en cascada per a canvis de CSS en diversos elements

Si heu de canviar el CSS en més de vint elements utilitzant `$.fn.css`, considereu fer els canvis d'estils afegint-los en una etiqueta *style*. D'aquesta manera s'incrementa un 60% el rendiment.

```
// correcte fins a 20 elements, lent en més elements
$('a.swedberg').css('color', '#asd123');
$('<style type="text/css">a.swedberg { color : #asd123 }</style>')
   .appendTo('head');
```

9.9. Utilitzar $.data en lloc de $.fn.data

Utilitzar `$.data` en un element del DOM en lloc de `$.fn.data` en una selecció pot ser fins a deu vegades més ràpid. Abans de fer-ho, cal estar segur de comprendre la diferència entre un element DOM i una selecció jQuery.

```
// regular
$(elem).data(key,value);

// 10 vegades més ràpid
$.data(elem,key,value);
```

9.10. No s'ha d'actuar en elements no existents

jQuery no us dirà si està tractant d'executar codi en una selecció buida; aquesta s'executarà com si res no estigués malament. Dependrà de vosaltres comprovar si la selecció conté elements.

```
// MALAMENT: el codi a continuació executa tres funcions
// sense comprovar si hi ha elements
// en la selecció
$('#nosuchthing').slideUp();

// Millor
var $mySelection = $('#nosuchthing');
if ($mySelection.length) { $mySelection.slideUp(); }

// MOLT MILLOR: afegir una extensió doOnce
jQuery.fn.doOnce = function(func){
   this.length && func.apply(this);
   return this;

}


$('li.cartitems').doOnce(function(){

   // fer alguna cosa

});
```

Aquest consell és especialment aplicable per a widgets de jQuery UI, els quals posseeixen molta càrrega fins i tot quan la selecció no conté elements.

9.11. Definició de variables

Les variables es poden definir en una sola declaració en lloc de diverses.

```
// antic
var test = 1;
var test2 = function() { ... };
var test3 = test2(test);

// millor manera
var test = 1,
   test2 = function() { ... },
   test3 = test2(test);
```

En funcions autoexecutables, les definicions de variables es poden passar totes juntes.

```
(function(foo, bar) { ... })(1, 2);
```

9.12. Condicionals

```
// antic
if (type == 'foo' || type == 'bar') { ... }

// millor
if (/^(foo|bar)$/.test(type)) { ... }

// cerca en objecte literal
if (({ foo : 1, bar : 1 })[type]) { ... }
```

9.13. No s'ha de tractar jQuery com si fos una caixa negra

Utilitzeu el codi font de la biblioteca com si fos la vostra documentació, i deseu l'enllaç http://bit.ly/jqsource com a marcador per tenir-lo de referència.

10. Organització del codi

10.1. Introducció

Quan s'emprèn la tasca de fer aplicacions complexes del costat del client, cal considerar la manera com s'organitzarà el codi. Aquest capítol està dedicat a analitzar alguns patrons d'organització de codi per a utilitzar en una aplicació efectuada amb jQuery. A més, s'explorarà el sistema de gestió de dependències de RequireJS.

10.1.1. Conceptes clau

Abans de començar amb els patrons d'organització de codi, és important entendre alguns conceptes clau:

- El codi ha d'estar dividit en unitats funcionals: mòduls, serveis, etc.; i s'ha d'evitar la temptació de tenir-ho tot en un únic bloc `$(document).ready()`. Aquest concepte es coneix com a encapsulació.

- No repetir codi. Identificar peces similars i utilitzar tècniques d'herència.

- Malgrat la naturalesa de jQuery, no totes les aplicacions JavaScript treballen (o tenen la necessitat de posseir una representació) en el DOM.

- Les unitats de funcionalitat han de tenir una articulació flexible (en anglès, *loosely coupled*), és a dir, una unitat de funcionalitat ha de ser capaç d'existir per si mateixa i la comunicació amb altres unitats s'ha de fer a través d'un sistema de missatges com els esdeveniments personalitzats o pub/sub. D'altra banda, sempre que es pugui, s'ha de mantenir allunyada la comunicació directa entre unitats funcionals.

El concepte d'articulació flexible pot ser especialment problemàtic per a desenvolupadors que fan la seva primera incursió en aplicacions complexes. Per tant, si esteu començant a crear aplicacions, només cal que sigueu conscients d'aquest concepte.

10.2. Encapsulació

El primer pas per a l'organització del codi és separar l'aplicació en diferents peces.

Moltes vegades, aquest esforç sol ser suficient per a mantenir el codi en ordre.

10.2.1. L'objecte literal

Un objecte literal és, potser, la manera més simple d'encapsular codi relacionat. Aquest no ofereix cap privacitat per a propietats o mètodes, però és útil per a eliminar funcions anònimes, centralitzar opcions de configuració i facilitar el camí per a la reutilització i refactorització.

Un objecte literal

```
var myFeature = {
   myProperty : 'hello',

   myMethod : function() {
      console.log(myFeature.myProperty);
   },

   init : function(settings) {
      myFeature.settings = settings;
   },

   readSettings : function() {
      console.log(myFeature.settings);
   }
};

myFeature.myProperty; // 'hello'
myFeature.myMethod(); // registra 'hello'
myFeature.init({ foo : 'bar' });
myFeature.readSettings(); // registra { foo : 'bar' }
```

L'objecte posseeix una propietat i diversos mètodes, els quals són públics (és a dir, qualsevol part de l'aplicació els pot veure). Com es pot aplicar aquest patró amb jQuery? Per exemple, en el següent codi escrit en l'estil tradicional:

```
// fent clic en un ítem de la llista es carrega cert contingut,
// després utilitzant l'ID d'aquest ítem s'oculten
// els ítems limítrofs
$(document).ready(function() {
   $('#myFeature li')
      .append('<div/>')
      .click(function() {
         var $this = $(this);
         var $div = $this.find('div');
         $div.load('foo.php?item=' +
            $this.attr('id'),
            function() {
               $div.show();
```

```
                $this.siblings()
                    .find('div').hide();
            }
        );
    });
});
```

Si l'exemple mostrat representa el 100% de l'aplicació, és convenient deixar-ho com està, ja que no val la pena fer una reestructuració. En canvi, si la peça és part d'una aplicació més gran, estaria bé separar aquesta funcionalitat d'altres no relacionades. Per exemple, és convenient moure la URL a la qual es fa la petició fora del codi i passar-la a l'àrea de configuració. També trencar la cadena de mètodes per a fer després més fàcil la modificació.

Utilitzar un objecte literal per a una funcionalitat jQuery

```
var myFeature = {
    init : function(settings) {
        myFeature.config = {
            $items : $('#myFeature li'),
            $container : $('<div class="container"></div>'),
            urlBase : '/foo.php?item='
        };

        // permet sobreescriure la configuració predeterminada
        $.extend(myFeature.config, settings);

        myFeature.setup();
    },

    setup : function() {
        myFeature.config.$items
            .each(myFeature.createContainer)
            .click(myFeature.showItem);
    },

    createContainer : function() {
        var $i = $(this),
            $c = myFeature.config.$container.clone()
                .appendTo($i);

        $i.data('container', $c);
    },

    buildUrl : function() {
        return myFeature.config.urlBase +
            myFeature.$currentItem.attr('id');
```

```
    },

    showItem : function() {
        var myFeature.$currentItem = $(this);
        myFeature.getContent(myFeature.showContent);
    },

    getContent : function(callback) {
        var url = myFeature.buildUrl();
        myFeature.$currentItem
            .data('container').load(url, callback);
    },

    showContent : function() {
        myFeature.$currentItem
            .data('container').show();
        myFeature.hideContent();
    },

    hideContent : function() {
        myFeature.$currentItem.siblings()
            .each(function() {
                $(this).data('container').hide();
            });
    }
};

$(document).ready(myFeature.init);
```

La primera característica que cal tenir en compte és que el codi és més llarg que l'original; com s'ha dit anteriorment, si aquest és l'àmbit de l'aplicació, utilitzar un objecte literal seria probablement una exageració.

Amb la nova organització, els avantatges obtinguts són:

- Separació de cada funcionalitat en petits mètodes. En un futur, si es vol canviar la manera de mostrar el contingut, serà clar on caldrà fer-ho. En el codi original, aquest pas és molt més difícil de localitzar.

- S'han eliminat els usos de funcions anònimes.

- Les opcions de configuració es mouen a una ubicació central.

- S'eliminen les limitacions que posseeixen les cadenes de mètodes, i això fa que el codi sigui més fàcil de refactoritzar, barrejar i reorganitzar.

Per les seves característiques, la utilització d'objectes literals permet una clara millora per a trams llargs de codi inserits en un bloc `$(document).ready()`. No obstant això, no són més avançats que tenir diverses declaracions de funcions dins d'un bloc `$(document).ready()`.

10.2.2. El patró modular

El patró modular supera algunes limitacions de l'objecte literal, i ofereix privacitat per a variables i funcions, i al seu torn (si es col) ofereix una API pública.

El patró modular

```
var feature =(function() {

   // variables i funcions privades
   var privateThing = 'secret',
      publicThing = 'not secret',

      changePrivateThing = function() {
         privateThing = 'super secret';
      },

      sayPrivateThing = function() {
         console.log(privateThing);
         changePrivateThing();
      };

   // API pública
   return {
      publicThing : publicThing,
      sayPrivateThing : sayPrivateThing
   }

})();

feature.publicThing; // registra 'not secret'

feature.sayPrivateThing();
// registra 'secret' i canvia el valor
// de privateThing
```

En l'exemple, s'autoexecuta una funció anònima que retorna un objecte. Dins de la funció, es defineixen algunes variables. Com que aquestes són definides dins de la funció, des de fora no s'hi té accés, tret que es posin dins de l'objecte que es retorna. Això implica que cap codi fora de la funció té accés a la variable

`privateThing` o a la funció `sayPrivateThing`. No obstant això, `sayPrivateThing` posseeix accés a `privateThing` i `changePrivateThing` ja que estan definits en el mateix àmbit.

El patró és poderós, ja que permet tenir variables i funcions privades, exposant una API limitada consistent a retornar propietats i mètodes d'un objecte.

A continuació es mostra una revisió de l'exemple vist anteriorment, amb les mateixes característiques, però exposant un únic mètode públic del mòdul, `showItemByIndex()`.

Utilitzar el patró modular per a una funcionalitat jQuery

```
$(document).ready(function() {
   var feature = (function() {

      var $items = $('#myFeature li'),
         $container = $('<div class="container"></div>'),
         $currentItem,

         urlBase = '/foo.php?item=',

         createContainer = function() {
            var $i = $(this),
               $c = $container.clone().appendTo($i);

            $i.data('container', $c);
         },

         buildUrl = function() {
            return urlBase + $currentItem.attr('id');
         },

         showItem = function() {
            var $currentItem = $(this);
            getContent(showContent);
         },

         showItemByIndex = function(idx) {
            $.proxy(showItem, $items.get(idx));
         },

         getContent = function(callback) {
            $currentItem.data('container').load(buildUrl(), callback);
         },

         showContent = function() {
```

```
            $currentItem.data('container').show();
            hideContent();
         },

         hideContent = function() {
            $currentItem.siblings()
               .each(function() {
                  $(this).data('container').hide();
               });
         };

      $items
         .each(createContainer)
         .click(showItem);

      return { showItemByIndex : showItemByIndex };
   })();

   feature.showItemByIndex(0);
});
```

10.3. Gestió de dependències

Quan un projecte assoleix una certa grandària, comença a ser difícil el maneig dels mòduls d'una aplicació, ja que és necessari saber ordenar-los de manera correcta, i començar a combinar-los en un únic arxiu per a aconseguir la menor quantitat de peticions. També és possible que es vulgui carregar codi «al vol» després de la càrrega de la pàgina.

Nota

Aquest subapartat està basat en l'excel·lent documentació de RequireJS i és utilitzada amb el permís de James Burke, autor de RequireJS.

RequireJS és una eina de gestió de dependències creada per James Burke, la qual ajuda a manejar els mòduls, carregar-los en un ordre correcte i combinar-los de manera fàcil sense haver de fer cap canvi. Al seu torn, atorga una manera fàcil de carregar codi una vegada carregada la pàgina, i permet minimitzar el temps de descàrrega.

RequireJS posseeix un sistema modular que no cal que se segueixi per a obtenir-ne els beneficis. El format modular de RequireJS permet l'escriptura de codi encapsulat, la incorporació d'internacionalització (i18n) als paquets (que permet utilitzar-los en diferents llenguatges) i fins i tot la utilització de serveis JSONP com a dependències.

10.3.1. Obtenir RequireJS

La manera més fàcil d'utilitzar RequireJS amb jQuery és descarregant el paquet de jQuery amb RequireJS ja incorporat en ell. Aquest paquet exclou porcions de codi que dupliquen funcions de jQuery. També és útil descarregar un exemple de projecte jQuery que utilitza RequireJS.

10.3.2. Utilitzar RequireJS amb jQuery

Utilitzar RequireJS és simple. Només cal incorporar a la pàgina la versió de jQuery que posseeix RequireJS incorporat i, a continuació, sol·licitar els arxius de l'aplicació. L'exemple següent assumeix que tant jQuery com els altres arxius són dins de la carpeta `scripts/`.

Utilitzar RequireJS: un exemple simple

```
<!DOCTYPE html>
<html>
   <head>
  <title>jQuery+RequireJS Sample Page</title>
      <script src="scripts/require-jquery.js"></script>
      <script>require(["app"]);</script>
   </head>
   <body>
      <h1>jQuery+RequireJS Sample Page</h1>
   </body>
</html>
```

La crida a `require(["app"])` li diu a RequireJS que carregui l'arxiu `scripts/app.js`. RequireJS carregarà qualsevol dependència passada a `require()` sense l'extensió `.js` des del mateix directori que en què es troba l'arxiu `require-jquery.js`, encara que també es pot especificar la ruta de la manera següent:

```
<script>require(["scripts/app.js"]);</script>
```

L'arxiu `app.js` és una altra crida a `require.js` per a carregar tots els arxius necessaris per a l'aplicació. En l'exemple següent, `app.js` sol·licita dues extensions `jquery.alpha.js` i `jquery.beta.js` (no són extensions reals, només exemples). Aquestes extensions estan en la mateixa carpeta que `require-jquery.js`:

Un simple arxiu JavaScript amb dependències

```
require(["jquery.alpha", "jquery.beta"], function() {
   //Les extensions jquery.alpha.js i jquery.beta.js han estan carregades.
   $(function() {
```

```
        $('body').alpha().beta();
    });
});
```

10.3.3. Crear mòduls reutilitzables amb RequireJS

RequireJS fa que sigui fàcil definir mòduls reutilitzables a través de `require.def()`. Un mòdul RequireJS pot tenir dependències que poden ser utilitzades per a definir un mòdul, a més de poder retornar un valor –un objecte, una funció o una altra cosa– que pot ser fins i tot utilitzat per altres mòduls.

Si el mòdul no posseeix cap dependència, tan sols s'ha d'especificar el nom com a primer argument de `require.def()`. El segon argument és un objecte literal que defineix les propietats del mòdul. Per exemple:

Definició d'un mòdul RequireJS que no posseeix dependències

```
require.def("my/simpleshirt",
    {
        color: "black",
        size: "unisize"
    }
);
```

L'exemple ha de ser desat en l'arxiu `my/simpleshirt.js`.

Si el mòdul posseeix dependències, es poden especificar en el segon argument de `require.def()` a través d'un vector, i després passar una funció com a tercer argument. Aquesta funció serà anomenada per a definir el mòdul una vegada carregades totes les dependències. Aquesta funció rep els valors retornats per les dependències com un argument (en el mateix ordre en què són requerides en el vector) i després la mateixa ha de retornar un objecte que defineixi el mòdul.

Definició d'un mòdul RequireJS amb dependències

```
require.def("my/shirt",
    ["my/cart", "my/inventory"],
    function(cart, inventory) {
        //retorna un objecte que defineix "my/shirt"
        return {
            color: "blue",
            size: "large"
            addToCart: function() {
                inventory.decrement(this);
                cart.add(this);
            }
```

```
        }
    }
);
```

En aquest exemple, es crea el mòdul `my/shirt`. Aquest depèn de `my/cart` i `my/inventory`. En el disc, els arxius estan estructurats de la manera següent:

```
my/cart.js
my/inventory.js
my/shirt.js
```

La funció que defineix `my/shirt` no és cridada fins que `my/cart` i `my/inventory` hagin estat carregades, i aquesta funció rep com a arguments els mòduls com `cart` i `inventory`. L'ordre dels arguments de la funció ha de coincidir amb l'ordre en què les dependències es requereixen en el vector. L'objecte retornat defineix el mòdul `my/shirt`. Definint els mòduls d'aquesta manera, `my/shirt` no existeix com un objecte global, ja que múltiples mòduls poden existir a la pàgina al mateix temps.

Els mòduls no han de retornar un objecte; es permet qualsevol tipus de valor.

Definició d'un mòdul RequireJS que retorna una funció

```
require.def("my/title",
    ["my/dependency1", "my/dependency2"],
   function(dep1, dep2) {
      // torna una funció per a definir "my/title".
      // Aquest retorna o estableix
      // el títol de la finestra
      return function(title) {
         return title ? (window.title = title) : window.title;
      }
   }
);
```

Només un mòdul ha de ser requerit per arxiu JavaScript.

10.3.4. Optimitzar el codi amb les eines de RequireJS

Una vegada incorporat RequireJS per al maneig de dependències, l'optimització del codi és molt fàcil. Descarregueu el paquet de RequireJS i col·loqueu-lo en qualsevol lloc, preferentment fora de l'àrea de desenvolupament web. Per als propòsits d'aquest exemple, el paquet de RequireJS està situat en una carpeta paral·lela al directori `webapp` (la qual conté la pàgina HTML i tots els arxius JavaScript de l'aplicació). L'estructura de directoris és:

```
requirejs/ (utilitzat per a executar les eines)
```

```
webapp/app.html
webapp/scripts/app.js
webapp/scripts/require-jquery.js
webapp/scripts/jquery.alpha.js
webapp/scripts/jquery.beta.js
```

Després, a la carpeta on es troben `require-jquery.js` i `app.js`, cal crear un arxiu anomenat `app.build.js` amb el contingut següent:

Arxiu de configuració per a les eines d'optimització de RequireJS

```
{
   appDir: "../",
   baseUrl: "scripts/",
   dir: "../../webapp-build",
   //Comentar la línia següent si es vol
   //minificar el codi pel compilador
   //en el seu mode "simple"
   optimize: "none",

   modules: [
      {
         name: "app"
      }
   ]
}
```

Per a utilitzar l'eina, cal tenir instal·lat Java 6. Closure Compiler és utilitzat per a la minificació del codi (en cas que `optimize: "none"` estigui comentat).

Per a començar a processar els arxius, cal obrir una finestra de comandos, adreçar-se al directori `webapp/scripts` i executar:

```
# per a sistemes que no són windows
../../requirejs/build/build.sh app.build.js

# per a sistemes windows
..\..\requirejs\build\build.bat app.build.js
```

Una vegada executat, l'arxiu `app.js` de la carpeta `webapp-build` contindrà tot el codi d'app.js més el de `jquery.alpha.js` i `jquery.beta.js`. Si s'obre l'arxiu `app.html` (també en la carpeta `webapp-build`), es podrà observar que no es realitza cap petició per a carregar `jquery.alpha.js` i `jquery.beta.js`.

10.4. Exercicis

10.4.1. Crear un mòdul *portlet*

Obriu l'arxiu `/ejercicios/portlets.html` en el navegador. Feu l'exercici utilitzant l'arxiu `/ejercicios/js/portlets.js`. L'exercici consisteix a crear una funció creadora de *portlet* que utilitzi el patró modular, de tal manera que el següent codi funcioni:

```
var myPortlet = Portlet({
   title : 'Curry',
   source : 'data/html/curry.html',
   initialState : 'open' // or 'closed'
});

myPortlet.$element.appendTo('body');
```

Cada *portlet* haurà de ser un `div` amb un títol, un àrea de contingut, un botó per a obrir/tancar el *portlet*, un botó per a remoure'l i un altre per a actualitzar-lo. El *portlet* retornat per la funció haurà de tenir l'API pública següent:

```
myPortlet.open(); // força a obrir
myPortlet.close(); // força a tancar
myPortlet.toggle(); // alterna entre els estats obert i tancat
myPortlet.refresh(); // actualitza el contingut
myPortlet.destroy(); // remou el portlet de la pàgina
myPortlet.setSource('data/html/onions.html'); // canvia el codi
```

11. Esdeveniments personalitzats

11.1. Introducció als esdeveniments personalitzats

Tots estem familiaritzats amb els esdeveniments bàsics –`click`, `mouseover`, `focus`, `blur`, `submit`, etc.– que sorgeixen a partir de la interacció de l'usuari amb el navegador.

Els esdeveniments personalitzats permeten conèixer el món de la programació orientada a esdeveniments (en anglès, *event-driven programming*). En aquest apartat, s'utilitzarà el sistema d'esdeveniments personalitzats de jQuery per a crear una simple aplicació de cerca a Twitter.

En un primer moment, pot ser difícil entendre el requisit d'utilitzar esdeveniments personalitzats, ja que els esdeveniments convencionals permeten satisfer totes les necessitats. No obstant això, els esdeveniments personalitzats ofereixen una nova forma de pensar la programació en JavaScript. En lloc d'enfocar-se en l'element que executa una acció, els esdeveniments personalitzats posen l'atenció en l'element on s'esdevindrà l'acció. Aquest concepte proporciona diversos beneficis:

- Els comportaments de l'element objectiu poden ser executats per diferents elements utilitzant el mateix codi.

- Els comportaments poden ser executats en múltiples i similars elements objectius alhora.

- Els comportaments són associats d'una manera més clara amb l'element objectiu, i això fa que el codi sigui més fàcil de llegir i mantenir.

Un exemple és la millor forma d'explicar l'assumpte. Suposeu que teniu un llum incandescent en una habitació d'una casa. El llum actualment està encès i és controlat per dos interruptors de tres posicions i un *clapper* (interruptor activat per aplaudiments):

```
<div class="room" id="kitchen">
   <div class="lightbulb on"></div>
   <div class="switch"></div>
   <div class="switch"></div>
   <div class="clapper"></div>
</div>
```

Executant el *clapper* o algun dels interruptors, l'estat del llum canvia. Als interruptors o al *clapper* no els interessa si el llum està encès o apagat, només volen canviar-ne l'estat.

Sense la utilització d'esdeveniments personalitzats, es pot escriure la rutina de la manera següent:

```
$('.switch, .clapper').click(function() {
   var $light = $(this).parent().find('.lightbulb');
   if ($light.hasClass('on')) {
      $light.removeClass('on').addClass('off');
   } else {
      $light.removeClass('off').addClass('on');
   }
});
```

D'altra banda, utilitzant esdeveniments personalitzats, el codi queda així:

```
$('.lightbulb').on('changeState', function(e) {
   var $light = $(this);
   if ($light.hasClass('on')) {
      $light.removeClass('on').addClass('off');
   } else {
      $light.removeClass('off').addClass('on');
   }
});

$('.switch, .clapper').click(function() {
   $(this).parent().find('.lightbulb').trigger('changeState');
});
```

Alguna cosa important ha passat: el comportament del llum s'ha mogut, abans estava en els interruptors i en *el clapper*, i ara es troba en el mateix llum.

També es pot fer l'exemple una mica més interessant. Suposeu que s'ha afegit una altra habitació a la casa, juntament amb un interruptor general, com es mostra a continuació:

```
<div class="room" id="kitchen">
   <div class="lightbulb on"></div>
   <div class="switch"></div>
   <div class="switch"></div>
   <div class="clapper"></div>
</div>
<div class="room" id="bedroom">
   <div class="lightbulb on"></div>
   <div class="switch"></div>
```

```
    <div class="switch"></div>
    <div class="clapper"></div>
</div>
<div id="master_switch"></div>
```

Si hi ha algun llum encès a la casa, es poden apagar a través de l'interruptor general. I, de la mateixa manera, si hi ha llums apagats es poden encendre amb aquest interruptor. Per a fer aquesta tasca, s'agreguen dos esdeveniments personalitzats més al llum: `turnOn` i `turnOff`. A través d'una lògica en l'esdeveniment `changeState` es decideix quin esdeveniment personalitzat cal utilitzar:

```
$('.lightbulb')
    .on('changeState', function(e) {
        var $light = $(this);
        if ($light.hasClass('on')) {
            $light.trigger('turnOff');
        } else {
            $light.trigger('turnOn');
        }
    })
    .on('turnOn', function(e) {
        $(this).removeClass('off').addClass('on');
    })
    .on('turnOff', function(e) {
        $(this).removeClass('off').addClass('on');
    });

$('.switch, .clapper').click(function() {
    $(this).parent().find('.lightbulb').trigger('changeState');
});

$('#master_switch').click(function() {
    if ($('.lightbulb.on').length) {
        $('.lightbulb').trigger('turnOff');
    } else {
        $('.lightbulb').trigger('turnOn');
    }
});
```

Observeu que el comportament de l'interruptor general s'ha vinculat a l'interruptor general, mentre que el comportament dels llums pertany als llums.

Nota

Si esteu acostumats a la programació orientada a objectes, pot ser útil pensar en els esdeveniments personalitzats com a mètodes d'objectes. En termes generals, l'objecte al qual pertany el mètode es crea a partir del selector jQuery. Vincular l'esdeveniment persona-

litzat `changeState` a tots els elements `$('.light')` és similar a tenir una classe anomenada `Light` amb un mètode `changeState`, i després instanciar nous objectes `Light` per cada element.

Recapitulació: $.fn.on i $.fn.trigger

Al món dels esdeveniments personalitzats hi ha dos mètodes importants de jQuery: `$.fn.on` i `$.fn.trigger`. En l'apartat dedicat a esdeveniments s'ha explicat la utilització d'aquests dos mètodes per a treballar amb esdeveniments de l'usuari; en aquest apartat és important recordar dos punts:

- El mètode `$.fn.on` pren com a arguments un tipus d'esdeveniment i una funció controladora d'esdeveniment. Opcionalment, pot rebre informació associada a l'esdeveniment com a segon argument, desplaçant com a tercer argument la funció controladora d'esdeveniment. Qualsevol informació passada estarà disponible a la funció controladora a través de la propietat `data` de l'objecte de l'esdeveniment. Al seu torn, la funció controladora rep l'objecte de l'esdeveniment com a primer argument.

- El mètode `$.fn.trigger` pren com a arguments el tipus d'esdeveniment i, opcionalment, pot prendre un vector amb valors. Aquests valors seran passats a la funció controladora d'esdeveniments com a arguments després de l'objecte de l'esdeveniment.

A continuació es mostra un exemple d'utilització de `$.fn.on` i `$.fn.trigger` on s'utilitza informació personalitzada en tots dos casos:

```
$(document).on('myCustomEvent', { foo : 'bar' }, function(e, arg1, arg2) {
   console.log(e.data.foo); // 'bar'
   console.log(arg1); // 'bim'
   console.log(arg2); // 'baz'
});

$(document).trigger('myCustomEvent', [ 'bim', 'baz' ]);
```

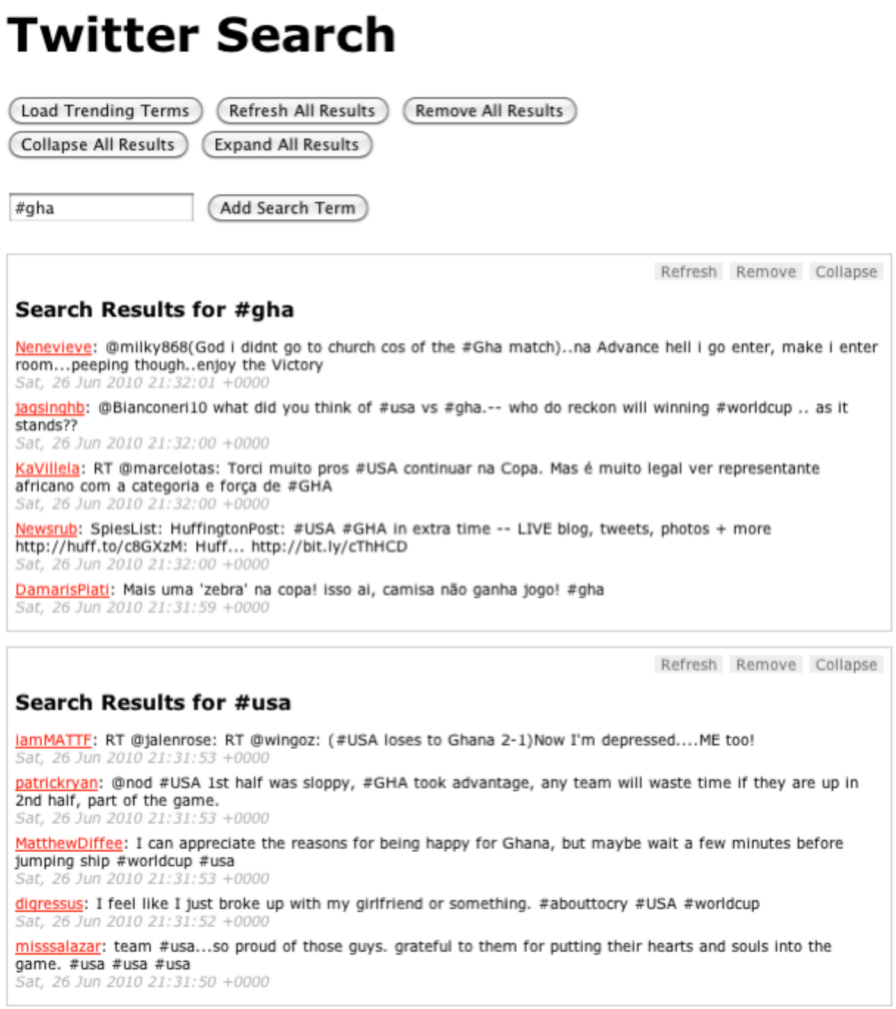
11.1.1. Un exemple d'aplicació

Per a demostrar el poder dels esdeveniments personalitzats, es desenvoluparà una simple eina per a buscar a Twitter. Aquesta eina oferirà diverses maneres a l'usuari de fer una cerca: ingressant el terme que es vol buscar en una caixa de text o consultant els «temes de moda» de Twitter.

Els resultats de cada terme es mostraran en un contenidor de resultats; aquests resultats es podran expandir, col·lapsar, refrescar i remoure, ja sigui de forma individual o conjunta.

El resultat final de l'aplicació serà el següent:

L'aplicació finalitzada



Iniciació

Es comença amb un HTML bàsic:

```
<h1>Twitter Search</h1>
<input type="button" id="get_trends"
   value="Load Trending Terms" />

<form>
   <input type="text" class="input_text"
      id="search_term" />
   <input type="submit" class="input_submit"
      value="Add Search Term" />
</form>
```

```
<div id="twitter">
   <div class="template results">
      <h2>Search Results for
      <span class="search_term"></span></h2>
   </div>
</div>
```

L'HTML posseeix un contenidor (#twitter) per al widget, una plantilla per als resultats (ocult amb CSS) i un simple formulari on l'usuari pot escriure el terme que vol buscar.

Hi ha dos tipus d'elements en els quals cal actuar: els contenidors de resultats i el contenidor *Twitter*.

Els contenidors de resultats són el cor de l'aplicació. Es crearà una extensió per a preparar cada contenidor una vegada aquest s'agrega al contenidor *Twitter*. A més, entre altres coses, l'extensió vincularà els esdeveniments personalitzats per cada contenidor i afegirà, a la part superior dreta de cada contenidor, botons que executaran accions. Cada contenidor de resultats tindrà els esdeveniments personalitzats següents:

- **refresh**. Assenyala que la informació del contenidor s'està actualitzant i dispara la petició que busca les dades per al terme de cerca.

- **populate**. Rep la informació JSON i la utilitza per a omplir el contenidor.

- **remove**. Remou el contenidor de la pàgina després que l'usuari confirmi l'acció. Aquesta confirmació es pot ometre si es passa `true` com a segon argument del controlador d'esdeveniment. L'esdeveniment, a més, remou el terme associat amb el contenidor de resultats de l'objecte global que conté els termes de cerca.

- **collapse**. Afegeix una classe al contenidor, la qual ocultarà el resultat a través de CSS. A més, canviarà el botó de «col·lapsar» a «expandir».

- **expand**. Remou la classe del contenidor que afegeix l'esdeveniment *collapse*. A més, canviarà el botó d'«expandir» a «col·lapsar».

A més, l'extensió és responsable d'afegir els botons d'accions al contenidor, vinculant un esdeveniment `click` a cada botó i utilitzant la classe de cada ítem per a determinar quin esdeveniment personalitzat serà executat en cada contenidor de resultats.

```
$.fn.twitterResult = function(settings) {
   return this.each(function() {
      var $results = $(this),
```

```
         $actions = $.fn.twitterResult.actions =
            $.fn.twitterResult.actions ||
            $.fn.twitterResult.createActions(),
         $a = $actions.clone().prependTo($results),
         term = settings.term;

      $results.find('span.search_term').text(term);

      $.each(
          ['refresh', 'populate', 'remove', 'collapse', 'expand'],
         function(i, ev) {
            $results.bind(
               ev,
               { term : term },
               $.fn.twitterResult.events[ev]
            );
         }
      );

      // utilitza la classe de cada acció per a determinar
      // quin esdeveniment s'executarà en el panell de resultats
      $a.find('li').click(function() {
         // passa l'element <li> fent clic en la funció
         // perquè es pugui manipular en cas que calgui
         $results.trigger($(this).attr('class'), [ $(this) ]);
      });
   });
};

$.fn.twitterResult.createActions = function() {
   return $('<ul class="actions" />').append(
      '<li class="refresh">Refresh</li>' +
      '<li class="remove">Remove</li>' +
      '<li class="collapse">Collapse</li>'
   );
};

$.fn.twitterResult.events = {
   refresh : function(e) {
         // indica que els resultats s'estan actualitzant
      var $this = $(this).addClass('refreshing');

      $this.find('p.tweet').remove();
      $results.append('<p class="loading">Loading ...</p>');

      // obté la informació de Twitter en format jsonp
      $.getJSON(
```

```
            'http://search.twitter.com/search.json?q=' +
               escape(e.data.term) + '&rpp=5&callback=?',
            function(json) {
               $this.trigger('populate', [ json ]);
            }
         );
      },

      populate : function(e, json) {
         var results = json.results;
         var $this = $(this);

         $this.find('p.loading').remove();

         $.each(results, function(i,result) {
            var tweet = '<p class="tweet">' +
               '<a href="http://twitter.com/' +
               result.from_user +
               '">' +
               result.from_user +
               '</a>: ' +
               result.text +
               ' <span class="date">' +
               result.created_at +
               '</span>' +
            '</p>';
            $this.append(tweet);
         });

         // indica que els resultats
         // ja s'han actualitzat
         $this.removeClass('refreshing');
      },

      remove : function(e, force) {
         if (
            !force &&
            !confirm('Remove panel for term ' + e.data.term + '?')
         ) {
            return;
         }
         $(this).remove();

         // indica que ja no es tindrà
         // un panell per al terme
         search_terms[e.data.term] = 0;
      },
```

```
    collapse : function(e) {
       $(this).find('li.collapse').removeClass('collapse')
          .addClass('expand').text('Expand');

       $(this).addClass('collapsed');
    },

    expand : function(e) {
       $(this).find('li.expand').removeClass('expand')
          .addClass('collapse').text('Collapse');

       $(this).removeClass('collapsed');
    }
};
```

El contenidor *Twitter* posseeix només dos esdeveniments personalitzats:

- **getResults**. Rep un terme de cerca i comprova si ja no existeix un contenidor de resultats per a aquest terme. En cas que no existeixi, hi afegeix un contenidor utilitzant la plantilla de resultats, el configura utilitzant l'extensió `$.fn.twitterResult` (mostrada anteriorment) i després executa l'esdeveniment `refresh` amb la finalitat de carregar correctament els resultats. Finalment, desa el terme buscat per no haver de tornar a demanar les dades sobre la cerca.

- **getTrends**. Consulta a *Twitter* la llista dels deu primers «termes de moda», interactua amb ells i executa l'esdeveniment `getResults` per cadascun, de tal manera que afegeix un contenidor de resultats per cada terme.

Vinculacions en el contenidor *Twitter*:

```
$('#twitter')
   .on('getResults', function(e, term) {
      // es comprova que ja no hi hagi una caixa per al terme
      if (!search_terms[term]) {
         var $this = $(this);
         var $template = $this.find('div.template');

         // fa una còpia de la plantilla
         // i la insereix com la primera caixa de resultats
         $results = $template.clone().
            removeClass('template').
            insertBefore($this.find('div:first')).
            twitterResult({
               'term' : term
            });
```

```
            // carrega el contingut utilitzant l'esdeveniment personalitzat "refresh"
            // vinculat al contenidor de resultats
            $results.trigger('refresh');
            search_terms[term] = 1;
         }
      })
      .on('getTrends', function(e) {
         var $this = $(this);
         $.getJSON('http://api.twitter.com/1/trends/1.json?callback=?', function(json) {
               var trends = json[0].trends;
               $.each(trends, function(i, trend) {
                  $this.trigger('getResults', [ trend.name ]);
               });
            });
      });
```

Fins ara, s'ha escrit una gran quantitat de codi que no fa res, la qual cosa no està malament. S'han especificat tots els comportaments que es volen per als elements nuclis i s'ha creat un marc sòlid per a la creació ràpida de la interfície.

A continuació, es connecta la caixa de cerca i el botó per a carregar els «temes de moda». En la caixa de text, es captura el terme ingressat i es passa al mateix temps que s'executa l'esdeveniment `getResults`. D'altra banda, fent clic en el botó per a carregar els «temes de moda», s'executa l'esdeveniment `getTrends`:

```
$('form').submit(function(e) {
   e.preventDefault();
   var term = $('#search_term').val();
   $('#twitter').trigger('getResults', [ term ]);
});

$('#get_trends').click(function() {
   $('#twitter').trigger('getTrends');
});
```

Afegint-hi botons amb un ID apropiat, es pot remoure, col·lapsar, expandir i refrescar tots els contenidors de resultats al mateix temps. Per al botó que remou el contenidor, cal tenir en compte que s'està passant `true` al controlador de l'esdeveniment com a segon argument, indicant que no es vol una confirmació de l'usuari per a remoure el contenidor.

```
$.each(['refresh', 'expand', 'collapse'], function(i, ev) {
   $('#' + ev).click(function(e) { $('#twitter div.results').trigger(ev); });
});

$('#remove').click(function(e) {
```

```
    if (confirm('Remove all results?')) {
       $('#twitter div.results').trigger('remove', [ true ]);
    }
});
```

Conclusió

Els esdeveniments personalitzats ofereixen una nova manera de pensar el codi: posen l'èmfasi en l'objectiu d'un comportament, no en l'element que l'activa. Si es pren el temps des del principi per a explicar les peces de la seva aplicació, i els comportaments que aquestes peces necessiten exhibir, els esdeveniments personalitzats proveeixen d'una manera poderosa de «parlar» amb aquestes peces, ja sigui d'una en una o en massa.

Una vegada que s'han descrit els comportaments, es converteix en un fet trivial executar-los des de qualsevol lloc, i això permet la ràpida creació i experimentació d'opcions d'interfície. Finalment, els esdeveniments personalitzats també permeten millorar la lectura del codi i el seu manteniment, i fan clara la relació entre un element i el seu comportament.

Podeu veure l'aplicació completa en els arxius `demos/custom-events/custom-events.html` i `demos/custom-events/js/custom-events.js` del material que compon aquest mòdul.

12. Funcions i execucions diferides a través de l'objecte $.Deferred

12.1. Introducció

A partir de la versió 1.5 de jQuery, la biblioteca va introduir una nova utilitat: l'objecte diferit `$.Deferred` (en anglès, *deferred object*). Aquest objecte introdueix noves formes per a la invocació i execució de les funcions de devolució (*callbacks*), i permet crear aplicacions més robustes i flexibles.

Enllaç d'interès

Per a obtenir més detalls sobre `$.Deferred`, consulteu http://api.jquery.com/category/deferred-object/.

12.2. L'objecte diferit i Ajax

El cas més comú en què es pot apreciar la utilitat de l'objecte diferit és en el maneig de les funcions de devolució en peticions Ajax.

Segons que es va poder apreciar en l'apartat dedicat a això, una manera d'invocar una petició Ajax és:

Manera tradicional d'utilitzar el mètode `$.ajax`

```
$.ajax({
   // la URL per a la petició
   url : 'post.php',

   // funcions de devolució a executar
   // en cas que la petició hagi estat
   // satisfactòria, amb error i/o completada
   success : function(data) {
      alert('Petició efectuada satisfactòriament');
   },
   error : function(jqXHR, status, error) {
      alert('Disculpeu, hi ha un problema');
   },
   complete : function(jqXHR, status) {
      alert('Petició efectuada');
   }
});
```

Com es pot observar, les funcions de devolució són configurades dins del mateix objecte `$.ajax`. Aquesta manera és incòmoda i poc flexible, ja que no permet desacoblar les funcions de devolució de la mateixa petició Ajax. I en grans aplicacions això pot arribar a ser un problema.

L'objecte diferit ens permet reescriure el codi anterior de la manera següent:

L'objecte diferit en una petició Ajax

```
// dins d'una variable es defineix
// la configuració de la petició ajax
var ajax = $.ajax({
   url : 'post.php'
});

// a través del mètode done() executem
// la funció de devolució satisfactòria (sucess)
ajax.done(function(){
   alert('Petició efectuada satisfactòriament');
});

// a través del mètode fail() executem
// la funció de devolució d'error (error)
ajax.fail(function(){
   alert('Disculpeu, hi ha hagut un problema');
});

// a través del mètode always() executem
// la funció de devolució de petició completada (complete)
ajax.always(function(){
   alert('Petició efectuada');
});
```

A través dels mètodes `deferred.done`, `deferred.fail` i `deferred.always` es poden desacoblar les funcions de devolució de la mateixa petició Ajax, cosa que en permet un maneig més còmode.

Cal tenir en compte que en cap moment es crida l'objecte diferit `$.Deferred`. Això és perquè jQuery ja l'incorpora implícitament dins del maneig de l'objecte `$.ajax`. Més endavant s'explicarà com utilitzar l'objecte `$.Deferred` de manera explícita.

De la mateixa manera, es poden crear cues de funcions de devolució o lligar-les a diferents lògiques/accions:

Cues de funcions de devolució en una petició Ajax

```
// definició de la petició Ajax
var ajax = $.ajax({
   url : 'post.php'
});

// primera funció de devolució a executar
ajax.done(function(){
   alert('Primera funció de devolució en cas satisfactori');
});

// segona funció de devolució a executar
// imediatament després de la primera
ajax.done(function(){
   alert('Segona funció de devolució en cas satisfactori');
});

// si l'usuari fa clic en #element
// s'agrega una tercera funció de devolució
$('#element').click(function(){

   ajax.done(function(){
      alert('Tercera funció de devolució si l'usuari fa clic');
   });

});

// en cas que hi hagi un error es defineix una altra
// funció de devolució
ajax.fail(function(){
   alert('Disculpeu, hi ha hagut un problema');
});
```

En executar-se la petició Ajax, i en cas que hagi estat satisfactòria, s'executen dues funcions de devolució, una darrere de l'altra. No obstant això, si l'usuari fa clic en `#element`, s'agrega una tercera funció de devolució, la qual també s'executa immediatament, sense tornar a fer la petició Ajax. Això és perquè l'objecte diferit (que es troba implícitament en la variable `ajax`) ja té informació associada sobre que la petició Ajax s'ha fet correctament.

12.2.1. deferred.then

Una altra manera d'utilitzar els mètodes `deferred.done` i `deferred.fail` és a través de `deferred.then`, que permet definir, en un mateix bloc de codi, les funcions de devolució a succeir en els casos satisfactoris i erronis.

Utilització del mètode `deferred.then`

```
// definició de la petició Ajax
var ajax = $.ajax({
   url : 'post.php'
});

// el mètode espera dues funciones de devolució
ajax.then(

   // la primera és la funció de devolució satisfactòria
   function(){
      alert('Petició efectuada satisfactòriament');
   },

   // la segona és la funció de devolució errònia
   function(){
      alert('Disculpeu, hi hagut un problema');
   }

);
```

12.3. Creació d'objectes diferits amb $.Deferred

De la mateixa manera que es poden desacoblar les funcions de devolució en una petició Ajax, també es pot fer en altres funcions utilitzant de manera explícita l'objecte `$.Deferred`.

Per exemple, una funció que verifica si un nombre és parell, de la manera tradicional es pot de així:

Funció sense utilitzar l'objecte `$.Deferred`

```
// funció que calcula si un nombre enter és parell o imparell
var isEven = function(number) {

   if (number%2 == 0){
      return true;
   } else {
      return false;
   }
```

```
}

// si és parell registra un missatge,
// en cas contrari registra un altre
if (isEven(2)){
   console.log('Es par');
} else {
   console.log('Es impar');
}
```

Utilitzant l'objecte `$.Deferred`, el mateix exemple es pot reescriure de la manera següent:

Funció utilitzant l'objecte `$.Deferred`

```
// funció que calcula si un nombre enter és parell o imparell
var isEven = function(number) {

   // desa en una variable l'objecte $.Deferred()
   var dfd = $.Deferred();

   // si és parell, resol l'objecte utilitzant deferred.resolve,
   // en cas contrari, el rebutja utilitzant deferred.reject
   if (number%2 == 0){
      dfd.resolve();
   } else {
      dfd.reject();
   }

   // retorna l'objecte diferit amb el seu estat definit
return dfd.promise();

}

// amb deferred.then es manegen les funcions de devolució
// en els casos en què el nombre sigui parell o imparell
isEven(2).then(

   // la primera és la funció de devolució satisfactòria
   function(){
      console.log('És parell');
   },

   // la segona és la funció de devolució errònia
   function(){
      console.log('És imparell');
```

```
    }

);
```

Els mètodes `deferred.resolve` i `deferred.reject` permeten **definir l'estat intern** de l'objecte `$.Deferred()`. Aquesta definició **és permanent, és a dir, no és possible modificar-la després** i és el que permet manejar el comportament i l'execució de les funcions de devolució posteriors per a cadascun dels casos.

Tingueu en compte que la funció `isEven` retorna el mètode `deferred.promise`. És una versió de l'objecte diferit, però que només permet llegir el seu estat o afegir noves funcions de devolució.

Nota

En els exemples que utilitzaven Ajax mostrats anteriorment, els mètodes `deferred.resolve` i `deferred.reject` són cridats de manera interna per jQuery dins de la configuració `sucess` i `error` de la petició. Per això mateix es deia que l'objecte diferit estava incorporat implícitament dins de l'objecte `$.ajax`.

Els mètodes `deferred.resolve` i `deferred.reject`, a més, permeten retornar valors per a ser utilitzats per les funcions de devolució.

Funció amb `deferred.resolve` i `deferred.reject` retornant valors reutilitzables

```
// funció que calcula si un nombre enter és parell o imparell
var isEven = function(number) {

   var dfd = $.Deferred();

   // resol o rebutja l'objecte utilitzat
   // i retorna un text amb el resultat
   if (number%2 == 0){
      dfd.resolve('El nombre ' + number + ' és parell');
   } else {
      dfd.reject('El nombre ' + number + ' és imparell');
   }

   // retorna l'objecte diferit amb el seu estat definit
return dfd.promise();

}

isEven(2).then(

   function(result){
      console.log(result); // Registra 'El nombre 2 és parell'
   },

   function(result){
      console.log(result);
   }
```

```
);
```

Nota

Es pot determinar l'estat d'un objecte diferit a través del mètode `deferred.state`. Aquest retorna un *string* amb algun d'aquests tres valors: `pending`, `resolved` o `rejected`. Per a obtenir més detalls sobre `deferred.state`, es pot consultar http://api.jquery.com/deferred.state/.

12.3.1. deferred.pipe

Hi ha casos en què cal modificar l'estat d'un objecte diferit o filtrar la informació que ve associada. Per a aquests casos existeix `deferred.pipe`. El seu funcionament és similar a `deferred.then`, amb la diferència que `deferred.pipe` retorna un nou objecte diferit modificat a través d'una funció interna.

Funció filtrant valors utilitzant`deferred.pipe`

```
// funció que calcula si un nombre enter és parell o imparell
var isEven = function(number) {

   var dfd = $.Deferred();

   if (number%2 == 0){
      dfd.resolve(number);
   } else {
      dfd.reject(number);
   }

   return dfd.promise();

}

// vector amb una sèrie de nombre parells i imparells
var numbers = [0, 2, 9, 10, 5, 8, 12];

// a través de deferred.pipe es pregunta si nombre es troba
// dins el vector numbers
isEven(2).pipe(

   function(number){

      // crea un nou objecte diferit
      var dfd = $.Deferred();

      if($.inArray(number, numbers) !== -1){
         dfd.resolve();
      } else {
```

```
            dfd.reject();
        }

        // retorna un nou objecte diferit
        return dfd.promise();

    }

).then(

    function(){
        // com que està dins del vector numbers i és parell,
        // es registra aquest missatge
        console.log('El nombre és parell i es troba dins de numbers');
    },

    function(){
        console.log('El nombre és imparell o no es troba dins de numbers');
    }

);
```

Enllaç d'interès

Per a obtenir més detalls sobre `deferred.pipe`, es pot consultar http://api.jquery.com/deferred.pipe/.

12.3.2. $.when

El mètode `$.when` permet executar funcions de devolució, quan un o més objectes diferits posseeixin algun estat definit.

Un cas comú d'utilització de `$.when` és quan es vol verificar que s'han efectuat dues peticions Ajax separades.

Utilització de `$.when`

```
// primera petició ajax
var comments = $.ajax({
    url : '/echo/json/'
});

// segona petició ajax
var validation = $.ajax({
    url : '/echo/json/'
});

// quan les dues peticions siguin realitzades
```

```
// executa alguna funció de devolució definida
// dins de deferred.then
$.when(comments, validation).then(
   function(){
      alert('Peticions efectuades');
   },
   function(){
      alert('Disculpeu, hi ha hagut un problema');
   }
);
```

Enllaç d'interès

Per a obtenir més detalls sobre `$.when`, es pot consultar http://api.jquery.com/jquery.when/.