7

Fondations de la bibliothèque dojo

La bibliothèque dojo est une boîte à outils Open Source visant à faciliter l'utilisation des technologies JavaScript et DHTML. Créée à partir de projets tels que nWidgets, burstlib et f(m) sous l'impulsion d'Alex Russell et Dylan Schiemann, son principal objectif est de résoudre les problèmes du DHTML, qui freinaient son adoption en masse afin de développer des applications Web riches.

dojo ne regroupe pas uniquement une collection de bibliothèques JavaScript. Elle offre un gestionnaire évolué de modules ainsi qu'un cadre pour structurer les applications Web riches fondées sur la technologie JavaScript. Pour simplifier, nous la considérons dans le contexte de cet ouvrage comme une bibliothèque.

Les premières lignes de code de dojo ont été écrites en septembre 2004, et les premières contributions ont débuté en mars 2005. À ce jour, trois versions majeures en ont été publiées. dojo est supportée par la fondation dojo, créée en 2005, qui regroupe ses principaux développeurs.

Nous avons divisé la présentation de cette bibliothèque en deux chapitres distincts. Le présent chapitre aborde sa mise en œuvre, sa structuration en modules, son support des éléments de base du langage ainsi que son support des techniques Ajax. Nous détaillons au chapitre 11 toutes les fonctionnalités de dojo relatives à la programmation Web.

Mécanismes de base

La bibliothèque est structurée en modules adressant diverses problématiques. Elle met en œuvre un mécanisme de chargement à la demande de ces modules afin d'alléger le code JavaScript de la bibliothèque utilisé au niveau du navigateur.

La figure 7.1 illustre les principaux groupes de modules de cette bibliothèque.

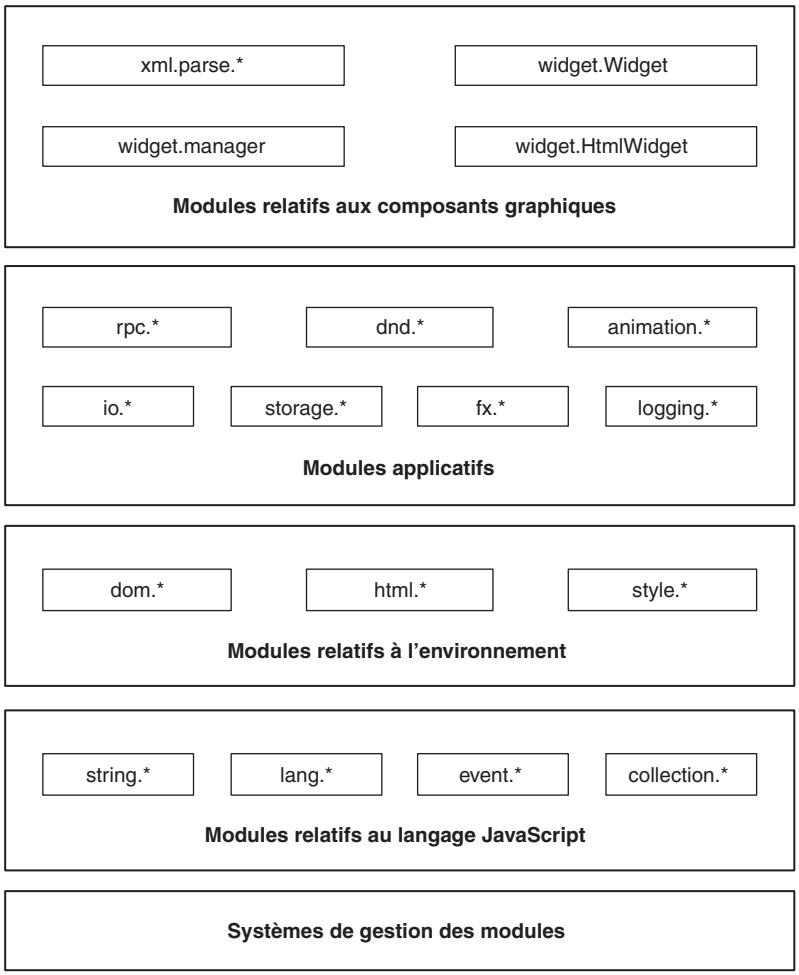

Figure 7.1
Architecture modulaire de la bibliothèque dojo

Ce mécanisme est récursif. Un module peut nécessiter le chargement d'autres modules et met en œuvre la correspondance entre les noms de modules et leurs fichiers source ainsi que le transfert de ces fichiers et le chargement du code JavaScript qu'ils contiennent.

Installation et configuration

La bibliothèque dojo propose différentes distributions selon l'utilisation souhaitée. L'objectif de ces distributions est d'incorporer différents modules directement dans le fichier **dojo.js,** le fichier pivot de la bibliothèque, plutôt que de les charger à la demande.

En effet, un chargement à la demande d'un grand nombre de fichiers peut engendrer un nombre trop important d'échanges réseau et pénaliser les performances.

Le tableau 7.1 récapitule les principales distributions disponibles ainsi que leur contenu.

Tableau 7.1 Distributions de la bibliothèque dojo

Nom	Description
ajax	Inclut dans le fichier **dojo.js** le noyau de dojo ainsi que tous les modules permettant de mettre en œuvre les techniques Ajax.
browserio	Inclut dans le fichier **dojo.js** le noyau de dojo ainsi que tous les modules relatifs à Ajax et fondés sur la classe `XMLHttpRequest`.
core	Inclut uniquement dans le fichier **dojo.js** le noyau de dojo.
editor	Inclut dans le fichier **dojo.js** le noyau de dojo ainsi que tous les composants graphiques nécessaires à l'éditeur Wysiwyg.
event	Inclut dans le fichier **dojo.js** le noyau de dojo ainsi que tous les modules relatifs au support des événements.
event_and_io	Inclut dans le fichier **dojo.js** le noyau de dojo ainsi que tous les modules relatifs au support des événements et d'Ajax.
kitchen_sink	Inclut dans le fichier **dojo.js** tous les modules de la bibliothèque.
minimal	Contient les modules minimaux permettant à la bibliothèque de fonctionner.
storage	Inclut dans le fichier **dojo.js** le noyau de dojo ainsi que tous les modules relatifs au stockage d'informations au niveau du client.
widget	Contient le noyau de dojo ainsi que tous les modules correspondant aux composants graphiques de la bibliothèque.

Le nom du fichier téléchargeable d'une distribution est structuré de la manière suivante : **dojo-<VERSION>-<NOM_DISTRIBUTION>.zip.** Dans chaque distribution, le fichier **build.txt** localisé à sa racine récapitule les différents modules contenus dans le fichier **dojo.js.** Ces différentes distributions sont téléchargeables à l'adresse *http://download.dojotoolkit.org/*.

Notons qu'il est possible de travailler avec une version non packagée récupérée directement à partir du gestionnaire de version.

L'installation de la bibliothèque consiste à copier les répertoires et fichiers de la distribution de dojo dans un répertoire accessible depuis le serveur Web du site. Ce répertoire doit pouvoir être accédé depuis un navigateur et comporter, au minimum, les éléments suivants :

```
src/
dojo.js
iframe_history.html
```

La configuration de la bibliothèque se réalise en deux étapes dans chaque page HTML. La première consiste à initialiser l'objet global `djConfig` (repère ❶), qui contient le paramétrage de la bibliothèque :

```
<html>
    <head>
        (...)
        <script>
            djConfig = { isDebug: true };←❶
        </script>
    </head>
    (...)
</html>
```

Le tableau 7.2 récapitule les principaux attributs de cet objet afin de paramétrer la bibliothèque dojo.

Tableau 7.2 Principaux attributs de configuration de l'objet *djConfig*

Attribut	Description
`allowQueryConfig`	Autorise l'utilisation des paramètres de requête afin de surcharger les attributs de configuration de l'objet `djConfig`. Sa valeur par défaut est `false`. Dans le cas où sa valeur vaut `true`, la chaîne de paramètres suivante dans l'adresse de la page permet de positionner la valeur de l'attribut `isDebug` avec la valeur `true` : */maPage.html?djConfig.isDebug=true*.
`baseRelativePath`	Doit être utilisé dans le cas où les sources de dojo et les pages qui l'utilisent ne sont pas fournies par la même adresse. Sa valeur par défaut est la chaîne de caractères vide.
`debugAtAllCosts`	Paramètre la bibliothèque afin de faciliter son débogage dans différents outils. Sa valeur par défaut est `false`.
`isDebug`	Permet d'activer les traces applicatives. Sa valeur par défaut est `false`.
`parseWidgets`	Spécifie si les balises correspondant aux composants graphiques de dojo doivent être évaluées. Sa valeur par défaut est `true`.
`preventBackButtonFix`	Permet d'activer ou non la gestion du bouton Précédent du navigateur. Si sa valeur est `false`, ce mécanisme est désactivé. Sa valeur par défaut est `true`.
`searchIds`	Si la valeur de l'attribut `parseWidgets` vaut `false`, permet de spécifier les identifiants des balises des composants graphiques de dojo à évaluer.

La seconde étape consiste à faire référence au fichier **dojo.js** de la distribution utilisée (repère ❶ dans le code suivant) au début des fichiers HTML :

```
<html>
    <head>
        (...)
        <script>
            djConfig = { isDebug: true };
        </script>
        <script language="JavaScript" type="text/javascript"
                                    src="../../dojo.js"></script>←❶
    </head>
    (...)
</html>
```

Les différentes distributions de dojo fournissent deux versions du fichier **dojo.js.** La première contient une version compressée des sources peu lisibles de la bibliothèque. La seconde, correspondant au fichier **dojo.js.uncompressed.js** et localisée au même endroit que le fichier précédent, n'est pas compressée et est donc facilement lisible.

Le chargement du premier fichier est plus rapide et doit donc être utilisé en production. Il peut cependant se révéler difficile à utiliser pour voir les traitements réalisés par dojo, ce qui n'est pas le cas du second.

À partir de ce moment, la bibliothèque dojo peut être utilisée dans la page HTML. Elle intègre la capacité de charger à la demande les modules nécessaires qui ne sont pas présents dans le fichier **dojo.js.**

Gestion des modules

Comme indiqué précédemment, la bibliothèque dojo est structurée en modules, lesquels rassemblent diverses entités JavaScript, telles que des fonctions ou des objets JavaScript.

Le package représente une unité de stockage d'un module et correspond donc à un fichier ou à une entité téléchargeable depuis un serveur Web. Par défaut, tous les packages se trouvent dans le répertoire **src/** situé sous le répertoire d'installation de la bibliothèque dojo.

Dans la mesure où le poids global du code de la bibliothèque est important, l'approche de dojo consiste à ne charger que les modules utilisés par l'application. La bibliothèque met pour cela en œuvre un mécanisme de chargement à la demande fondé sur les méthodes `dojo.require` et `dojo.provide`.

La première indique qu'un module ou un ensemble de modules est utilisé dans l'application. Elle correspond au mécanisme d'importation utilisé dans le langage Java et fondé sur le mot-clé `import`. Cette méthode est couramment utilisée dans les applications utilisant dojo.

La seconde spécifie le nom du module contenu dans un package. Ce concept est similaire à celui mis en œuvre dans le langage Java avec le mot-clé `package`. Cette méthode est utilisée uniquement dans les implémentations de modules de dojo ou de modules additionnels, tels que des composants graphiques.

Dans son application, le développeur doit donc spécifier les modules qu'il désire utiliser afin que la bibliothèque les charge automatiquement. Si le module est déjà présent, aucun chargement n'est effectué.

Le code suivant décrit l'importation du module `dojo.lang.declare` (repère ❶) :

```
<html>
    <head>
        <script language="JavaScript" type="text/javascript">
            djConfig = { isDebug: true }
        </script>
         <script language="JavaScript" type="text/javascript"
                                    src="../../dojo.js"></script>
```

```
        <script language="JavaScript" type="text/javascript">
            dojo.require("dojo.lang.declare");←❶
            (...)
        </script>
    </head>
    (...)
</html>
```

La figure 7.2 illustre les mécanismes mis en œuvre dans cet exemple.

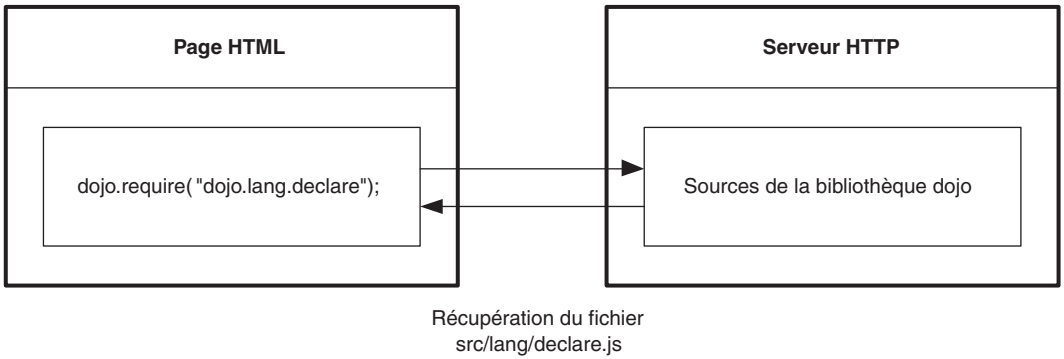

Figure 7.2
Mécanisme de chargement du module dojo.lang.declare

Dans le cas où tous les sous-modules d'un module doivent être importés, le caractère `*` peut être utilisé. La bibliothèque se fonde alors sur le fichier spécial **__package__.js,** localisé dans le répertoire du package, afin de déterminer les modules à importer.

Le code suivant décrit l'importation du module `dojo.lang.*` (repère ❶), qui contient les sous-modules `dojo.lang`, `dojo.lang.common`, `dojo.lang.assert`, `dojo.lang.array`, `dojo.lang.type`, `dojo.lang.func`, `dojo.lang.extras`, `dojo.lang.repr` et `dojo.lang.declare` :

```
<html>
    <head>
        <script language="JavaScript" type="text/javascript">
            djConfig = { isDebug: true }
        </script>
         <script language="JavaScript" type="text/javascript"
                                    src="../../dojo.js"></script>
         <script language="JavaScript" type="text/javascript">
             dojo.require("dojo.lang.*");←❶
             (...)
         </script>
    </head>
    (...)
</html>
```

La figure 7.3 illustre les mécanismes mis en œuvre dans cet exemple.

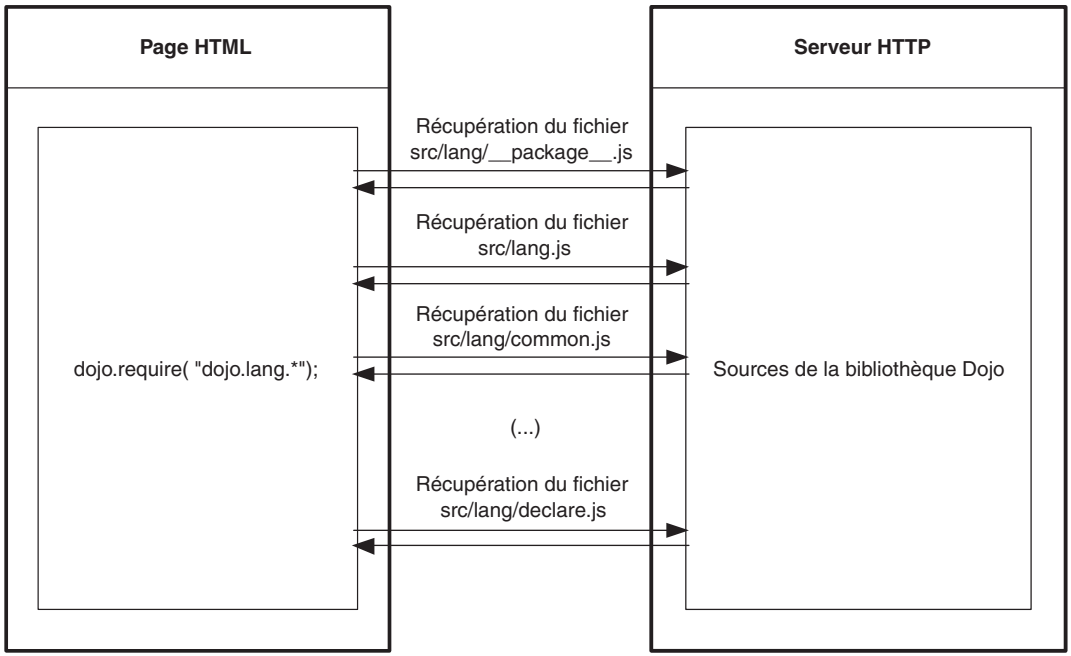

Figure 7.3
Mécanisme de chargement du module dojo.lang.*

Le tableau 7.3 récapitule les principaux modules disponibles dans la bibliothèque dojo.

Tableau 7.3 Principaux modules de la bibliothèque dojo

Module	Description
`dojo.date`	Fonctions de manipulation des dates
`dojo.dnd`	Support du glisser-déposer
`dojo.dom`	Fonctions facilitant l'utilisation de DOM
`dojo.event`	Support des événements DOM de dojo
`dojo.html`	Fonctions permettant de réaliser des traitements fondés sur des éléments HTML
`dojo.io`	Support relatif à la mise en œuvre de la technologie Ajax
`dojo.lang`	Fonctions de base facilitant l'utilisation de JavaScript
`dojo.lfx`	Fonctions permettant de réaliser des effets dans des pages HTML
`dojo.logging`	Support permettant de mettre en œuvre des traces applicatives
`dojo.profile`	Profileur JavaScript afin de mesurer les performances des traitements
`dojo.reflect`	Fonctions permettant l'accès à des informations sur les objets et les fonctions

Tableau 7.3 Principaux modules de la bibliothèque dojo *(suite)*

Module	Description
`dojo.regexp`	Fonctions de construction d'expressions régulières particulières
`dojo.string`	Fonctions relatives aux chaînes de caractères
`dojo.style`	Fonctions permettant de réaliser des traitements fondés sur des styles CSS
`dojo.svg`	Fonctions relatives à l'utilisation de la technologie SVG
`dojo.widget`	Composants graphiques de la bibliothèque
`dojo.xml`	Support pour parcourir des documents XML et mettre en œuvre la technologie XSLT

Nous n'abordons dans ce chapitre que les modules de base non graphiques de la bibliothèque. Les mécanismes de dojo relatifs au développement Web sont détaillés au chapitre 11.

Support de base

Cette famille de modules a pour objectif de faciliter l'utilisation du langage JavaScript. Elle correspond à tous les sous-modules de `dojo.lang`.

Ces derniers peuvent être importés en une seule instruction, comme dans le code suivant :

```
dojo.require("dojo.lang.*");
```

À l'instar de la bibliothèque prototype, dojo offre une fonction intéressante, `dojo.lang.tryThese`, qui permet d'essayer plusieurs fonctions successivement jusqu'à ce que l'une d'elles ne génère pas d'erreur. Cette fonction facilite notamment la mise en œuvre de fonctionnalités spécifiques à un navigateur.

Le code suivant décrit sa mise en œuvre :

```
function maFonction() {
    return dojo.lang.tryThese(
        function() {
            alert("appel de la première méthode");
            return test1(); //Erreur car la méthode n'existe pas
        },
        function() {
            alert("appel de la seconde méthode");
            return test2(); //Erreur car la méthode n'existe pas
        },
        function() {
            alert("appel de la troisième méthode");
            return "par défaut"; //Elément réellement retourné←❶
        }
    );
}
```

La fonction `maFonction` renvoie la valeur retournée par la dernière fonction (repère ❶) puisqu'elle est la seule à s'exécuter correctement. Ce type de fonction est particulièrement adapté à la sélection de l'objet à utiliser lors de la mise en œuvre des techniques Ajax.

La fonction `dojo.delayThese` offre également un intéressant mécanisme pour exécuter des fonctions successivement à un intervalle de temps spécifié. Cette fonction se fonde sur la méthode `setTimeout` de JavaScript.

Elle offre également la possibilité de spécifier une fonction de rappel exécutée avant chaque appel de fonction ainsi qu'une autre fonction après toutes les fonctions spécifiées.

Le code suivant décrit sa mise en œuvre :

```
dojo.lang.delayThese([
    function() {
        alert("appel de la première méthode");
    },
    function() {
        alert("appel de la seconde méthode");
    },
    function() {
        alert("appel de la troisième méthode");
    }
], 10);
```

Les fonctions spécifiées dans cet exemple sont appelées successivement avec dix secondes d'intervalle.

Notons la présence des fonctions `dojo.lang.setTimeout` et `dojo.lang.clearTimeout`, qui permettent d'encapsuler l'utilisation de la fonction JavaScript `setTimeout`.

Le code suivant décrit la mise en œuvre de la fonction `dojo.lang.setTimeout` :

```
dojo.lang.setTimeout(function() {
    alert("Execution de la fonction de rappel après 10 secondes");
}, 10000);
```

Vérification de types

dojo fournit une encapsulation de différentes fonctions de base de JavaScript afin de déterminer le type de variables JavaScript. Ces implémentations se fondent sur les concepts décrits à l'adresse *http://www.crockford.com/javascript/recommend.html.*

Le tableau 7.4 récapitule ces différentes fonctions de vérification de types de dojo.

Tableau 7.4 Fonctions de vérification de types de dojo

Méthode	Paramètre	Description
`dojo.lang.isAlien`	Un élément	Détermine si l'élément en paramètre n'est pas une fonction fournie par le langage JavaScript.
`dojo.lang.isArray`	Un élément	Détermine si l'élément en paramètre est un tableau.
`dojo.lang.isArrayLike`	Un élément	Détermine si l'élément en paramètre est un objet qui ressemble à un tableau, c'est-à-dire qui possède une propriété `length` correspondant à un nombre entier fini.
`dojo.lang.isBoolean`	Un élément	Détermine si l'élément en paramètre est un booléen.

Tableau 7.4 Fonctions de vérification de types de dojo *(suite)*

Méthode	Paramètre	Description
`dojo.lang.isFunction`	Un élément	Détermine si l'élément en paramètre est une fonction.
`dojo.lang.isNumber`	Un élément	Détermine si l'élément en paramètre est un nombre.
`dojo.lang.isObject`	Un élément	Détermine si l'élément en paramètre est un objet.
`dojo.lang.isString`	Un élément	Détermine si l'élément en paramètre est une chaîne de caractères.
`dojo.lang.isUndefined`	Un élément	Détermine si l'élément en paramètre est non défini.

Le code suivant décrit la mise en œuvre de ces différentes méthodes afin de déterminer le type de variables JavaScript :

```
var nombre = 10;
alert(dojo.lang.isNumber(nombre)); // Affiche true
var tableau = [];
alert(dojo.lang.isArray(tableau)); // Affiche true
alert(dojo.lang.isFunction(tableau)); // Affiche false
var boolean = true;
alert(dojo.lang.isBoolean(boolean)); // Affiche true
var chaine = "Ceci est un test";
alert(dojo.lang.isString(chaine)); // Affiche true
```

Fonctions

À l'instar de la méthode `bind` de la bibliothèque prototype décrite au chapitre précédent, dojo offre une fonction `dojo.lang.hitch` permettant de résoudre la problématique du contexte d'exécution d'une fonction JavaScript.

Cette fonction prend en paramètres un objet ainsi qu'une fonction qui sera exécutée en se fondant sur l'objet précédent.

Le code suivant reprend l'exemple du chapitre précédent en l'adaptant à cette fonction :

```
var monInstance = new Object();
monInstance.maVariable = "test";
function maFonction() {
    alert("Valeur de maVariable : "+this.maVariable);
}
var nouvelleMaFonction = dojo.lang.hitch(monInstance, maFonction);
nouvelleMaFonction();
//Affiche correctement la valeur de maVariable
```

L'appel direct à la fonction `maFonction` génère une erreur puisque le mot-clé `this` ne peut être utilisé dans ce contexte d'exécution.

La fonction `dojo.lang.curry` offre la possibilité de créer une fonction en définissant certains de ses paramètres avec des valeurs particulières. La fonction créée peut alors être utilisée avec un nombre de paramètres moins important.

Le code suivant décrit la mise en œuvre de cette fonction :

```
function maFonction(parametre1, parametre2, parametre3) {
    return parametre1+","+parametre2+","+parametre3;
}
var nouvelleFonction = dojo.lang.curry(
                              null, maFonction, "valeur1");
var retour = nouvelleFonction("valeur2", "valeur3");
// retour contient la valeur « valeur1,valeur2,valeur3 »
```

Le premier paramètre, dont la valeur est nulle dans cet exemple, permet de spécifier le contexte d'exécution de la fonction.

Support de la programmation orientée objet

La bibliothèque dojo met à disposition dans ce module différentes fonctions utilitaires permettant de réaliser des opérations de base sur les objets JavaScript.

Le tableau 7.5 récapitule ces différentes fonctions.

Tableau 7.5 Fonctions de gestion des objets JavaScript

Fonction	Paramètre	Description
`dojo.inherits`	Deux fonctions de construction d'un objet	Permet de faire hériter une classe d'une autre en initialisant l'attribut `prototype` de la classe avec un objet correspondant à la classe mère.
`dojo.lang.extend`	Une fonction de construction d'un objet et un tableau associatif	Ajoute les éléments d'un tableau associatif à l'attribut `prototype` de la fonction de construction d'un objet. Cette fonction est équivalente à `dojo.extend`. Cette dernière doit désormais être utilisée.
`dojo.lang.mixin`	Un objet et un tableau associatif	Ajoute les éléments d'un tableau associatif à un objet. Cette fonction est équivalente à `dojo.mixin`. Cette dernière doit désormais être utilisée.
`dojo.lang.shallowCopy`	Un objet et un drapeau	Permet de cloner un objet. Le drapeau en paramètre détermine si le clonage est réalisé de manière récursive.

La fonction `dojo.extend` permet d'ajouter les méthodes contenues dans un tableau associatif à l'attribut `prototype` de la fonction de construction d'un objet. Ce mécanisme offre la possibilité d'enrichir une classe avec différentes méthodes.

Le code suivant décrit la mise en œuvre de la classe `StringBuffer`, décrite au chapitre 3, avec le support de cette fonction (repère ❶) :

```
function StringBuffer() {
    this.chainesInternes = [];
}
dojo.extend(StringBuffer, {←❶
    append : function(chaine) {
        this.chainesInternes.push(chaine);
    },
    clear : function() {
        this.chainesInternes.splice(0,
                        this.chainesInternes.length);
```

```
    },
    toString : function() {
        return this.chainesInternes.join("");
    }
});
var sb = new StringBuffer();
sb.append("première chaîne");
sb.append(" et ");
sb.append("deuxième chaîne ");
var resultat = sb.toString();
//resultat contient « première chaîne et deuxième chaine »
```

La fonction `dojo.inherits` permet de faire étendre une classe d'une autre. Elle initialise l'attribut `prototype` de la fonction de construction correspondant à la classe fille avec une instance de la classe mère. De par son implémentation, cette fonction ne permet pas de mettre en œuvre l'héritage multiple.

Le code suivant décrit l'utilisation de cette fonction (repère ❶) en reprenant les classes `StringBuffer` et `ExtendedStringBuffer` abordées au chapitre 3 :

```
function StringBuffer() {
    this.chainesInternes = [];
}
dojo.extend(StringBuffer, {
    this.append = function(chaine) {
        this.chainesInternes.push(chaine);
    },
    this.clear = function() {
        this.chainesInternes.splice(0,
                        this.chainesInternes.length);
    },
    this.toString = function() {
        return this.chainesInternes.join("");
    }
}
function ExtendedStringBuffer() {
}
dojo.extend(ExtendedStringBuffer, {
    uneMethode : function() {
        (...)
    }
}
dojo.inherits(ExtendedStringBuffer, StringBuffer);←❶
```

La bibliothèque dojo fournit également un support plus poussé de la programmation orientée objet par l'intermédiaire de la fonction `dojo.lang.declare`, qui est abordée plus en détail par la suite.

Tableaux

La bibliothèque dojo fournit différentes fonctions facilitant l'utilisation et la manipulation des tableaux JavaScript simples et associatifs.

Le tableau 7.6 récapitule ces différentes fonctions.

Tableau 7.6 Fonctions de manipulation des tableaux

Fonction	Paramètre	Description
`dojo.lang.every`	Un tableau, une fonction de rappel et un objet	Détermine si tous les éléments d'un tableau correspondent à un critère. La vérification de ce dernier se réalise par le biais d'une fonction de rappel passée en paramètre. L'exécution de la fonction de rappel est réalisée dans le contexte de l'objet correspondant au dernier paramètre.
`dojo.lang.filter`	Un tableau, une fonction de rappel et un objet	Permet de filtrer les éléments d'un tableau en se fondant sur une fonction de rappel. L'exécution de la fonction de rappel est réalisée dans le contexte de l'objet correspondant au dernier paramètre.
`dojo.lang.find`	Un tableau, un élément, un drapeau d'identité et un drapeau déterminant si le parcours du tableau se fait de sa fin vers son début.	Retourne l'indice du premier élément du tableau correspondant à l'élément passé en paramètre. Retourne `-1` si aucun élément ne correspond.
`dojo.lang.findLast`	Un tableau, un élément, un drapeau d'identité	Retourne l'indice du dernier élément du tableau correspondant à l'élément passé en paramètre. Retourne `-1` si aucun élément ne correspond.
`dojo.lang.forEach`	Un tableau et une fonction de rappel	Permet de parcourir un tableau en se fondant sur une fonction de rappel. Cette dernière reçoit en paramètre à chaque itération l'élément courant, son indice ainsi que le tableau parcouru.
`dojo.lang.has`	Un tableau associatif et un nom de propriété	Détermine si le tableau associatif possède la propriété dont le nom est passé en paramètre.
`dojo.lang.inArray`	Un tableau et un élément	Détermine si un élément est présent dans un tableau.
`dojo.lang.indexOf`	Un tableau, un élément, un drapeau d'identité et un drapeau déterminant si le parcours du tableau se fait de sa fin vers son début.	Équivalent de la fonction `dojo.lang.find`.
`dojo.lang.isEmpty`	Un tableau	Détermine si le tableau est vide. Cette fonction supporte aussi bien les tableaux simples que les tableaux associatifs.
`dojo.lang.lastIndexOf`	Un tableau, un élément et un drapeau d'identité	Équivalent de la fonction `dojo.lang.findLast`.
`dojo.lang.map`	Un tableau, un objet et une fonction de rappel	Permet de construire un tableau à partir d'un autre en se fondant sur une fonction de rappel. Cette dernière a la responsabilité de renvoyer l'objet à insérer dans le nouveau tableau. L'exécution de la fonction de rappel est réalisée dans le contexte de l'objet correspondant au dernier paramètre.
`dojo.lang.some`	Un tableau, une fonction de rappel et un objet	Détermine si au moins un élément d'un tableau correspond à un critère. La vérification de ce dernier se réalise par le biais d'une fonction de rappel passée en paramètre. L'exécution de la fonction de rappel est réalisée dans le contexte de l'objet correspondant au dernier paramètre.
`dojo.lang.toArray`	Un objet similaire à un tableau et un indice de début	Permet de convertir un objet similaire à un tableau en tableau.
`dojo.lang.unnest`	Des éléments simples ou des tableaux	Permet de construire un tableau simple à partir de tableaux imbriqués les uns dans les autres.

Puisque JavaScript considère les objets en tant que tableaux associatifs, les fonctions du tableau 7.6 qui s'appliquent à ces tableaux peuvent être utilisées avec des objets.

La fonction `dojo.lang.isEmpty` détermine si le tableau en paramètre contient des éléments, comme le montre le code suivant :

```
var monTableau1 = [ "élément1","élément2","élément3" ];
var monTableau2 = [];
var tableau1Vide = dojo.lang.isEmpty(monTableau1);
// tableau1Vide contient false
var tableau2Vide = dojo.lang.isEmpty(monTableau2);
// tableau2Vide contient true
```

Cette fonction peut être utilisée avec des tableaux associatifs, comme dans le code ci-dessous :

```
dojo.require("dojo.lang.*");
var monTableauAssociatif1 = { "clé1": "valeur1" };
var monTableauAssociatif2 = {};
var tableauAssociatif1Vide = dojo.lang.isEmpty(monTableauAssociatif1);
// tableauAssociatif1Vide contient false
var tableauAssociatif2Vide = dojo.lang.isEmpty(monTableauAssociatif2);
// tableauAssociatif2Vide contient true
```

La fonction `dojo.lang.forEach` permet de parcourir un tableau en se fondant sur une fonction de rappel. Cette dernière est appelée à chaque itération de la boucle avec l'élément courant, son indice ainsi que le tableau parcouru.

Le code suivant décrit la mise en œuvre de cette fonction :

```
var monTableau = [ "élément1","élément2","élément3" ];
dojo.lang.forEach(monTableau, function(element, indice, tableau) {
    alert(indice+" : "+element);
});
```

L'exemple ci-dessus affiche les différents éléments avec leur indice correspondant : 0 pour `"élément1"`, 1 pour `"élement2"` et 2 pour `"élément3"`.

Plusieurs fonctions sont également mises à disposition afin de tester la présence d'un élément dans un tableau. C'est le cas des fonctions similaires `dojo.lang.has`, décrite dans le code suivant, et `dojo.lang.inArray` :

```
var monTableau = [ "élément1","élément2","élément3" ];
var indiceElement = dojo.lang.has("élément2");
```

Les fonctions `dojo.lang.find` et `dojo.lang.findLast` recherchent l'indice du premier élément dans un tableau correspondant à la valeur spécifiée en paramètre. Ces deux fonctions parcourent respectivement le tableau du début à la fin et inversement.

Elles prennent en paramètres le tableau ainsi que l'élément à rechercher. Un troisième paramètre permet de spécifier si la comparaison des éléments doit être stricte (sans conversion de type) en se fondant sur l'opérateur `===`. Si aucun élément n'est trouvé, la valeur de retour est `-1`.

Le code suivant décrit la mise en œuvre de ces fonctions :

```
var monTableau = [ "élément","élément2","élément" ];
var indiceElement1 = dojo.lang.find("élément");
// indiceElement1 contient 0
var indiceElement2 = dojo.lang.findLast("élément");
// indiceElement2 contient 2
```

Les fonctions `dojo.lang.indexOf` et `dojo.lang.lastIndexOf` correspondent aux fonctions décrites ci-dessus.

Les fonctions `dojo.lang.some` et `dojo.lang.every` permettent de déterminer si les éléments d'un tableau correspondent à un critère. Ce dernier est implémenté au moyen d'une fonction de rappel passée en paramètre.

Le mécanisme de ces fonctions consiste à parcourir le tableau et, pour chaque élément, à appeler la fonction de rappel. Dans le cas de la fonction `dojo.lang.some`, la valeur `true` est renvoyée si au moins un des retours de la fonction de rappel pour un élément du tableau correspond à `true`. Ainsi, au moins un élément du tableau correspond au critère.

Le code suivant décrit la mise en œuvre de la fonction `dojo.lang.some` :

```
var monTableau = [ "élément","élément2","élément" ];
var retourSome = dojo.lang.some(monTableau, function(element, indice, tableau) {
    if( element=="élément" ) {
        return true;
    } else {
        return false;
    }
});
// retourSome contient la valeur true
```

La valeur de la variable `retourSome` correspond à `true`. Si le tableau ne contenait aucun élément ayant pour valeur `"élément"`, la valeur de cette variable serait `false`.

Dans le cas de la fonction `dojo.lang.every`, la valeur `true` est renvoyée si tous les retours de la fonction de rappel pour les éléments du tableau correspondent à `true`. Ainsi tous les éléments du tableau correspondent au critère.

Le code suivant décrit la mise en œuvre de la fonction `dojo.lang.every` :

```
var monTableau = [ "élément","élément","élément" ];
var retourEvery = dojo.lang.every(monTableau, function(element, indice, tableau) {
    if( element=="élément" ) {
        return true;
    } else {
        return false;
    }
});
// retourEvery contient la valeur true
```

La valeur de la variable `retourEvery` correspond à `true`. Si le tableau contenait un élément dont la valeur était différente de `"élément"`, la valeur de cette variable serait `false`.

Programmation orientée objet

Dans le package `dojo.lang.declare`, dojo fournit différents mécanismes visant à faciliter la mise en œuvre de la programmation orientée objet avec JavaScript. Tous ces mécanismes se fondent sur la fonction `dojo.lang.declare`.

Nous allons reprendre dans cette section les différentes classes abordées au chapitre 3.

Mise en œuvre des classes

La fonction `dojo.lang.declare` prend en paramètres le nom de la classe, le ou les classes, éventuellement une fonction d'initialisation ainsi qu'un tableau associatif contenant les différents attributs et méthodes de la classe.

Le code suivant décrit la mise en œuvre de la classe `StringBuffer` en se fondant sur cette méthode :

```
dojo.lang.declare("lang.StringBuffer", null, {
    initializer: function() {
        this.chainesInternes = [];
    },
    append: function(chaine) {
        this.chainesInternes.push(chaine);
    },
    clear: function() {
        this.chainesInternes.splice(0, this.chainesInternes.length);
    },
    toString: function() {
        this.chainesInternes.join("");
    }
});
```

Toutes les méthodes présentes dans le tableau associatif sont ajoutées au prototype de la classe et sont donc partagées par toutes ses instances.

Initialisation des classes

dojo supporte la notion de constructeur, qu'elle appelle *initialiseur,* dont l'objectif est d'initialiser les éléments spécifiques à l'instance. Un tel initialiseur peut être mis en œuvre par le biais de la fonction d'initialisation ou par celui d'une méthode de la classe nommée `initializer`.

L'objectif de ces méthodes est de permettre l'initialisation d'éléments spécifiques à une instance d'une classe, tels que les attributs.

Le tableau associatif décrit à la section précédente ne doit pas être utilisé pour initialiser les éléments spécifiques aux instances, tels que les attributs. Par contre, il peut être utilisé afin de mettre en œuvre des attributs statiques, c'est-à-dire partagés par toutes les instances, comme le montre le code suivant :

```
dojo.lang.declare("lang.StringBuffer", null, {
    attributStatique: "valeur"
});
var sb1 = new lang.StringBuffer();
var sb2 = new lang.StringBuffer();
sb1.attributStatique = "autre valeur";
/* sb1.attributStatique et sb2. attributStatique contiennent toutes les deux la valeur
 ➥« autre valeur » */
```

La première approche consiste simplement à spécifier les traitements d'initialisation dans une méthode nommée `initializer` (repère ❶), comme dans le code suivant :

```
dojo.lang.declare("lang.StringBuffer", null, {
    initializer: function(chaine) {←❶
        this.chainesInternes.push(chaine);
    },
    chainesInternes: [],
    append: function(chaine) {
        this.chainesInternes.push(chaine);
    },
    clear: function() {
        this.chainesInternes.splice(0, this.chainesInternes.length);
    },
    toString: function() {
        this.chainesInternes.join("");
    }
});
```

Le seconde approche, fondée sur la fonction d'initialisation (repère ❶), utilise le second paramètre de la méthode `dojo.declare`, comme dans le code suivant :

```
dojo.lang.declare("lang.StringBuffer", null, function(chaine) {←❶
        this.chainesInternes = [];
        this.chainesInternes.push(chaine);
}, {
    append: function(chaine) {
        this.chainesInternes.push(chaine);
    },
    clear: function() {
        this.chainesInternes.splice(0, this.chainesInternes.length);
    },
    toString: function() {
        this.chainesInternes.join("");
    }
});
```

Support de l'héritage

La méthode `dojo.declare` permet de mettre en œuvre l'héritage par l'intermédiaire de son second paramètre.

Le code suivant décrit la mise en œuvre de la classe `StringBuffer` (repère ❶) et de sa sous-classe `ExtendedStringBuffer` (repère ❷) en se fondant sur cette méthode :

```
dojo.lang.declare("lang.StringBuffer", null, {←❶
    initializer: function(args) {
        this.chainesInternes = [];
    },
    append: function(chaine) {
        this.chainesInternes.push(chaine);
    },

    clear: function() {
        this.chainesInternes.splice(0,
                        this.chainesInternes.length);
    },

    toString: function() {
        return this.chainesInternes.join("");
    }
});
dojo.lang.declare("lang.ExtendedStringBuffer",lang.StringBuffer, {←❷
    initializer: function() {
    },
    append: function(chaine) {
        this.chainesInternes.push(«tests: "+chaine);
    },
});
```

La méthode `dojo.declare` offre la possibilité de gérer séparément les méthodes de la classe courante ainsi que celles de la classe mère. Il est de la sorte possible d'appeler une méthode de la classe mère par l'intermédiaire de la méthode `this.inherited` (repère ❶), comme le montre le code suivant :

```
dojo.lang.declare("java.lang.ExtendedStringBuffer", java.lang.StringBuffer, {
    initializer: function() {
    },
    append: function(chaine) {
        this.inherited("append",chaine);←❶
    },
});
```

Lors de l'appel du constructeur de la classe `ExtendedStringBuffer`, les méthodes `initializer` des deux classes sont appelées successivement.

Chaînes de caractères et dérivés

dojo offre des supports permettant de gérer les chaînes de caractères ainsi que leurs dérivés.

Ces derniers incluent les chaînes de caractères elles-mêmes, avec le module `dojo.string`, les dates, avec le module `dojo.date`, et les expressions régulières, avec le module `dojo.regexp`.

Chaînes de caractères

Le module de dojo fournissant des fonctions de manipulation des chaînes de caractères est `dojo.string`. Dans ce module, les noms des fonctions sont préfixés par `dojo.string`.

Le tableau 7.7 récapitule ces différentes fonctions.

Tableau 7.7 Fonctions du module *dojo.string*

Fonction	Paramètre	Description
`dojo.string.capitalize`	Une chaîne de caractères	Met en majuscule la première lettre de chaque mot de la chaîne de caractères.
`dojo.string.endsWith`	Une chaîne de caractères, une chaîne de comparaison et éventuellement un drapeau	Détermine si la fin d'une chaîne de caractères correspond à celle spécifiée en paramètre. Un drapeau permet de spécifier si la comparaison doit être sensible à la casse.
`dojo.string.endsWithAny`	Une chaîne de caractères et une liste de chaînes de comparaison	Détermine si la fin d'une chaîne de caractères correspond à l'une de celles spécifiées en paramètre.
`dojo.string.escape`	Un type de contenu et une chaîne de caractères	Convertit et encode une chaîne de caractères pour un type de contenu spécifié en paramètre. Les types supportés sont `html` pour le langage HTML, `xhtml` pour le langage xHTML, `sql` pour le langage SQL, `regexp` pour les expressions régulières, `js` pour JavaScript et `ascii` pour l'encodage ASCII.
`dojo.string.has`	Une chaîne de caractères et un ensemble de chaînes	Détermine si une des chaînes de caractères passées en paramètre est contenue dans une chaîne.
`dojo.string.isBlank`	Une chaîne de caractères	Retourne la valeur `true` si la chaîne de caractères est uniquement composée d'espaces et `false` dans le cas contraire.
`dojo.string.normalizeNewlines`	Une chaîne de caractères et la chaîne correspondant à un retour à la ligne	Permet d'homogénéiser tous les retours à la ligne d'une chaîne de caractères en se fondant sur la chaîne représentant ces retours.
`dojo.string.pad`	Une chaîne de caractères, la longueur de la chaîne résultante et éventuellement un caractère ainsi qu'un sens d'ajout	Ajoute avant ou après une chaîne de caractères un nombre spécifié en paramètre du caractère désiré. Si le caractère n'est pas spécifié, le caractère `0` est utilisé. Le sens d'ajout peut prendre la valeur `1` pour un ajout en début de chaîne (par défaut) et `-1` pour un ajout en fin de chaîne.
`dojo.string.padLeft`	Une chaîne de caractères, la longueur de la chaîne résultante et éventuellement un caractère	Ajoute avant une chaîne de caractères un nombre spécifié en paramètre du caractère désiré. Si le caractère n'est pas spécifié, le caractère `0` est utilisé.
`dojo.string.padRight`	Une chaîne de caractères, la longueur de la chaîne résultante et éventuellement un caractère	Ajoute après une chaîne de caractères un nombre spécifié en paramètre du caractère désiré. Si le caractère n'est pas spécifié, le caractère `0` est utilisé.
`dojo.string.repeat`	Une chaîne de caractères, un nombre et éventuellement un séparateur	Permet de construire une chaîne de caractères en répétant la chaîne en paramètre du nombre de fois spécifié. Un séparateur peut être utilisé afin de séparer les chaînes répétées.
`dojo.string.startsWith`	Une chaîne de caractères, une chaîne de comparaison et éventuellement un drapeau	Détermine si le début d'une chaîne de caractères correspond à celle spécifiée en paramètre. Un drapeau permet de spécifier si la comparaison doit être sensible à la casse.
`dojo.string.startsWithAny`	Une chaîne de caractères et une liste de chaînes de comparaison	Détermine si le début d'une chaîne de caractères correspond à l'une de celles spécifiées en paramètre.

Tableau 7.7 Fonctions du module *dojo.string (suite)*

Fonction	Paramètre	Description
`dojo.string.summary`	Une chaîne de caractères et une longueur	Construit une sous-chaîne se finissant par « … » si la chaîne en paramètre dépasse la longueur spécifiée. Dans le cas contraire, la chaîne elle-même est retournée.
`dojo.string.trim`	Une chaîne de caractères	Supprime les espaces autour de la chaîne spécifiée en paramètre.
`dojo.string.trimEnd`	Une chaîne de caractères	Supprime les espaces à la fin de la chaîne spécifiée en paramètre.
`dojo.string.trimStart`	Une chaîne de caractères	Supprime les espaces au début de la chaîne spécifiée en paramètre.

Un premier ensemble de fonctions offrent la possibilité de vérifier qu'une chaîne de caractères correspond à une condition. C'est le cas des fonctions `dojo.string.startsWith`, `dojo.string.startsWithAny`, `dojo.string.endsWith` et `dojo.string.endsWithAny`, qui permettent respectivement de déterminer si une chaîne de caractères commence ou finit par une autre.

Le code suivant décrit la mise en œuvre des deux premières, sachant que les deux restantes s'utilisent de la même manière :

```
var chaine1 = "Ceci est une chaîne de test";
var chaine2 = "Une autre chaîne de test";

var test1Chaine1 = dojo.string.startsWith(chaine1, "Ceci");
// test1Chaine1 contient la valeur true
var test2Chaine1 = dojo.string.startsWithAny(chaine1, "Test", "Ceci", "Un");
// test2Chaine1 contient la valeur true
var test1Chaine2 = dojo.string.startsWith(chaine2, "Ceci");
// test1Chaine2 contient la valeur false
var test2Chaine2 = dojo.string.startsWithAny(chaine2, "Test", "Ceci", "Un");
// test2Chaine2 contient la valeur true
```

La fonction `dojo.string.has` détermine si une chaîne de caractères d'ensemble est contenue dans une autre, comme dans le code suivant :

```
var chaine1 = "Ceci est une chaîne de test";
var chaine2 = "Une autre chaîne de test";
var testChaine1 = dojo.string.has(chaine1, "Ceci");
// testChaine1 contient la valeur true
var test1Chaine2 = dojo.string.has(chaine2, "test");
// test1Chaine2 contient la valeur true
var test2Chaine2 = dojo.string.has(chaine2, "Test");
// test2Chaine2 contient la valeur false
```

Un autre ensemble de fonctions offrent la possibilité de réaliser des opérations sur des chaînes de caractères. C'est le cas notamment de la fonction `dojo.string.trim` et de ses déclinaisons afin de supprimer des espaces « autour » d'une chaîne de caractères et de la fonction `dojo.string.pad` et de ses déclinaisons afin d'ajouter un nombre de caractères identiques avant ou après une chaîne de caractères.

Le code suivant décrit la mise en œuvre de la fonction `dojo.string.trim` afin de supprimer les espaces au début et à la fin d'une chaîne de caractères :

```
var chaine = "   Ceci est une chaîne de caractères   ";
var nouvelleChaine = dojo.string.trim(chaine);
/* nouvelleChaine contient la chaîne de caractères « Ceci est une chaîne de caractères » */
```

Le code suivant décrit la mise en œuvre de la fonction `dojo.string.pad` afin d'ajouter au début et à la fin d'une chaîne de caractères trois espaces :

```
var chaine = "Ceci est une chaîne de caractères";
var chainePad = dojo.string.pad(chaine, chaine.length+3, " ", 1);
chainePad = dojo.string.pad(chainePad, chainePad.length+3, " ", -1);
/* chainePad contient la chaîne de caractères «   Ceci est une chaîne de caractères   » */
```

La fonction `dojo.string.capitalize` permet de mettre en majuscules les premiers caractères des mots d'une chaîne de caractères, comme dans le code suivant :

```
var chaine = "Ceci est une chaîne de caractères";
var chaineCapitalisee = dojo.string.capitalize(chaine);
/* chaineCapitalisee contient la chaîne de caractères « Ceci Est Une Chaîne De
   ➥Caractères » */
```

dojo fournit en outre la classe `dojo.string.Builder` afin de construire des chaînes de caractères. Cette classe correspond à l'équivalent de la classe `StringBuffer` fournie par le langage Java.

Le code suivant décrit la mise en œuvre de cette classe :

```
var builder = new dojo.string.Builder();
builder.append("Ceci est");
builder.append(" une chaîne de test");

var chaine = builder.toString();
/* chaine contient la chaîne de caractères « Ceci est une chaîne de test » */
```

Dates

Le module de dojo fournissant des fonctions de manipulation des dates est `dojo.date`. Dans ce module, les noms des fonctions sont préfixés par `dojo.date`.

Le tableau 7.8 récapitule les principales fonctions de ce module.

Tableau 7.8 Principales fonctions du module *dojo.date*

Fonction	Paramètre	Description
`dojo.date.add`	Une date, une unité et un nombre	Ajoute une période à une date. L'unité choisie permet de spécifier si la période est exprimée en secondes, minutes, heures, jours ou mois.
`dojo.date.compare`	Deux dates et éventuellement une option de comparaison	Permet de comparer deux dates. L'option permet de spécifier si la comparaison porte uniquement sur la date, l'heure ou les deux.
`dojo.date.format`	Une date et un format	Convertit une date en une chaîne de caractères en se fondant sur un format.
`dojo.date.getDayName`	Une date	Retourne le nom du jour de la semaine de la date spécifiée.
`dojo.date.getDayShortName`	Une date	Retourne le nom abrégé du jour de la semaine de la date spécifiée.
`dojo.date.getDaysInMonth`	Une date	Retourne le nombre de jours pour le mois courant de la date spécifiée.
`dojo.date.getMonthName`	Une date	Retourne le nom du mois de la date spécifiée.
`dojo.date.getMonthShortName`	Une date	Retourne le nom abrégé du mois de la date spécifiée.
`dojo.date.getTimezoneName`	Une date	Retourne le nom du fuseau horaire de la date spécifiée.
`dojo.date.getWeekOfYear`	Une date et éventuellement le premier jour de l'année	Retourne le numéro de la semaine dans l'année pour la date spécifiée.
`dojo.date.isLeapYear`	Une date	Détermine si l'année de la date spécifiée est bissextile.
`dojo.date.setDayOfYear` et `dojo.date.getDayOfYear`	Une date et un numéro Une date	Positionne et retourne le numéro du jour dans l'année pour la date spécifiée.
dojo.date.toRelativeString	Une date	Détermine l'écart en la date courante et une date spécifiée.

Les principales fonctions du tableau 7.8 se décomposent en deux ensembles. Le premier permet de récupérer et de positionner des informations pour des dates. Le second correspond à des fonctions de manipulation, de test et de formatage de dates.

La fonction `dojo.date.isLeapYear` permet de déterminer si l'année d'une date est bissextile, comme le montre le code suivant :

```
var date1 = new Date();
date1.setFullYear(2006);
var date1Bissextile = dojo.date.isLeapYear(date1);
/* date1Bissextile contient false car l'année 2006 n'est pas bissextile */
var date2 = new Date();
date2.setFullYear(2007);
var date2Bissextile = dojo.date.isLeapYear(date2);
/* date2Bissextile contient false car l'année 2007 n'est pas bissextile */
var date3 = new Date();
date3.setFullYear(2008);
var date3Bissextile = dojo.date.isLeapYear(date3);
/* date3Bissextile contient true car l'année 2008 est bissextile */
```

La fonction `dojo.date.format` offre la possibilité de convertir une date en chaîne de caractères en se fondant sur un format. Ce dernier utilise différents éléments afin de préciser les positions des éléments d'une date dans une chaîne.

Le tableau 7.9 récapitule les principaux de ces éléments.

Tableau 7.9 Éléments du format de date de la fonction *dojo.date.format*

Élément	Description
`%a` et `%A`	Correspond aux noms abrégé et complet du jour pour le paramètre de localisation.
`%b` et `%B`	Correspond aux noms abrégé et complet du mois pour le paramètre de localisation.
`%d`	Correspond au jour dans le mois courant.
`%H` et `%I`	Correspond à l'heure courante, respectivement sur 24 heures et 12 heures.
`%j`	Correspond au numéro du jour de l'année courante.
`%m`	Correspond au numéro du mois de l'année courante.
`%M`	Correspond à la minute courante.
`%S`	Correspond à la seconde courante.
`%y`	Correspond à l'année sur quatre caractères.

Le code suivant décrit la mise en œuvre de cette fonction de formatage de dates :

```
// Définition de la date: 10/09/2006 à 12h30min30s
var date = new Date();
date.setFullYear(2006);
date.setMonth(08);
date.setDate(10);
date.setHours(12);
date.setMinutes(30);
date.setSeconds(30);
var format = "%d/%m/%y, %H:%M:%S";

var chaineDate = dojo.date.format(date,format);
// chaineDate contient la valeur « 10/09/2006, 12:30:30 »
```

La fonction `dojo.date.compare` permet de comparer deux dates. Elle renvoie la valeur `0` si les deux dates sont égales, `1` si la première est supérieure à la seconde et `-1` dans le cas contraire. Elle offre en outre la possibilité d'utiliser une option de comparaison afin de spécifier le type de comparaison : comparaison des dates ou des heures uniquement ou des deux.

Le code suivant décrit la mise en œuvre de cette fonction :

```
/* date1 correspond à la date courante dont la valeur des heures est 0 */
var date1=new Date();
date1.setHours(0);
/* date2 correspond à la date courante dont les valeurs de l'année et des heures
➥sont respectivement 2005 et 12 */
var date2=new Date();
```

```
date2.setFullYear(2005);
date2.setHours(12);
var comparaison1 = dojo.date.compare(date1, date1);
/* comparaison1 contient 0 car les dates sont égales */
var comparaison2 = dojo.date.compare(date1, date2,
                         dojo.date.compareTypes.DATE);
/* comparaison2 contient 1 car la date de date1 est supérieure
   à celle de date2 */
var comparaison3 = dojo.date.compare(date2, date1,
                         dojo.date.compareTypes.DATE);
/* comparaison3 contient -1 (test inverse) */
var comparaison4 = dojo.date.compare(date1, date2,
                         dojo.date.compareTypes.TIME);
/* comparaison4 contient -1 car l'heure de date1 est inférieure à
   celle de date2 */
var comparaison5 = dojo.date.compare(date1, date2,
          dojo.date.compareTypes.DATE|dojo.date.compareTypes.TIME);
/* comparaison5 contient 1 (test inverse) */
```

La fonction `dojo.date.add` offre la possibilité d'ajouter une période à une date. Elle est très pratique pour déterminer la date obtenue après un nombre de jours, par exemple. Cette période peut être spécifiée en jours, comme dans le code suivant :

```
// Définition de la date: 29/09/2006 à 12h30min30s
var date = new Date();
date.setFullYear(2006);
date.setMonth(08);
date.setDate(29);
date.setHours(12);
date.setMinutes(30);
date.setSeconds(30);
var dateObtenue = dojo.date.add(date, dojo.date.dateParts.DAY, 5);
// dateObtenue correspond à la date: 04/10/2006 à 12h30min30s
```

La fonction `dojo.date.add` se fonde sur la structure `dojo.date.dateParts` afin de définir l'unité de temps de la période ajoutée. Cette structure inclut l'année, le mois, le jour, l'heure, la minute, la seconde et la milliseconde.

Expressions régulières

Le module `dojo.regexp` de dojo fournit des fonctions implémentant diverses expressions régulières afin de vérifier suivant des critères précis et de formater une chaîne de caractères passée en paramètre. Dans ce module, les noms des fonctions sont préfixés par `dojo.regexp`.

Le tableau 7.10 récapitule ces différentes fonctions.

Tableau 7.10 Fonctions du module *dojo.regexp*

Fonction	Paramètre	Description
`currency`	Un objet de configuration	Retourne une expression régulière afin de vérifier qu'une chaîne de caractères correspond à une valeur monétaire.
`emailAddress`	Un objet de configuration	Retourne une expression régulière afin de vérifier qu'une chaîne de caractères correspond à une adresse e-mail.
`emailAddressList`	Un objet de configuration	Retourne une expression régulière afin de vérifier qu'une chaîne de caractères correspond à une liste d'adresses e-mail.
`host`	Un objet de configuration	Retourne une expression régulière afin de vérifier qu'une chaîne de caractères correspond à un nom de domaine ou une adresse IP.
`integer`	Un objet de configuration	Retourne une expression régulière afin de vérifier qu'une chaîne de caractères correspond à un nombre entier.
`ipaddress`	Un objet de configuration	Retourne une expression régulière afin de vérifier qu'une chaîne de caractères correspond à une adresse IP.
`numberFormat`	Un objet de configuration	Retourne une expression régulière afin de vérifier qu'une chaîne de caractères contient bien des nombres à des positions spécifiées.
`realNumber`	Un objet de configuration	Retourne une expression régulière afin de vérifier qu'une chaîne de caractères correspond à un nombre réel.
`time`	Un objet de configuration	Retourne une expression régulière afin de vérifier qu'une chaîne de caractères correspond à un format de représentation d'un temps.
`url`	Un objet de configuration	Retourne une expression régulière afin de vérifier qu'une chaîne de caractères correspond à une URL valide.
`us.state`	Un objet de configuration	Retourne une expression régulière afin de vérifier qu'une chaîne de caractères correspond à l'abréviation d'un territoire ou d'un État d'Amérique du Nord.

Un premier ensemble de fonctions se fondent sur un objet de configuration spécifiant différentes options afin de créer les expressions régulières. C'est le cas des fonctions `dojo.regexp.emailAddress` et `dojo.regexp.emailAddressList`, qui vérifient la validité du format d'adresses e-mail.

Le code suivant décrit leur mise en œuvre :

```
// Vérification d'adresses email
var chaineEmail1 = "email@eyrolles.com";
var chaineEmail2 = "email_eyrolles.com"
var reAdresseEmail = new RegExp(dojo.regexp.emailAddress());
var chaineEmail1Correcte = reAdresseEmail.test(chaineEmail1);
// chaineEmail1Correcte contient la valeur true
var chaineEmail2Correcte = reAdresseEmail.test(chaineEmail2);
// chaineEmail2Correcte contient la valeur false
// Vérification de listes d'adresses email
var chaineListeEmail1 = "email@eyrolles.com,email1@eyrolles.com";
var chaineListeEmail2 = "email_eyrolles.com,email1_eyrollescom"
var reListeAdresseEmail = new RegExp(dojo.regexp.emailAddressList());
var chaineEmail1Correcte = reListeAdresseEmail.test(chaineListeEmail1);
// chaineEmail1Correcte contient la valeur true
var chaineEmail2Correcte = reListeAdresseEmail.test(chaineListeEmail2);
// chaineEmail2Correcte contient la valeur false
```

Un autre ensemble se fonde sur des formats afin de vérifier le bon format d'une chaîne de caractères. Les fonctions de ce type offrent la possibilité de vérifier le format de chaînes utilisant des nombres, telles que les heures et les numéros de téléphone.

La première vérification est mise en œuvre par la fonction `dojo.regexp.time`. Un format doit être spécifié en paramètre, format se fondant sur les éléments récapitulés au tableau 7.11.

Tableau 7.11 Éléments du format de la fonction *dojo.regexp.time*

Élément	Description
`h`	Correspond à la valeur des heures de 0 à 12. Cette valeur n'est pas précédée de zéro dans le cas d'une valeur inférieure à 10.
`H`	Correspond à la valeur des heures de 0 à 24. Cette valeur n'est pas précédée de zéro dans le cas d'une valeur inférieure à 10.
`hh`	Correspond à la valeur des heures de 0 à 12. Cette valeur doit être précédée de zéro dans le cas d'une valeur inférieure à 10.
`HH`	Correspond à la valeur des heures de 0 à 24. Cette valeur doit être précédée de zéro dans le cas d'une valeur inférieure à 10.
`m`	Correspond à la valeur des minutes. Cette valeur n'est pas précédée de zéro dans le cas d'une valeur inférieure à 10.
`mm`	Correspond à la valeur des minutes. Cette valeur doit être précédée de zéro dans le cas d'une valeur inférieure à 10.
`s`	Correspond à la valeur des secondes. Cette valeur n'est pas précédée de zéro dans le cas d'une valeur inférieure à 10.
`ss`	Correspond à la valeur des secondes. Cette valeur doit être précédée de zéro dans le cas d'une valeur inférieure à 10.
`t`	Correspond à la valeur « AM » ou « PM » en majuscules ou minuscules.

La seconde vérification se fonde sur la fonction `dojo.regexp.numberFormat`, dont le format passé en paramètre utilise le caractère # afin de spécifier la place des nombres dans la chaîne de caractères à vérifier. Le caractère `?` peut également être mis en œuvre afin de spécifier des nombres optionnels. Ainsi, les formats peuvent être de la forme suivante :

```
var format1 = "(+###) ## ## ## ## ##";
// Pour un numéro de téléphone avec indicatif : (+033) 01 02 03 04 05
var format2 = "###-######";
// Pour une chaîne du type : 342-54303
var format3 = "### ##### ##???";
// Pour une chaine du type 435 58394 455 ou 435 58394 45532
```

Le code suivant décrit la mise en œuvre des fonctions `dojo.regexp.time` (repère ❶) et `dojo.regexp.numberFormat` (repère ❷) :

```
var chaineHeureCorrecte = "13:45:22";
var chaineHeureIncorrecte = "1:45:22 PM";
var reHeure = new RegExp(dojo.regexp.time({ ←❶
    format: "HH:mm:ss"
}));
var heureCorrecte = reHeure.test(chaineHeureCorrecte);
// heureCorrecte contient la valeur true
var heureIncorrecte = reHeure.test(chaineHeureIncorrecte);
```

```
// heureIncorrecte contient la valeur false
var chaineNombre = "4353-756398";
var reChaineNombre = new RegExp(dojo.regexp.numberFormat({ ←❷
    format: "####-######"
}));
var chaineNombreCorrecte = reChaineNombre.test(chaineNombre);
// chaineNombreCorrecte contient la valeur true
```

Support des collections

dojo implémente différents types de collections afin de permettre la mise en œuvre de structures de données évoluées, telles que les listes, les arbres, les queues, les piles et les tables de hachage.

Cette section détaille les principales collections mises à disposition par dojo ainsi que le support de dojo permettant de les parcourir.

Collections de base

La bibliothèque dojo met en œuvre différentes implémentations des listes et des tables de hachage par le biais de classes telles que `dojo.collections.ArrayList` et `dojo.collections.Dictionnary`.

La première implémente une liste non triée, à l'instar de la classe `ArrayList` fournie par Java. Elle fournit différentes méthodes afin d'insérer et de supprimer des éléments, de rechercher des éléments et de parcourir la liste.

Le tableau 7.12 récapitule les différentes méthodes de la classe `dojo.collections.ArrayList`.

Tableau 7.12 Méthodes de la classe *dojo.collections.ArrayList*

Méthode	Paramètre	Description
`add`	L'élément à ajouter	Ajoute un élément à la liste.
`addRange`	Le tableau contenant les éléments à ajouter	Ajoute une liste d'éléments à la liste.
`clear`	-	Supprime tous les éléments contenus dans la liste.
`clone`	-	Retourne une copie de la liste.
`contains`	Un élément à rechercher	Détermine si l'élément en paramètre est contenu dans la liste.
`forEach`	Une fonction de rappel ainsi qu'éventuellement son contexte d'exécution	Permet de parcourir la collection en se fondant sur une fonction de rappel. Cette dernière doit prendre en paramètre l'élément courant, son indice ainsi qu'un tableau représentant la liste.
`getIterator`	-	Retourne un itérateur afin de parcourir la liste.
`indexOf`	Un élément	Retourne la position de l'élément passé en paramètre dans la liste. S'il n'est pas présent, la valeur `-1` est renvoyée.
`insert`	Un élément ainsi que l'index d'insertion	Insère un élément à une position donnée.
`item`	Un indice	Retourne l'élément correspondant à la position spécifiée en paramètre.

Tableau 7.12 Méthodes de la classe *dojo.collections.ArrayList (suite)*

Méthode	Paramètre	Description
`remove`	Un élément	Supprime de la liste l'élément passé en paramètre.
`removeAt`	Un indice	Supprime de la liste l'élément dont l'index est passé en paramètre.
`reverse`	-	Inverse l'ordre des éléments de la liste.
`setByIndex`	Un indice et un élément	Insère un élément à un indice spécifié dans la liste.
`sort`		Trie les éléments de la liste.
`toArray`	-	Convertit la liste en tableau.
`toString`	Rien ou un délimiteur	Retourne une représentation sous forme de chaîne de caractères de la liste.

Le code suivant décrit la façon d'utiliser cette classe afin de créer une liste, d'ajouter des éléments et de parcourir son contenu :

```
var liste = new dojo.collections.ArrayList();
liste.add("élément1");
liste.add("élément3");
liste.setByIndex(1, "élément2");
// Affiche les différents éléments de la liste, insérés ci-dessus
liste.forEach(function(element, indice, tableau) {
    alert(indice+": "+element);
});
```

Dans le code ci-dessus, `element` contient la valeur de l'élément courant de la liste lors du parcours de cette dernière et `indice` un nombre entier correspondant à sa position.

La seconde classe correspond à une table de hachage, c'est-à-dire une paire clé-valeur. Elle fournit différentes méthodes afin d'insérer et de supprimer des éléments, de rechercher des éléments et de parcourir la table.

Le tableau 7.13 récapitule les différentes méthodes de la classe `dojo.collections.Dictionary`.

Tableau 7.13 Méthodes de la classe *dojo.collections.Dictionary*

Méthode	Paramètre	Description
`add`	Une clé et une valeur	Ajoute dans la table de hachage une clé ainsi que sa valeur.
`clear`	-	Supprime tous les éléments contenus dans la table de hachage.
`clone`	-	Retourne une copie de la table de hachage.
`contains`	Une clé	Détermine si une clé est contenue dans la table pour la clé spécifiée.
`containsValue`	Une valeur	Détermine si un élément est contenu dans la table pour la valeur spécifiée.
`entry`	Une clé	Retourne l'entrée contenant une clé et une valeur et correspondant à la clé spécifiée en paramètre.
`forEach`	Une fonction de rappel ainsi que, éventuellement, son contexte d'exécution	Permet de parcourir la collection en se fondant sur une fonction de rappel. Cette dernière doit prendre en paramètre l'élément courant, son indice ainsi qu'un tableau représentant la liste.

Tableau 7.13 Méthodes de la classe *dojo.collections.Dictionary (suite)*

Méthode	Paramètre	Description
`getIterator`	-	Retourne l'itérateur permettant de parcourir la table de hachage.
`getKeyList`	-	Retourne la liste des clés de la table de hachage.
`getValueList`	-	Retourne la liste des valeurs de la table de hachage.
`remove`	Une clé	Supprime un élément de la table de hachage en se fondant sur sa clé.
`item`	Une clé	Retourne la valeur correspondant à une clé dans la table de hachage.

Le code suivant décrit la façon d'utiliser cette classe afin de créer une table de hachage, d'ajouter des éléments et de parcourir son contenu :

```
var tableHachage = new dojo.collections.Dictionary();
tableHachage.add("clé1", "élément1");
tableHachage.add("clé2", "élément2");
tableHachage.add("clé3", "élément3");
var listeCles = tableHachage.getKeyList();
/* listeCles correspond au tableau [ "clé1", "clé2", "clé2" ] */
var listeValeurs = tableHachage.getValueList();
/* listeValeurs correspond au tableau [ "élément1", " élément2", " élément3" ] */
// Affiche les différents éléments de la table, insérés ci-dessus
tableHachage.forEach(function(element, indice, tableau) {
    alert(indice+": "+element.key+", "+element.value);
});
```

Dans le code ci-dessus, `element` contient la paire clé-valeur (respectivement les attributs `key` et `value`) de l'élément courant de la table de hachage lors du parcours de cette dernière et `indice` un nombre entier correspondant à sa position.

Nous verrons plus loin dans ce chapitre que ces classes peuvent utiliser le support des classes d'itération afin d'être parcourues.

Structures de données avancées

Les structures de données avancées fournies par dojo correspondent aux queues, aux piles, aux graphes et aux arbres binaires. Nous ne détaillons dans cette section que la mise en œuvre des deux premières structures.

La première est implémentée par la classe `dojo.collections.Queue`. Cette structure correspond à une file d'attente, ou FIFO (First In First Out), dont le premier élément qui lui est ajouté est le premier à en sortir. Une mise en œuvre de cette structure a été abordée au chapitre 2 en se fondant sur les tableaux JavaScript.

Le tableau 7.14 récapitule les méthodes de la classe `dojo.collections.Queue`.

Tableau 7.14 Méthodes de la classe *dojo.collections.Queue*

Méthode	Paramètre	Description
`clear`	-	Supprime tous les éléments contenus dans la queue.
`clone`	-	Retourne une copie de la queue.
`contains`	Une valeur	Détermine si un élément est contenu dans la queue pour la clé spécifiée.
`copyTo`	Un tableau et un indice	Copie le contenu dans la queue du tableau passé en paramètre à partir de la position spécifiée.
`dequeue`	-	Retourne l'élément suivant de la queue et le supprime de celle-ci.
`enqueue`	Un élément	Ajoute un élément à la fin de la queue.
`forEach`	Une fonction de rappel ainsi qu'éventuellement son contexte d'exécution	Permet de parcourir la collection en se fondant sur une fonction de rappel. Cette dernière doit prendre en paramètre l'élément courant, son indice ainsi qu'un tableau représentant la liste.
`getIterator`	-	Retourne l'itérateur permettant de parcourir la queue.
`peek`	-	Retourne l'élément suivant de la queue sans en modifier son contenu.
`toArray`	-	Retourne une représentation sous forme de tableau de la liste.

Le code suivant décrit la mise en œuvre de cette classe afin de traiter des données en se fondant sur une queue :

```
var queue = new dojo.collections.Queue();
queue.enqueue("élément1");
queue.enqueue("élément2");
queue.enqueue("élément3");
var premierElementSansRetrait = queue.peek();
// premierElementSansRetrait contient la valeur "élément1"
var elementExtrait = queue.dequeue();
// elementExtrait contient la valeur "élément1"
elementExtrait = queue.dequeue();
// elementExtrait contient la valeur "élément2"
elementExtrait = queue.dequeue();
// elementExtrait contient la valeur "élément3"
```

La seconde structure est implémentée par la classe `dojo.collections.Stack`. Elle correspond à une pile, ou LIFO (Last In First Out), dans laquelle le dernier élément qui lui est ajouté est le premier à en sortir. Une mise en œuvre de cette structure a également été abordée au chapitre 2 en se fondant sur les tableaux JavaScript.

Le tableau 7.15 récapitule les méthodes de la classe `dojo.collections.Stack`.

Tableau 7.15 Méthodes de la classe *dojo.collections.Stack*

Méthode	Paramètre	Description
`clear`	-	Supprime tous les éléments contenus dans la pile.
`clone`	-	Retourne une copie de la pile.
`contains`	Une valeur	Détermine si un élément est contenu dans la pile pour la clé spécifiée.
`copyTo`	Un tableau et un indice	Copie le contenu dans la pile du tableau passé en paramètre à partir de la position spécifiée.

Tableau 7.15 Méthodes de la classe *dojo.collections.Stack (suite)*

Méthode	Paramètre	Description
`forEach`	Une fonction de rappel ainsi qu'éventuellement son contexte d'exécution	Permet de parcourir la collection en se fondant sur une fonction de rappel. Cette dernière doit prendre en paramètre l'élément courant, son indice ainsi qu'un tableau représentant la liste.
`getIterator`	-	Retourne l'itérateur permettant de parcourir la pile.
`peek`	-	Retourne l'élément suivant de la pile sans en modifier le contenu.
`pop`	-	Retourne l'élément suivant au sommet de la pile et le supprime de celle-ci.
`push`	Un élément	Ajoute un élément au sommet de la pile.
`toArray`	-	Retourne une représentation sous forme de tableau de la pile.

Le code suivant décrit la mise en œuvre de cette classe afin de traiter des données en se fondant sur une pile :

```
var pile = new dojo.collections.Stack();
pile.push("élément1");
pile.push("élément2");
pile.push("élément3");
var premierElementSansRetrait = pile.peek();
// premierElementSansRetrait contient la valeur "élément3"
var elementExtrait = pile.pop();
// elementExtrait contient la valeur "élément3"
elementExtrait = pile.pop();
// elementExtrait contient la valeur "élément2"
elementExtrait = pile.pop();
// elementExtrait contient la valeur "élément1"
```

Itérateurs

dojo fournit différentes classes pour parcourir les collections décrites aux sections précédentes. Ces classes correspondent à des itérateurs, entités permettant de se déplacer progressivement dans une collection.

Le premier, implémenté par le biais de la classe `dojo.collections.Iterator`, offre la possibilité de parcourir des listes telles que celles implémentées par les classes `dojo.collections.ArrayList` et `dojo.collections.SortedList`.

Le tableau 7.16 récapitule les méthodes disponibles pour cette classe.

Tableau 7.16 Méthodes de la classe *dojo.collections.Iterator*

Méthode	Paramètre	Description
`atEnd`	-	Détermine si l'itérateur se trouve à la fin du parcours de la collection.
`get`	-	Retourne l'élément courant de l'itération.
`map`	Une fonction de rappel ainsi qu'éventuellement son contexte d'exécution	Retourne un tableau rempli en se fondant sur la fonction de rappel et les éléments contenus dans l'itérateur d'une collection.
`reset`	-	Réinitialise l'itérateur au début du parcours de la collection.

Le code suivant décrit la mise en œuvre de cette classe afin de parcourir une liste fondée sur la classe `dojo.collections.ArrayList` :

```
var liste = new dojo.collections.ArrayList();
liste.add("élément1");
liste.add("élément2");
liste.add("élément3");
// Affiche les différents éléments de la liste, insérés ci-dessus
for(var iterateur = liste.getIterator(); !iterateur.atEnd(); ) {
    var element = iterateur.get();
    alert("Element: "+element);
}
```

dojo fournit également la classe `dojo.collections.DictionaryIterator` afin de parcourir des tables de hachage telles que celle implémentée par la classe `dojo.collections.Dictionary`. Elle met à disposition les mêmes méthodes que la classe `dojo.collections.Iterator`.

L'élément renvoyé par la méthode `get` de cette classe est désormais une structure contenant la clé et la valeur d'une entrée dans la table de hachage. Cette structure est implémentée par l'intermédiaire de la classe `dojo.collections.DictionaryEntry`.

Le code suivant décrit la mise en œuvre de cette classe afin de parcourir une liste fondée sur la classe `dojo.collections.Dictionary` :

```
var tableHachage = new dojo.collections.Dictionary();
tableHachage.add("clé1", "élément1");
tableHachage.add("clé2", "élément2");
tableHachage.add("clé3", "élément3");
// Affiche les différents éléments de la table, insérés ci-dessus
for(var i = tableHachage.getIterator(); !i.atEnd(); ) {
    var element = i.get();
    alert("Element: "+element.key+", "+element.value);
}
```

Support du DOM

La bibliothèque dojo fournit deux fonctions permettant d'accéder aux éléments de l'arbre DOM d'une page HTML. Aucun module ne doit être spécifié afin de les utiliser, car elles font partie des fonctions centrales de dojo.

La fonction `dojo.byId` retourne un élément DOM en se fondant sur son identifiant. Elle encapsule la méthode `getElementById` de l'objet `document` d'une page HTML.

La fonction `dojo.byTag` retourne un tableau d'éléments DOM en se fondant sur un nom de balise. Elle encapsule la méthode `getElementsByTagName` de l'objet `document` d'une page HTML.

Afin de détailler le support du DOM par dojo, nous allons reprendre dans cette section l'exemple utilisé au chapitre 4 :

```
<div id="maZone">
    <span id="monTexte"><b>Un texte</b></span>
    <div id="monAutreZone">
         <p>Un autre texte</p>
    </div>
</div>
```

L'exemple suivant décrit la mise en œuvre des fonctions `dojo.byId` et `dojo.byTag` en se fondant sur le fragment HTML ci-dessus :

```
var zone = dojo.byId("maZone");
// zone contient la référence à la première balise
var listeBalises = dojo.byTag("div");
// listeBalises contient les différentes balises dont le nom est div
```

En plus de ces deux fonctions, dojo offre un support permettant de manipuler l'arbre DOM d'une page Web par l'intermédiaire de son module `dojo.dom`. Ce module fournit un ensemble de constantes et de fonctions à cet effet.

Le tableau 7.17 récapitule les principales constantes relatives au type du nœud DOM.

Tableau 7.17 Principales constantes relatives au type du nœud DOM

Constante	Valeur	Description
`dojo.dom.ELEMENT_NODE`	1	Correspond à un nœud.
`dojo.dom.ATTRIBUTE_NODE`	2	Correspond à un attribut.
`dojo.dom.TEXT_NODE`	3	Correspond à un nœud de texte.
`dojo.dom.CDATA_SECTION_NODE`	4	Correspond à un nœud de type CDATA.
`dojo.dom.ENTITY_NODE`	6	Correspond à une entité XML.
`dojo.dom.COMMENT_NODE`	8	Correspond à un commentaire.
`dojo.dom.DOCUMENT_NODE`	9	Correspond à un nœud racine d'un document XML.

Ce module met également à disposition différentes fonctions utilitaires afin de faciliter la manipulation des éléments DOM. Le tableau 7.18 récapitule ces fonctions.

Tableau 7.18 Fonctions de manipulation d'éléments DOM

Fonction	Paramètre	Description
`dojo.dom.copyChildren`	Un nœud source, un nœud cible et un drapeau	Copie tous les éléments enfants d'une balise source vers une balise destination. Si la valeur du troisième paramètre vaut `true`, tous les éléments enfants de type texte sont supprimés.
`dojo.dom.createDocument`	-	Crée un nœud de type document.
`dojo.dom.createDocument-FromText`	-	Crée un nœud de type document et l'initialise en se fondant sur du code XML contenu dans une chaîne de caractères.
`dojo.dom.firstElement`	Un nœud et un nom de balise	Retourne le premier élément enfant de type : `dojo.dom.ELEMENT_NODE` d'un nœud. Si un nom de balise est spécifié, la fonction retourne le premier élément correspondant.

Tableau 7.18 Fonctions de manipulation d'éléments DOM *(suite)*

Fonction	Paramètre	Description
`dojo.dom.getAncestors`	Un nœud, une fonction de rappel et un drapeau	Retourne la liste des parents d'un nœud en appliquant éventuellement un filtrage. Ce dernier est implémenté par une fonction de rappel. Si elle n'est pas spécifiée, aucun filtrage n'est réalisé. Si la valeur du troisième élément vaut `true`, retourne uniquement le premier élément correspondant.
`dojo.dom.getAncestorsByTag`	Un nœud, un nom de balise et un drapeau	Retourne la liste des parents d'un nœud correspondant au nom de balise spécifié. Si la valeur du troisième élément vaut `true`, retourne uniquement le premier élément correspondant.
`dojo.dom.getFirstAncestor-ByTag`	Un nœud et un nom de balise	Retourne le premier parent d'un nœud correspondant au nom de balise spécifié.
`dojo.dom.getUniqueId`	-	Génère un identifiant unique.
`dojo.dom.hasParent`	Un nœud	Détermine si un nœud est un enfant d'un autre.
`dojo.dom.innerXML`	Un nœud	Correspond au contenu d'un nœud au format XML.
`dojo.dom.insertAfter`	Un nœud de référence, un nœud à insérer et un drapeau	Ajoute un nœud au même niveau et après un nœud de référence. Si les deux nœuds en paramètres sont égaux ou le nœud à insérer se trouve déjà avec le nœud de référence, le nœud n'est ajouté que si le drapeau vaut `true`.
`dojo.dom.insertAtIndex`	Un nœud à insérer, un nœud cible pour l'insertion et un indice	Ajoute un nœud dans la liste des enfants d'un nœud cible. Si la valeur de l'indice est supérieure au nombre des enfants du nœud, il est ajouté en dernière position.
`dojo.dom.insertAtPosition`	Un nœud de référence, un nœud à insérer et une position	Ajoute un nœud au même niveau qu'un nœud de référence. La position du nœud inséré dépend du paramètre de position, lequel peut prend les valeurs `before`, `after`, `first` ou `last`. Son omission correspond à la valeur `last`.
`dojo.dom.insertBefore`	Un nœud de référence, un nœud à insérer et un drapeau	Ajoute un nœud au même niveau et avant un nœud de référence. Si les deux nœuds en paramètres sont égaux ou le nœud à insérer se trouve déjà avec le nœud de référence, le nœud n'est ajouté que si le drapeau vaut `true`.
`dojo.dom.isDescendantOf`	Un nœud, un nœud parent potentiel et un drapeau	Détermine si un nœud correspond à un enfant d'un autre nœud. Si la valeur du troisième élément vaut `true`, réalise la fonctionnalité au sens strict.
`dojo.dom.isNode`	Un élément de l'arbre DOM	Détermine si le paramètre est un nœud.
`dojo.dom.isTag`	Un nœud et différents noms de balises	Détermine si un nœud correspond à un des noms de balise passés en paramètres. Si tel est le cas, le nom correspondant est retourné.
`dojo.dom.lastElement`	Un nœud et un nom de balise	Retourne le dernier élément enfant de type : `dojo.dom.ELEMENT_NODE` d'un nœud. Si un nom de balise est spécifié, la fonction retourne le dernier élément correspondant.
`dojo.dom.moveChildren`	Un nœud source, un nœud cible et un drapeau	Déplace tous les éléments enfants d'une balise source vers une balise destination. Si la valeur du troisième paramètre vaut vrai, tous les éléments enfant de type texte sont supprimés.
`dojo.dom.nextElement`	Un nœud et un nom de balise	Retourne l'élément suivant de type : `dojo.dom.ELEMENT_NODE` d'un nœud. Si un nom de balise est spécifié, la fonction retourne l'élément suivant correspondant.

Tableau 7.18 Fonctions de manipulation d'éléments DOM *(suite)*

Fonction	Paramètre	Description
`dojo.dom.prependChild`	Un nœud à insérer et un nœud sous lequel insérer	Ajoute un nœud en première position des enfants d'un autre nœud.
`dojo.dom.prevElement`	Un nœud et un nom de balise	Retourne l'élément précédant de type : `dojo.dom.ELEMENT_NODE` d'un nœud. Si un nom de balise est spécifié, la fonction retourne l'élément précédant correspondant.
`dojo.dom.removeChildren`	Un nœud	Supprime tous les éléments enfants d'un nœud.
`dojo.dom.removeNode`	Un nœud	Supprime le nœud spécifié de son nœud père.
`dojo.dom.replaceChildren`	Un nœud source et un nœud à ajouter	Supprime tous les éléments enfants d'un nœud et lui ajoute le nœud spécifié en second paramètre.
`dojo.dom.setAttributeNS`	Un nœud, un espace de nommage et un nom et une valeur d'attribut	Correspond à l'implémentation de la méthode `setAttributNS` du DOM niveau 2.
`dojo.dom.textContent`	Un nœud et éventuellement une chaîne de caractères	Correspond à l'implémentation de l'attribut `textContent` du DOM niveau 3.

La fonction `dojo.dom.moveChildren` permet de déplacer tous les nœuds enfants d'un nœud vers un autre. Le code suivant décrit la façon d'utiliser cette fonction afin de déplacer les enfants de la balise `span` d'identifiant `monTexte` dans la balise `div` d'identifiant `monAutreZone` :

```
var parentSource = dojo.byId("monTexte");
var parentCible = dojo.byId("monAutreZone");
dojo.dom.moveChildren(parentSource, parentCible, false);
```

Ce code permet de modifier l'arbre DOM en mémoire de la page afin d'obtenir le résultat suivant :

```
<div id="maZone">
    <span id="monTexte"></span>
    <div id="monAutreZone">
         <b>Un texte</b> ← ❶
         <p>Un autre texte</p>
    </div>
</div>
```

Le repère ❶ met en valeur le bloc déplacé par l'utilisation de la fonction `dojo.dom.moveChildren`.

La fonction `dojo.dom.removeChildren` offre la possibilité de supprimer tous les nœuds enfants d'un nœud. Le code suivant permet de supprimer tout le contenu de la balise `span` d'identifiant `monTexte` :

```
var source = dojo.byId("monTexte");
dojo.dom.removeChildren(source);
```

Le code ci-dessus permet d'obtenir l'arbre DOM en mémoire de la page suivante :

```
<div id="maZone">
    <span id="monTexte"></span>←❶
    <div id="monAutreZone">
         <p>Un autre texte</p>
    </div>
</div>
```

Le repère ❶ met en valeur le bloc dans lequel les nœuds enfants ont été supprimés par l'utilisation de la fonction `dojo.dom.removeChildren`.

D'autres fonctions, telles que `dojo.dom.insertBefore` et `dojo.dom.insertAfter`, permettent d'insérer des éléments à différentes positions dans l'arbre DOM d'une page.

Ces fonctions ajoutent respectivement des nœuds avant et après un nœud de référence. Le code suivant implémente la façon d'ajouter un texte en italique (balise `i`) avant et après le texte en gras (balise `b`) :

```
var baliseItaliqueAvant = document.createElement("i");
var texteItalique = document.createTextNode("Italique");
baliseItaliqueAvant.appendChild(texteItalique);
var baliseItaliqueApres = baliseItaliqueAvant.clone(true);
var source = dojo.byId("monTexte");
var baliseGras = source.childNodes[0];
dojo.dom.insertBefore(baliseItaliqueAvant, baliseGras);
dojo.dom.insertAfter(baliseItaliqueApres, baliseGras);
```

Le code ci-dessus permet d'obtenir l'arbre DOM en mémoire de la page suivante :

```
<div id="maZone">
    <span id="monTexte">
        <i>Italique</i>←❶
        <b>Un texte</b>←❷
        <i>Italique</i>←❸
    </span>
    <div id="monAutreZone">
         <p>Un autre texte</p>
    </div>
</div>
```

Les repères ❶ et ❸ mettent respectivement en valeur les blocs ajoutés avant et après le bloc identifié par le repère ❷ par l'utilisation des fonctions `dojo.dom.insertBefore` et `dojo.dom.insertAfter`.

Dans le code JavaScript ci-dessus, les fonctions `dojo.createElement` et `createTextNode` du DOM sont utilisées afin de créer les nœuds à insérer. Ces fonctions ont été abordées au chapitre 4.

La fonction `dojo.dom.prependChild` peut être utilisée en complément des deux précédentes afin d'ajouter un enfant à un nœud, même s'il n'en possède pas.

Pour finir, différentes fonctions offrent la possibilité d'accéder à différents nœuds de l'arbre DOM de manière relative à un nœud. Ainsi, les fonctions `dojo.dom.firstElement`, `dojo.dom.lastElement`, `dojo.dom.prevElement` et `dojo.dom.nextElement` permettent respectivement d'avoir le premier et le dernier nœud d'un ensemble de nœuds ainsi que les nœuds précédant et suivant un nœud.

Le code suivant décrit la mise en œuvre de ces fonctions sur le fragment HTML utilisé dans cette section :

```
var element = dojo.byId("maZone");
var premierElement = dojo.dom.firstElement(element);
// premierElement correspond à l'élément d'identifiant "monTexte"
var dernierElement = dojo.dom.lastElement(element);
/* dernierElement correspond à l'élément d'identifiant
   "monAutreZone" */
var precedentElement = dojo.dom.prevElement(element);
/* precedentElement est null puisqu'il n'y a pas
   d'élément précédent */
var suivantElement = dojo.dom.nextElement(element);
/* dernierElement correspond à l'élément d'identifiant
   "monAutreZone" */
```

Support des techniques Ajax

La bibliothèque dojo offre un intéressant support afin de mettre en œuvre simplement les techniques Ajax ainsi que l'appel à des services distants.

Dans cette section, nous abordons les mécanismes fondés sur la fonction générique `dojo.io.bind`, dont l'objectif est d'abstraire des mécanismes sous-jacents et de permettre ainsi de s'adapter au mieux au contexte d'exécution de l'application qui utilise cette fonction.

Fonction dojo.io.bind

La fonction `dojo.io.bind` correspond à l'entité centrale du support d'Ajax par la bibliothèque dojo. Elle intègre tous les mécanismes permettant d'envoyer une requête à partir d'une page sans la recharger et de recevoir la réponse correspondante.

Elle prend en paramètre un tableau associatif, qui permet de spécifier les différents attributs de configuration ainsi que les fonctions de rappel de la requête.

Le code suivant décrit un exemple simple d'utilisation de la fonction `dojo.io.bind` :

```
var proprietes = {
    url: "monUrl",
    mimetype: "text/plain",
    load: function(type, data) {
        // traitement de données texte
        (...)
    }
};
dojo.io.bind(proprietes);
```

Le tableau 7.19 récapitule les principaux attributs supportés par la fonction `dojo.io.bind`.

Tableau 7.19 Principaux attributs supportées par la fonction *dojo.io.bind*

Attribut	Description
`content`	Contenu de la requête sous forme de tableau associatif
`formNode`	Élément correspondant au formulaire à soumettre.
`method`	Méthode HTTP à utiliser afin d'envoyer la requête
`mimetype`	Type de contenu renvoyé par la ressource. La fonction gère de façon différente les différents types de contenus.
`multipart`	Spécifie si la requête est *multipart,* c'est-à-dire si elle contient plusieurs parties, pour l'envoi d'un formulaire. Cette fonctionnalité n'est pas supportée par toutes les couches de transport.
`sync`	Spécifie si la requête est synchrone. Les valeurs possibles sont `true` ou `false`.
`timeoutSeconds`	Délai d'attente maximal durant lequel la réponse doit être reçue.
`transport`	Spécifie explicitement la couche de transport à utiliser.
`url`	URL de la ressource
`useCache`	Spécifie si un cache doit être mis en œuvre.

Le tableau 7.20 récapitule les fonctions de rappel gérées par la fonction `dojo.io.bind`.

Tableau 7.20 Fonctions de rappel supportées par la fonction *dojo.io.bind*

Fonction	Paramètre	Description
`error`	Le type d'événement et l'erreur survenue	Appelée en cas d'erreur lors de l'envoi de la requête.
`handle`	Le type d'événement, les données et éventuellement l'instance de transport utilisée	Fonction de rappel générique. Elle est appelée en cas de succès ou d'erreur de la requête dans le cas où les méthode `error` et `load` ne sont pas spécifiées.
`load`	Le type d'événement, les données et éventuellement l'instance de transport utilisée	Appelée en cas de succès de la requête lorsque les résultats de la réponse sont reçus.
`timeout`	-	Appelée lorsque la réponse n'a pas été reçue dans le délai d'attente maximal spécifié. Est utilisée conjointement avec l'attribut `timeoutSeconds`.

Traitement du résultat d'une requête

Les paramètres passés à la requête permettent de spécifier des fonctions de rappel afin de traiter les données renvoyées. En cas de succès de la requête, dojo offre la possibilité d'utiliser la fonction de rappel `handle` ou `load`.

La fonction `load` permet de traiter exclusivement une requête exécutée avec succès. Cette fonction prend en paramètre le cas dans lequel on se trouve, les données et éventuellement une instance correspondant à l'événement.

Le code suivant décrit la mise en œuvre de cette fonction de rappel afin de traiter les données reçues suite à une requête :

```
var proprietes = {
    url: "monUrl",
    mimetype: "text/plain",
    load: function(type, data) {
        /* type contient « load » et data les données reçues sous forme de texte */
    }
};
dojo.io.bind(proprietes);
```

La fonction `handle` ne peut être utilisée que lorsque la fonction `load` n'est pas définie. Elle est beaucoup plus générale puisqu'elle permet de traiter aussi bien le succès que l'échec d'une requête.

Le code suivant décrit la mise en œuvre de cette fonction de rappel :

```
var proprietes = {
    url: "monUrl",
    mimetype: "text/plain",
    handle: function(type, data) {
        if( type == "load" ) {
            // Tout s'est bien passé
        } else if( type == "error" ) {
            // Une erreur s'est produite
        } else {
            // Dans tous les autres cas
        }
    }
};
dojo.io.bind(proprietes);
```

Gestion des erreurs

Bien que la fonction de rappel `handle` permette de gérer les erreurs survenues lors d'une requête Ajax, la fonction de rappel `error` peut être mise en œuvre afin de les gérer de manière spécifique, à l'instar de la fonction `load`.

La fonction `error` prend en paramètre le cas dans lequel on se trouve ainsi que l'erreur qui s'est produite.

Le code suivant décrit la mise en œuvre de la fonction de rappel `error` pour gérer les erreurs survenues lors de la requête :

```
var proprietes = {
    url: "monUrl",
    mimetype: "text/plain",
    error: function(type, error) {
        (...)
    }
};
dojo.io.bind(proprietes);
```

Formats de réponse

dojo supporte nativement différents formats de réponse. Il peut ainsi réaliser un traitement avant de passer les données reçues aux fonctions de rappel.

La propriété `mimetype` offre la possibilité de spécifier à dojo le format de données attendu. La bibliothèque sait de la sorte comment traiter les données reçues dans la réponse de la requête Ajax.

Le tableau 7.21 récapitule les formats de réponse supportés.

Tableau 7.21 Formats de réponse supportés par la fonction *dojo.io.bind*

Format	Description
`text/plain`	Correspond à un format texte. Dans ce cas, dojo ne réalise aucun traitement avant de fournir le contenu aux fonctions de rappel.
`text/javascript`	Correspond à un format contenant du code JavaScript. dojo évalue ce dernier et fournit l'objet résultant aux fonctions de rappel.
`text/xml`	Correspond à un format XML. Dans ce cas, dojo fournit le contenu aux fonctions de rappel sous forme de document XML.
`text/json`	Correspond à un format JSON. Dans ce cas, dojo construit des objets JavaScript correspondant aux données reçues en se fondant sur ce format.

Le code suivant décrit la mise en œuvre du format `text/json` :

```
var proprietes = {
    url: "monUrl",
    mimetype: "text/json",
    load: function(type, donnees, transport) {
        for( var element in donnees ) {
            dojo.debug(element, ":", donnees[element]);
        }
    }
};
dojo.io.bind(proprietes);
```

Cet exemple affiche le résultat suivant en se fondant sur le contenu de la réponse décrit plus bas :

```
DEBUG: now : Wed Aug 23 2006 15:44:19 GMT+0200
DEBUG: element1 : Valeur de l'élément 1
DEBUG: element2 : Valeur de l'élément 2
```

Le contenu de la réponse reçue correspond au texte suivant :

```
{
    now: new Date(),
    element1: "Valeur de l'élément 1",
    element2: "Valeur de l'élément 2"
}
```

Classes de transport disponibles

Bien que la bibliothèque dojo offre une unique fonction, `dojo.io.bind`, elle implémente différents mécanismes internes afin de mettre en œuvre Ajax. Ces implémentations, qui ne sont pas toutes fondées sur la classe `XMLHttpRequest`, permettent de résoudre des problématiques particulières, telles que les contraintes de sécurités issues de cette classe, qui empêchent l'appel d'une adresse d'un autre domaine.

Le tableau 7.22 récapitule les principales couches de transport fournies par la bibliothèque.

Tableau 7.22 Principales couches de transport fournies par dojo

Couche de transport	Description
`IframeTransport`	Se fonde sur une iframe cachée utilisée afin d'exécuter des requêtes Ajax. Le principal intérêt de cette couche de transport réside dans sa capacité à mettre en œuvre l'envoi de fichier (upload) avec Ajax tout en offrant la possibilité de réaliser des requêtes Ajax sur un autre domaine.
`ScriptSrcTransport`	Se fonde sur un élément HTML `script` généré à la volée afin de mettre en œuvre Ajax. Le principal intérêt de cette approche réside dans la possibilité de réaliser des requêtes Ajax sur un autre domaine.
`XMLHTTPTransport`	Se fonde sur la classe `XMLHttpRequest` pour exécuter les requêtes Ajax.

dojo détecte automatiquement l'implémentation de transport la plus appropriée. Il est cependant possible d'en spécifier une explicitement par l'intermédiaire de l'attribut `transport` dans les paramètres de la fonction `dojo.io.bind`.

Le code suivant décrit l'utilisation de ce mécanisme afin d'utiliser une couche de transport fondée sur `XMLHttpRequest` (repère ❶) :

```
var proprietes = {
    url: "monUrl",
    (...)
    mimetype: "text/plain",
    transport: "XMLHTTPTransport" ← ❶
};
dojo.io.bind(proprietes);
```

Ces différentes couches de transport ne possèdent pas les mêmes caractéristiques et sont mises en œuvre de manière transparente suivant les fonctionnalités utilisées. C'est le cas, par exemple, de l'envoi de fichier et de la gestion des boutons Précédent et Suivant dans le cadre d'Ajax, qui se fondent sur la classe `IframeTransport`.

Support de RPC

dojo offre la possibilité d'appeler un service distant de manière normalisée. La bibliothèque supporte différents protocoles dont JSON-RPC et celui des services Yahoo!.

Ces différents supports se fondent sur la méthode `dojo.io.bind` décrite à la section précédente afin d'envoyer les requêtes et de recevoir leurs réponses.

Plutôt que d'implémenter chaque fois les mécanismes d'appel aux services, dojo offre la possibilité de décrire par l'intermédiaire d'un tableau associatif les caractéristiques des services. Le format de ce tableau est appelé SMD (Simple Method Description). Ce mécanisme permet de généraliser les traitements décrits dans le code suivant :

```
function traiterResultats(type, data, event) {
    (...)
}
var methodeService = function(parametre1, parametre2) {
    var parametresRequete = { parametre1: parametre1,
                              parametre2: parametre2 };
    dojo.io.bind({
        url: "monUrl",
        load: traiterResultats,
        mimetype: "text/plain",
        content: parametresRequete
    });
};
// Execution du service
methodeService("parametre1", "parametre2");
```

Avec la syntaxe SMD, le service ci-dessus est décrit de la manière suivante :

```
{
    "serviceType": "JSON-RPC",
    "serviceURL": "monUrl",
    "methods": [
        {
            "name": "methodeService",
            "parameters": [
                { "name": "parametre1" },
                { "name": "parametre2" }
            ]
        }
    ]
}
```

Afin de mettre en œuvre JSON-RPC, dojo fournit la classe `dojo.rpc.JsonService`, qui prend en compte la description précédente. Cette classe étend la classe générique `dojo.rpc.RpcService`, laquelle contient toute la logique d'interprétation du format de description des services.

Le code suivant décrit la mise en œuvre de la classe `dojo.rpc.JsonService` :

```
// le contenu de description correspond au code JSON ci-dessus
var description = "maDescriptionDeService.smd";
var monService = dojo.rpc.JsonService(description);
monService.methodeService("parametre1", "parametre2");
```

Afin de recevoir la réponse de la requête, l'appel des méthodes du service renvoie des objets sur lesquels des observateurs peuvent être enregistrés par l'intermédiaire de leur méthode `addCallback`. Ces derniers sont notifiés lorsque la réponse à la requête est reçue.

De plus, la méthode de l'observateur prend en paramètre le résultat contenu dans la réponse.

Le code suivant décrit la mise en œuvre de cette méthode :

```
function traiterResultats(resultat) {
    (...)
}
(...)
var reception = monService.methodeService("parametre1", "parametre2");
reception.addCallback(traiterResultats);
```

Nous détaillons au chapitre 14 l'utilisation de la classe `dojo.rpc.YahooService` afin d'accéder aux services offerts par Yahoo!.

Soumission de formulaire

La bibliothèque dojo offre la possibilité d'envoyer de manière transparente les données d'un formulaire en utilisant une requête Ajax.

La mise en œuvre de cette fonctionnalité est très simple, puisqu'elle se fonde sur l'option `formNode`, qui référence le nœud correspondant au formulaire. La bibliothèque se charge de déterminer les éléments du formulaire et de les utiliser afin de construire les données contenues dans la requête Ajax.

Le code suivant décrit la mise en œuvre de cette fonctionnalité :

```
var proprietes = {
    url: "monUrl",
    mimetype: "text/plain",
    formNode: dojo.byId("monFormulaire"),
    load: function(type, data) {
        // traitement de données texte
        (...)
    }
};
dojo.io.bind(proprietes);
```

Le code ci-dessus se fonde sur le formulaire d'une page HTML décrit ci-dessous :

```
<html>
    (...)
    <body>
        (...)
        <form id="monFormulaire">
            <input type="hidden" name="champ1" value="valeur1"/>
            <input type="text" name="champ2" value="valeur2"/>
        </form>
        (...)
    </body>
</html>
```

En résumé

La bibliothèque dojo met l'accent sur la simplicité d'utilisation pour son support des techniques Ajax. Ainsi, la fonction centrale du support, `dojo.io.bind`, masque toute la complexité de mise en œuvre d'Ajax et intègre des mécanismes de détermination automatique des algorithmes à utiliser en fonction du contexte.

Dans cette section, nous avons détaillé les fonctionnalités de base de la fonction `dojo.io.bind`, telles que l'envoi de requêtes Ajax, le traitement des réponses, la gestion des erreurs ou le support des différents types de données reçues.

Cette fonction offre la possibilité de gérer les boutons Précédent et Suivant du navigateur par l'intermédiaire de fonctions de rappel. Elle permet également de mettre à jour l'adresse d'une page HTML suite à une requête Ajax afin de permettre leur historisation.

Traces applicatives

La bibliothèque dojo met à disposition un outil générique permettant de gérer des traces applicatives. Il est défini dans le module `dojo.logging.Logger`, qui doit être inclus par l'intermédiaire de la méthode `dojo.require`.

L'inclusion du module crée un objet `dojo.log` qui permet d'écrire des traces pour un niveau donné.

Le tableau 7.23 récapitule les différentes méthodes utilisables de cet objet afin d'écrire des traces applicatives. Chacune de ces méthodes prend en paramètre un message sous forme de chaîne de caractères.

Tableau 7.23 Méthodes de mise en œuvre des traces applicatives

Méthode	Description
`dojo.log.debug`	Ajoute une trace applicative avec le niveau `debug`.
`dojo.log.info`	Ajoute une trace applicative avec le niveau `info` (information).
`dojo.log.warn`	Ajoute une trace applicative avec le niveau `warn` (avertissement).
`dojo.log.err`	Ajoute une trace applicative avec le niveau `err` (erreur).
`dojo.log.crit`	Ajoute une trace applicative avec le niveau `crit` (erreur critique).
`dojo.log.exception`	Ajoute une trace applicative pour la levée d'une exception.

L'activation du mode de débogage se réalise par le biais de l'objet `djConfig` et de sa propriété `isDebug`. Si la valeur de cette dernière est `true`, toutes les traces applicatives sont affichées. Dans le cas contraire, ne sont affichés que les messages à caractère fatal, tels ceux relatifs à des levées d'exception.

Le code suivant décrit la configuration des traces affichées :

```
<html>
    <head>
        (...)
        <script>djConfig = { isDebug: true }</script>
        <script language="JavaScript" type="text/javascript" src="../../dojo.js">
        ➥</script>
        <script language="JavaScript" type="text/javascript">
            dojo.require("dojo.logging.Logger");
            (...)
        </script>
    </head>
    (...)
</body>
```

L'utilisation de ce support se réalise simplement par le biais des fonctions récapitulées au tableau 7.23, comme dans le code suivant :

```
<html>
    <head>
        (...)
        <script>djConfig = { isDebug: true }</script>
        <script language="JavaScript" type="text/javascript"
                                      src="../../dojo.js"></script>
        <script language="JavaScript" type="text/javascript">
            dojo.require("dojo.logging.Logger");
            dojo.addOnLoad(function() {
                var message1 = "affichage avec dojo.log.debug()";
                dojo.log.debug(message1);
                var message2 = "affichage avec dojo.log.info()";
                dojo.log.info(message2);
                var message3 = "affichage avec dojo.log.warn()";
                dojo.log.warn(message3);
                var message4 = "affichage avec dojo.log.err()";
                dojo.log.err(message4);
                var message5 = "affichage avec dojo.log.crit()";
                dojo.log.crit(message5);
                var message6 = "affichage avec dojo.log.exception()";
                try{
                    dojo.raise("une exception");
                } catch(err) {
                    dojo.log.exception(message6, err, true);
                }
            });
        </script>
    </head>
    <body>
        <p>Page de tests de l'outil de traces de dojo...</p>
    </body>
</html>
```

dojo offre la possibilité de filtrer les traces en fonction de leur criticité. Le code suivant en donne un exemple de mise en œuvre :

```
// Positionnement du niveau souhaité
dojo.log.setLevel(dojo.log.getLevel("WARNING"));
// Ecriture des traces applicatives
dojo.log.info("Trace applicative d'informations.");
dojo.log.warn("Trace applicative d'avertissement.");
```

Dans le code ci-dessus, seul le second message est affiché.

La figure 7.4 illustre l'affichage des traces applicatives de la page HTML, lesquelles sont ajoutées à la fin des pages.

```
Page de tests de l'outil de traces de Dojo...

DEBUG: 12:36:27: affichage avec dojo.log.debug()
INFO: 12:36:27: affichage avec dojo.log.info()
WARNING: 12:36:27: affichage avec dojo.log.warn()
ERROR: 12:36:27: affichage avec dojo.log.err()
CRITICAL: 12:36:27: affichage avec dojo.log.crit()
FATAL: une exception
ERROR: 12:36:27: affichage avec dojo.log.exception() Error : une exception : file:///C:/_applications/dojo-nightly/src/bootstrap1.js : line 249
```

Figure 7.4
Affichage des traces applicatives dans une page HTML

Notons la présence du module `dojo.debug.Firebug`, qui permet d'écrire les traces dans la console de l'outil de débogage de Firefox.

Mesure des performances

La bibliothèque dojo fournit un intéressant module permettant de mesurer les temps d'exécution de fonctions ou de méthodes d'objets. Ce support est localisé dans le module `dojo.profile`.

Comme le montre le code suivant, avant son utilisation et son importation (repère ❷), l'application doit réinitialiser (repère ❶) les enregistrements des temps d'exécution existants :

```
delete dojo["profile"];←❶
dojo.require("dojo.profile");←❷
(...)
```

Par la suite, afin d'enregistrer les temps d'exécution des traitements, le code suivant indique comment le support met à disposition les méthodes `start` (repère ❶) et `end` (repère ❷) afin de délimiter respectivement le début et la fin de l'enregistrement tout en lui assignant un nom logique (repère ❶) :

```
function fonction1() { (...) }
dojo.profile.start("fonction1");←❶
fonction1();
dojo.profile.end("fonction1");←❷
```

Notons que l'objet `dojo.profile` a été automatiquement instancié lors de l'inclusion du module.

Dans le code suivant, l'objet `dojo.profile` met à disposition la méthode `dump` (repère ❶) afin d'afficher les résultats dans une page Web :

```
(...)
dojo.profile.start("fonction1");
fonction1();
dojo.profile.end("fonction1");
(...)
dojo.profile.dump(true);←❶
```

Notons que la méthode `dump` prend un paramètre permettant de spécifier si le tableau de résultat doit être automatiquement inséré dans la page Web (valeur `true`). Si la valeur du paramètre est `false`, le tableau retourné par la méthode doit être ajouté manuellement à la page.

Le code suivant fournit un exemple de mise en œuvre de ce module par l'intermédiaire d'un exemple complet :

```
// Réinitialisation des enregistrements
delete dojo["profile"];
// Importation du module
dojo.require("dojo.profile");

function fonction1() {
    var nombre = 0;
    for(var cpt=0;cpt<100;cpt++) {
        nombre = nombre % 5 + cpt;
    }
}

function fonction2() {
    var chaine = "";
    var motif = "element,";
    for(var cpt=0;cpt<1000;cpt++) {
        chaine = chaine + motif;
    }
}

// Mesure du temps d'exécution de la fonction fonction1
dojo.profile.start("fonction1");
fonction1();
dojo.profile.end("fonction1");
// Mesure du temps d'exécution de la fonction fonction1
dojo.profile.start("fonction2");
fonction2();
dojo.profile.end("fonction2");
// Affichage des temps d'exécution mesurés
dojo.profile.dump(true);
```

La figure 7.5 illustre l'affichage du tableau récapitulant les temps d'exécution mesurés.

Page de calcul de performances de Dojo...

Identifier	Calls	Total	Avg
fonction2	1	47	47
fonction1	1	0	0

Figure 7.5
Tableau récapitulatif des temps d'exécution

Conclusion

La bibliothèque dojo fournit d'intéressants mécanismes afin de mettre en œuvre des applications JavaScript évoluées. Intégrant un mécanisme de chargement à la demande des modules souhaités, elle permet d'alléger la quantité de code JavaScript chargé dans le navigateur.

La bibliothèque est structurée en modules adressant une fonctionnalité particulière. Les différentes fonctionnalités mises à disposition correspondent à des fonctions facilitant la mise en œuvre de JavaScript, les types de base et les collections ainsi que les technologies DOM et Ajax.

La bibliothèque dojo fournit un intéressant support afin de développer des applications Web. Ce support intègre des modules contenant des fonctions utilitaires relatives aux langages HTML et CSS. Il possède en outre des mécanismes permettant de gérer les événements et de structurer les composants graphiques.

La bibliothèque dojo contient enfin des implémentations prédéfinies de divers composants graphiques, tels que les arbres, les tableaux, les menus et les fenêtres flottantes.

Nous détaillons au chapitre 11 le support de cette bibliothèque pour le développement d'applications Web riches.