Dojo Toolkit

Créer des applications AJAX/RIA en JavaScript

Auteur	Thomas Corbière
Site Web	www.itsi-formation.com
Dernière modification	20/02/2012 (révision 3)
Dojo Toolkit	1.6

Objectifs

Dojo Toolkit est un framework JavaScript libre facilitant le développement Web 2.0. Ce stage vous permettra d'exploiter les fonctionnalités les plus utiles de Dojo Toolkit pour créer des applications Web riches (RIA). Vous apprendrez à utiliser AJAX, à concevoir des interfaces et des formulaires évolués ainsi qu'à intégrer des Widgets préfabriqués ou vos propres composants.

Participants

Ce cours s'adresse au développeur Web réalisant des interfaces utilisateurs en HTML/CSS et souhaitant les dynamiser avec JavaScript.

Prérequis

De bonnes connaissances en HTML, CSS et JavaScript ; connaissances de base en programmation orientée objet.

Travaux pratiques

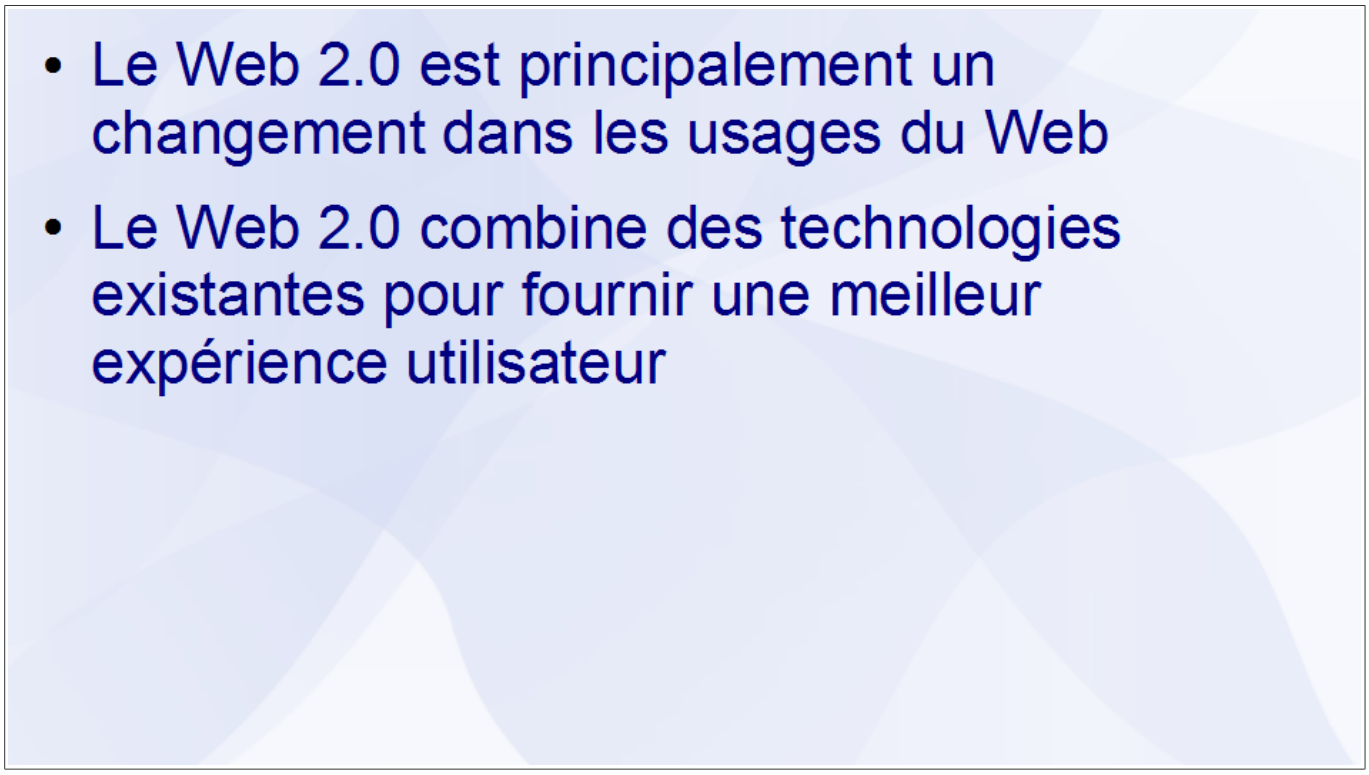
Les travaux pratiques ont pour objet la réalisation du prototype d'une application. L'interface graphique de l'application sera réalisée en exploitant les Widgets fournis avec Dojo Toolkit puis progressivement enrichie pour prendre en compte les interactions avec l'utilisateur.

Liste des chapitres

Chapitre 1 – Introduction à Dojo Toolkit

Sommaire

Concepts et technologies du Web 2.0



Dans le Web 2.0, l'utilisateur occupe la place centrale. Le Web devient ainsi un espace de socialisation, de collaboration et de partage où le contenu est créé directement par les utilisateurs sans nécessité de compétences techniques.

Le Web 2.0 affirme également le Web comme plate-forme de développement universelle avec l'émergence d'applications Internet riches (RIA) capables de supplanter les applications de bureau traditionnelles.

D'un point de vue technologique, le Web 2.0 utilise des technologies pré-datant son invention de façon à rendre les sites et applications Web plus conviviaux et intuitifs :

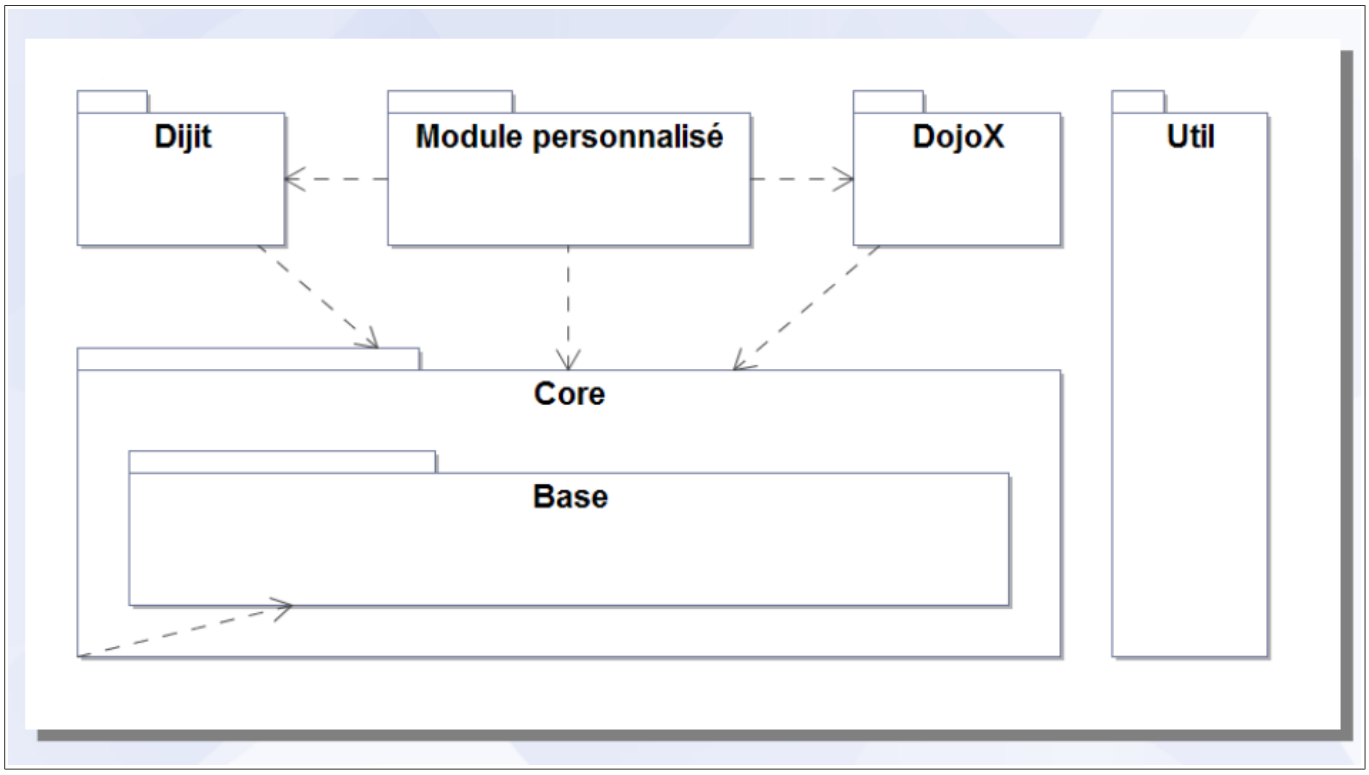
- HTML et CSS représentent les interfaces graphiques
- JavaScript dynamise les interfaces avec des animations et le support du glisser-déposer
- l'objet `XMLHttpRequest` permet de mettre à jour les données d'une page sans avoir à la recharger complètement

On peut toutefois noter que le Web 2.0 fait appel de façon intensive au Javascript qui n'était jusqu'alors qu'utilisé de façon limité. Le recours à un framework JavaScript devient donc indispensable pour simplifier le développement et assurer la compatibilité avec l'ensemble des navigateurs.

Présentation de Dojo Toolkit

Dojo Toolkit est un framework JavaScript librement utilisable dans toute application, qu'elle soit commerciale ou non. Il n'existe aucune obligation de rendre public le code source de ses propres modules. Il est développé sous l'égide de la Dojo Foundation dont les contributeurs comptent parmi les grands noms du secteur (Google, IBM, Orange, Zend Technologies…).

L'architecture



Dojo Toolkit dispose d'un très grand nombres de fonctionnalités organisées sous forme de modules.

Le module Dojo Base est le noyau compact et hautement optimisé de Dojo Toolkit. Il sert de fondation à l'ensemble des autres modules. Il dispose des fonctionnalités les plus souvent employées : la manipulation du DOM, les requêtes AJAX, les animations, la gestion des événements…

Le module Dojo Core comprend des fonctionnalités moins universelles mais complémentaires : animations avancées, gestion du glisser-déposer, gestion de l'historique…

Le module Dijit est la bibliothèque de Widgets accessibles et internationalisés de Dojo Toolkit. Il comprend les Widgets de mise en page, les Widgets applicatifs et les Widgets de formulaire.

Le module DojoX comprend les modules d'extensions et expérimentaux. Ses fonctionnalités les plus mises en avant sont la création de graphiques, la bibliothèque de Widgets pour applications mobiles, le DataGrid…

Le module Util comprend des outils complémentaire à Dojo Toolkit qui ne s'utilisent pas directement dans une application : un générateur de documentation technique, un framework de tests, un outil d'optimisation du code JavaScript…

Les fonctionnalités des modules autres que Dojo Base ne sont pas directement accessibles et doivent être importées avant de pouvoir être utilisées.

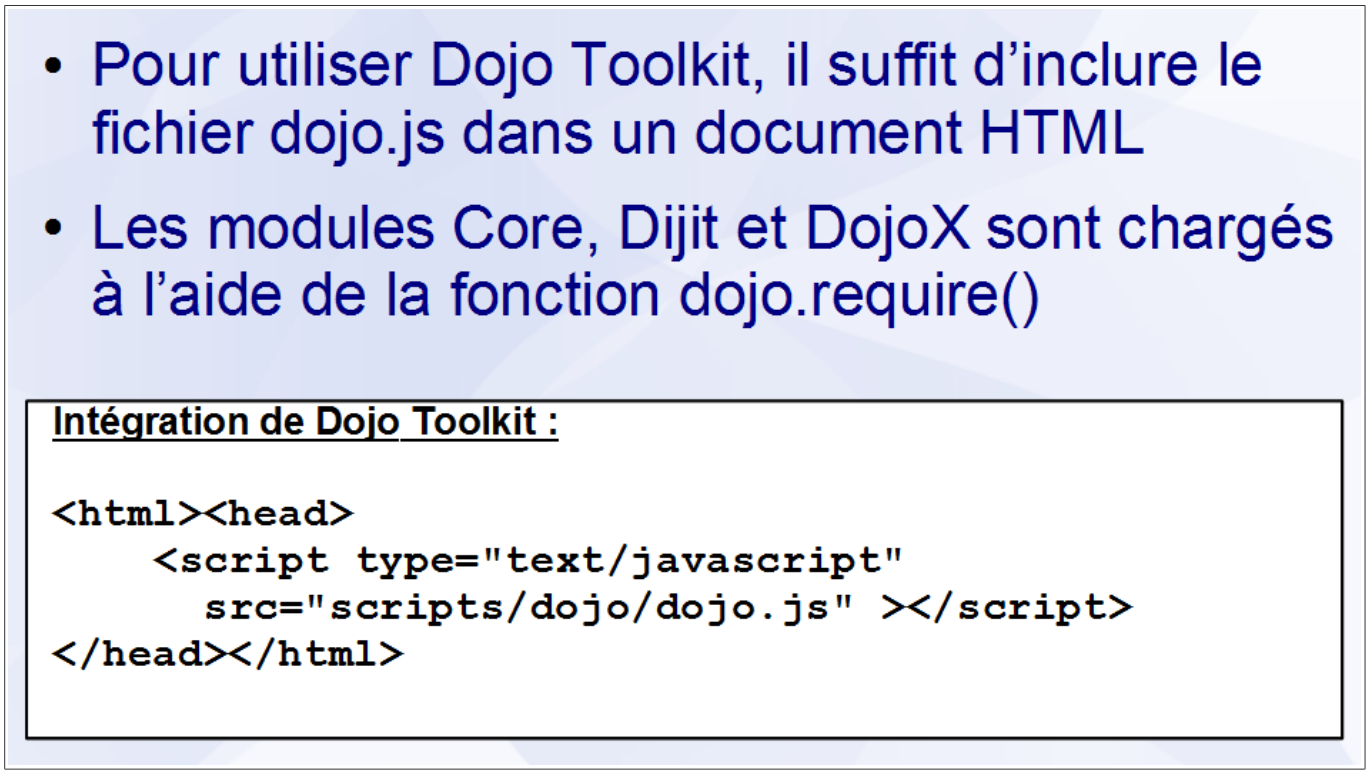
Les différentes éditions

Dojo Toolkit est disponible en trois éditions :

- Dojo Base – comprend uniquement le module Dojo Base
- Dojo Toolkit Release – comprend les modules Dojo Base, Dojo Core, Dijit et DojoX optimisés pour la mise en production
- Dojo Toolkit SDK – comprend les modules Dojo Base, Dojo Core, Dijit et DojoX non compressés ainsi que les outils complémentaires

Lors du développement, il est préférable d'utiliser l'édition Dojo Toolkit SDK pour faciliter le débogage de l'application. Lors de la mise en production, pour obtenir de meilleurs performances, il faut utiliser Dojo Toolkit Release ou créer une édition personnalisée.

Intégration à une application



L'intégration de Dojo Toolkit peut se faire :

- sans installation en utilisant un Content Delivery Network (CDN)
  - `http://ajax.googleapis.com/ajax/libs/dojo/1.6.1/dojo/dojo.xd.js`
  - `http://yandex.st/dojo/1.6.1/dojo/dojo.xd.js`
- avec installation de Dojo Toolkit sur le serveur hébergeant l'application

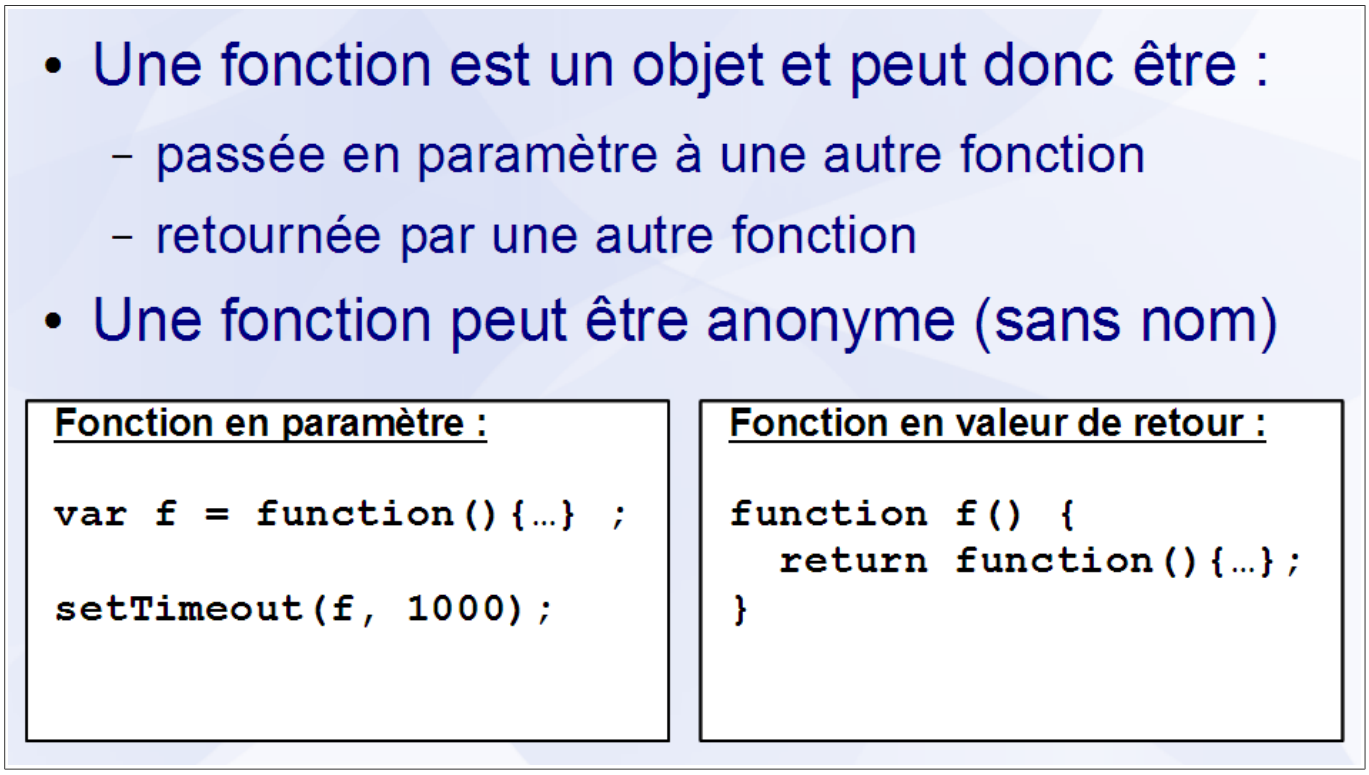
Chapitre 2 – Les concepts avancés du JavaScript

Sommaire

Introduction

- Le langage JavaScript est complexe et difficile à maîtriser
- Utiliser efficacement Dojo Toolkit nécessite une bonne compréhension des concepts avancés du JavaScript
- Dojo Toolkit dispose de fonctions simplifiant l'utilisation de ces concepts avancés

Les fonctions

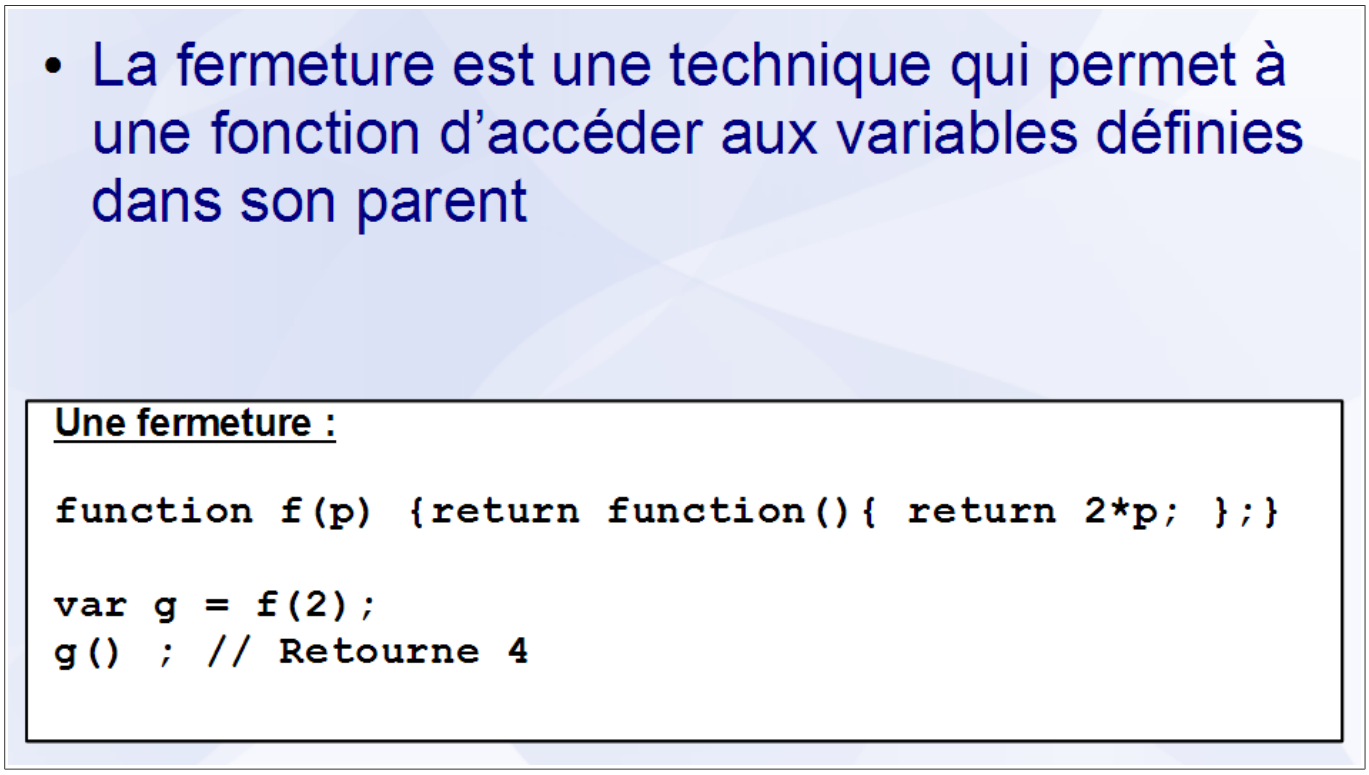


En JavaScript, une fonction est une instance de la classe `Function`. Elle se différencie des autres objets car un bloc d'instructions lui est rattaché. En dehors de cette particularité, une fonction peut s'utiliser comme n'importe quel autre objet :

- affectation à une variable
- passage en paramètre à une fonction
- valeur de retour d'une fonction

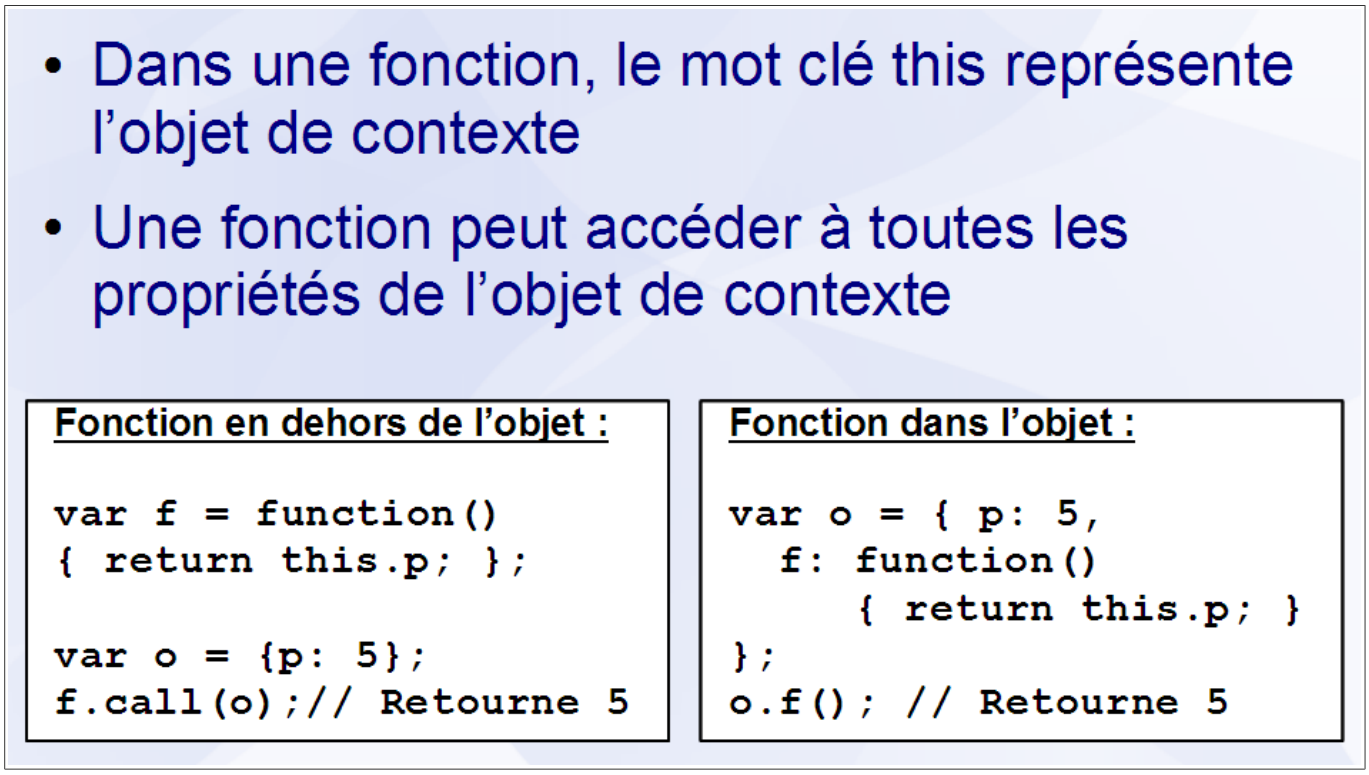
De plus, une fonction peut être anonyme ou, plus exactement, quelle que soit la façon dont elle est déclarée, elle est affectée à une variable qui elle possède un nom. C'est à partir du nom de cette variable que la fonction est invoquée.

Les fermetures



Lorsqu'une fonction est invoquée, un objet d'invocation est créé pour contenir ses paramètres et ses variables locales. La création d'une fonction dans une autre fonction lui associe l'objet d'invocation de son parent ce qui lui donne accès aux variables de ce dernier, même après la fin de son exécution. Ce mécanisme s'appelle la fermeture (*closure* en anglais).

L'objet de contexte

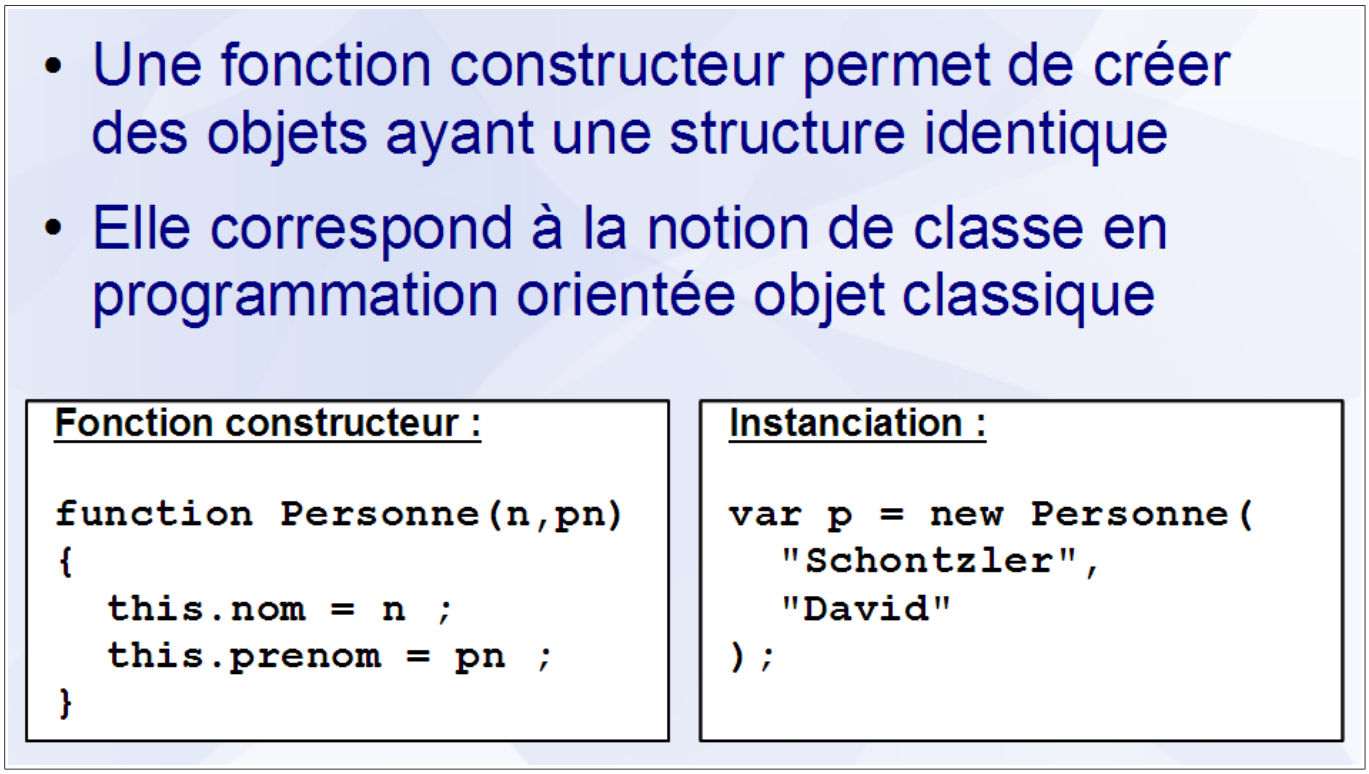


Toute fonction s'exécute avec un objet de contexte représenté, dans le corps de celle-ci, par le mot clé `this`. Par défaut, une fonction s'exécute dans le contexte de l'objet global qui, dans un navigateur, correspond à l'objet `window`.

Les méthodes `call()` et `apply()` d'une fonction permettent d'exécuter la fonction dans le contexte d'un objet passé en paramètre. La méthode `call()` accepte en paramètre un objet de contexte suivi des mêmes paramètres que la fonction invoquée tandis que la méthode `apply()` accepte en paramètre un objet de contexte suivi d'un tableau de paramètres à passer à la fonction invoquée.

Invoquée avec la notation pointée, une fonction utilise automatiquement l'objet qui précède son appelle comme objet de contexte.

Les fonctions constructeurs

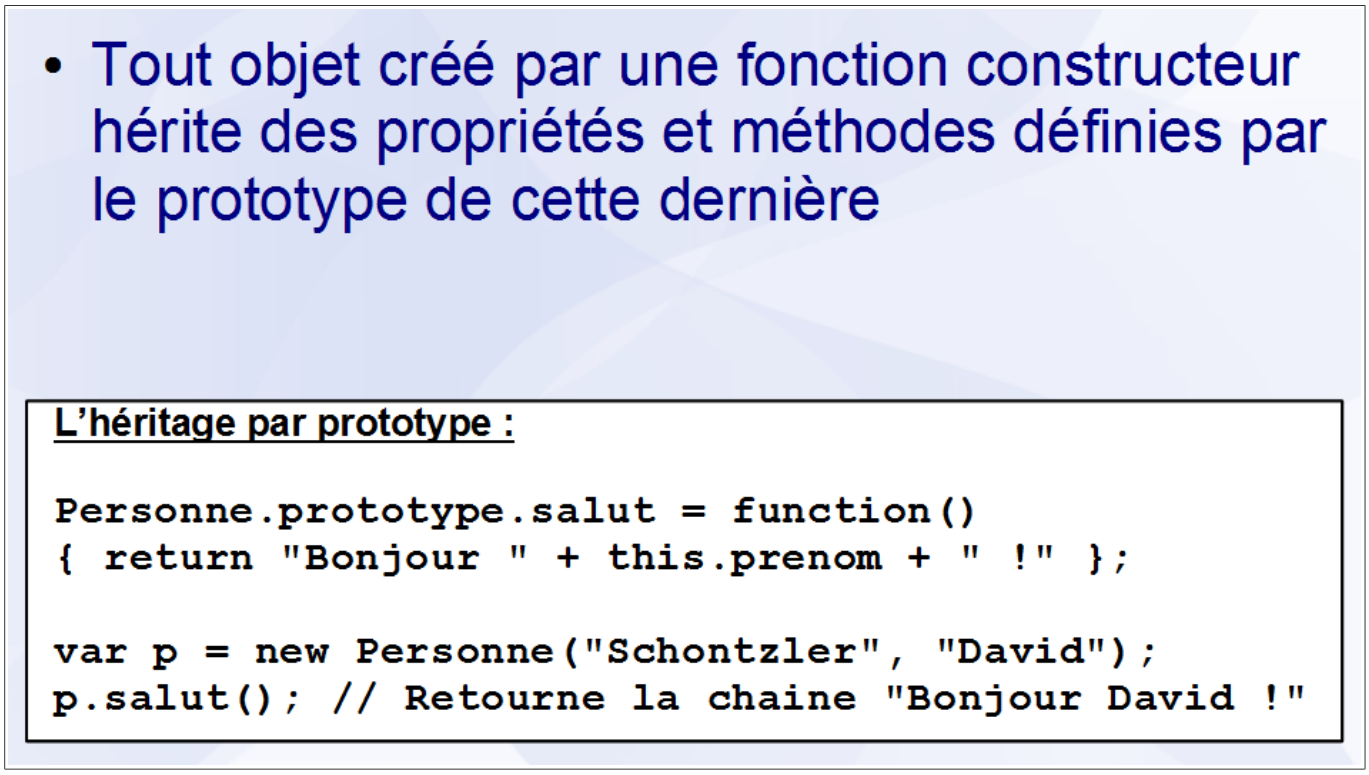


Il n'existe en JavaScript aucune différence entre une fonction standard et une fonction constructeur. Cette dernière est simplement invoquée avec l'opérateur `new`, ce qui a pour effet de créer un nouvel objet dont elle initialise les propriétés. Une même fonction peut être utiliser comme une fonction standard ou comme une fonction constructeur.

Par exemple :

```
function Personne()
{
    if(this instanceof Personne)
    {
        /* Invocation en tant que constructeur */
    }
    else
    {
        /* Invocation en tant que fonction */
    }
}
```

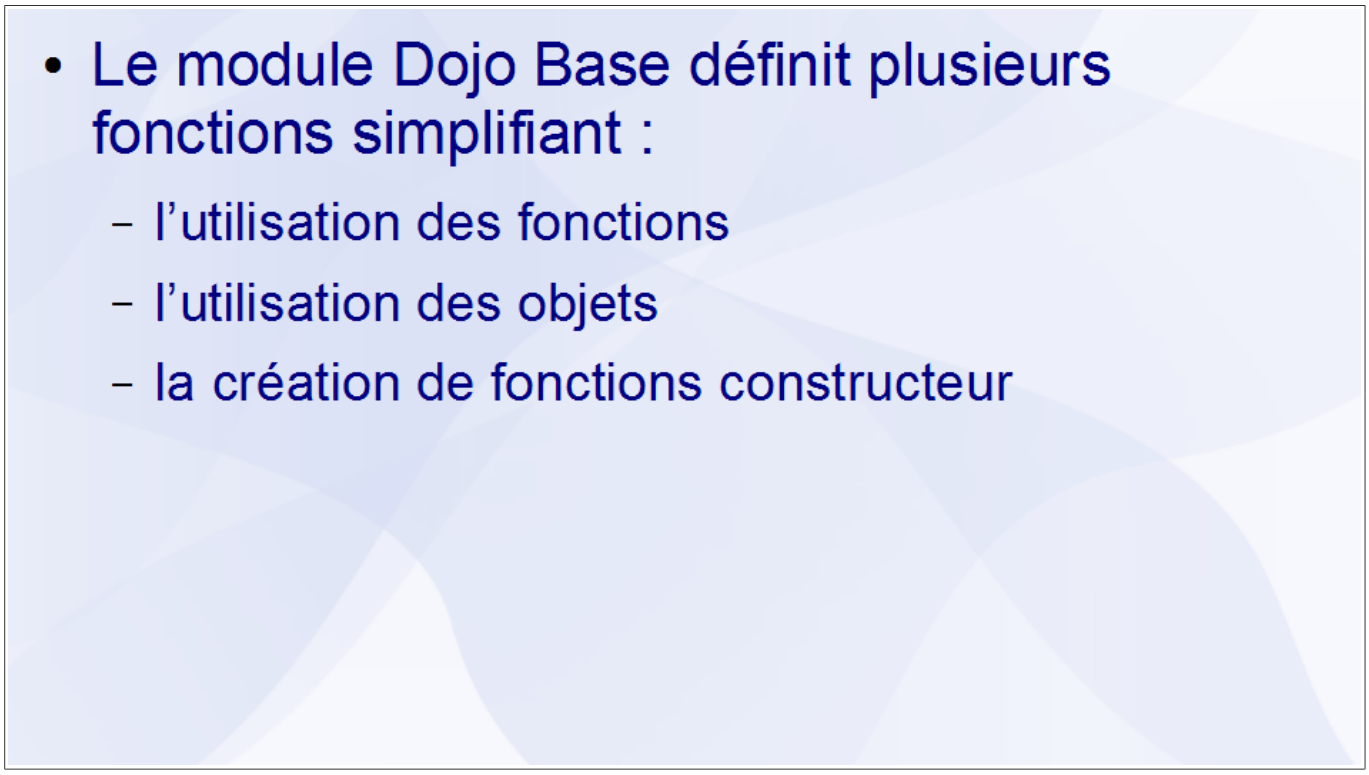
L'héritage par prototype



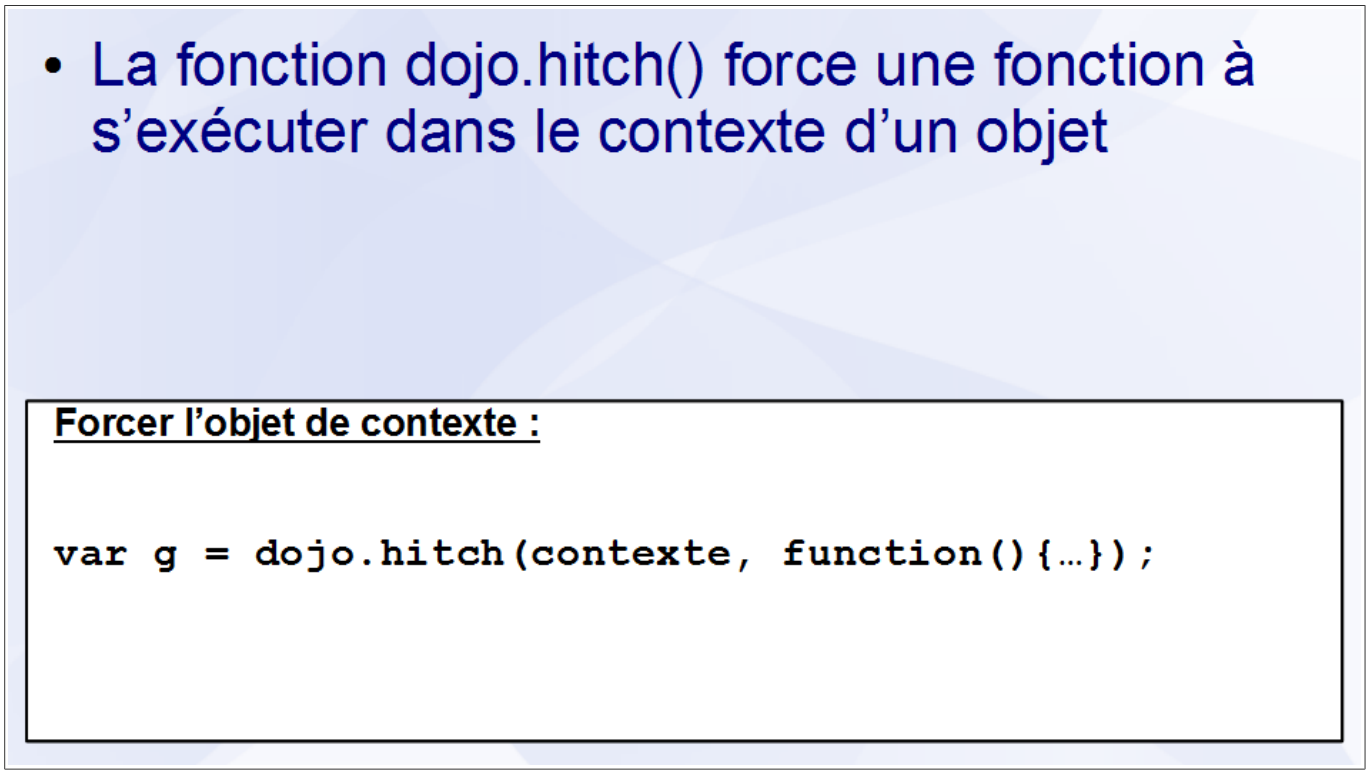
Le langage JavaScript est un langage orientée objet basé sur les prototypes. Chaque objet hérite automatiquement des propriétés et méthodes définies par le prototype de la fonction constructeur qui l'a créé. Un objet ne pouvant être construit que par une seule fonction constructeur, le JavaScript ne supporte que l'héritage simple où un objet n'a qu'un seul parent.

L'un des avantages de ce mode d'héritage est qu'il est possible de modifier le prototype de la fonction constructeur à tout moment, les changements étant visibles par toutes les instances de la classe, y compris celles construites avant la modification du prototype.

Les fonctions utilitaires pour la POO



Forcer l'objet de contexte

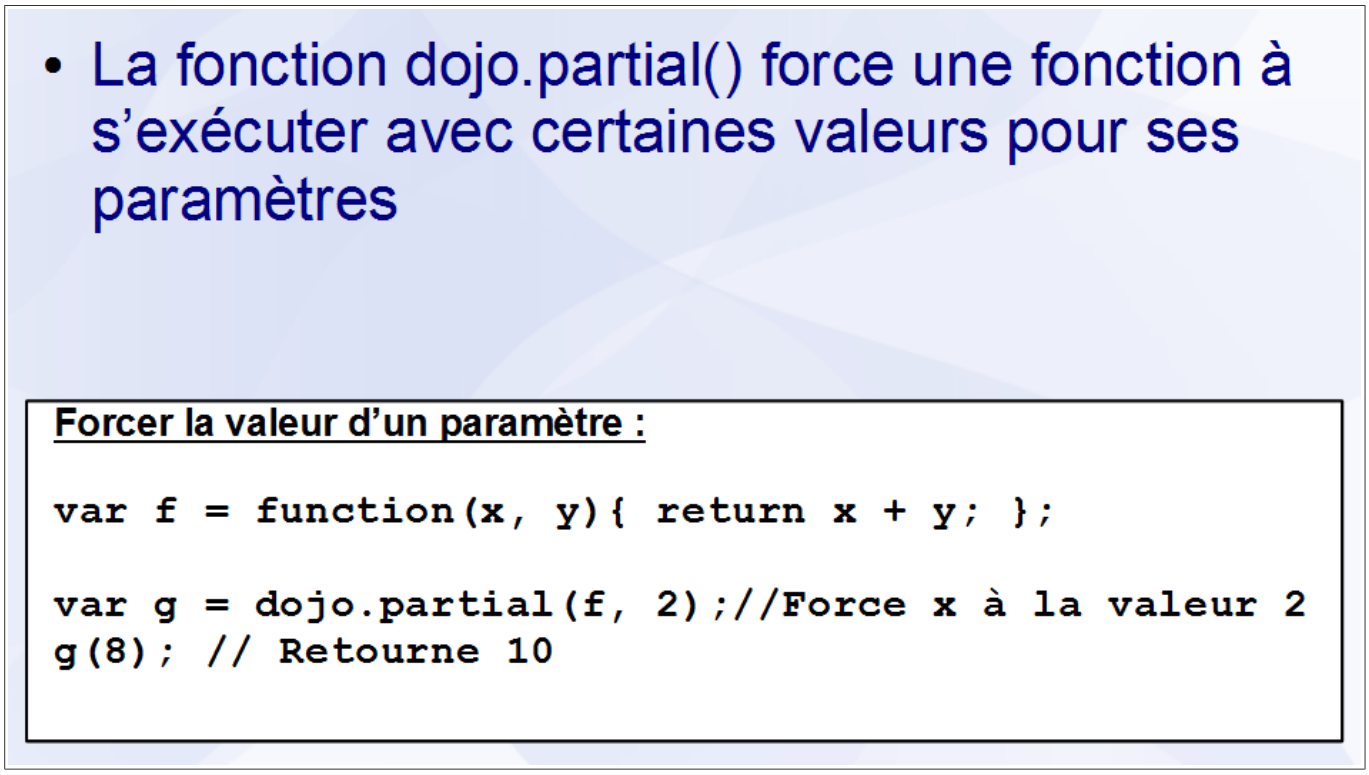


La fonction `dojo.hitch()` force une fonction à s'exécuter dans le contexte d'un objet. Elle accepte les paramètre les suivants :

Paramètre	Type	Description
`scope`	`Object`	Objet de contexte à utiliser.
`method`	`Function` `String`	Fonction à associer à l'objet de contexte ou nom d'une méthode de l'objet de contexte.
`param`	`-`	(optionnel, multiple) Valeur à utiliser pour les n premiers paramètres.

La fonction `dojo.hitch()` retourne une fonction s'exécutant dans le contexte d'un objet et forçant la valeur de certains paramètres de la fonction d'origine.

Forcer la valeur d'un paramètre

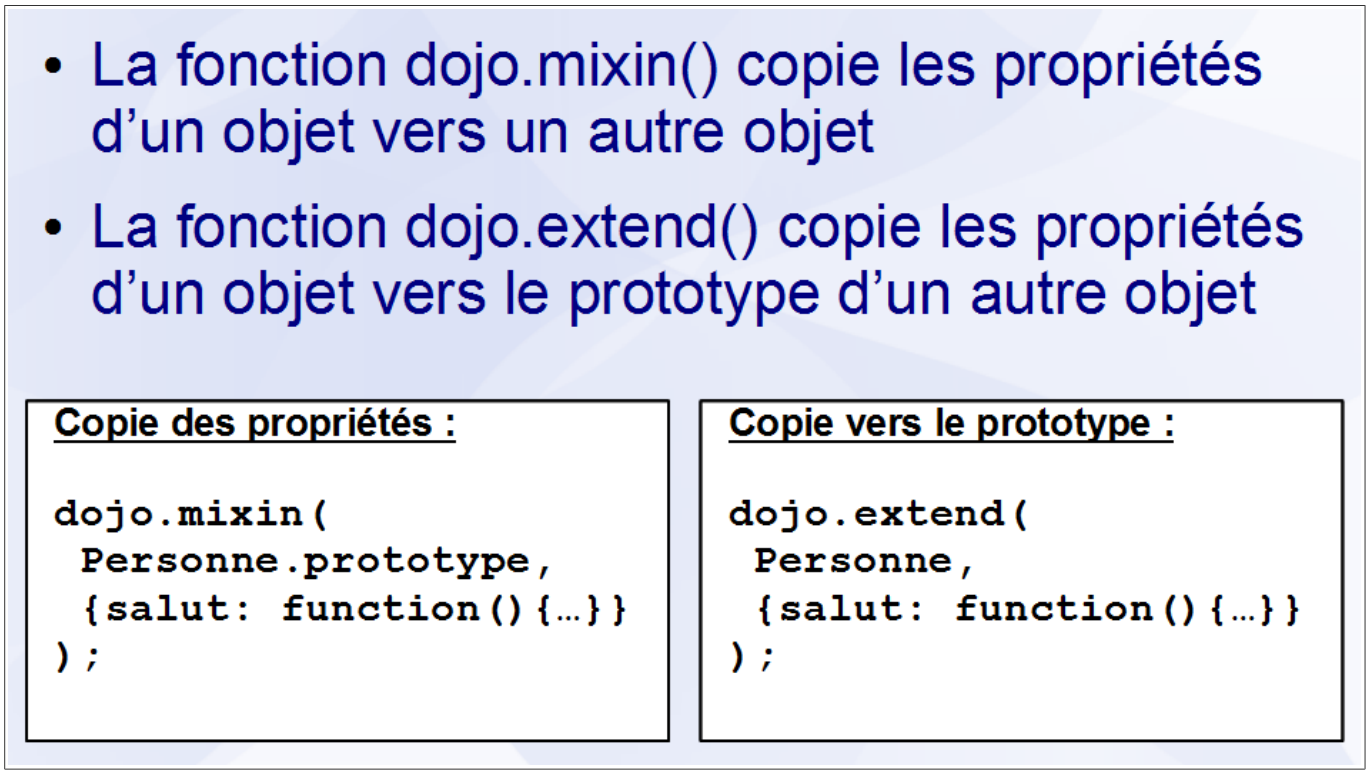


La fonction `dojo.partial()` force une fonction à s'exécuter avec certaines valeurs pour ses paramètres. Elle accepte les paramètre les suivants :

Paramètre	Type	Description
`method`	`Function`	Fonction à utiliser.
`param`	-	(optionnel, multiple) Valeur à utiliser pour les n premiers paramètres.

La fonction `dojo.partial()` retourne une fonction forçant la valeur de certains paramètres de la fonction d'origine.

Copier les propriétés d'un objet



La fonction `dojo.mixin()` copie les propriétés d'un objet vers un autre objet. Elle accepte les paramètre les suivants :

Paramètre	Type	Description
`obj`	`Object`	Objet auquel les propriétés doivent être ajoutées.
`props`	`-`	(optionnel, multiple) Objet dont les propriétés doivent être copiées.

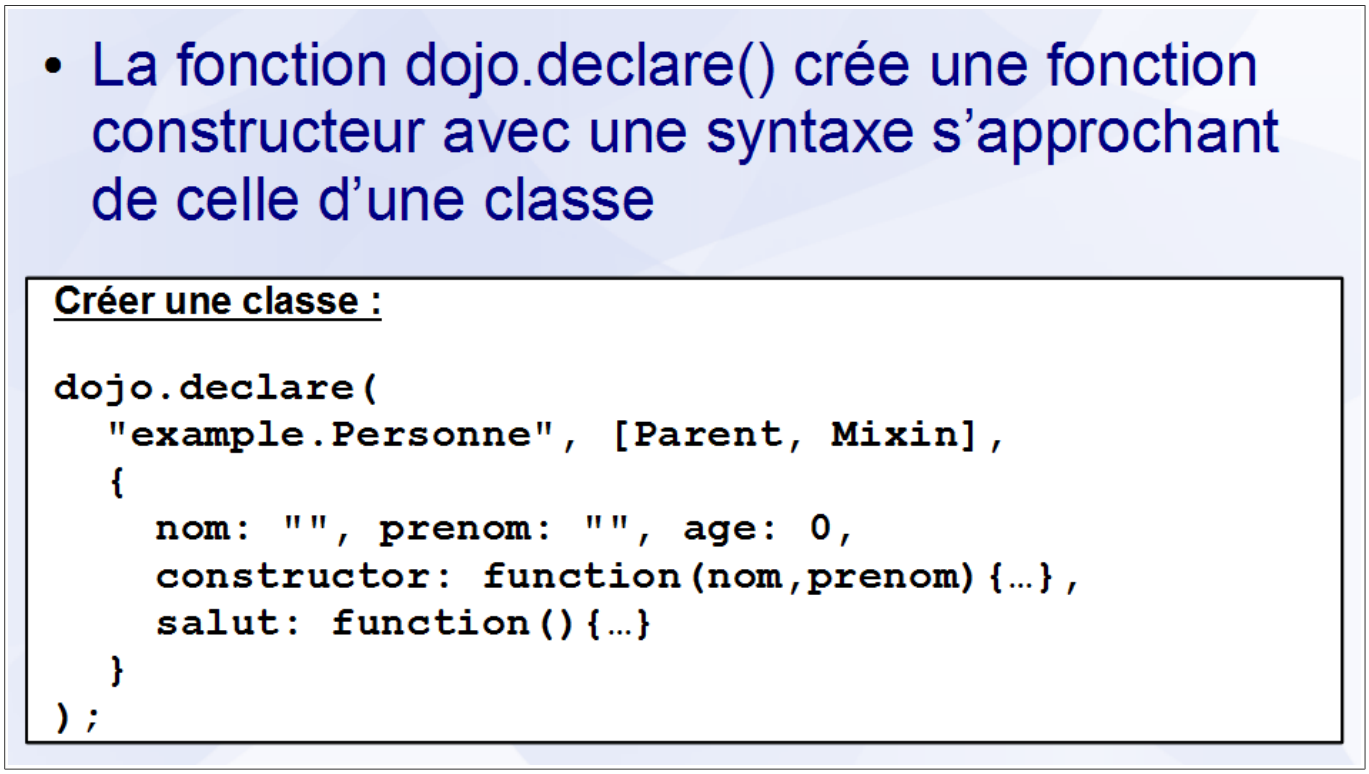
La fonction `dojo.mixin()` retourne l'objet auquel les propriétés ont été ajoutées.

La fonction `dojo.extend()` copie les propriétés d'un objet vers le prototype d'un autre objet. Elle accepte les paramètre les suivants :

Paramètre	Type	Description
`obj`	`Object`	Objet dont le prototype doit être modifié.
`props`	-	(optionnel, multiple) Objet dont les propriétés doivent être copiées.

La fonction `dojo.extend()` retourne l'objet dont le prototype a été modifié.

Créer une classe



La fonction `dojo.declare()` crée une fonction constructeur avec une syntaxe s'approchant de celle d'une classe. Elle accepte les paramètre les suivants :

Paramètre	Type	Description
`className`	`String`	(optionnel) Nom de la classe.
`superclass`	`Array`	Liste de classes dont on hérite : • La première classe est utilisée comme classe parent • Les autres classes sont utilisées en tant que mixins
`props`	`Object`	Liste des propriétés et des méthodes de la classe. quatre de ces propriétés ont une signification particulière : • `preamble` – méthode exécutée avant le constructeur • `constructor` – méthode constructeur • `postscript` – méthode exécutée après le constructeur • `inherited` – méthode permettant d'accéder aux méthodes de la classe parent

La fonction `dojo.declare()` retourne la fonction constructeur créée.

Conclusion

- La fonction dojo.hitch() permet de forcer une fonction à s'exécuter dans un contexte donné
- La fonction dojo.partial() permet de fixer la valeur de certains paramètres
- Les fonctions dojo.mixin() et dojo.extend() permettent de copier les propriétés d'un objet
- La fonction dojo.declare() permet de simplifier la création des fonctions constructeurs

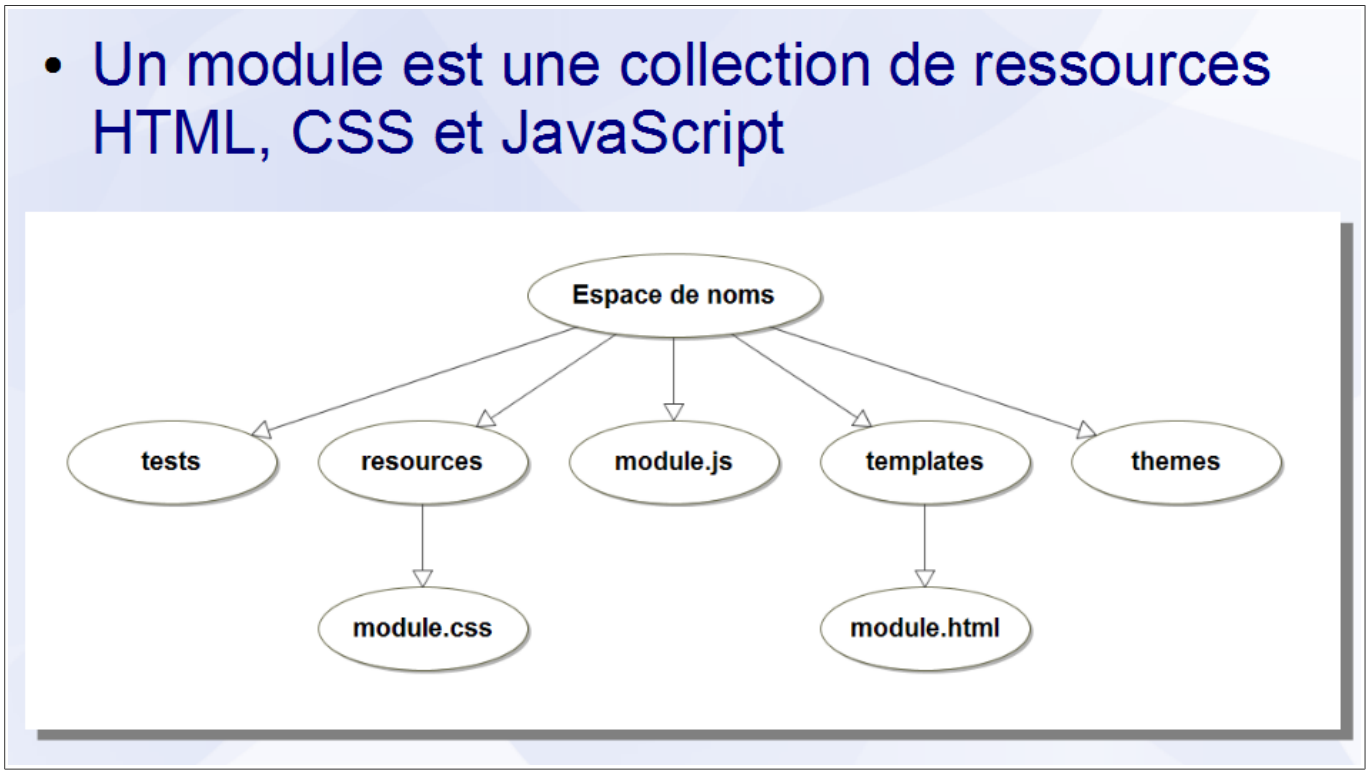
Chapitre 3 – La gestion des modules

Sommaire

Introduction

- Les sites et applications Web 2.0 nécessitent un grand nombre de scripts JavaScript
- Ces scripts sont réunis en modules pour en faciliter le développement et la maintenance
- Dojo Toolkit dispose d'un système de gestion de modules permettant de définir et d'importer des modules

Structure d'un module

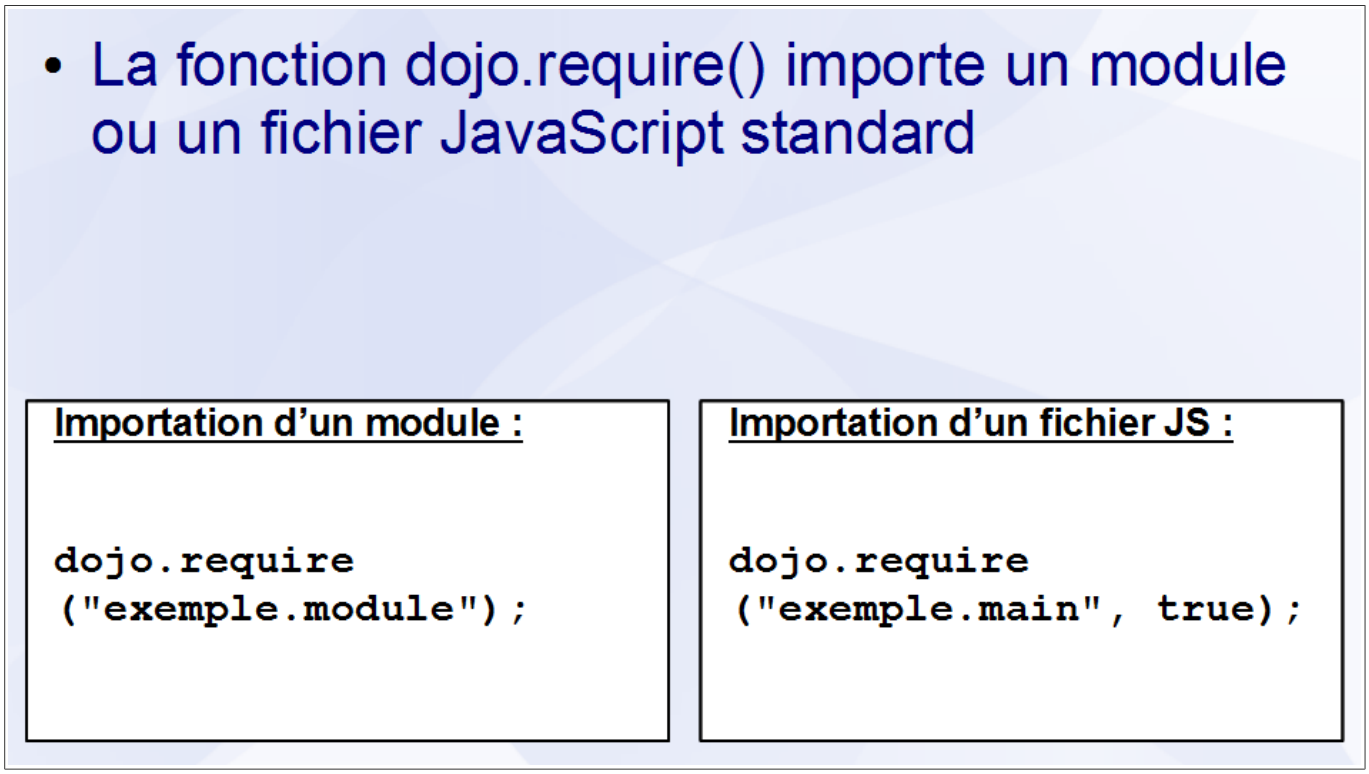


Sur le disque, un module est représenté par un répertoire contenant des fichiers HTML, CSS et JavaScript organisés de la façon suivante :

- `module.js` – fichier JavaScript correspondant au module
- `module.html` du sous-répertoire `templates` – représentation graphique du module s'il s'agit d'un Widget
- sous-répertoire `themes` – feuilles de styles CSS et images utilisées pour la mise en page d'une famille de Widgets
- sous-répertoire `resources` – autres fichiers nécessaires au bon fonctionnement du module
- sous-répertoire `tests` – suite de tests unitaires vérifiant le bon fonctionnement du module

Ce répertoire sert également d'espace de noms au module et fournit ainsi un nom unique à chaque ressource qui le constitue. Dojo Toolkit emploie la notation pointée pour désigner l'une de ces ressources, chaque étape de la notation correspondant à un répertoire exceptée la dernière qui correspond au nom du fichier JavaScript. Par exemple, `dojo.fx.easing` représente le module `easing` dans l'espace de noms `dojo.fx`, le fichier correspondant étant `dojo/fx/easing.js`.

Importation d'un module



La fonction `dojo.require()` importe un module ou un fichier JavaScript standard. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`moduleName`	`String`	Nom du module en notation pointée.
`omitModuleCheck`	`Boolean`	(optionnel) Si `true`, ne vérifie pas que le module crée un objet du même nom.

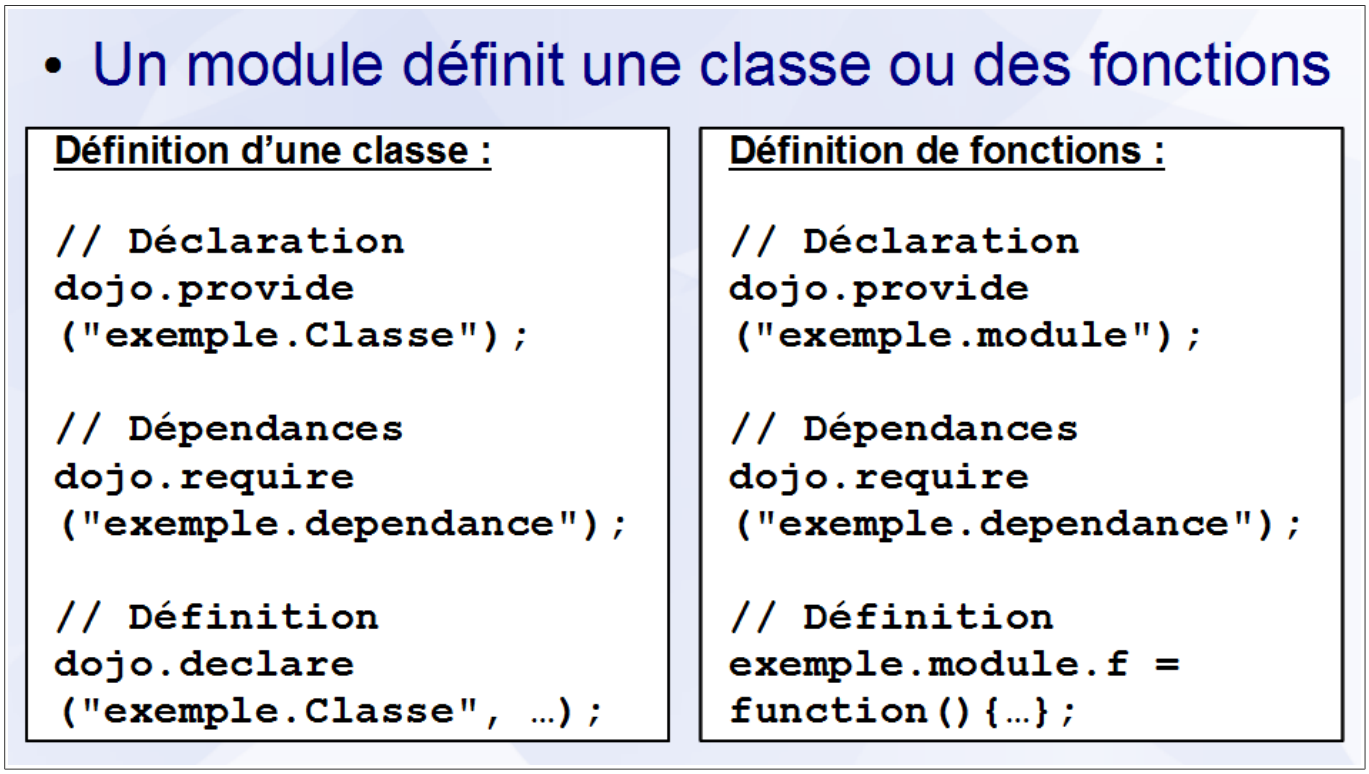
Avec une intégration locale de Dojo Toolkit, le chargeur de modules opère de façon synchrone, c'est à dire que la fonction `dojo.require()` ne retourne qu'après le chargement du module qui peut donc être utilisé immédiatement.

Avec une intégration de Dojo Toolkit via un CDN, le chargeur de modules opère de façon asynchrone, c'est à dire que la fonction `dojo.require()` retourne immédiatement sans attendre que le module ne soit chargé. Il est donc indispensable d'exécuter le code dépendant de ce module avec la fonction `dojo.ready()`.

```
dojo.require("exemple.module") ;
dojo.ready(function () { /* Code dépendant du module */ }) ;
```

Dans les deux cas, si le module a déjà été chargé, la fonction retourne immédiatement.

Définition d'un module



Le fichier JavaScript définissant un module contient trois sections :

- la première section utilise la fonction `dojo.provide()` pour enregistrer le module auprès du système de gestion de modules
- la deuxième section importe les modules nécessaires au fonctionnement du module en cours de définition
- la troisième section contient la définition proprement dite du module qui peut être une classe ou un ensemble de fonctions

Configuration d'un module

Avec le chargement de Dojo Toolkit :

```
<script type="text/javascript"
  src="scripts/dojo/dojo.js"
  data-dojo-config="locale: 'fr', exemple: {…},
    modulePaths: {exemple: '../../exemple'}"
></script>
```

Avant le chargement de Dojo Toolkit :

```
<script type="text/javascript" >
  var dojoConfig = {locale: "fr", exemple: {…},
    modulePaths: {exemple: "../../exemple"}};
</script>
```

La variable `dojoConfig` définit la configuration de Dojo Toolkit, ainsi que l'emplacement et la configuration des modules additionnels. La liste de ses propriétés est disponible en annexe.

La propriété `dojo.config` permet d'accéder aux propriétés définit par la variable `dojoConfig`.

```
if(dojo.config.propriete) {
    /* Code dépendant de la propriété */
}
```

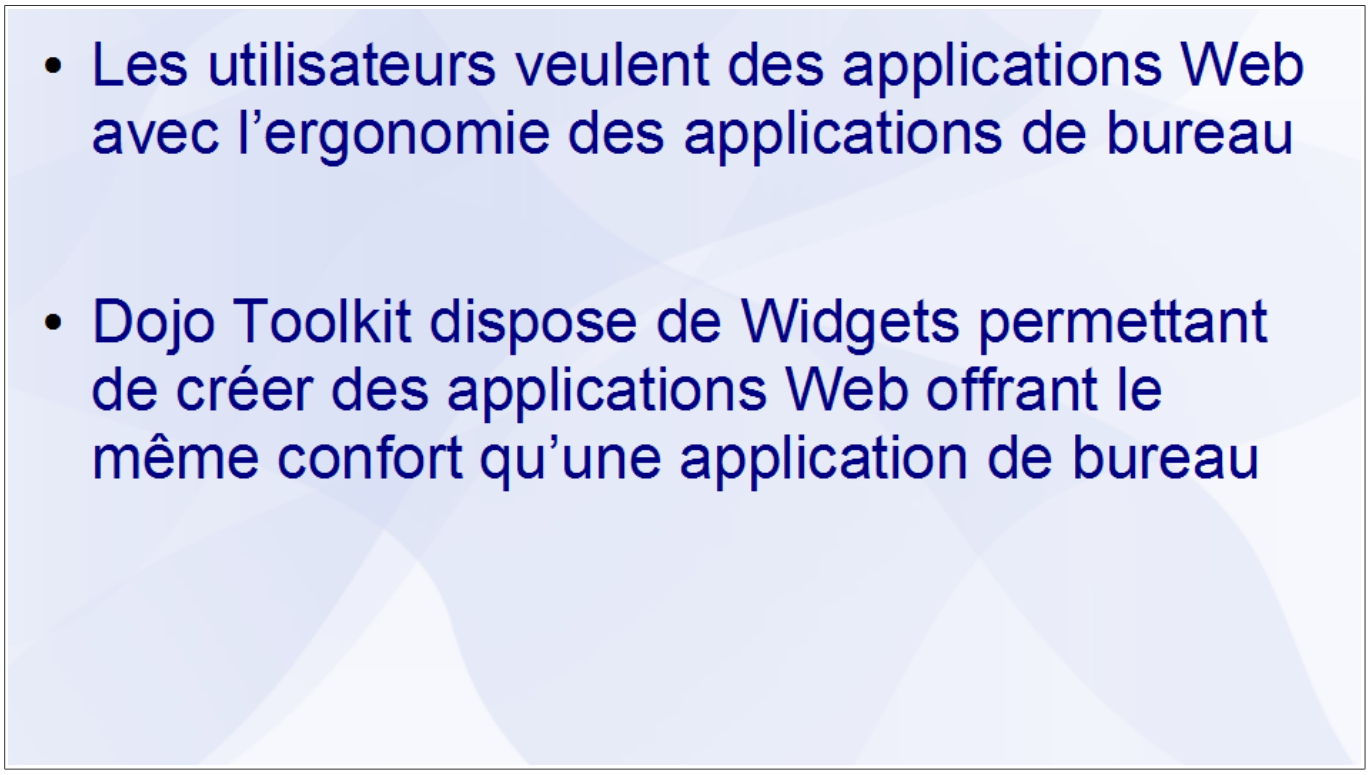
Conclusion

- Un module est une collection de ressources
- La fonction dojo.require() importe les modules
- La définition d'un module comprend :
  - une déclaration dojo.provide()
  - une liste de dépendances
  - la définition d'une classe ou de fonctions
- La variable dojoConfig définit la configuration de Dojo Toolkit et des modules additionnels

Chapitre 4 – Les Widgets

Sommaire

Introduction



Les Widgets sont des composants de l'interface graphique d'une application.

Le module Dijit est la bibliothèque de Widgets internationalisés et accessibles de Dojo Toolkit. Il couvre les Widgets de formulaire, les Widgets de mise en page et les Widgets applicatifs.

Le module DojoX définit des Widgets expérimentaux ou d'une utilité moins universelle. Il couvre la création de graphique, les Widgets pour appareils mobiles et les Widgets *DataGrid*.

Utilisation des thèmes et des Widgets

Utilisation du thème Claro :

```
<head>
  <link rel="stylesheet"
    href="scripts/dijit/themes/claro/claro.css" />
</head>
<body class="claro" ></body>
```

En JavaScript :

```
var b = new
  dijit.form.Button(
    {label: "Créer"},
    element
  );
```

Avec la syntaxe déclarative :

```
<div data-dojo-type =
  "dijit.form.Textarea"
  data-dojo-props="…"
></div>
```

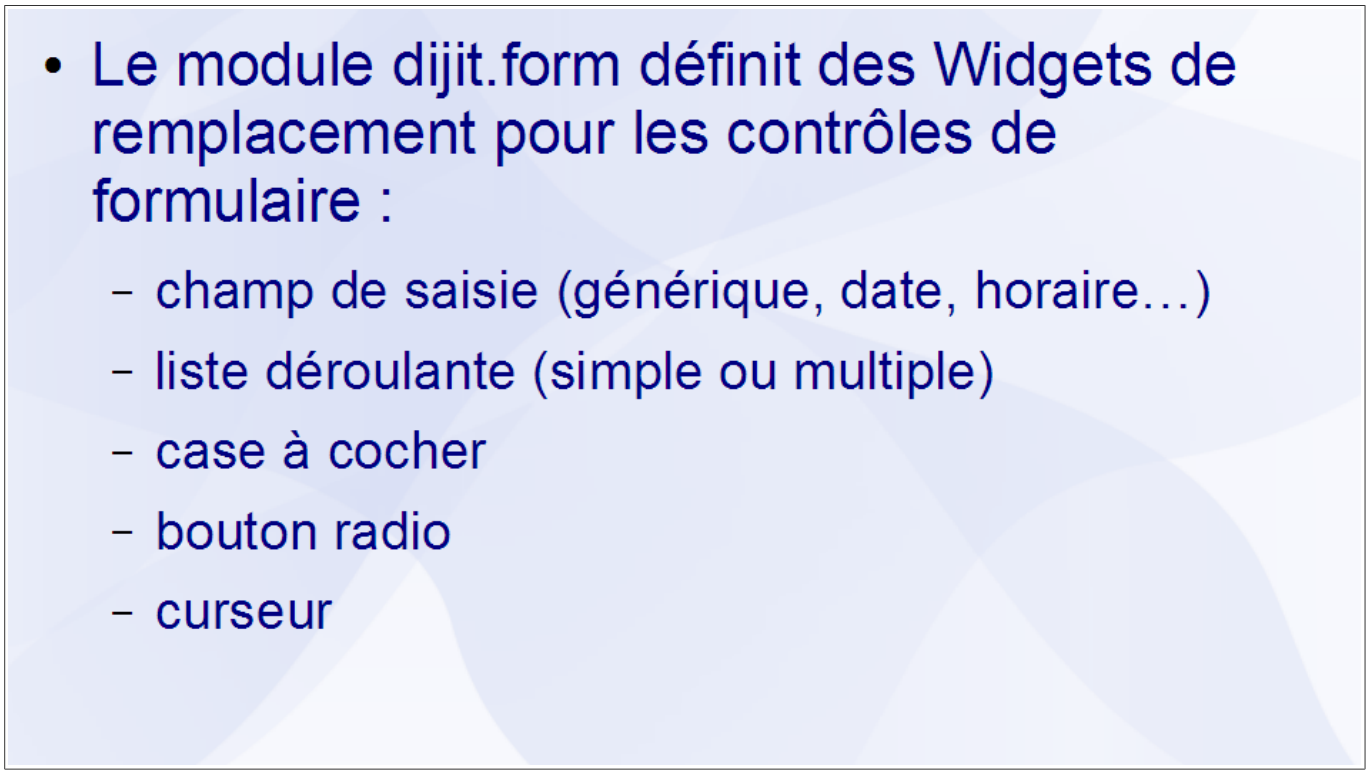
Avant d'utiliser un Widget, il est indispensable de charger un thème. Ce dernier est une collection de feuilles de styles CSS et d'images définissant la mise en page d'une famille de Widgets. L'utilisation d'un thème se fait en deux étapes :

1. Charger la feuille de styles principale du thème
2. Ajouter le nom du thème comme classe de l'élément `body`

Les thèmes suivants sont disponibles : Claro, Tundra, Soria et Nihilo.

Une fois le thème chargé, il est possible de créer des Widgets en utilisant soit la syntaxe JavaScript, soit la syntaxe déclarative. Pour utiliser cette dernière, la variable `dojoConfig` doit avoir sa propriété `parseOnLoad` à `true`. L'attribut `data-dojo-type` permet à l'analyseur de connaître la classe du Widget à créer tandis que l'attribut `data-dojo-props` spécifie les options à passer au constructeur. Celui-ci est standardisé pour tous les Widgets et accepte en paramètres un tableau associatif d'options et l'identifiant de l'élément qu'ils remplacent dans le document.

Les Widgets de formulaire

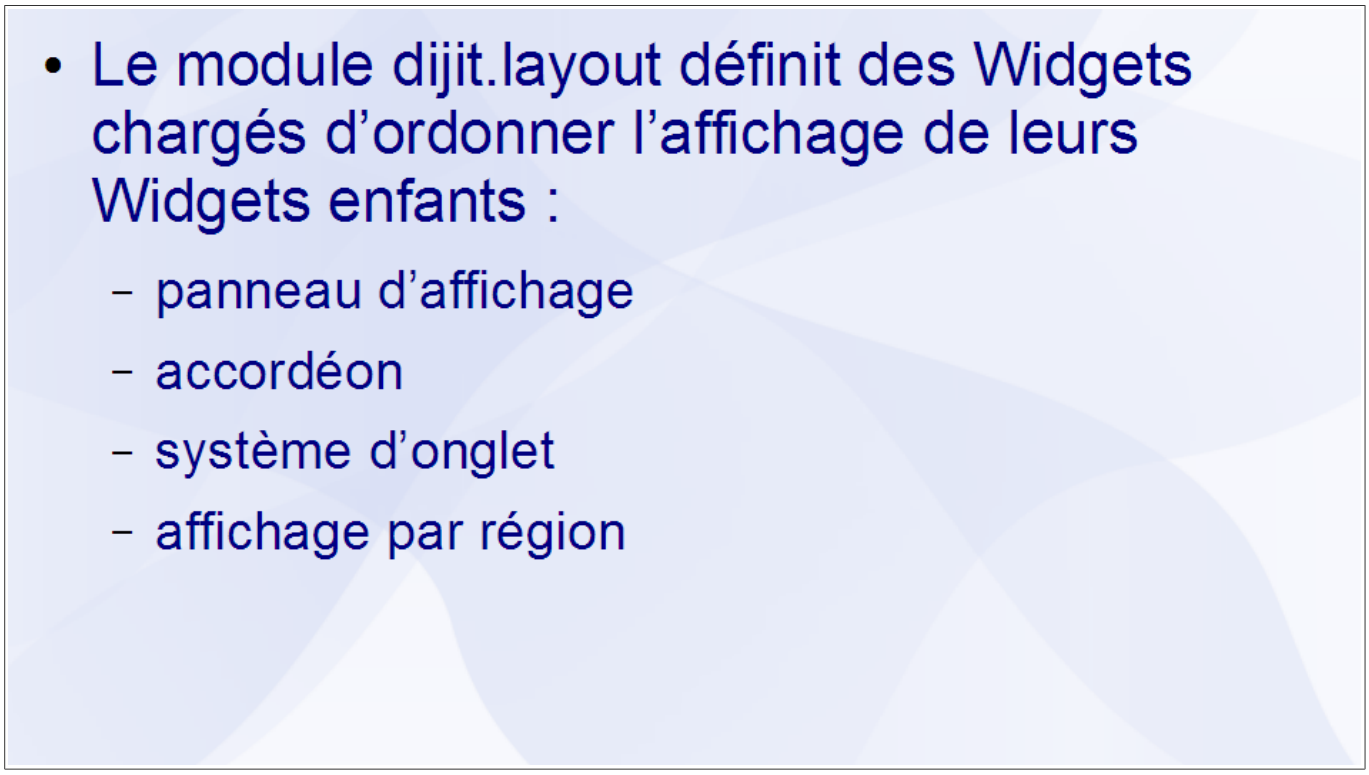


Le module `dijit.form` définit les Widgets suivants :

Widget	Description
`Form`	Remplacement de l'élément `form` vérifiant la validité de ses enfants.
`Button`	Remplacement des boutons de formulaire.
`TextBox`	Remplacement d'un champs de saisie.
`ValidationTextBox`	Champ de saisie permettant de valider la valeur saisie avec une expression régulière.
`NumberTextBox`	Champs de saisie n'acceptant que des nombres.
`NumberSpinner`	Variante de `NumberTextBox` avec des flèches pour modifier la valeur.
`DateTextBox`	Champs de saisie n'acceptant que des dates.
`TimeTextBox`	Champs de saisie n'acceptant que des valeurs horaires.
`CurrencyTextBox`	Champs de saisie n'acceptant que des valeurs monétaires.
`SimpleTextarea`	Remplacement de l'élément `textarea`.
`Textarea`	Remplacement de l'élément `ftextarea` dont la taille s'adapte automatiquement.
`FilteringSelect`	Remplacement de l'élément `select`.

Widget	Description
`MultiSelect`	Remplacement de l'élément `select` multiple.
`ComboBox`	Combinaison d'un champ de saisie et d'une liste déroulante.
`CheckBox`	Remplacement d'une case à cocher.
`RadioButton`	Remplacement d'un bouton radio.
`Slider`	Widget permettant de sélectionner une valeur à l'aide d'un curseur.

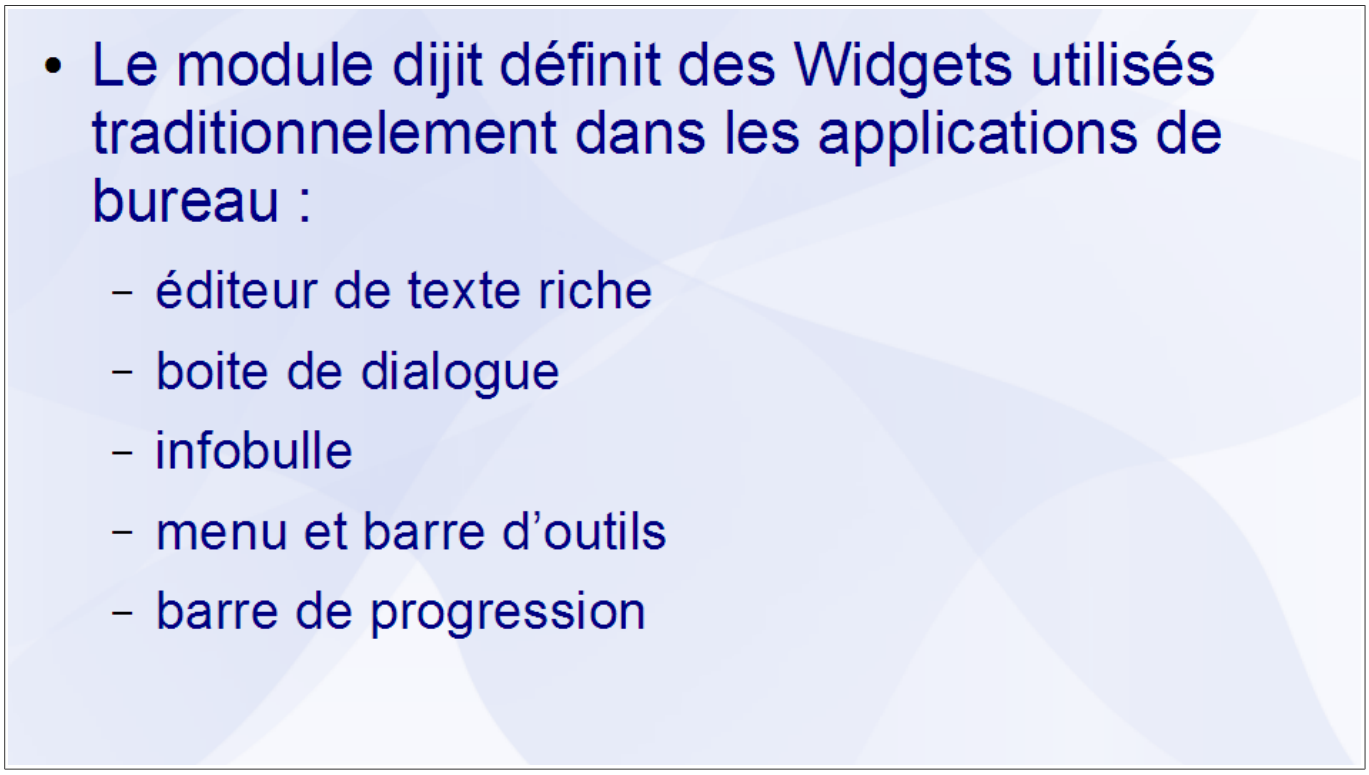
Les Widgets de mise en page



Le module `dijit.layout` définit les Widgets suivants :

Widget	Description
`ContentPane`	Panneau d'affichage pouvant charger son contenu via une requête AJAX.
`BorderContainer`	Définit 5 zones (`top`, `left`, `center`, `right`, `bottom`) dans lesquelles il affiche ses enfants.
`StackContainer`	Conteneur affichant un seul de ses enfants à la fois.
`TabContainer`	Variante du `StackContainer` présentant les titres de tous ses enfants sous la forme d'onglets.
`AccordionContainer`	Variante du `StackContainer` présentant les titres de tous ses enfants à la manière d'un accordéon.

Les Widgets applicatifs



Le module `dijit` définit les Widgets suivants :

Widget	Description
`Calendar`	Widget représentant un calendrier.
`Dialog`	Widget permettant d'afficher une boite de dialogue.
`Editor`	Éditeur de texte riche.
`ProgressBar`	Widget représentant une barre de progression.
`MenuBar`	Widget représentant une barre de menus.
`Menu`	Widget représentant un menu.
`Toolbar`	Widget représentant une barre d'outil.
`Tooltip`	Widget représentant une infobulle.
`Tree`	Widget représentant une structure arborescente.

Les Widgets personnalisés

Définition d'un Widget personnalisé :

```
dojo.declare("module.Widget", [dijit._Widget], {

  // Méthodes du cycle de vie
  preamble: function(){…},
  constructor: function(){…},
  postMixInProperties: function(){…},
  buildRendering: function(){…},
  postCreate: function(){…},
  startup: function(){…},
  destroyRecursive: function(){…},
  uninitialize: function(){…}
});
```

Dans Dojo Toolkit, un Widget est une classe qui hérite de `dijit._Widget`. Cette dernière définit plusieurs méthodes qui constitue de cycle de vie du Widget :

Méthode	**Description**
`preamble()`	S'exécute juste avant le constructeur.
`constructor()`	Initialise les propriétés de l'instance.
`postMixInProperties()`	S'exécute après la résolution de l'héritage du Widget et juste avant la création de sa représentation dans le document.
`buildRendering()`	Crée le DOM du Widget dans la propriété `domNode`.
`postCreate()`	S'exécute après la création du DOM du Widget et son ajout au document.
`startup()`	Déclenche la mise en page d'un Widget et de ses enfants. Pour les Widgets contenant des enfants, cette méthode doit obligatoirement être appelé manuellement s'ils ont été créés en JavaScript.
`destroyRecursive()`	(finale) Déclenche la destruction des enfants du Widget et appelle sa méthode `uninitialize()`.
`uninitialize()`	Définie des opérations à effectuer à la destruction du Widget.

Le système de gabarit

Définition d'un Widget utilisant le système de gabarit :

```
dojo.declare("m.W",
[dijit._Widget, dijit._Templated], {
templateString: dojo.cache("m","templates/W.html")
});
```

Gabarit associé au Widget :

```
<div data-dojo-attach-event="onclick:Activated" >
  <div data-dojo-attach-point="messageNode" >
    ${message}
  </div>
</div>
```

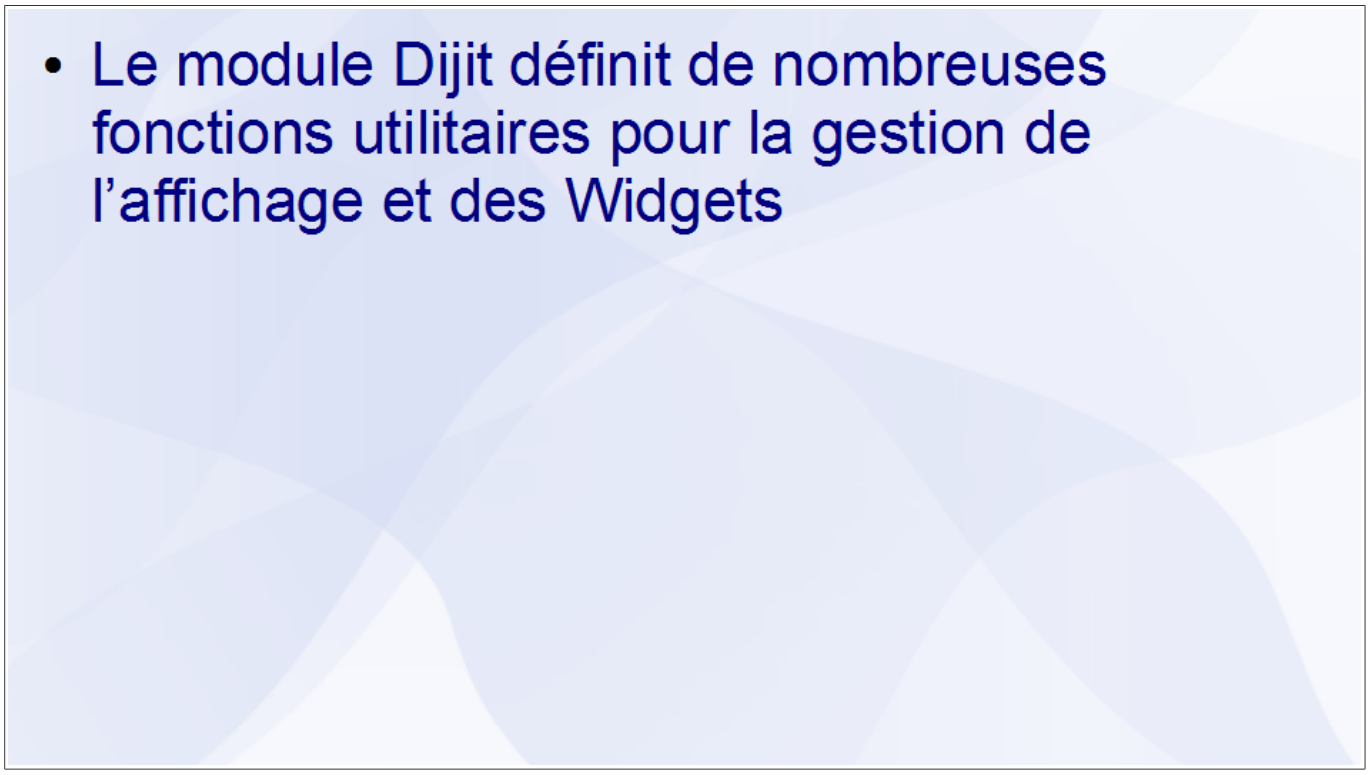
La classe `dijit._Templated` permet de créer le DOM d'un Widget à partir d'un gabarit HTML. Elle définit les deux propriétés suivantes :

Propriété	Type	Description
`templateString`	`String`	Code HTML du gabarit.
`widgetsInTemplate`	`Boolean`	Si `true`, il est possible d'utiliser des Widgets avec la syntaxe déclarative dans le gabarit.

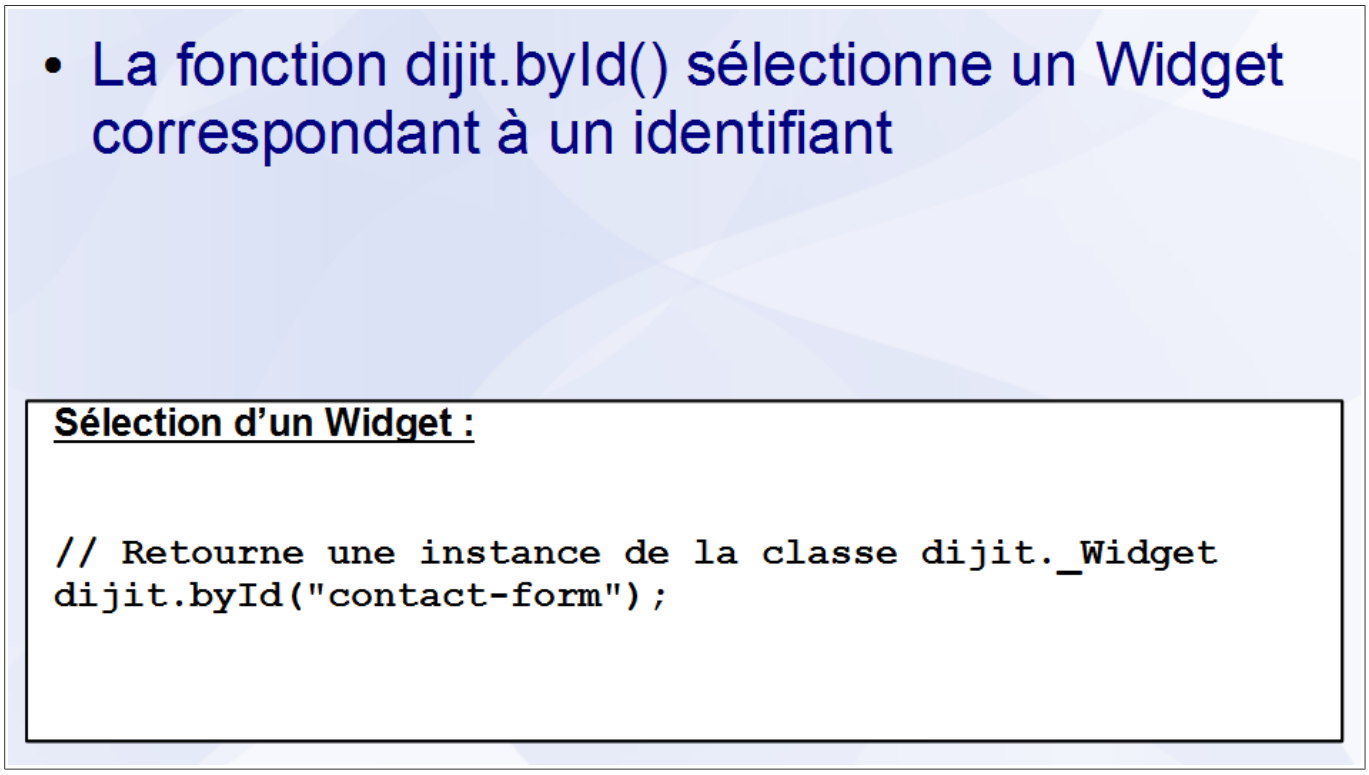
Dans le code HTML du gabarit :

- l'attribut `data-dojo-attach-event` permet d'associer un événement du DOM à une méthode du Widget
- l'attribut `data-dojo-attach-point` affecte le nœud du DOM de l'élément à la propriété spécifiée du Widget
- la syntaxe `${`*`nomPropriete`*`}` permet d'afficher la valeur d'une propriété du Widget dans le gabarit

Les fonctions utilitaires pour les Widgets



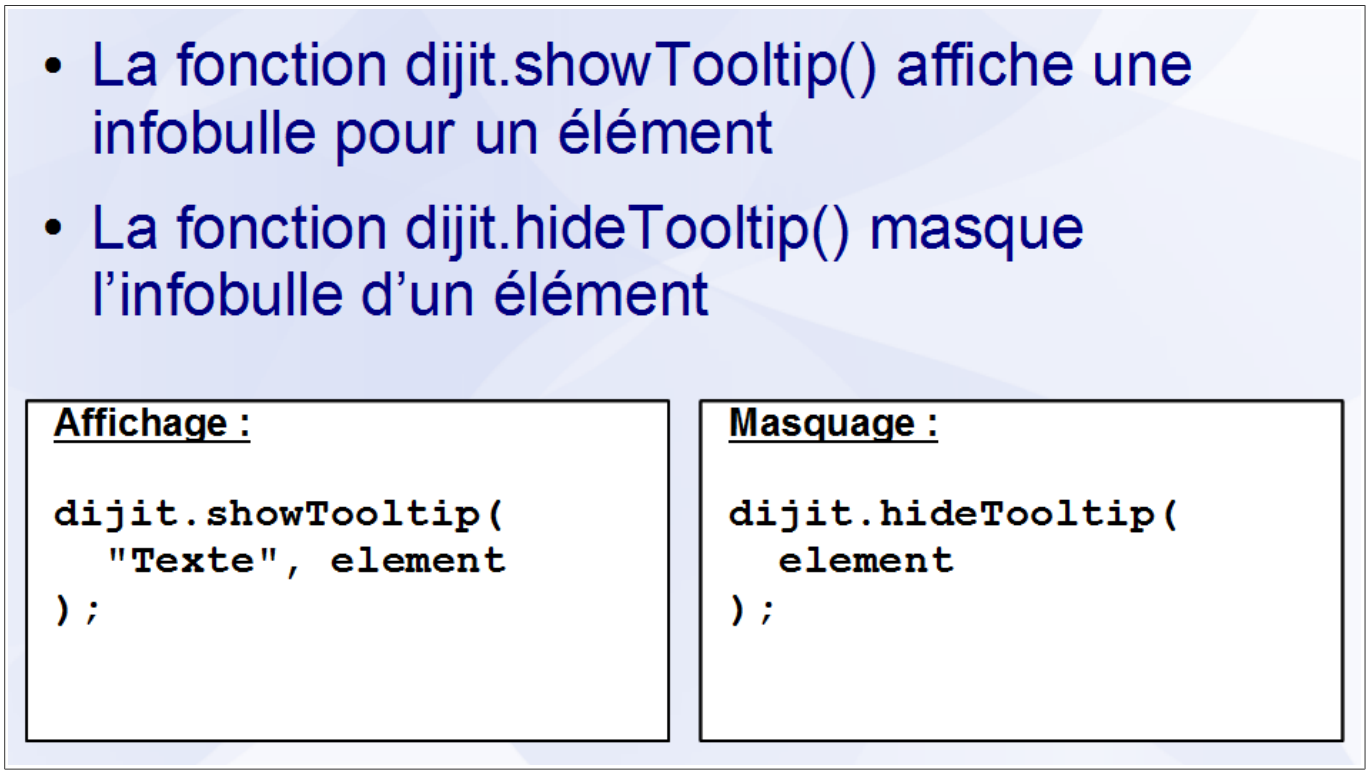
Sélection d'un Widget



La fonction `dijit.byId()` sélectionne un Widget correspondant à un identifiant. Elle accepte en paramètre un identifiant et retourne le Widget correspondant s'il existe ou `null` dans le cas contraire.

Attention à ne pas confondre cette fonction avec `dojo.byId()` qui permet de sélectionner un élément du DOM et non un Widget.

Manipulation des infobulles



La fonction `dijit.showTooltip()` affiche une infobulle pour un élément. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`innerHTML`	`String`	Texte à afficher dans l'infobulle.
`aroundNode`	`Element`	Élément associé à l'infobulle.

La fonction `dijit.hideTooltip()` masque l'infobulle d'un élément. Elle accepte en paramètre un nœud du DOM.

Conclusion

- Dojo Toolkit dispose de plusieurs thèmes définissant l'apparence des Widgets
- Un Widget peut être instancié en JavaScript ou avec la syntaxe déclarative
- Les Widgets héritent de la classe dijit._Widget
- La classe dijit._Templated permet de créer le DOM d'un Widget à partir d'un gabarit HTML

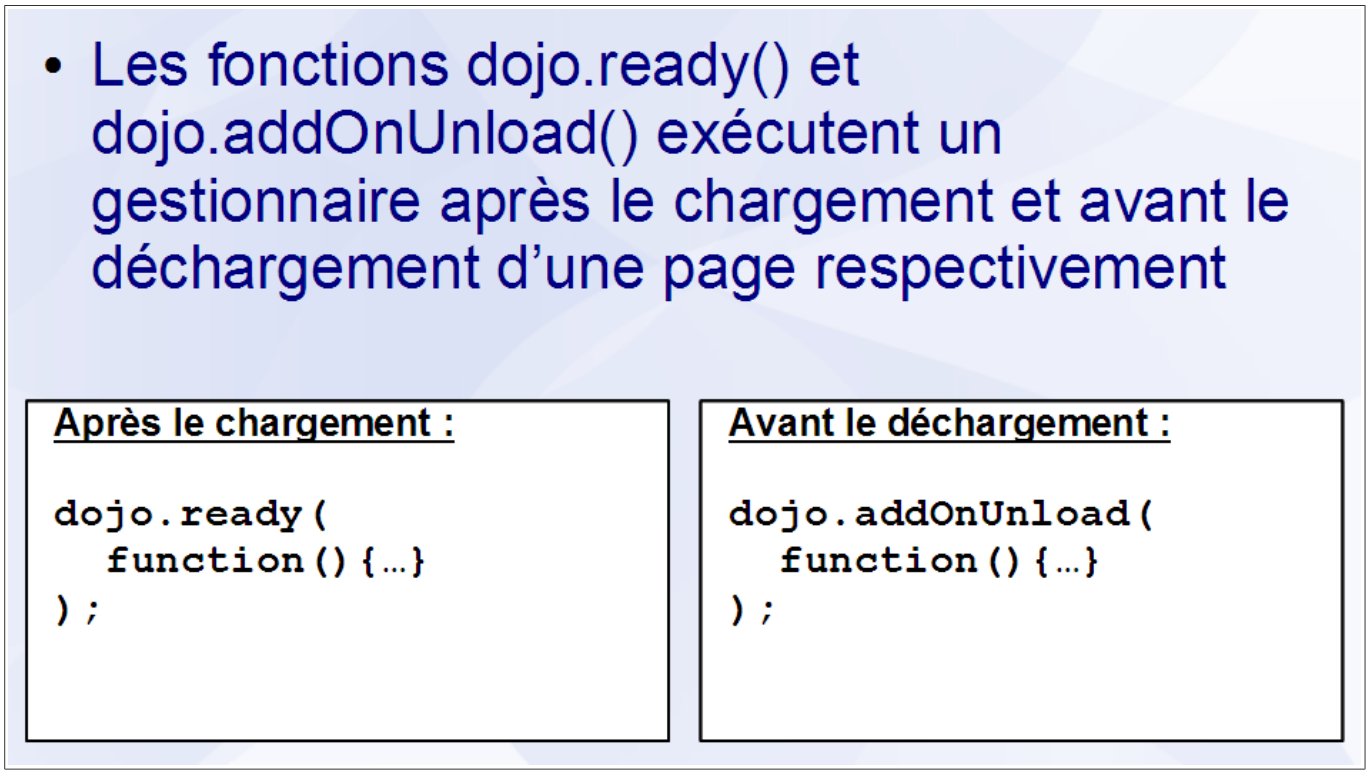
Chapitre 5 – La gestion des événements

Sommaire

Introduction

- La programmation côté client en JavaScript est principalement événementielle
- Le développeur écrit des gestionnaires qui sont déclenchés automatiquement quand survient un événement dont ils ont la charge
- Dojo Toolkit dispose de plusieurs fonctions facilitant la gestion des événements

Les événements du cycle de vie du document



La fonction `dojo.ready()` exécute un gestionnaire après :

- le chargement du DOM de la page (événement `DOMContentLoaded`)
- le chargement des modules importés avec la fonction `dojo.require()`
- l'instanciation des Widgets présent dans le document (`parseOnLoad` vaut `true`)

Elle accepte les paramètres suivants :

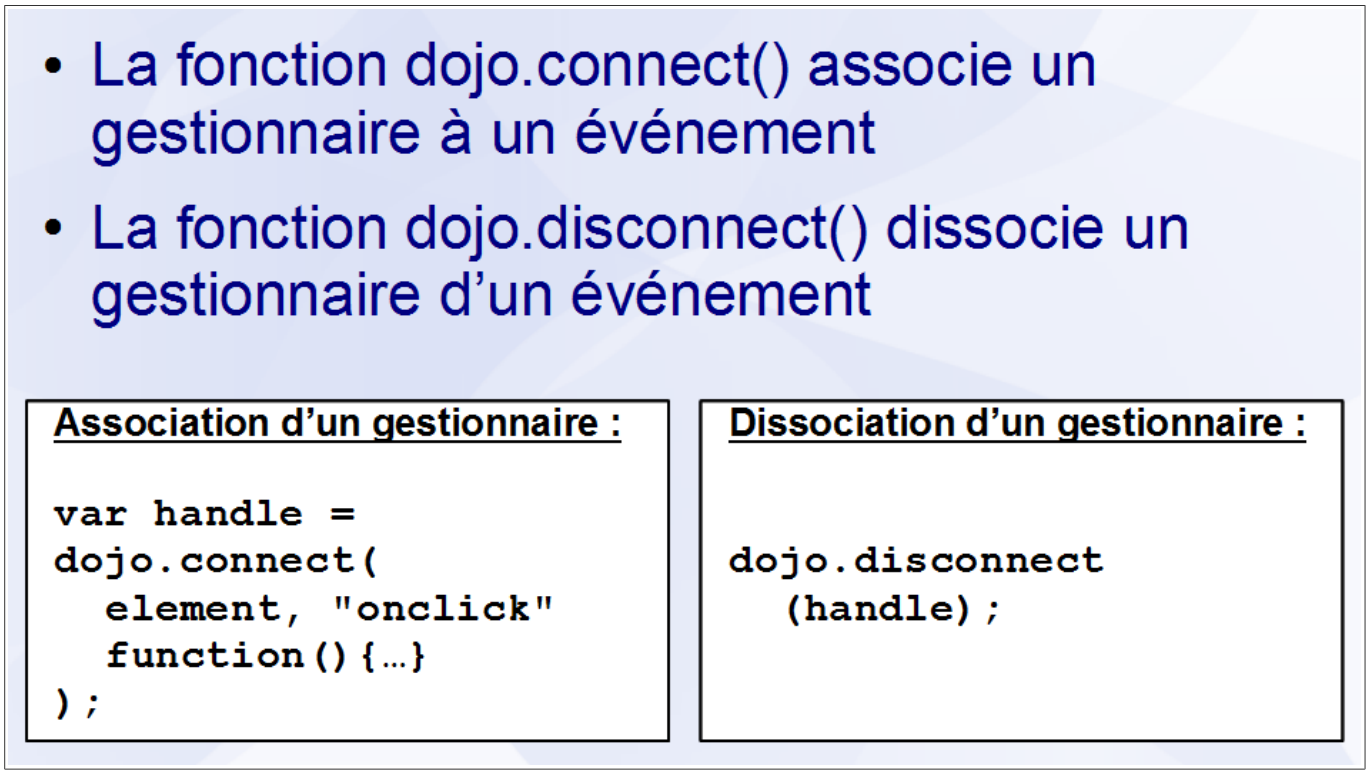
Paramètre	Type	Description
`obj`	`Object`	(optionnel) Objet de contexte pour le gestionnaire.
`functionName`	`Function` `String`	Fonction à exécuter ou nom d'une méthode de l'objet de contexte.

La fonction `dojo.ready()` est un alias de la fonction `dojo.addOnLoad()`.

La fonction `dojo.addOnUnload()` exécute un gestionnaire avant le déchargement d'une page. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`obj`	`Object`	(optionnel) Objet de contexte pour le gestionnaire.
`functionName`	`Function` `String`	Fonction à exécuter ou nom d'une méthode de l'objet de contexte.

Les événements du DOM



La fonction `dojo.connect()` associe un gestionnaire à un événement du DOM (tout en minuscule) ou à une méthode d'un objet quelconque (avec la même casse). Elle accepte les paramètres suivants :

Paramètre	Type	Description
`obj`	`Object`	Objet source de l'événement.
`event`	`String`	Nom de l'événement ou nom d'une méthode de l'objet source.
`context`	`Object`	(optionnel) Objet de contexte pour le gestionnaire.
`method`	`Function` `String`	Fonction à exécuter ou nom d'une méthode de l'objet de contexte.

La fonction `dojo.connect()` retourne un objet qui, utilisé avec la fonction `dojo.disconnect()`, permet de dissocier le gestionnaire d'un événement ou de la méthode d'un objet.

La liste des événements du DOM est fournie en annexe.

Les événements d'un Widget

- Les événements d'un Widget sont gérés de la même façon que les événements du DOM
- La syntaxe déclarative peut aussi être utilisée pour gérer les événements d'un Widget

Syntaxe déclarative :

```
<div data-dojo-type="dijit.form.Button" >
  <script type="dojo/connect"
    data-dojo-event="onClick"
    data-dojo-args="evt" >…</script>
</div>
```

Tous les Widgets disposent de méthodes reprenant le nom des événements du DOM mais avec des majuscules pour les distinguer : `onClick()`, `onDblClick()`, `onKeyDown()`, `onKeyPress()`, `onKeyUp()`, `onMouseDown()`, `onMouseMove()`, `onMouseOut()`, `onMouseOver()`, `onMouseLeave()`, `onMouseEnter()`, `onMouseUp()`, `onFocus()`, `onBlur()`.

Tout comme pour les événements du DOM, les fonctions `dojo.connect()` et `dojo.disconnect()` peuvent être utilisées pour associer et dissocier un gestionnaire à l'une de ces méthodes. Chaque widget possède également une méthode `connect()` fonctionnant à la manière de `dojo.connect()` mais qui spécifie automatique le widget comme objet de contexte.

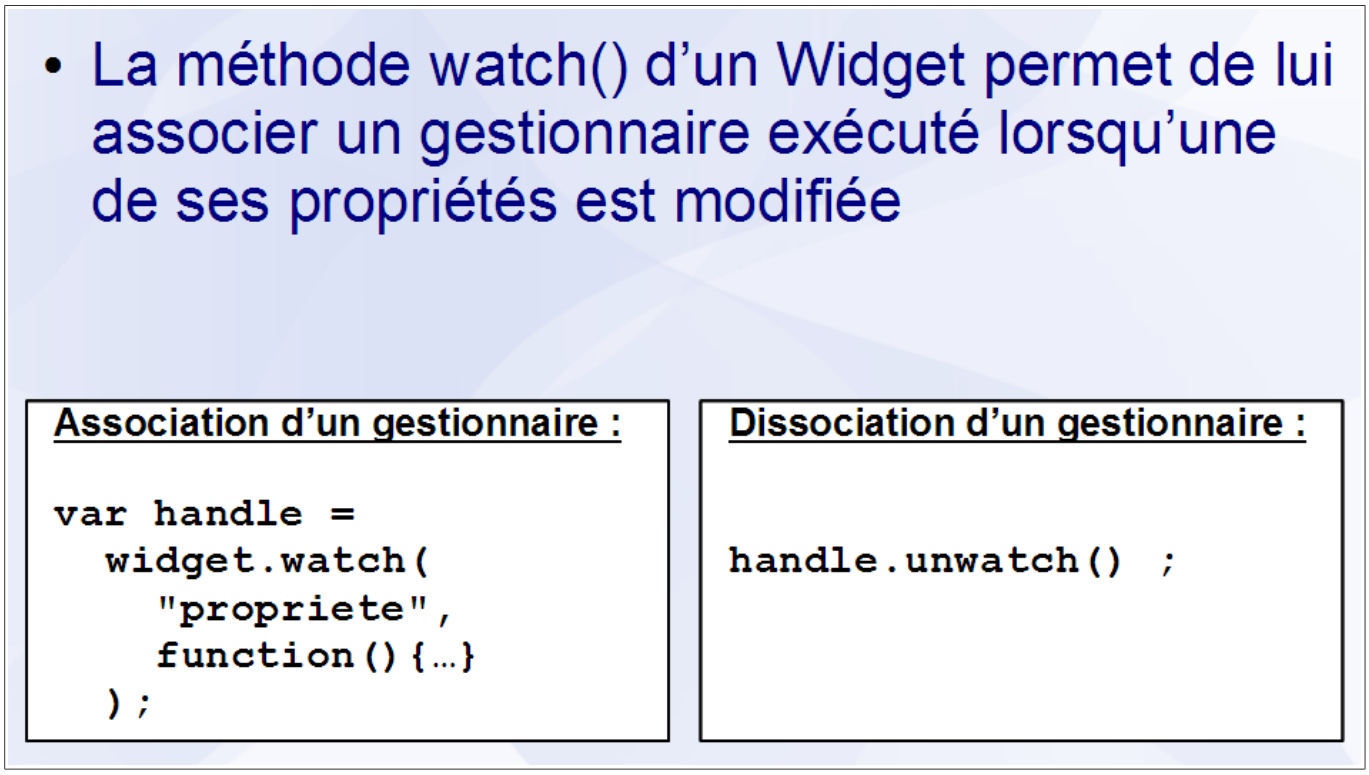
Par exemple :

```
dojo.connect(element, evenement, widget, methode) ;

// Équivalent avec la méthode connect() du Widget
widget.connect(element, evenement, methode) ;
```

Il est également possible de définir un gestionnaire d'événement pour un Widget avec la syntaxe déclarative.

Le suivi des propriétés d'un Widget

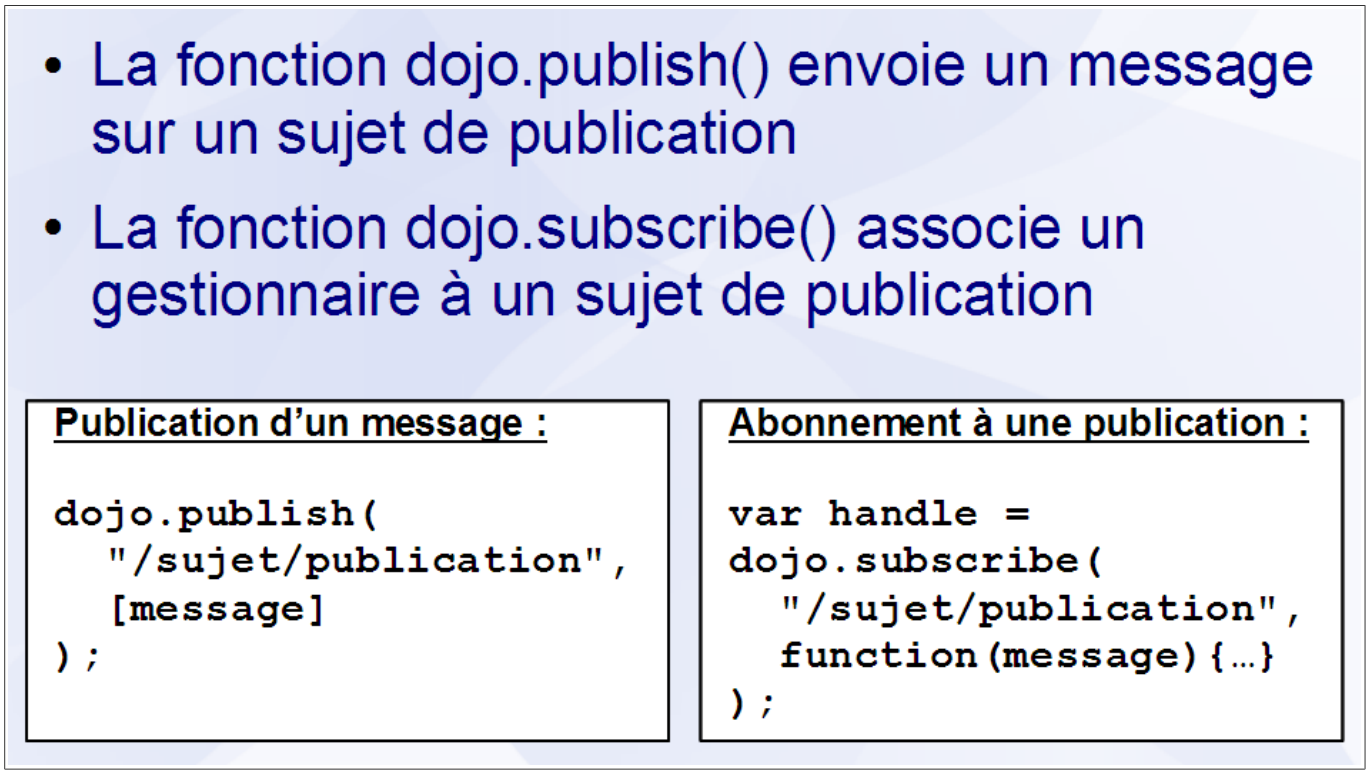


La méthode `watch()` d'un Widget permet de lui associer un gestionnaire qui sera exécuté à la modification de l'une de ses propriétés. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`name`	`String`	Nom de la propriété à suivre.
`callback`	`Function`	Fonction à exécuter.

La méthode `watch()` d'un Widget retourne un objet dont la méthode `unwatch()` permet d'arrêter le suivi de la propriété par le gestionnaire.

L'architecture Publish/Subscribe



Dans le modèle traditionnel de gestion des événements, il existe un lien fort entre la source de l'événement et le gestionnaire d'événement. Chacun a connaissance de l'existence de l'autre. Hors, un couplage élevé entre plusieurs composants entraîne des difficultés de maintenance et de montée en charge.

Dans l'architecture Publish/Subscribe, la source de l'événement et le gestionnaire d'événement n'ont pas connaissance l'un de l'autre. C'est le nom du sujet de publication qui les unit. Il est ainsi plus facile de faire évoluer l'un car cela est sans incidence sur l'autre.

La fonction `dojo.publish()` envoie un message sur un sujet de publication. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`topic`	`String`	Nom du sujet de publication.
`args`	`Array`	Tableau de paramètres qui est envoyé aux abonnés.

La fonction `dojo.subscribe()` associe un gestionnaire à un sujet de publication. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`topic`	`String`	Nom du sujet de publication.
`context`	`Object`	(optionnel) Objet de contexte pour le gestionnaire.
`method`	`Function` `String`	Fonction à exécuter ou nom d'une méthode de l'objet de contexte.

La fonction `dojo.subscribe()` retourne un objet qui, utilisé avec la fonction `dojo.unsubscribe()`, permet de dissocier le gestionnaire du sujet de publication.

Conclusion

- La fonction dojo.ready() permet d'exécuter un gestionnaire après le chargement de la page
- La fonction dojo.connect() permet d'associer un gestionnaire à un événement
- La fonction dojo.publish() permet d'envoyer un message sur un sujet de publication
- La fonction dojo.subscribe() permet d'associer un gestionnaire à un sujet de publication

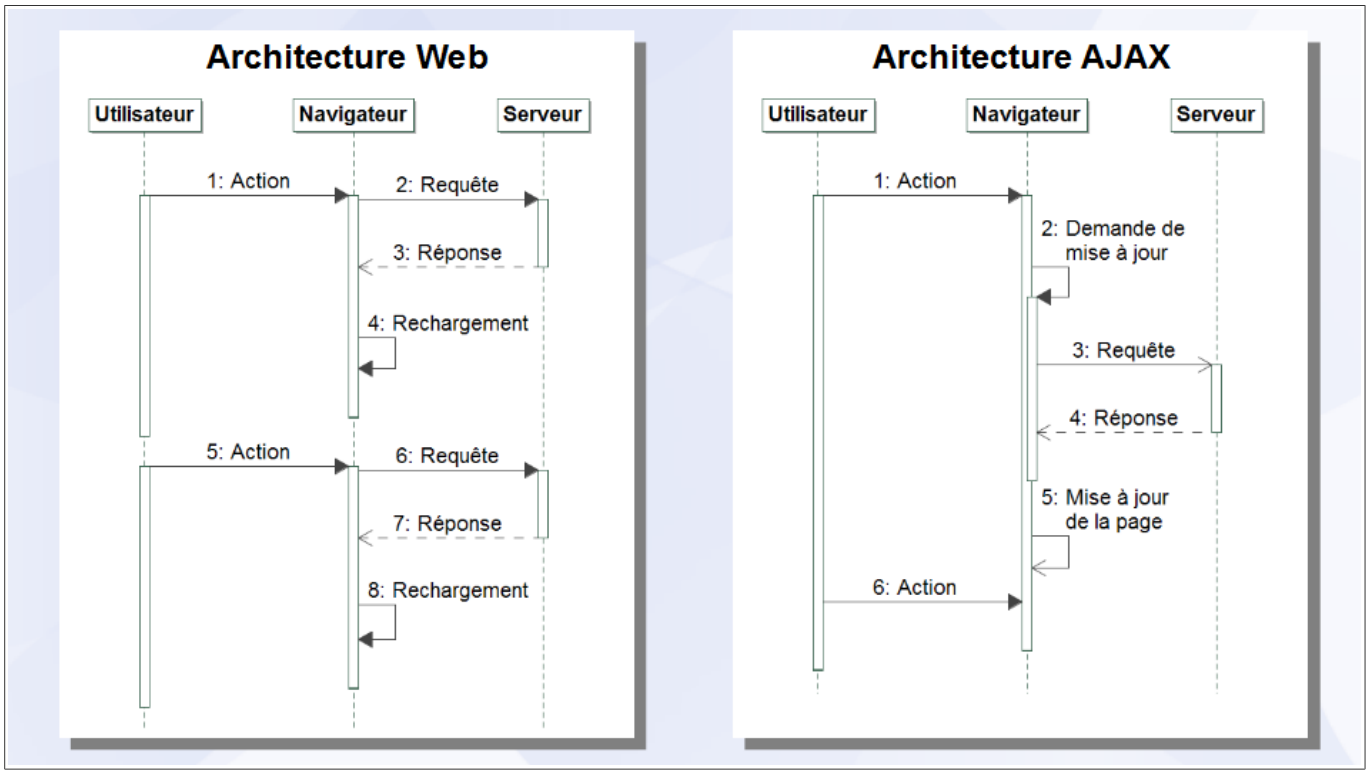
Chapitre 6 – La communication client/serveur

Sommaire

Introduction

- Une application Web traditionnelle recharge la page affichée à chaque action de l'utilisateur
- Ces rechargements diminuent le temps de réponse de l'application et nuisent au confort d'utilisation
- Dojo Toolkit dispose de fonctions permettant de charger des données sans recharger la page

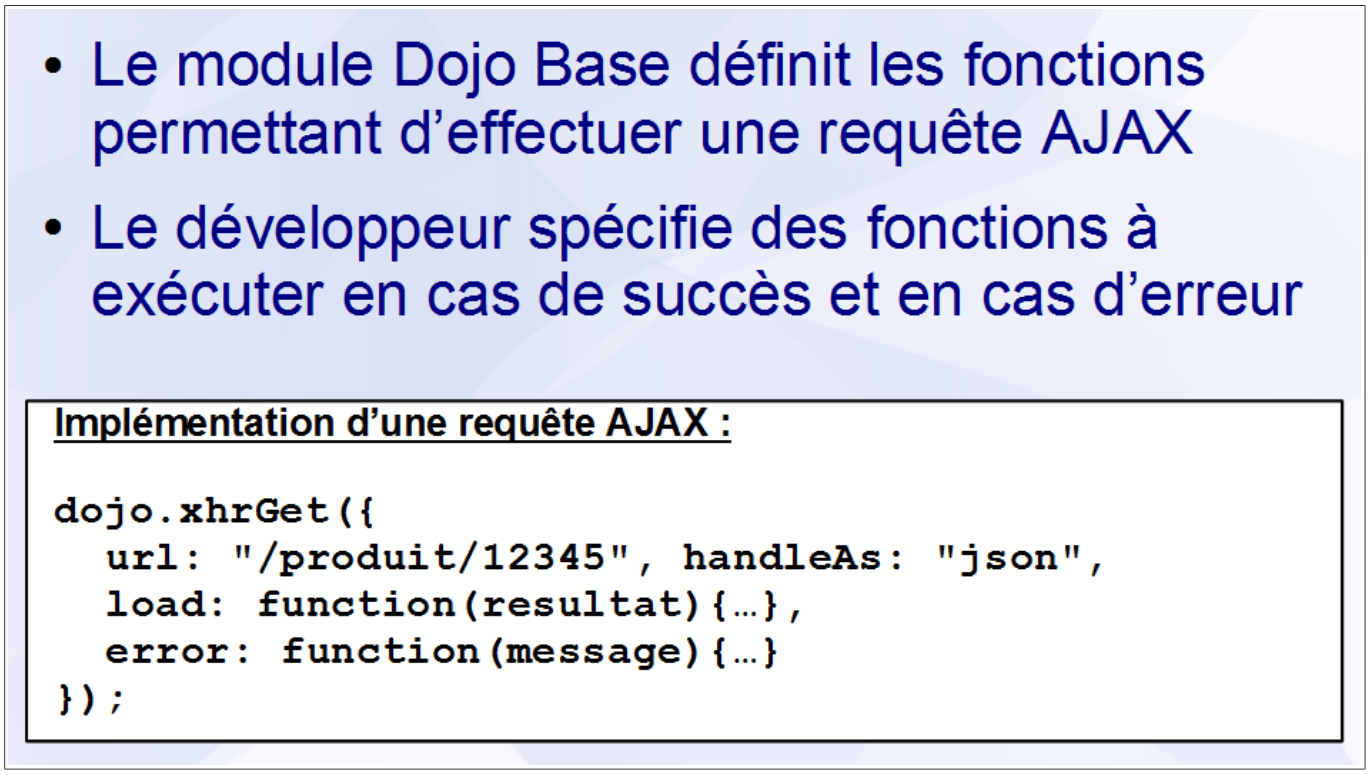
L'architecture AJAX



Dans l'architecture Web traditionnelle, une action de l'utilisateur déclenche l'envoi d'une requête synchrone – bloquant toute activité sur la page – vers le serveur Web qui répond en envoyant une nouvelle page à afficher. Le navigateur, après réception de la réponse, recharge le contenu de la fenêtre. Ces nombreux rechargements auquel s'ajoute le temps de réponse du serveur font que les applications Web traditionnelles semblent plus poussives que les applications de bureau.

Dans une architecture AJAX, l'application Web n'est constituée que d'une seule page Web dont le contenu est mis à jour dynamiquement. Lors d'une action de l'utilisateur, une requête asynchrone – ne bloquant pas les autres activités sur la page – est envoyée au serveur Web via l'objet `XMLHttpRequest`, ce qui ne provoque pas de rechargement de la page. Le développeur peut alors spécifier une fonction qui sera exécutée à la réception de la réponse du serveur. Cette architecture permet d'obtenir des applications plus réactives, ce qui améliore le confort d'utilisation.

Implémentation d'une requête AJAX



Le module Dojo Base définit une fonction permettant d'effectuer une requête AJAX pour chacune des quatre principales méthodes du protocole HTTP :

- `dojo.xhrGet()` – Envoi une requête HTTP avec la méthode GET
- `dojo.xhrPost()` – Envoi une requête HTTP avec la méthode POST
- `dojo.xhrPut()` – Envoi une requête HTTP avec la méthode PUT
- `dojo.xhrDelete()` – Envoi une requête HTTP avec la méthode DELETE

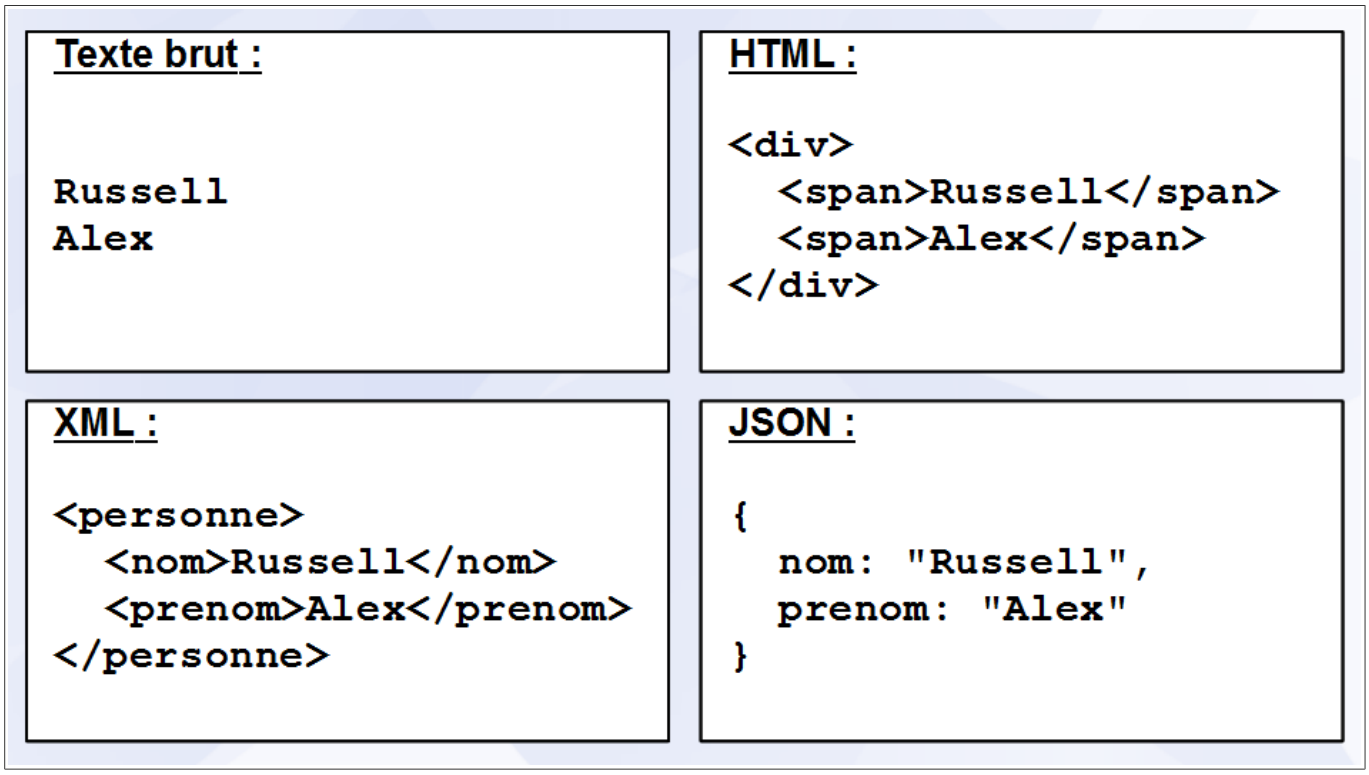
Les fonctions généralement employés sont `dojo.xhrGet()` pour recevoir des données du serveur et `dojo.xhrPost()` pour envoyer des données au serveur. Les fonctions `dojo.xhrPut()` et `dojo.xhrDelete()` sont utilisées conjointement avec les deux précédentes pour communiquer avec des services Web utilisant une architecture REST.

Ces fonctions acceptent en paramètre un objet avec les propriétés suivantes (liste partielle) :

Propriété	Type	Description
`url`	`String`	URL utilisée pour la requête.
`headers`	`Object`	En-têtes HTTP à envoyer avec la requête. L'en-tête `X-Requested-With: XMLHttpRequest` est envoyé automatiquement.
`content`	`Object`	Paramètres à envoyer avec la requête : • dans l'URL pour la méthode GET • dans le corps de la requête pour la méthode POST
`form`	`Element`	Élément `form` dont les champs seront envoyés avec la requête (remplace `url` et `content`).
`handleAs`	`String`	Type de données envoyé par le serveur : `text`, `json`, `xml` ou `javascript`.
`load`	`Function`	Fonction à exécuter en cas de succès.
`error`	`Function`	Fonction à exécuter en cas d'erreur.
`handle`	`Function`	Fonction à exécuter quelque soit le résultat de la requête.
`timeout`	`Number`	Durée d'attente de la réponse en millisecondes.
`failOk`	`Boolean`	Si `true`, l'échec de la requête est autorisé et aucun message d'erreur n'est affiché dans la console.
`sync`	`Boolean`	Si `true`, la requête est synchrone.

Ces fonctions retournent une instance de la classe `dojo.Deferred` (non étudiée).

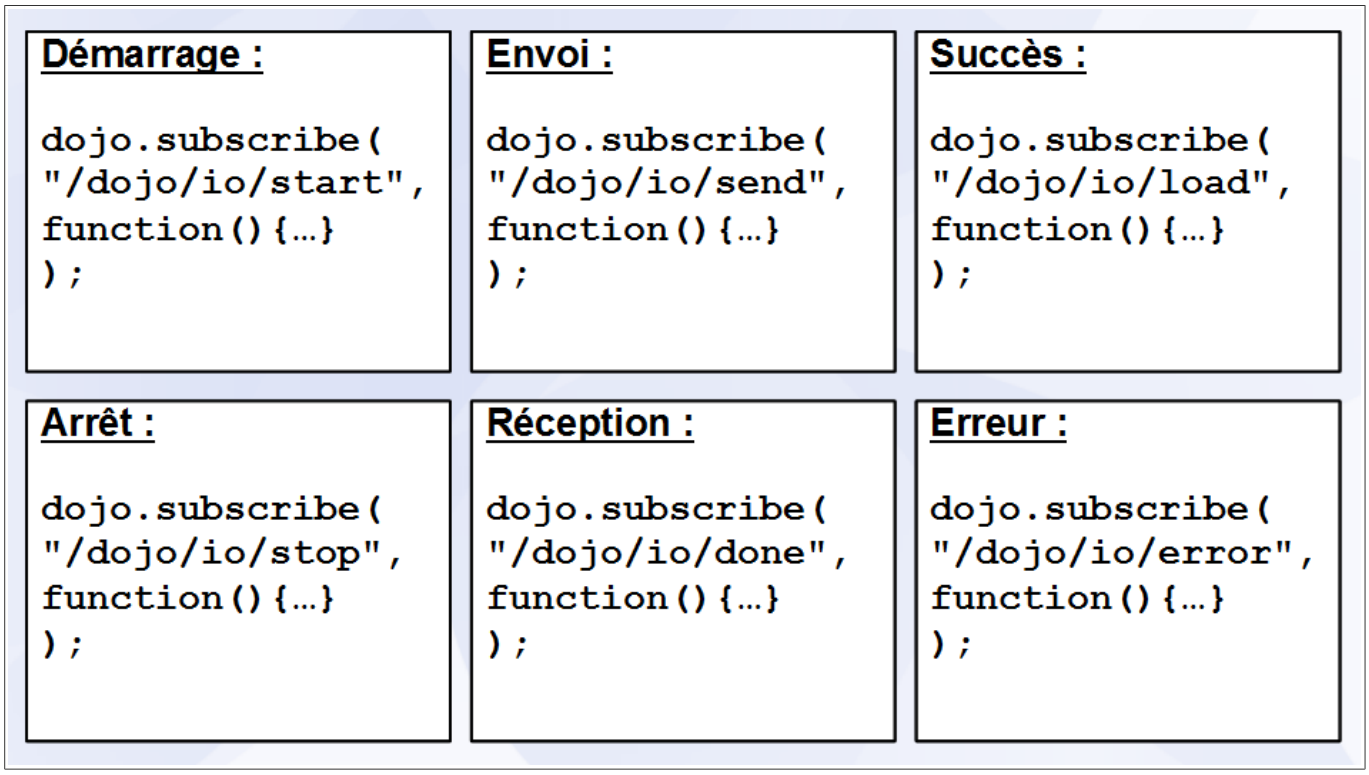
Les formats de données



Une requête AJAX peut être utilisée pour obtenir des données dans un format quelconque. Cependant, les quatre formats suivant sont le plus souvent employés :

Format	**Description**
Texte brut	Texte sans format particulier.
HTML	Le format HTML, reçu sous la forme de texte brut, présente l'avantage de pouvoir directement être inséré dans le document pour en modifier l'affichage.
XML	Le format XML permet de représenter des structures complexes. Le résultat de la requête AJAX est un objet `Document` pouvant être manipuler avec les fonctions de manipulation du DOM.
JSON	Le format JavaScript Object Notation (JSON) est inspiré du langage JavaScript et permet d'obtenir en résultat de la requête AJAX des données utilisables directement par le programme. Ce format est de plus en plus utilisé à la place de XML car plus léger que ce dernier.

Les événements du système d'entrées/sorties



Toutes les fonctions de Dojo Toolkit effectuant des opérations d'entrées/sorties peuvent publier des messages sur les sujets de publication suivants :

Sujet de publication	Description
`/dojo/io/start`	Début d'une opération d'entrée/sortie.
`/dojo/io/send`	Envoi de la requête.
`/dojo/io/load`	Succès de la requête.
`/dojo/io/error`	Échec de la requête.
`/dojo/io/done`	Fin de la requête (succès ou échec).
`/dojo/io/stop`	Requête annulée.

Conclusion

- Les requêtes AJAX sont souvent asynchrones
- Les fonctions dojo.xhrGet() et dojo.xhrPost() permettent d'effectuer des requêtes AJAX
- Plusieurs formats de données sont utilisés
- Les fonctions effectuant des entrées/sorties publient des messages sur plusieurs sujets de publication

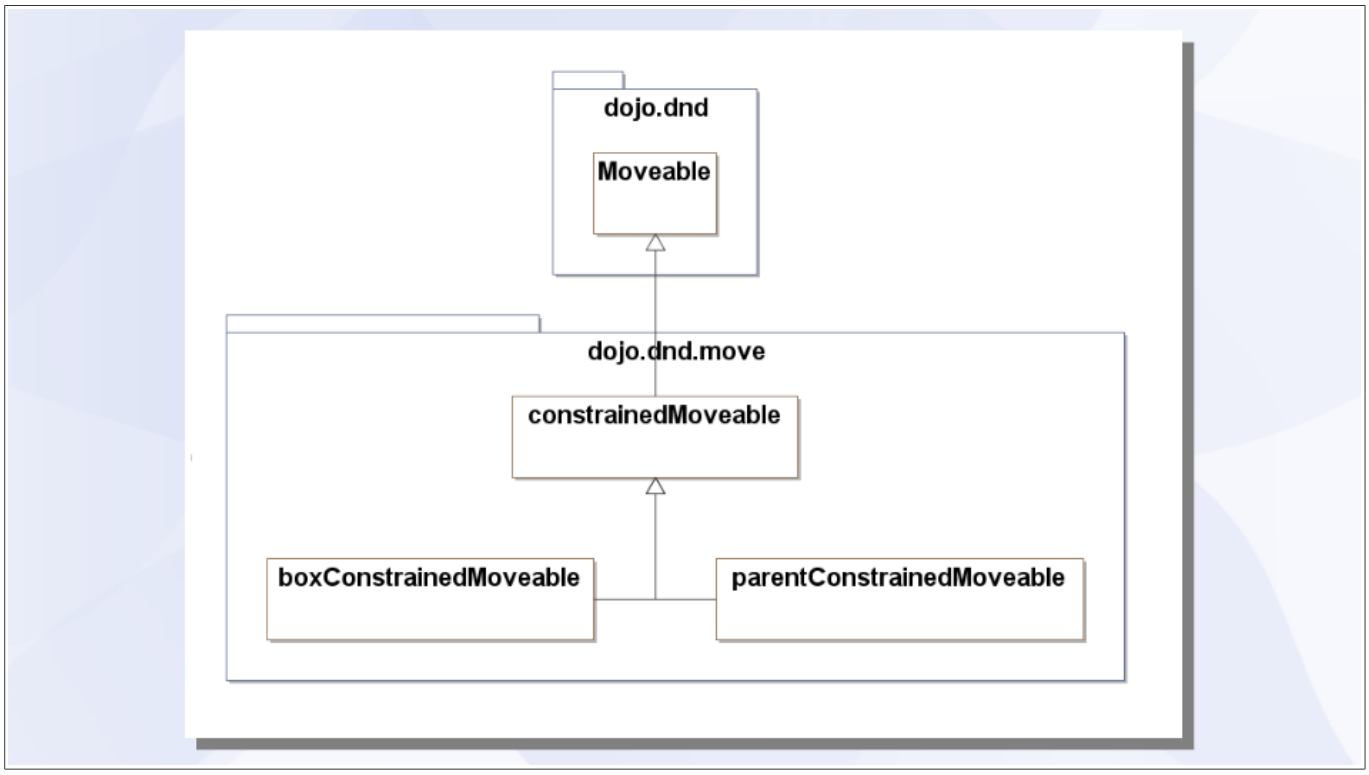
Chapitre 7 – Le glisser-déposer

Sommaire

Introduction

- L'une des fonctionnalités les plus distinctives des applications de bureau est le glisser-déposer
- Dojo Toolkit dispose de plusieurs modules permettant d'implémenter le glisser-déposer dans une application Web

Les éléments déplaçables



La classe `dojo.dnd.Moveable` rend un élément quelconque du document déplaçable sans contrainte à l'aide de la souris. Elle doit être importée avant de pouvoir être utilisée.

Les classes du module `dojo.dnd.move` rendent un élément quelconque du document déplaçable avec les restrictions suivantes :

- `dojo.dnd.move.constrainedMoveable` – l'élément n'est déplaçable que dans une boite calculée lors du premier déplacement
- `dojo.dnd.move.boxConstrainedMoveable` – l'élément n'est déplaçable que dans la boite spécifiée à la construction de l'objet
- `dojo.dnd.move.parentConstrainedMoveable` – l'élément n'est déplaçable qu'à l'intérieur de son parent

Le module `dojo.dnd.move` doit être importé avant d'utiliser les classes qu'il définit.

Implémentation d'un élément déplaçable

JavaScript :

```
var carre = new dojo.dnd.Moveable(
  "carre", {handle: "poignee", skip: true}
);
```

Syntaxe déclarative :

```
<div id="carre" data-dojo-type="dojo.dnd.Moveable"
  data-dojo-props="handle: 'poignee', skip: true"
>
  <div id="poignee" ></div>
</div>
```

Chacune de ces classes acceptent en paramètres l'identifiant de l'élément à rendre déplaçable et un tableau associatif d'options. Chaque classe définit ses propres options en plus de celles de son parent.

La classe `dojo.dnd.Moveable` a pour options :

Propriété	Type	Description
`handle`	`String`	Identifiant de l'élément servant de poignée à l'élément déplaçable. Si elle est définit, l'élément ne peut être déplacé qu'à l'aide de sa poignée.
`delay`	`Number`	Nombre de pixels de décalage à partir duquel le déplacement commence.
`skip`	`Boolean`	Si `true`, désactive le glisser-déposer pour les éléments de formulaire.

La classe `dojo.dnd.move.constrainedMoveable` a pour options :

Propriété	Type	Description
`constraints`	`Function`	Calcule et retourne une boite de contrainte lors du premier déplacement de l'élément.
`within`	`Boolean`	Si `true`, l'élément doit entièrement être contenu dans sa boite de contraintes.

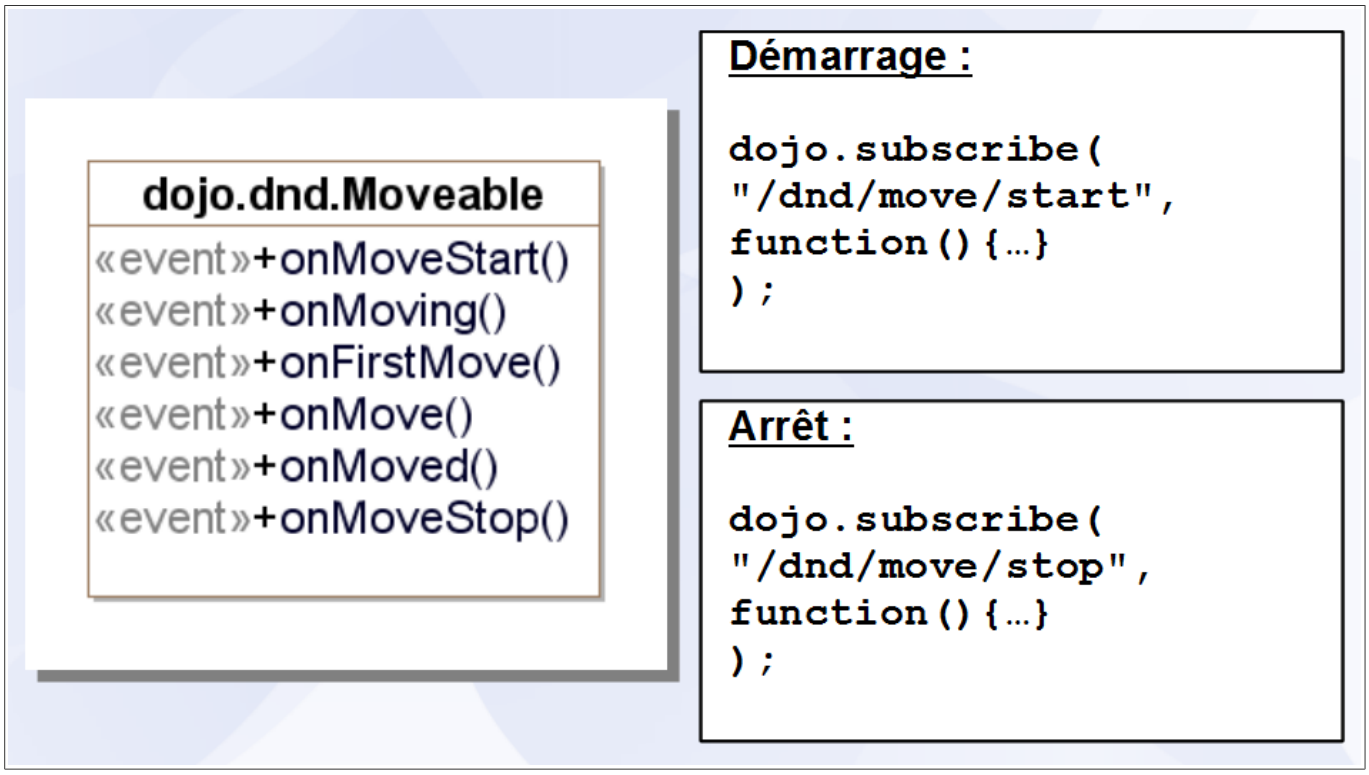
La classe `dojo.dnd.move.boxConstrainedMoveable` a pour options :

Propriété	Type	Description
`box`	`Object`	Boite de contraintes de l'élément déplaçable : • `t` – abscisse de la boite • `l` – ordonnée de la boite • `w` – longueur de la boite • `h` – hauteur de la boite

La classe `dojo.dnd.move.parentConstrainedMoveable` a pour options :

Propriété	Type	Description
`area`	`String`	Zone du parent où l'élément peut se déplacer : • `margin` – boite de marge du parent • `border` – boite de bordure du parent • `padding` – boite d'espacement du parent • `content` – boite de contenu du parent

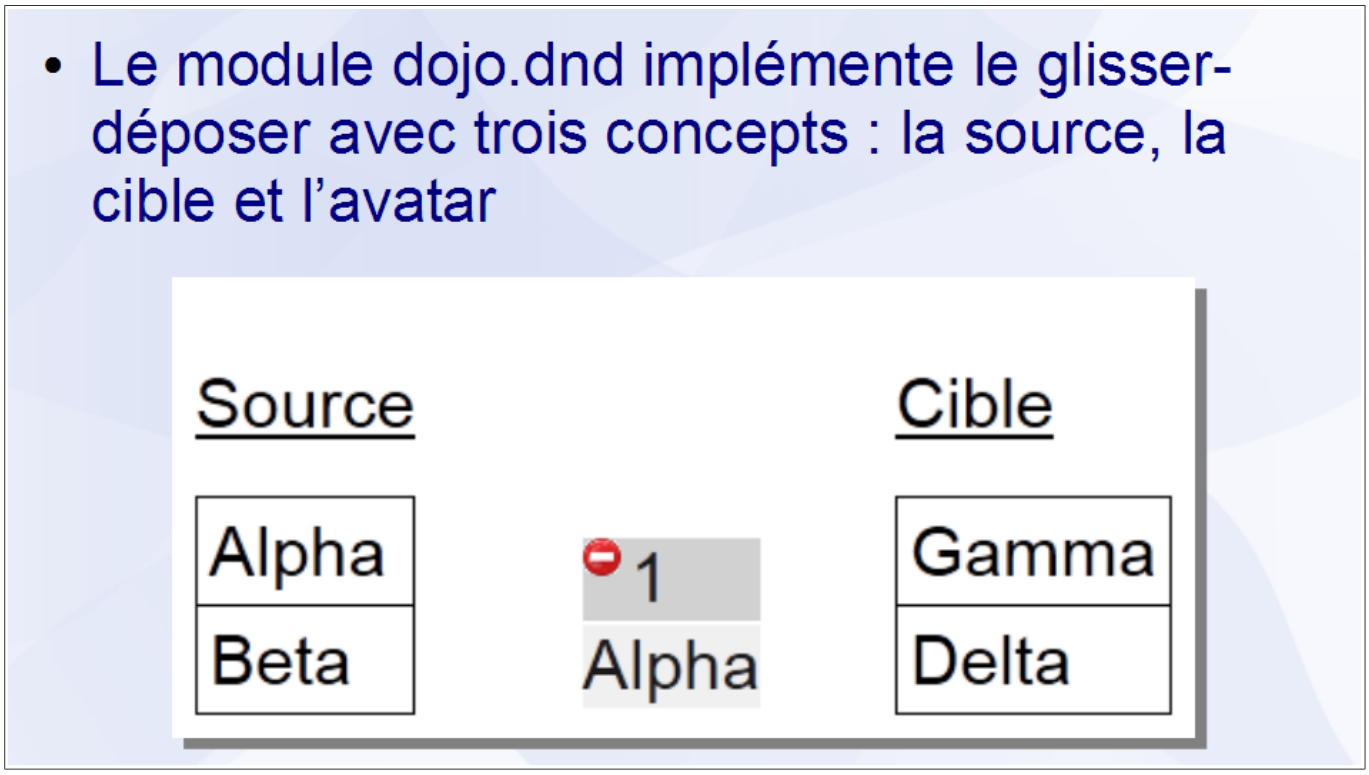
Les événements et publications des éléments déplaçables



La classe `dojo.dnd.Moveable` définit les événements et sujets de publication suivants :

Événement – Publication	Description
`onMoveStart()` `/dnd/move/start`	Événement déclenché au début du déplacement de l'élément.
`onMoving()`	Événement déclenché au début de chaque étape du déplacement de l'élément.
`onFirstMove()`	Événement déclenché au premier déplacement de l'élément.
`onMove()`	Événement déclenché à chaque étape du déplacement de l'élément.
`onMoved()`	Événement déclenché à la fin de chaque étape du déplacement de l'élément.
`onMoveStop()` `/dnd/move/stop`	Événement déclenché à la fin du déplacement de l'élément.

Les sources et les cibles



Le module `dojo.dnd` implémente un système de glisser-déposer plus complexe que le module `dojo.dnd.move` mais avec plus de fonctionnalités et supportant mieux un grand nombre d'éléments déplaçables. Il repose sur trois concepts :

- la source (instance de `dojo.dnd.Source`) contient des éléments pouvant être déplacés
- la cible (instance de `dojo.dnd.Target`) peut contenir des éléments déplaçables en provenance d'une ou plusieurs sources
- l'avatar (instance de `dojo.dnd.Avatar`) est une représentation de l'élément cours de déplacement

Le module `dojo.dnd.Source` doit être importé avant d'utiliser les classes ci-dessus.

La feuille de styles `dojo/resources/dnd.css` peut être importée dans l'application pour obtenir une mise en page basique de l'avatar. Elle peut également servir de base au développement d'une feuille de styles plus élaborée.

Implémentation d'une source

JavaScript :

```
var source = new dojo.dnd.Source("src");
source.insertNodes(false, [
  {data: "A", type: ["VOY"]},
  {data: "B", type: ["CON"]},
]);
```

Syntaxe déclarative :

```
<div id="src" data-dojo-type="dojo.dnd.Source" >
  <div class="dojoDndItem" dndType="VOY" >A</div>
  <div class="dojoDndItem" dndType="CON" >B</div>
</div>
```

La classe `dojo.dnd.Source` accepte en paramètres l'identifiant de l'élément à transformer en source et un tableau associatif d'options. Elle a pour options (liste partielle) :

Propriété	Type	Description
`accept`	`Array`	Types des éléments acceptés par la source.
`delay`	`Number`	Nombre de pixels de décalage à partir duquel le déplacement commence.
`horizontal`	`Boolean`	Si `true`, la source contient des éléments affichés en ligne.
`withHandles`	`Boolean`	Si `true`, un élément ne peut être déplacé que par sa poignée. La poignée est un enfant de l'élément déplaçable ayant pour classe `dojoDndHandle`.

Une source peut être crée avec la syntaxe déclarative à partir d'éléments de différents types : `div`, `ul`, `ol`, `table`. Comme pour les Widgets, l'attribut `data-dojo-type` définit la classe à employer et l'attribut `data-dojo-props` permet de spécifier les options. Les enfants de la source ayant pour classe `dojoDndItem` sont automatiquement ajoutés à la source comme éléments déplaçables. Chacun de ces éléments peut utiliser les attributs `dndData` pour spécifier ses données et `dndType` pour spécifier son type. En l'absence de l'attribut `dndData`, le contenu de l'élément est utilisé.

Implémentation d'une cible

JavaScript :

```
var cible = new dojo.dnd.Target(
  "cible", {accept: ["VOY"]}
);
```

Syntaxe déclarative :

```
<div id="cible" data-dojo-type="dojo.dnd.Target"
  data-dojo-props="accept:['VOY']"
></div>
```

La classe `dojo.dnd.Target` accepte en paramètres l'identifiant de l'élément à transformer en cible et un tableau associatif d'options. Les options sont les mêmes que pour les sources, ces dernières étant également des cibles.

Une cible peut être crée avec la syntaxe déclarative à partir d'éléments de différents types : `div`, `ul`, `ol`, `table`. Comme pour les Widgets, l'attribut `data-dojo-type` définit la classe à employer et l'attribut `data-dojo-props` permet de spécifier les options.

Personnalisation de l'avatar

JavaScript :

```
var source = new dojo.dnd.Source("src", {
  creator: function(item, hint){
    return {data: item, type: item.type,
     node: dojo.create("div", {innerHTML: item.text})};
}});
```

Syntaxe déclarative :

```
<script type="dojo/method" data-dojo-event="creator"
  data-dojo-args="item,hint" >
  return {data: item, type: item.type,
    node: dojo.create("div", {innerHTML: item.text})};
</script>
```

La méthode `creator()` des classes `dojo.dnd.Source` et `dojo.dnd.Target`, est appelée pour générer le DOM servant à afficher un élément dans une source ou une cible. Lorsque son second paramètre à pour valeur `avatar`, alors il est appelée pour construire le DOM servant à afficher l'avatar. Le premier paramètre de la méthode contient l'objet inséré dans la source ou la cible à la création de l'élément déplaçable.

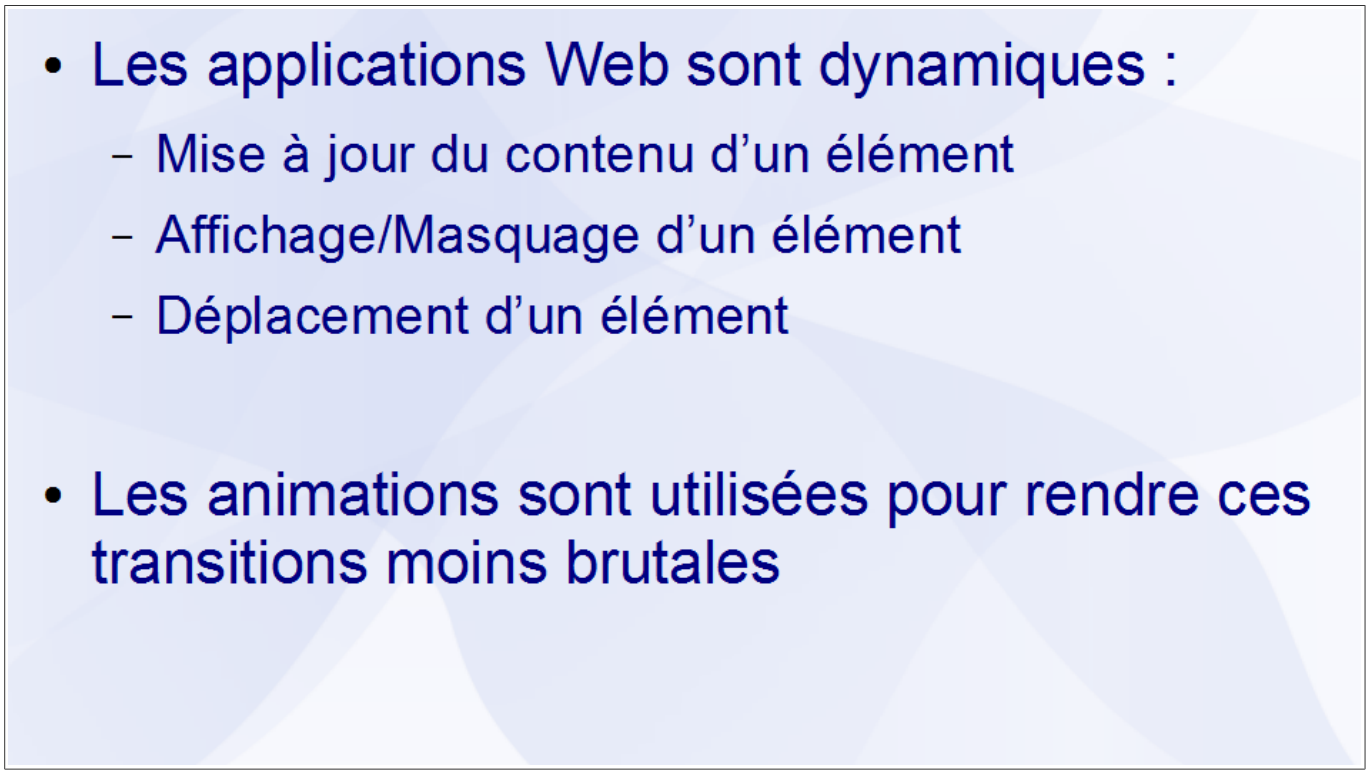
Conclusion

- La classe dojo.dnd.Moveable permet de rendre un élément déplaçable
- La classe dojo.dnd.Source permet de définir une source pour le glisser-déposer
- La classe dojo.dnd.Target permet de définir une cible pour le glisser-déposer
- Le glisser-déposer peut être implémenté en JavaScript ou avec la syntaxe déclarative

Chapitre 8 – Les animations

Sommaire

Introduction

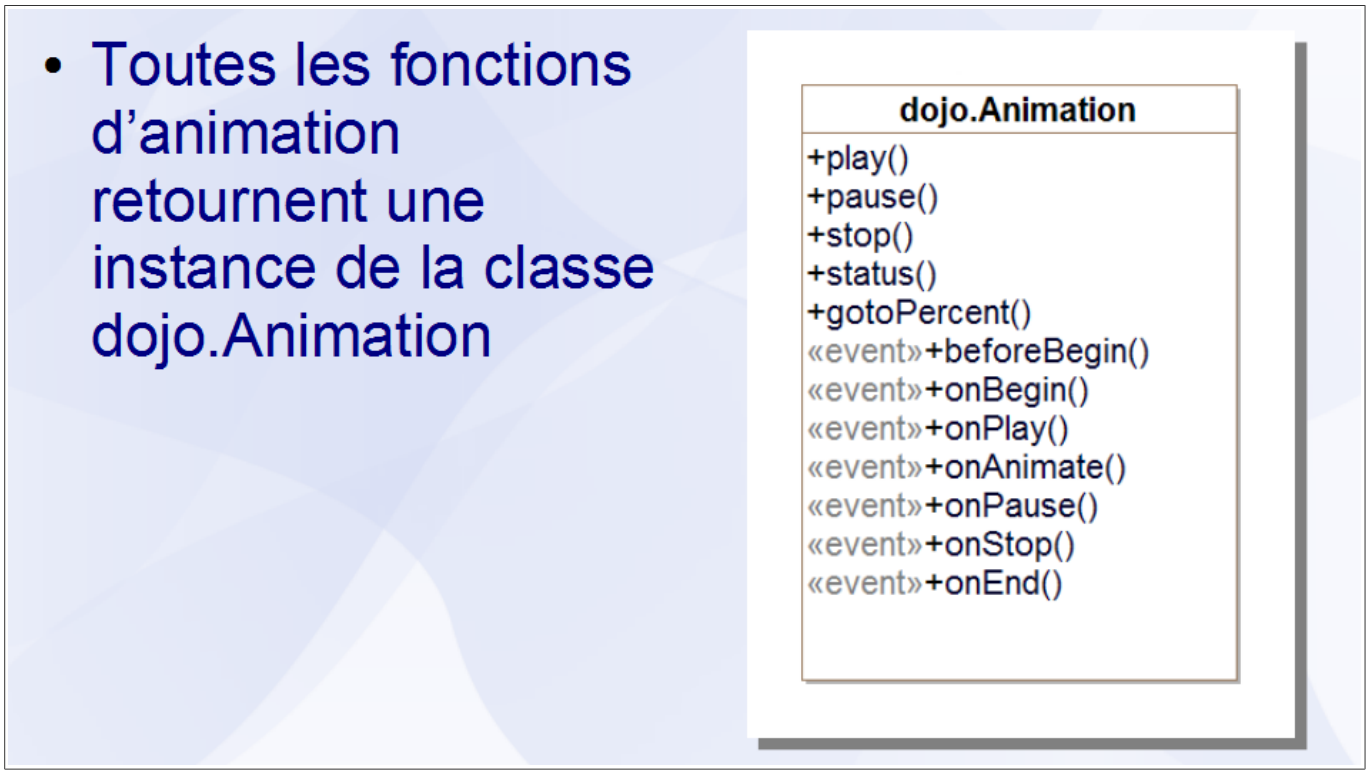


Dans une application Web traditionnelle, changer les informations présentées à l'utilisateur implique un rechargement complet du document.

Dans une application Web 2.0, grâce aux différentes techniques de communication avec un serveur et à la manipulation du DOM, le document est modifié dynamiquement mais ces changements instantanés peuvent perturber l'utilisateur. Ce dernier peut également ne pas remarquer qu'une information a été changée, il faut donc attirer son attention.

Les animations permettent d'améliorer l'ergonomie générale d'une application en atténuant ces problèmes.

La classe dojo.Animation



Une instance de la classe `dojo.Animation` est généralement obtenue en utilisant l'une des fonction d'animation. Le développeur peut interagir avec elle via les méthodes et les événements qu'elle expose.

La classe `dojo.Animation` définit les méthodes suivantes :

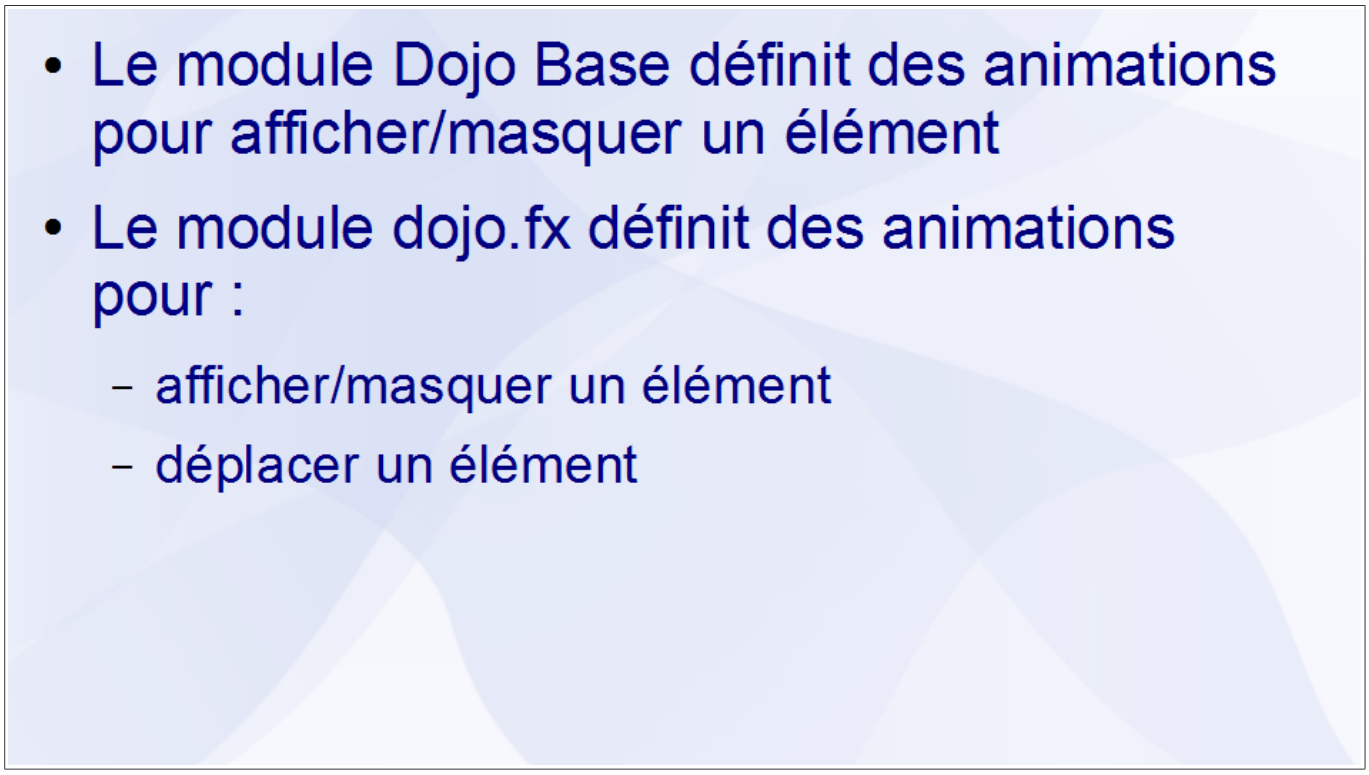
Méthode	**Description**
`play()`	Démarre l'animation.
`pause()`	Suspend l'animation.
`stop()`	Arrête l'animation.
`status`	Retourne l'état de l'animation : `paused`, `playing`, `stopped`.
`gotoPercent()`	Modifie l'état de la progression de l'animation.

Hormis `dojo.anim()`, les fonctions d'animation ne démarrent pas automatiquement les animations créées. Le développeur doit donc appeler la méthode `play()`.

La classe `dojo.Animation` définit les événements suivants :

Événement	Description
`beforeBegin()`	Événement synchrone déclenché avant le début de l'animation.
`onBegin()`	Événement déclenché au début de l'animation.
`onPlay()`	Événement déclenché au démarrage de l'animation.
`onAnimate()`	Événement déclenché à chaque étape de l'animation.
`onPause()`	Événement déclenché à la suspension de l'animation.
`onStop()`	Événement déclenché à l'arrêt de l'animation.
`onEnd()`	Événement déclenché à la fin de l'animation.

Les animations prédéfinies



Le module `dojox.fx` définit des animations complémentaires :

Méthode	Description
`dojox.fx.fadeTo()`	Rend un élément partiellement visible/invisible en modifiant son niveau de transparence.
`dojox.fx.crossFade()`	Rend un premier élément visible et un second élément invisible en modifiant leurs niveaux de transparence.
`dojox.fx.wipeTo()`	Rend un élément partiellement visible/invisible en modifiant sa hauteur.
`dojox.fx.slideBy()`	Déplace un élément relativement à sa position actuelle.
`dojox.fx.sizeTo()`	Redimensionne un élément aux dimensions spécifiées.
`dojox.fx.highlight()`	Met un élément en évidence.

Le module `dojox.fx` doit être importé avant de pouvoir utiliser ces fonctions.

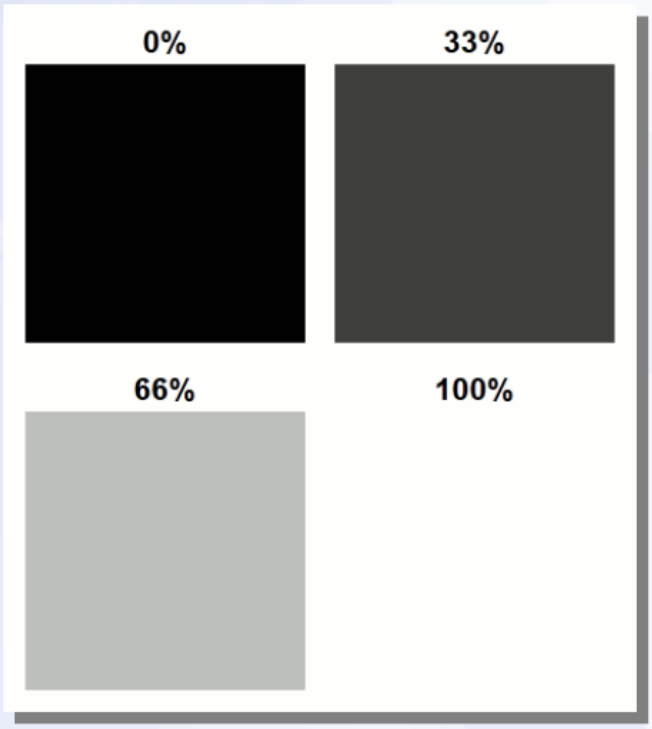
Animation du niveau de transparence

Rendre un élément visible :

```
dojo.fadeIn({
  node: "carre",
  duration: 1000
}).play();
```

Rendre un élément invisible :

```
dojo.fadeOut({
  node: "carre",
  duration: 1000
}).play();
```



Les fonctions `dojo.fadeIn()` et `dojo.fadeOut()` rendent un élément respectivement visible et invisible en modifiant son niveau de transparence. Elles acceptent en paramètre un objet avec les propriétés suivantes :

Propriété	Type	Description
`node`	`String`	Identifiant de l'élément à animer.
`duration`	`Number`	(optionnel) Durée de l'animation en millisecondes.
`easing`	`Function`	(optionnel) Fonction d'adoucissement à utiliser.

Les fonctions `dojo.fadeIn()` et `dojo.fadeOut()` retournent une instance de `dojo.Animation`.

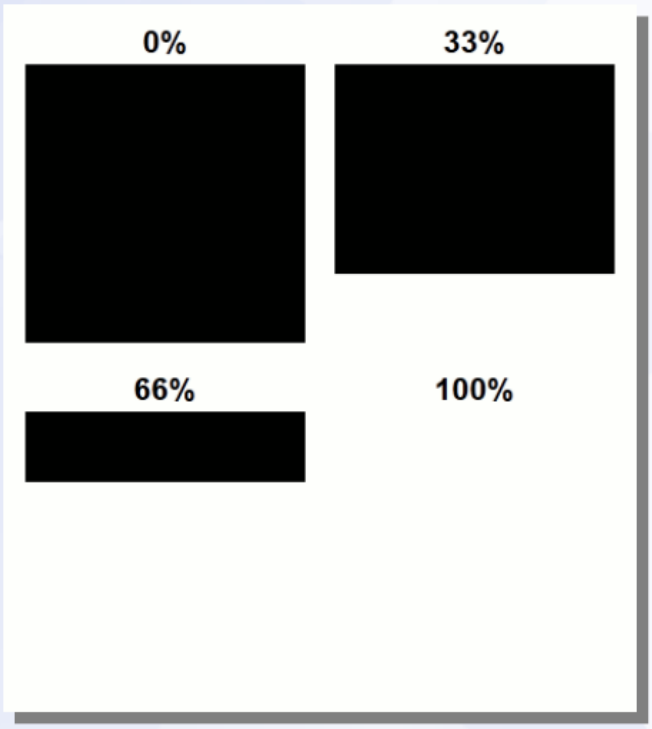
Animation de la hauteur

Rendre un élément visible :

```
dojo.fx.wipeIn({
  node: "carre",
  duration: 1000
}).play();
```

Rendre un élément invisible :

```
dojo.fx.wipeOut({
  node: "carre",
  duration: 1000
}).play();
```



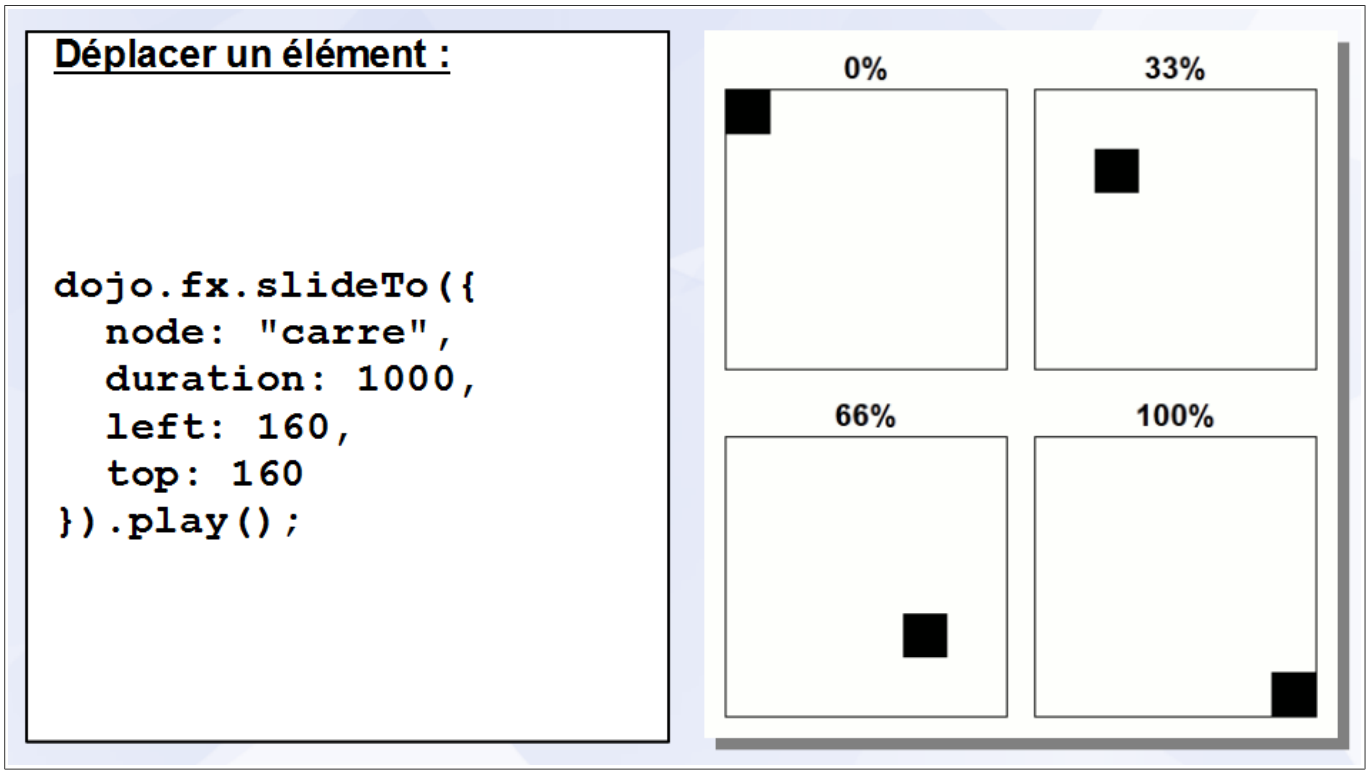
Les fonctions `dojo.fx.wipeIn()` et `dojo.fx.wipeOut()` rendent un élément respectivement visible et invisible en modifiant sa hauteur. Elles acceptent en paramètre un objet avec les propriétés suivantes :

Propriété	Type	Description
`node`	`String`	Identifiant de l'élément à animer.
`duration`	`Number`	(optionnel) Durée de l'animation en millisecondes.
`easing`	`Function`	(optionnel) Fonction d'adoucissement à utiliser.

Les fonctions `dojo.fx.wipeIn()` et `dojo.fx.wipeOut()` retournent une instance de `dojo.Animation`.

Le module `dojo.fx` doit être importé avant de pouvoir utiliser ces fonctions.

Animation du déplacement



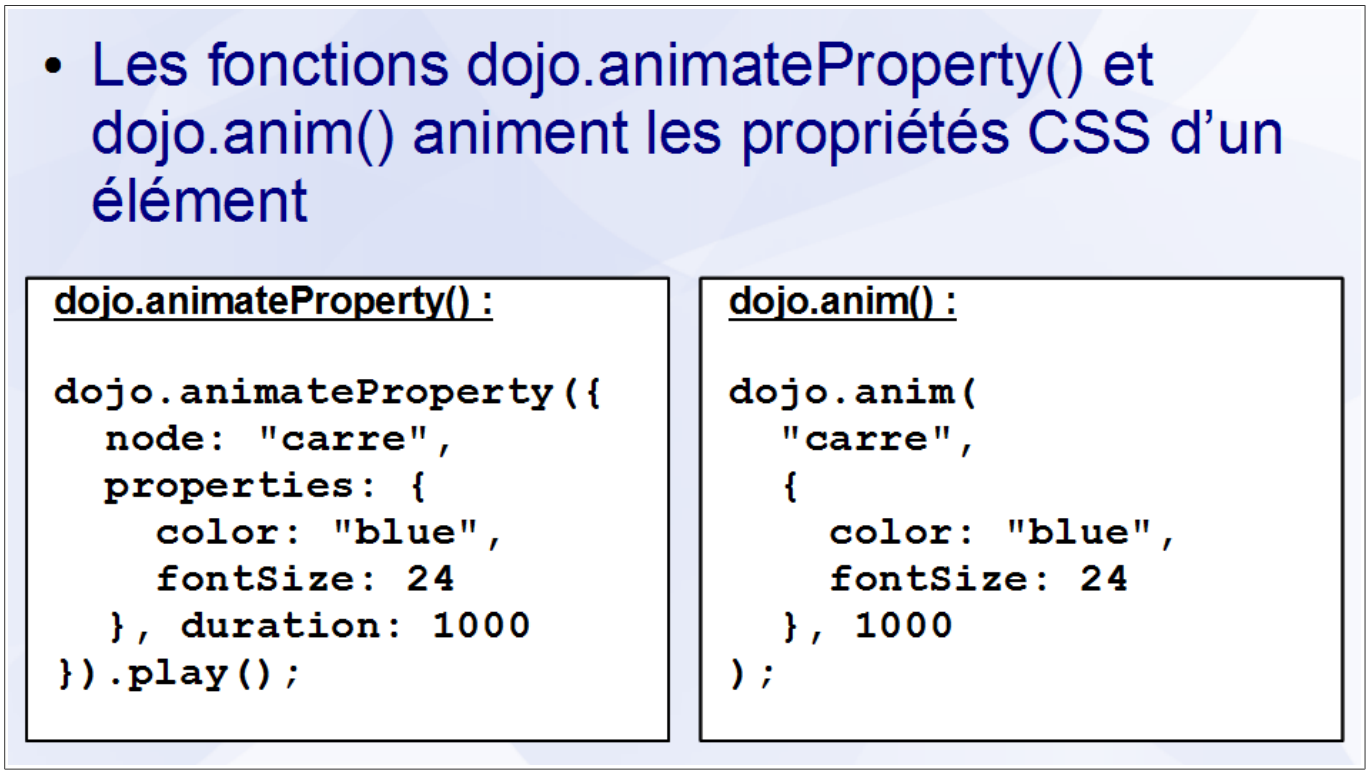
La fonction `dojo.fx.slideTo()` déplace un élément en modifiant ses coordonnées. Elle accepte en paramètre un objet avec les propriétés suivantes :

Propriété	Type	Description
`node`	`String`	Identifiant de l'élément à animer.
`duration`	`Number`	(optionnel) Durée de l'animation en millisecondes.
`easing`	`Function`	(optionnel) Fonction d'adoucissement à utiliser.
`top`	`Number`	(optionnel) Valeur de la propriété CSS `top` de l'élément à la fin de l'animation.
`left`	`Number`	(optionnel) Valeur de la propriété CSS `left` de l'élément à la fin de l'animation.

La fonction `dojo.fx.slideTo()` retourne une instance de `dojo.Animation`.

Le module `dojo.fx` doit être importé avant de pouvoir utiliser cette fonction.

Les animations personnalisées



`dojo.animateProperty()` est la fonction centrale pour la création des animations. Elle permet d'animer toutes les propriétés CSS utilisant une valeur numérique ou une couleur. La plupart des fonctions créant des animations, y compris `dojo.anim()`, font appel à elle.

Les différences entre `dojo.animateProperty()` et `dojo.anim()` sont les suivantes :

- `dojo.anim()` utilise des paramètres positionnels tandis que `dojo.animateProperty()` accepte en paramètre un objet avec toutes les propriétés de configuration de l'animation
- Les animations créées par `dojo.anim()` sont démarrées automatiquement

La fonction `dojo.animateProperty()` accepte en paramètre un objet avec les propriétés suivantes :

Propriété	Type	Description
`node`	`String`	Identifiant de l'élément à animer.
`properties`	`Object`	Propriétés CSS à animer.
`duration`	`Number`	(optionnel) Durée de l'animation en millisecondes.
`easing`	`Function`	(optionnel) Fonction d'adoucissement à utiliser.

Pour `dojo.anim()`, il faut passer ces même propriétés dans des paramètres individuels et dans le même ordre que le tableau.

Les fonctions `dojo.animateProperty()` et `dojo.anim()` retournent une instance de `dojo.Animation`.

L'objet passé à la propriété `properties` doit avoir la structure suivante :

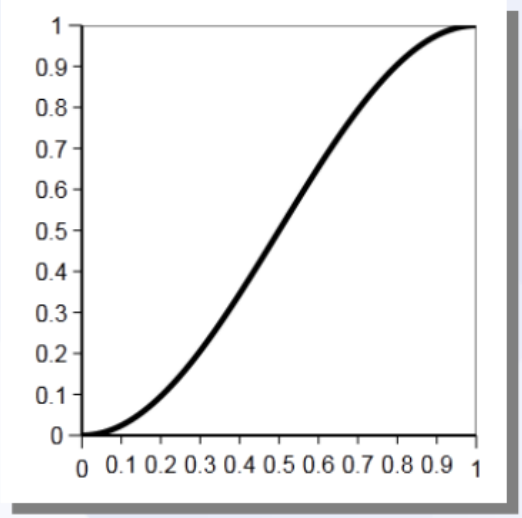
```
{
  // Chaque entrée de l'objet est le nom d'une propriété CSS

  width: 50, // Valeur de width à la fin de l'animation
             // Unité par défaut : px

  height: {
    start: 10,  // Valeur de height au début de l'animation
    end: 50,    // Valeur de height à la fin de l'animation
    units: "px" // Unité à utiliser
  }
}
```

Les fonctions d'adoucissement

- Elles contrôlent le rythme de l'animation
- dojo.fx.easing définit 31 de ces fonctions



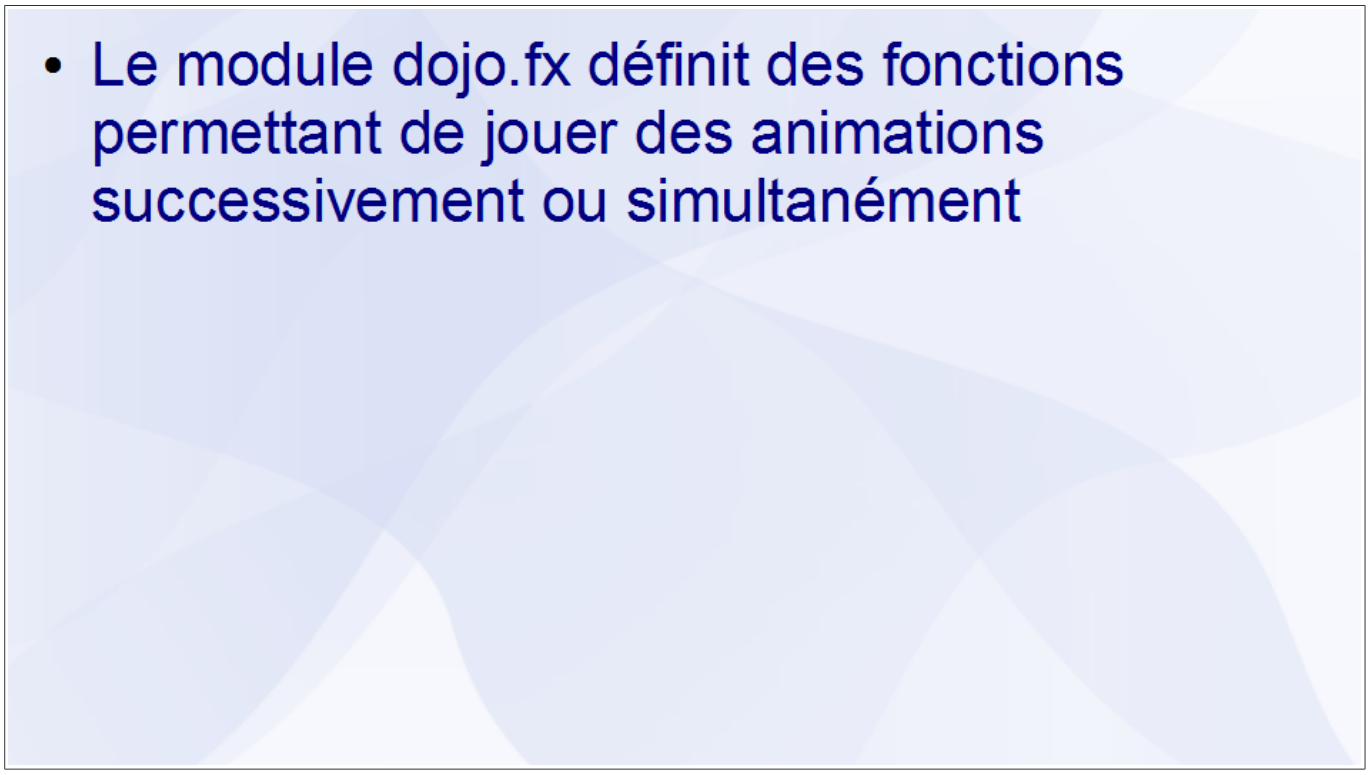
Le module `dojo.fx.easing` doit être importé avant de pouvoir utiliser les fonctions d'adoucissement.

Une fonction d'adoucissement est définie sur le domaine `[0,1]`, `0` et `1` représentant respectivement le début et la fin de l'animation.

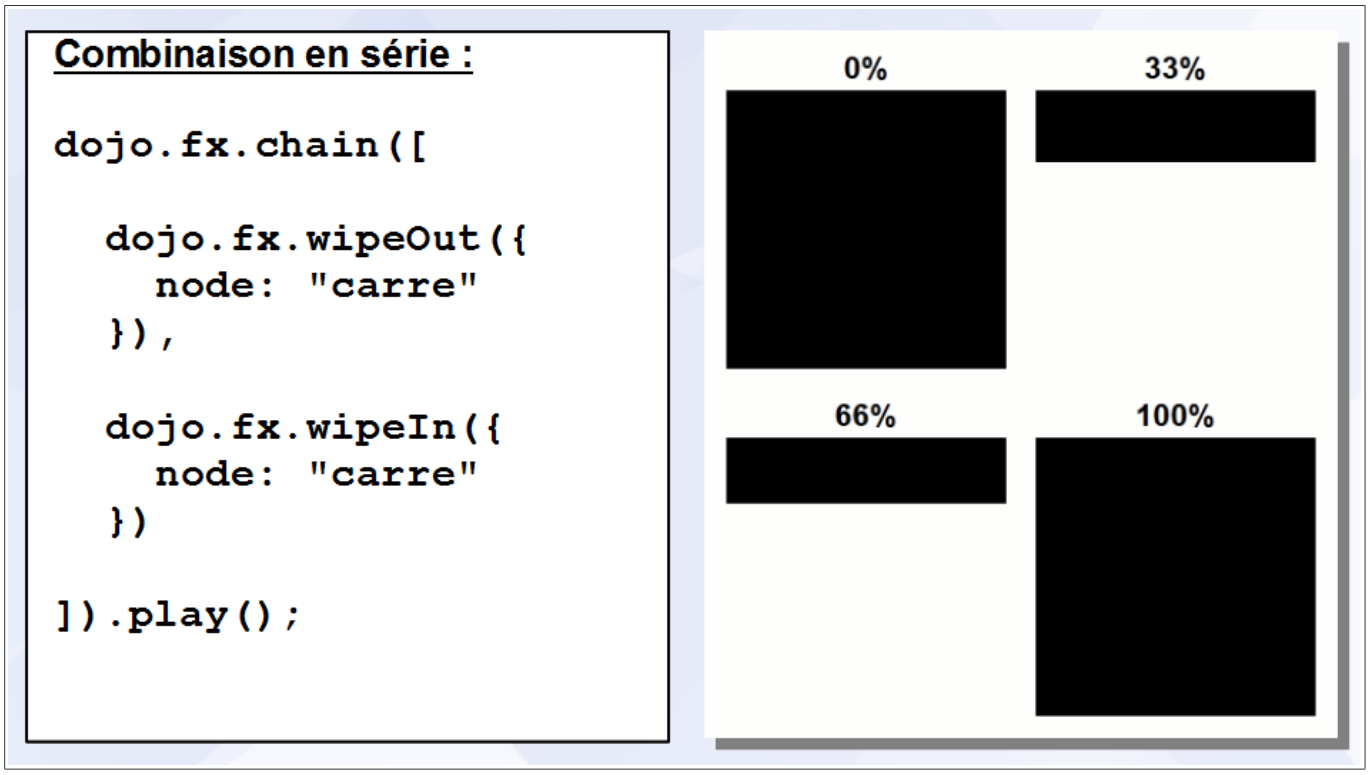
La valeur retournée représente la progression de la propriété CSS à animer, `0` et `1` représentant respectivement la valeur de la propriété CSS au début et à la fin de l'animation. Cette valeur peut être inférieure à `0` ou supérieure à `1`.

La liste des fonctions d'adoucissement est fournie en annexe.

Combinaison des animations



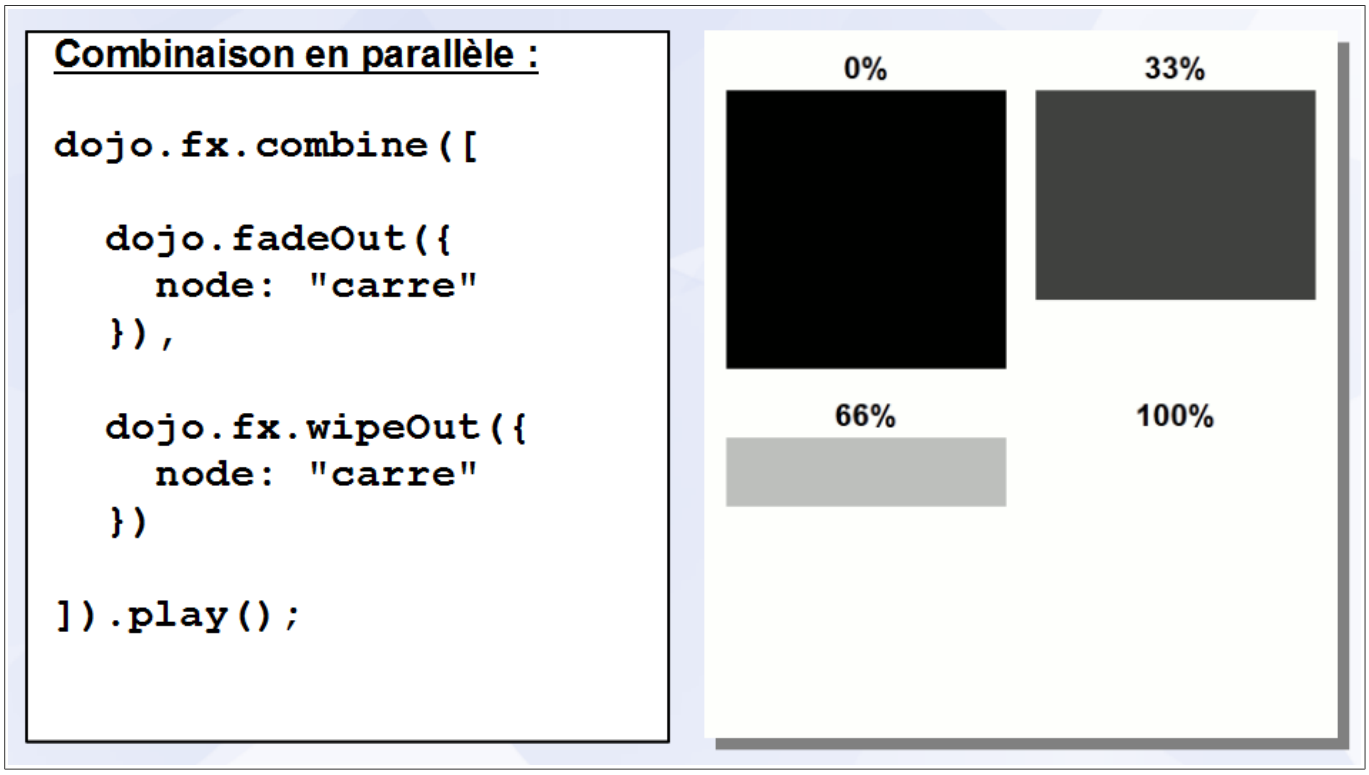
Combinaison en série



La fonction `dojo.fx.chain()` exécute plusieurs animations successivement. Elle accepte en paramètre un tableau d'instances de `dojo.Animation` et retourne une instance de `dojo.Animation` qui peut, à son tour, être combinée avec d'autres animations.

Le module `dojo.fx` doit être importé avant de pouvoir utiliser cette fonction.

Combinaison en parallèle



La fonction `dojo.fx.combine()` exécute plusieurs animations simultanément. Elle accepte en paramètre un tableau d'instances de `dojo.Animation` et retourne une instance de `dojo.Animation` qui peut, à son tour, être combinée avec d'autres animations.

Le module `dojo.fx` doit être importé avant de pouvoir utiliser cette fonction.

Conclusion

- La classe dojo.Animation est au cœur du système d'animation de Dojo Toolkit
- La fonction dojo.animateProperty() permet d'animer la plupart des propriétés CSS
- Les fonctions d'adoucissement contrôlent le rythme des animations
- dojo.chain() et dojo.combine() permettent de jouer plusieurs animations successivement ou simultanément

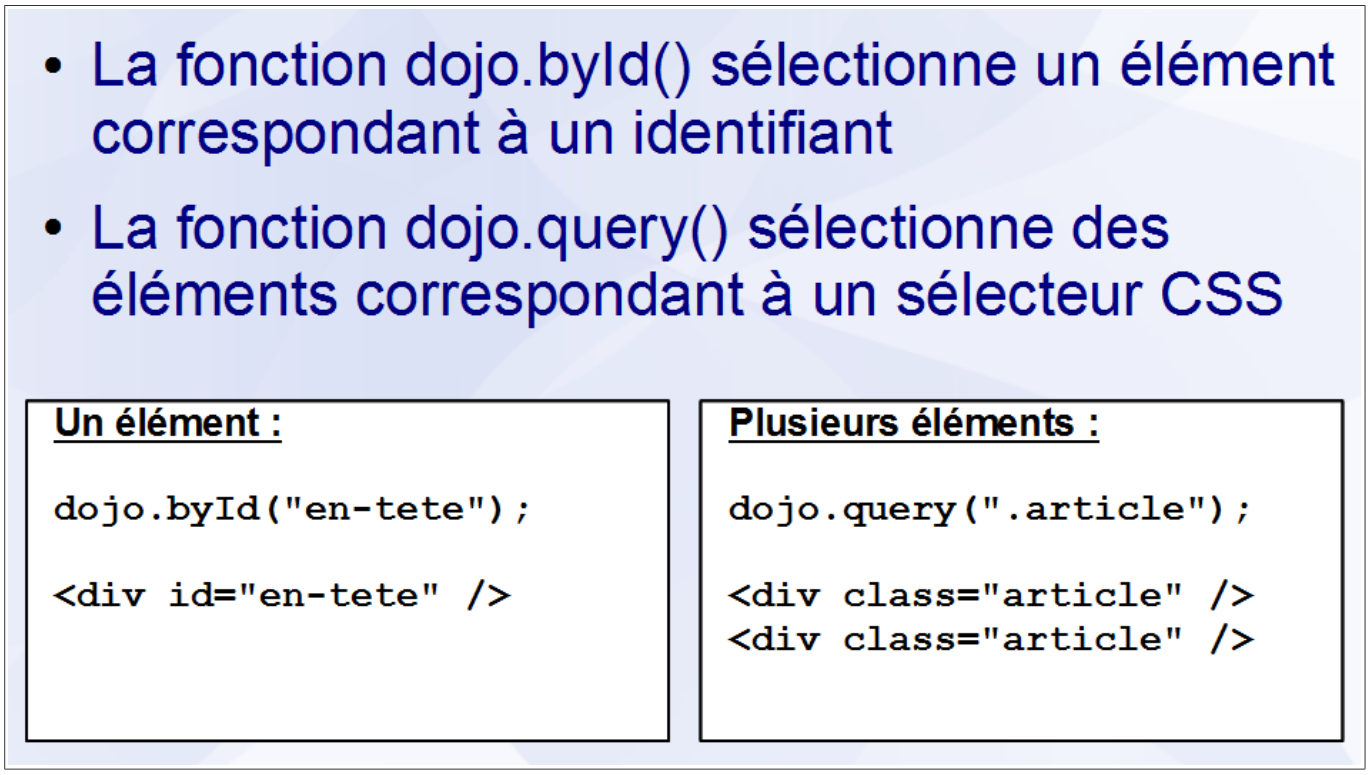
Chapitre 9 – Le Document Object Model

Sommaire

Introduction

- Le Document Object Model (DOM) est :
  - Une interface de programmation
  - Indépendante des plates-formes et des langages
  - Créée par le World Wide Web Consortium (W3C)
  - Pour manipuler les documents HTML et XML
- Dojo Toolkit dispose de plusieurs fonctions facilitant la manipulation du DOM

Sélection des éléments



La fonction `dojo.byId()` sélectionne un élément correspondant à un identifiant. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`id`	`String` `Element`	Identifiant de l'élément à rechercher ou un élément.
`doc`	`Document`	(optionnel) Document dans lequel effectué la recherche.

La fonction `dojo.byId()` retourne l'élément ayant l'identifiant recherché s'il est présent dans le document ou `null` dans le cas contraire. Si un élément est passé à la fonction, il est retourné immédiatement.

La fonction `dojo.byId()` remplace la fonction `document.getElementById()` du DOM standard.

La fonction `dojo.query()` sélectionne les éléments correspondant à un sélecteur CSS. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`query`	`String`	Sélecteur CSS à utiliser pour la recherche.
`root`	`Element`	(optionnel) Élément à partir duquel effectué la recherche.

La fonction `dojo.query()` retourne une instance de la classe `dojo.NodeList` contenant les éléments correspondant au sélecteur CSS.

La classe `dojo.NodeList` représente une collection de nœuds du DOM. Ses nombreuses méthodes permettent d'appliquer une même opération à tous les éléments de la collection. La plupart d'entre elles retournant une collection, il est possible de les chaîner.

Par exemple :

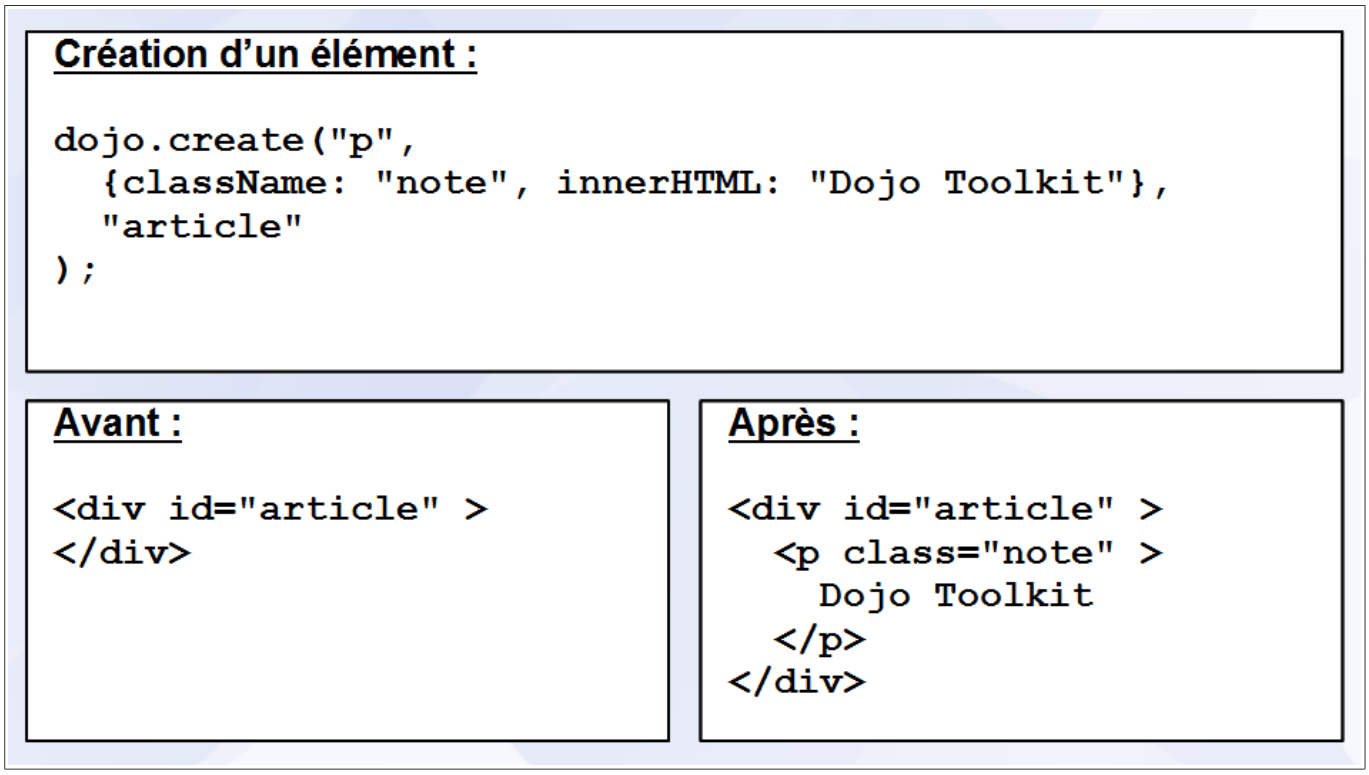
```
dojo.query("div.article")  // Sélection des éléments
    .addClass("nouveau")   // Ajout d'une classe aux éléments
    .fadeIn() ;            // Animation des éléments
```

La liste des sélecteurs CSS et la liste des méthodes de `dojo.NodeList` sont fournies en annexe.

Manipulation des éléments

- Les éléments sont les nœuds les plus courants dans un document HTML
- Un élément est délimité par une balise d'ouverture et une balise de fermeture entre lesquelles se trouve son contenu
- Le module Dojo Base dispose de plusieurs fonctions pour créer, modifier et supprimer des éléments

Création d'un élément

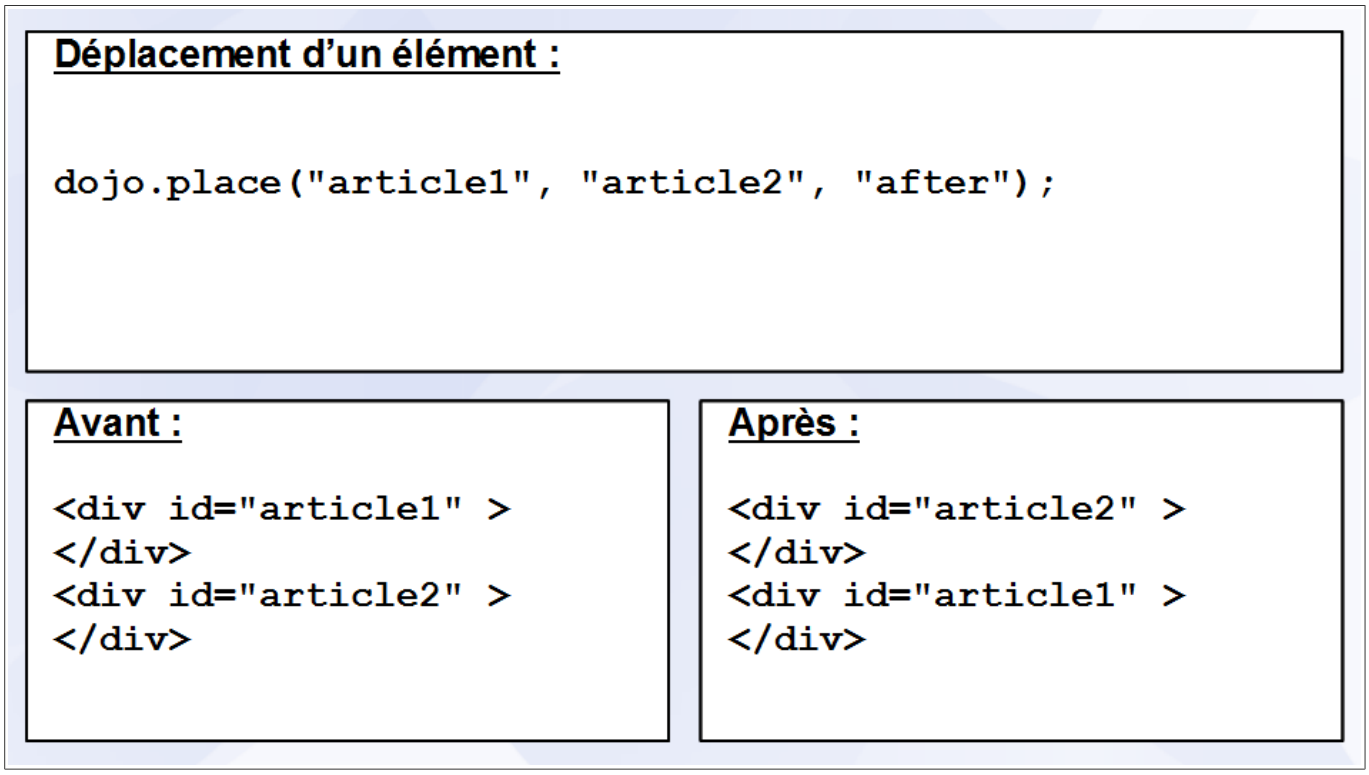


La fonction `dojo.create()` crée un élément et l'ajoute au document. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`tag`	`String`	Nom de l'élément à créer.
`attrs`	`Object`	(optionnel) Tableau associatif des attributs à ajouter à l'élément.
`refNode`	`String`	(optionnel) Élément de contexte pour l'ajout au document.
`pos`	`String`	(optionnel) Position par rapport à l'élément de contexte : • `first` – premier enfant de l'élément de contexte • `last` – dernier enfant de l'élément de contexte • `only` – enfant unique de l'élément de contexte • `before` – frère précédent de l'élément de contexte • `after` – frère suivant de l'élément de contexte • `replace` – remplace l'élément de contexte

La fonction `dojo.create()` retourne l'élément créé.

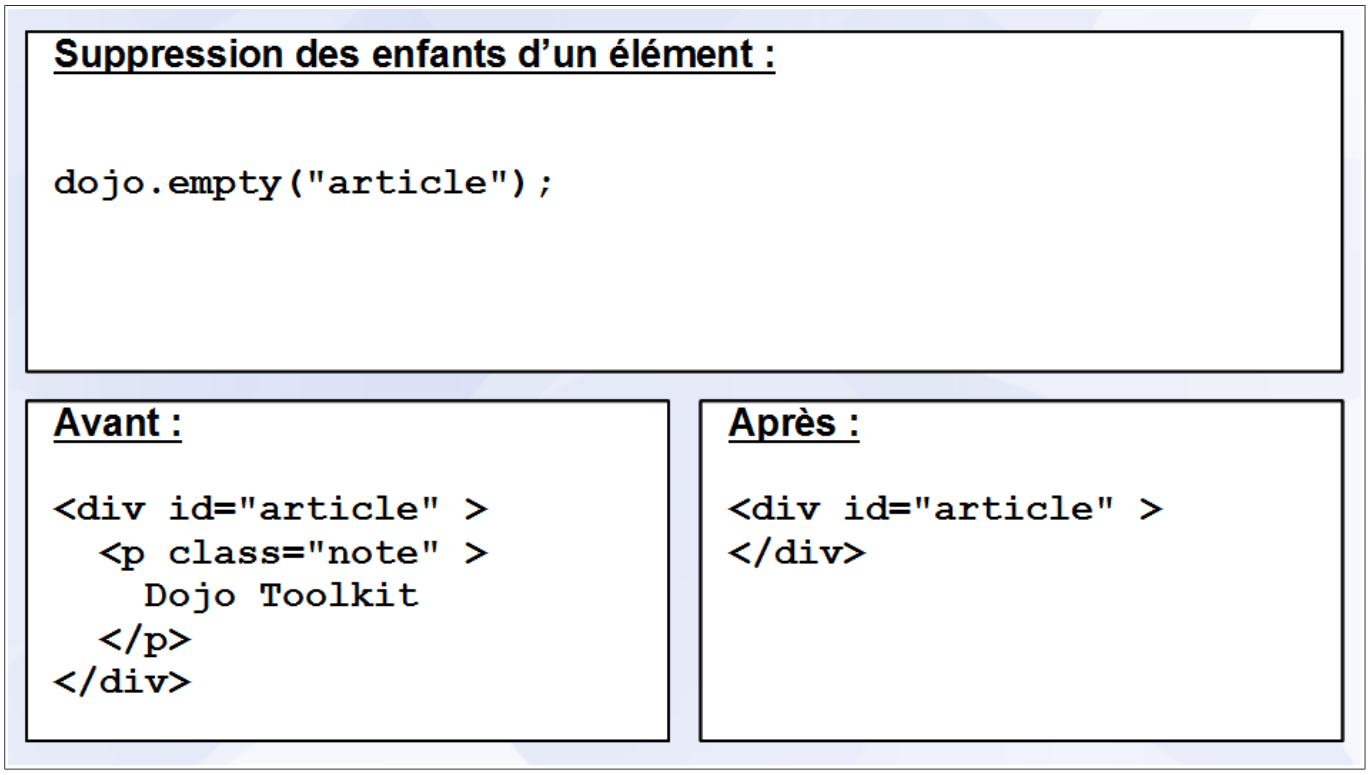
Déplacement d'un élément



La fonction `dojo.place()` déplace un élément dans le document. Elle accepte les paramètres suivants :

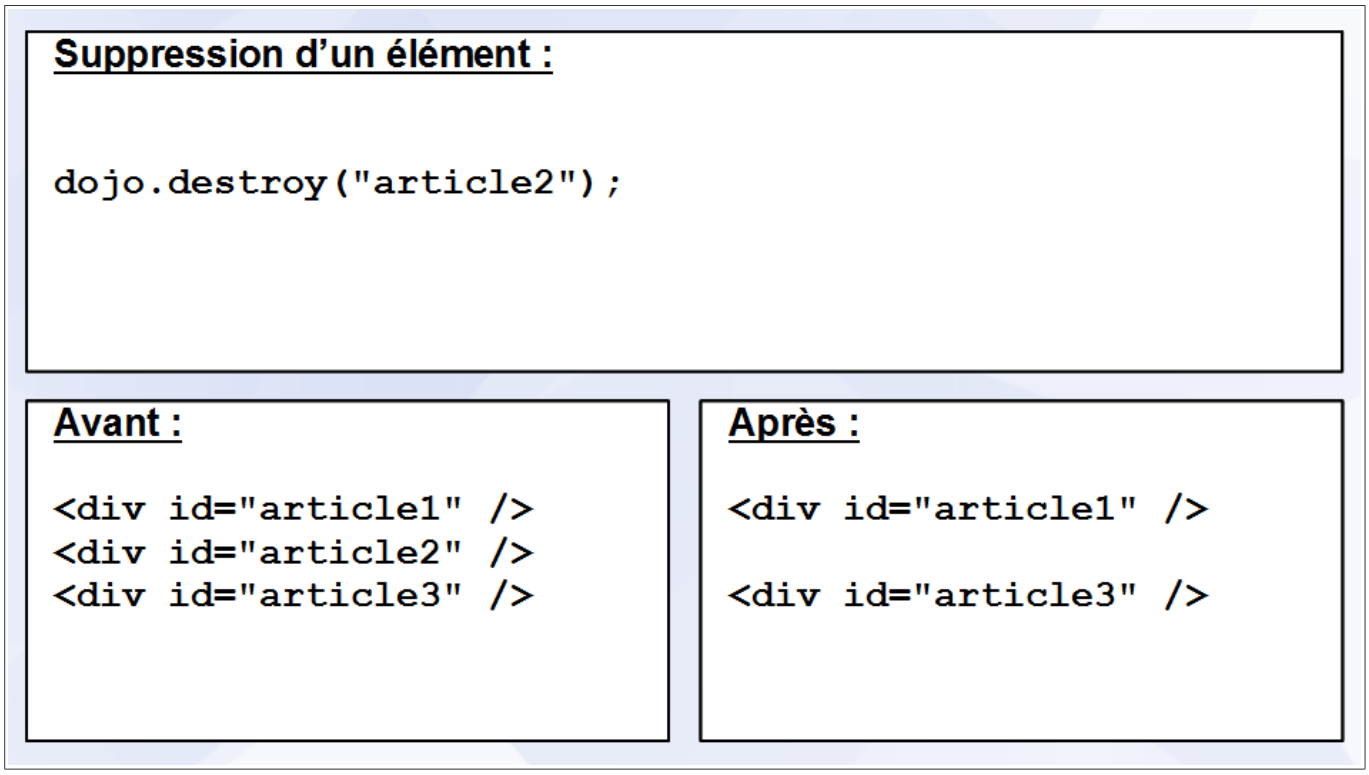
Paramètre	Type	Description
`node`	`String`	Identifiant de l'élément à déplacer.
`refNode`	`String`	(optionnel) Élément de contexte pour le déplacement dans le document.
`position`	`String`	(optionnel) Position par rapport à l'élément de contexte : • `first` – premier enfant de l'élément de contexte • `last` – dernier enfant de l'élément de contexte • `only` – enfant unique de l'élément de contexte • `before` – frère précédent de l'élément de contexte • `after` – frère suivant de l'élément de contexte • `replace` – remplace l'élément de contexte

Suppression des enfants d'un élément



La fonction `dojo.empty()` accepte en paramètre l'identifiant de l'élément dont il faut supprimer les enfants.

Suppression d'un élément

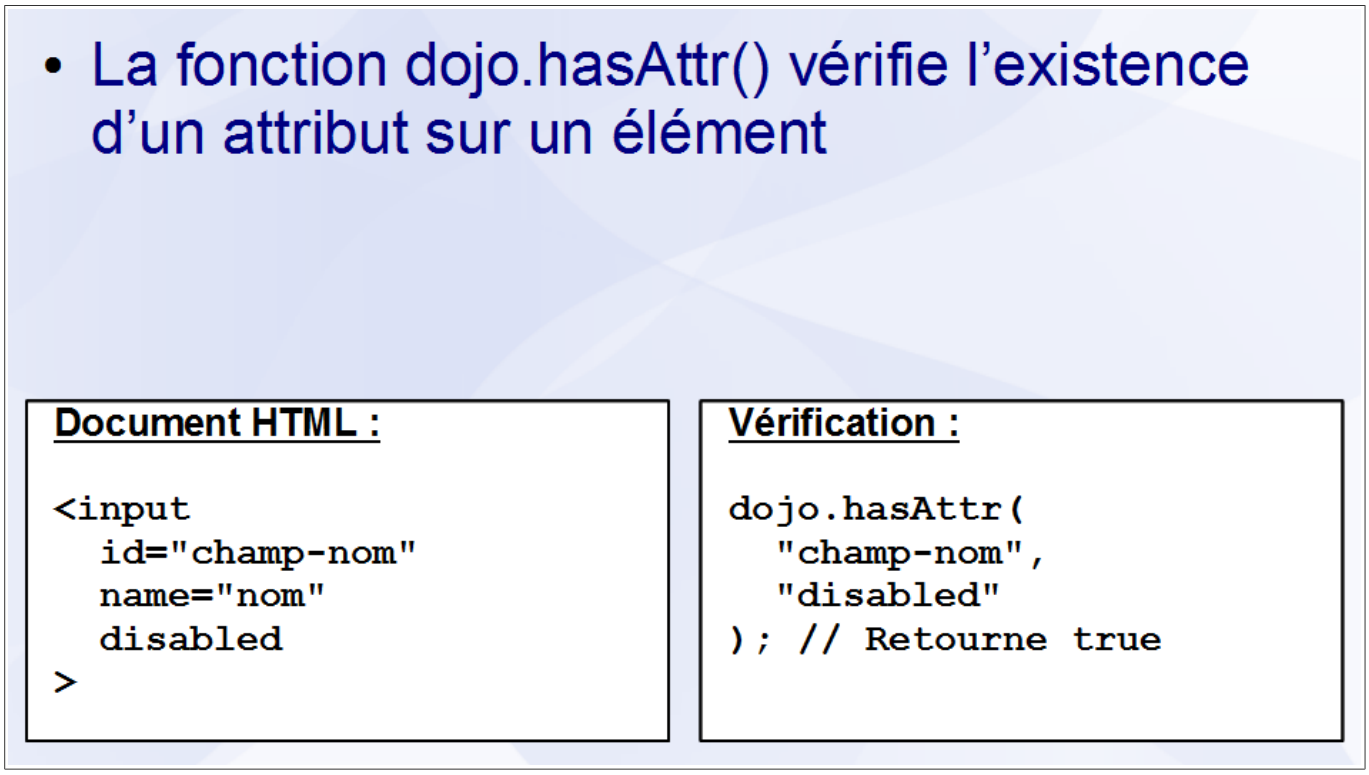


La fonction `dojo.destroy()` accepte en paramètre l'identifiant de l'élément à supprimer.

Manipulation des attributs

- La balise d'ouverture d'un élément peut contenir des attributs
- Un attribut est l'association d'un nom et d'une valeur
- Le module Dojo Base dispose de plusieurs fonctions pour récupérer, créer, modifier et supprimer les attributs d'un élément

Vérification de l'existence d'un attribut

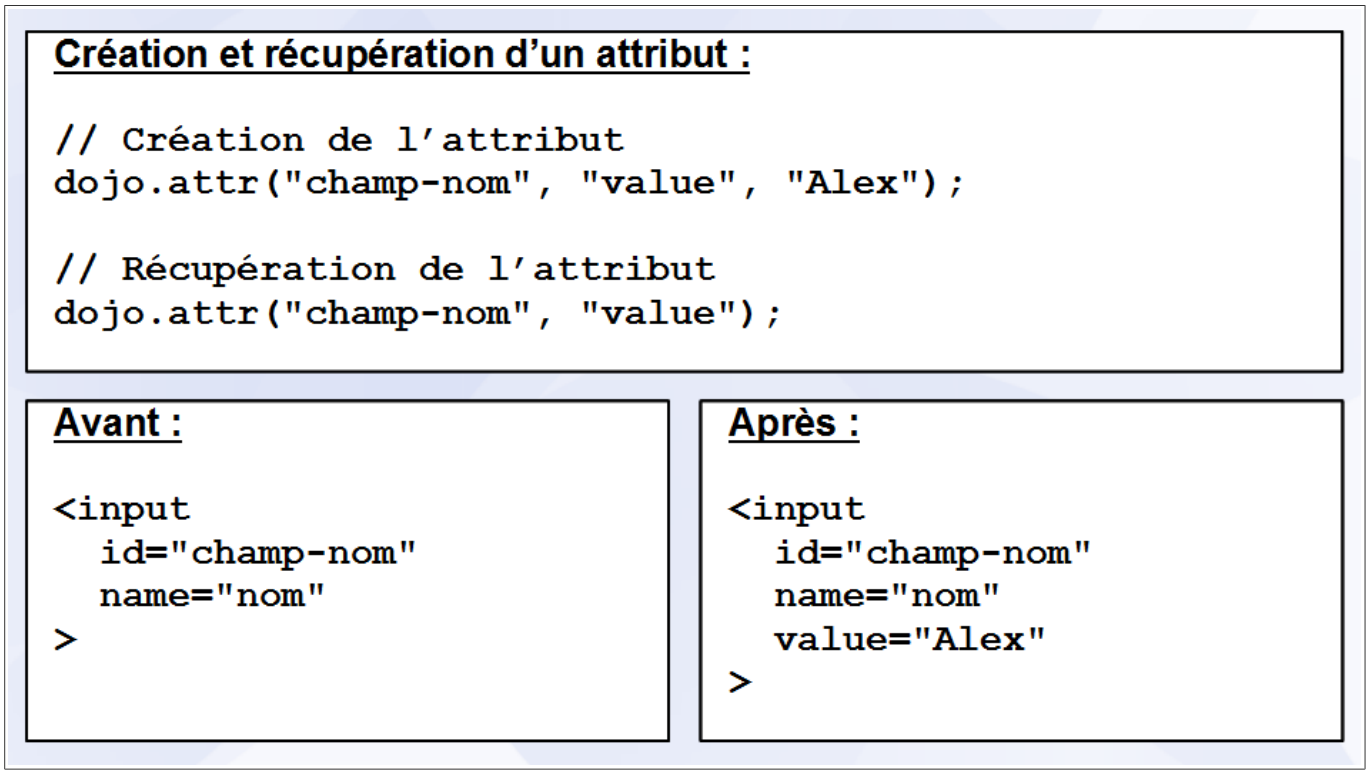


La fonction `dojo.hasAttr()` vérifie l'existence d'un attribut sur un élément. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`node`	`String`	Identifiant de l'élément.
`name`	`String`	Nom de l'attribut à rechercher.

La fonction `dojo.hasAttr()` retourne `true` si l'attribut existe ou `false` dans le cas contraire.

Récupération, création et modification d'un attribut



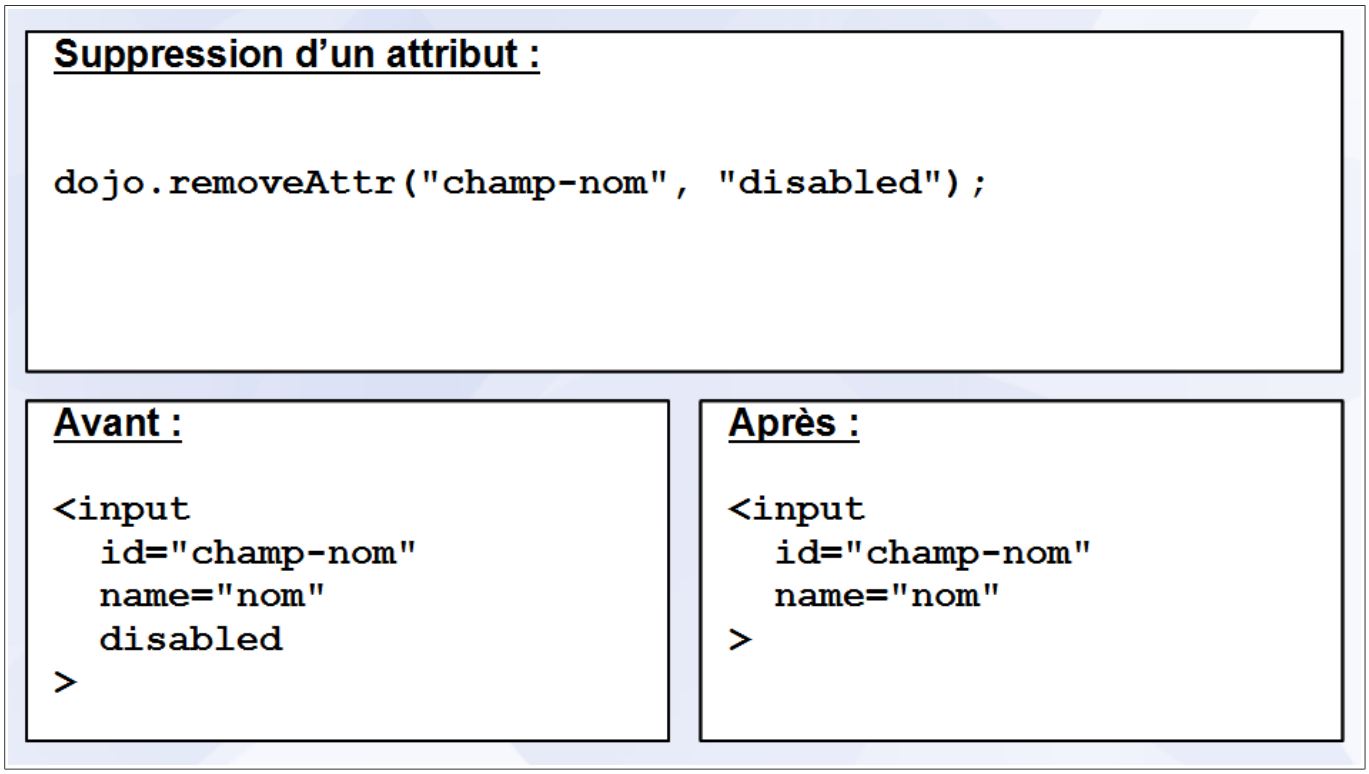
La fonction `dojo.attr()` récupère, crée ou modifie la valeur d'un attribut. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`node`	`String`	Identifiant de l'élément.
`name`	`String`	Nom de l'attribut.
`value`	`String`	(optionnel) Valeur de l'attribut.

Il est possible de passer un objet en second paramètre pour créer ou modifier plusieurs attributs à la fois :

```
dojo.attr("champ-nom", { value: "Alex", type: "hidden" }) ;
```

Suppression d'un attribut



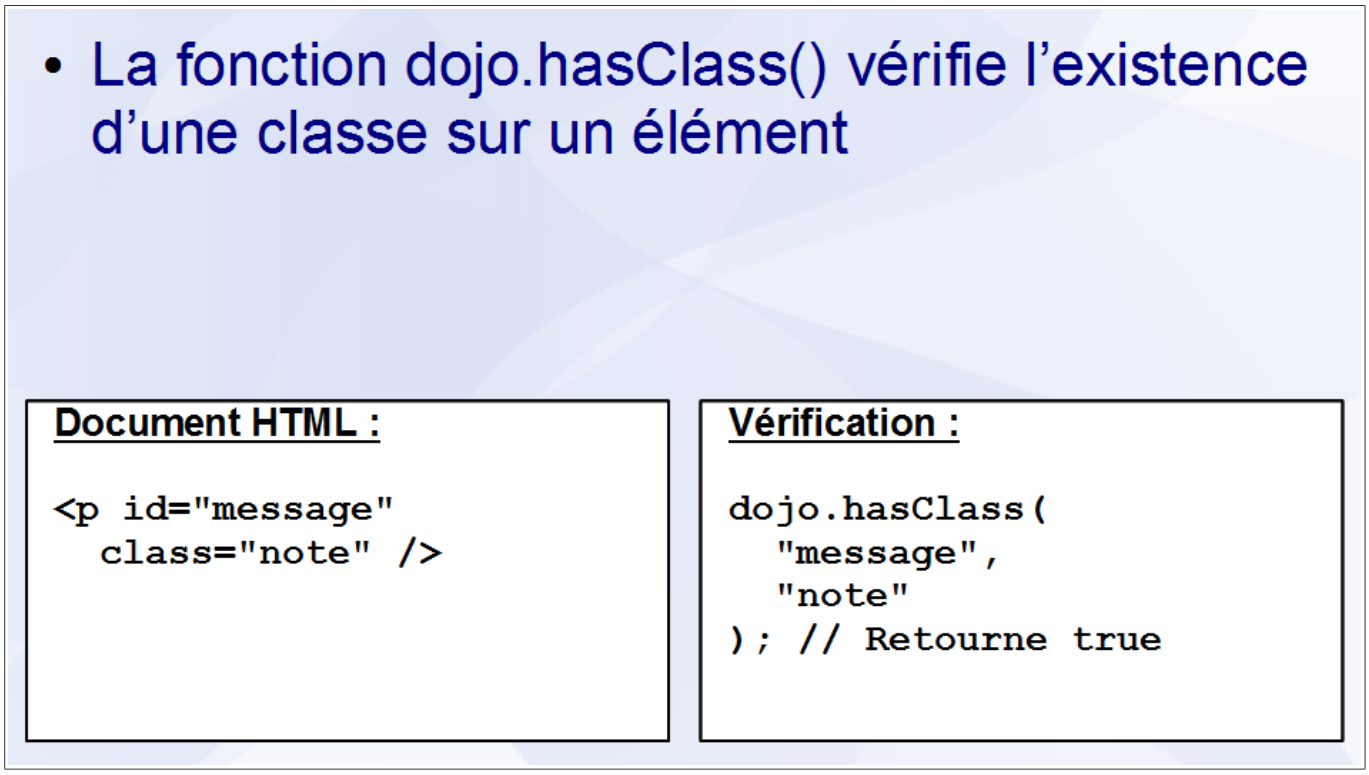
La fonction `dojo.removeAttr()` supprime un attribut. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`node`	`String`	Identifiant de l'élément.
`name`	`String`	Nom de l'attribut à supprimer.

Manipulation des classes

- Les éléments d'un document HTML dispose d'un attribut class
- La valeur de cet attribut est utilisée dans les feuilles de styles CSS pour spécifier la mise en page d'un groupe d'éléments
- Le module Dojo Base dispose de plusieurs fonctions pour récupérer, ajouter et supprimer des classes à un élément

Vérification de l'existence d'une classe

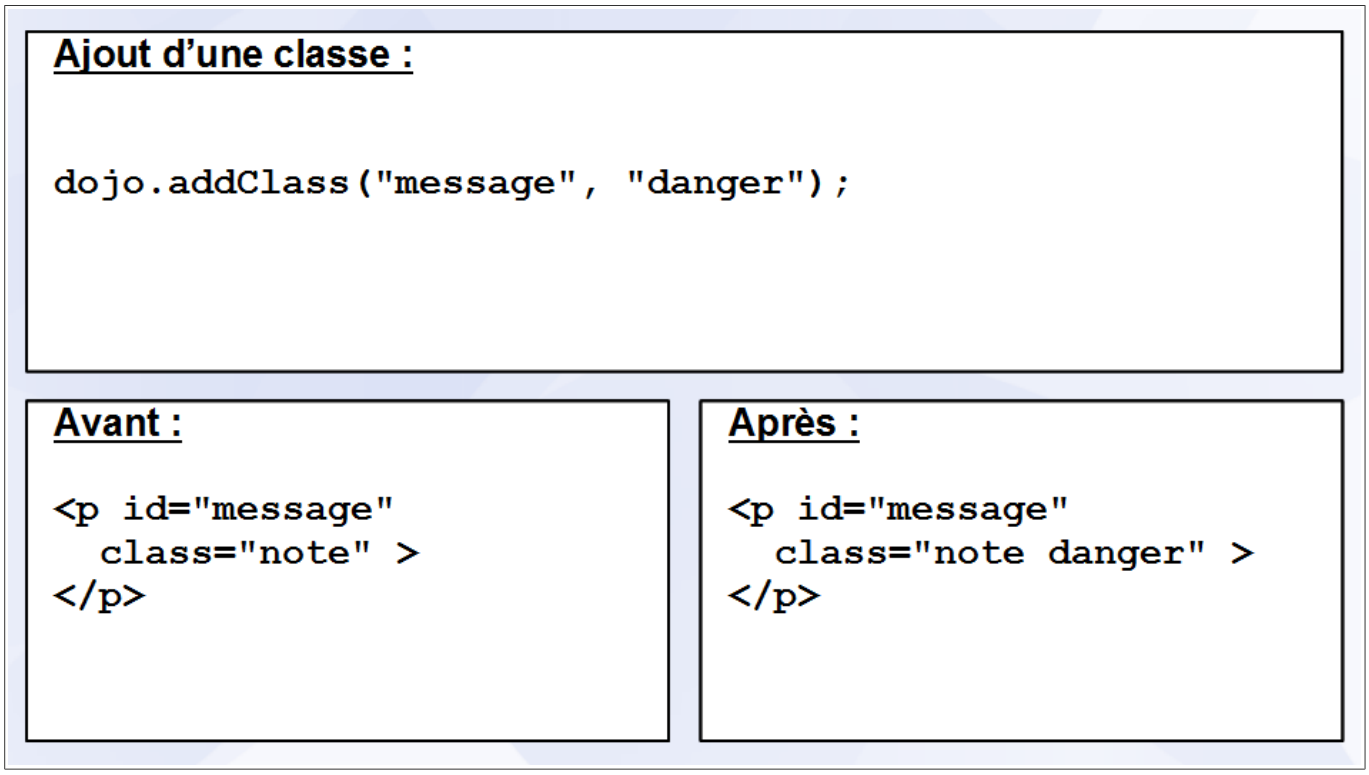


La fonction `dojo.hasClass()` vérifie l'existence d'une classe sur un élément. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`node`	`String`	Identifiant de l'élément.
`classStr`	`String`	Nom de la classe à rechercher.

La fonction `dojo.hasClass()` retourne `true` si la classe existe ou `false` dans le cas contraire.

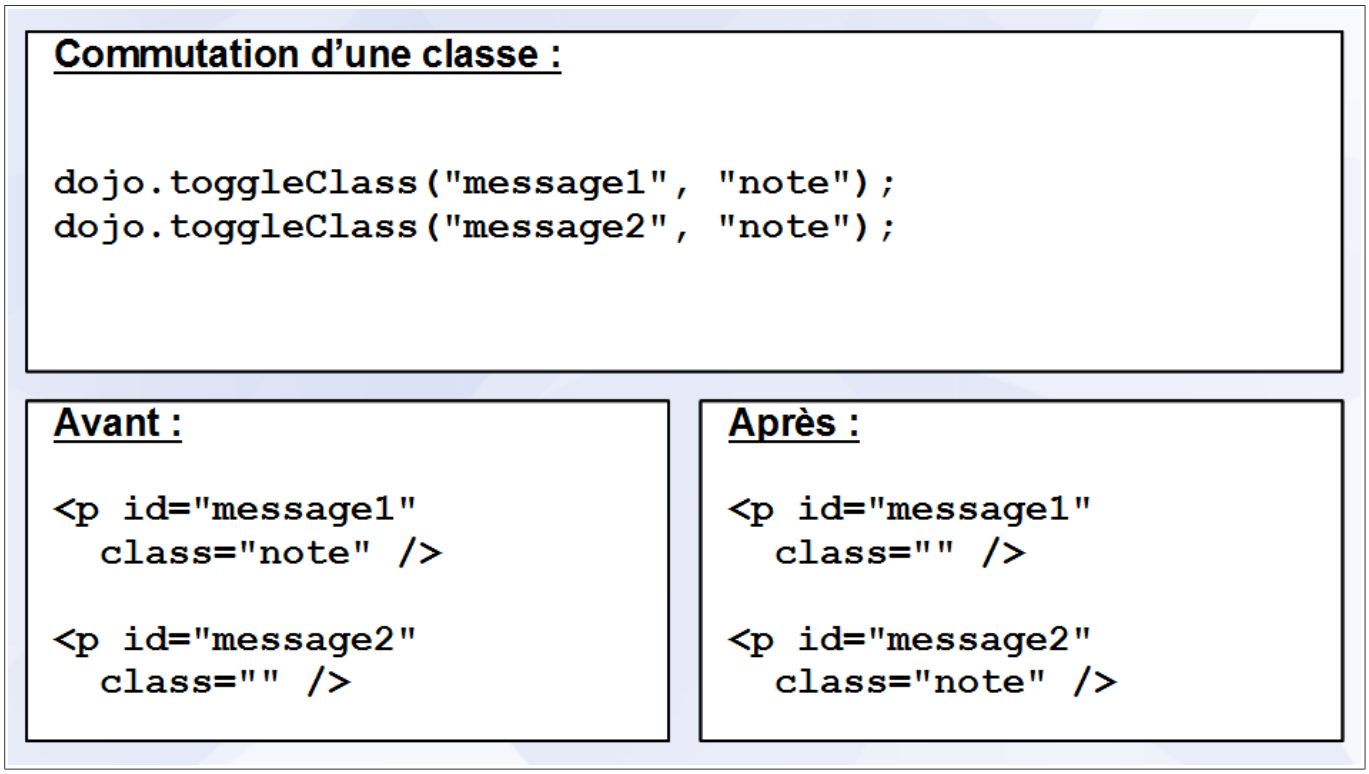
Ajout d'une classe



La fonction `dojo.addClass()` ajoute une ou plusieurs classes à un élément. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`node`	`String`	Identifiant de l'élément.
`classStr`	`String`	Liste de classes à ajouter.

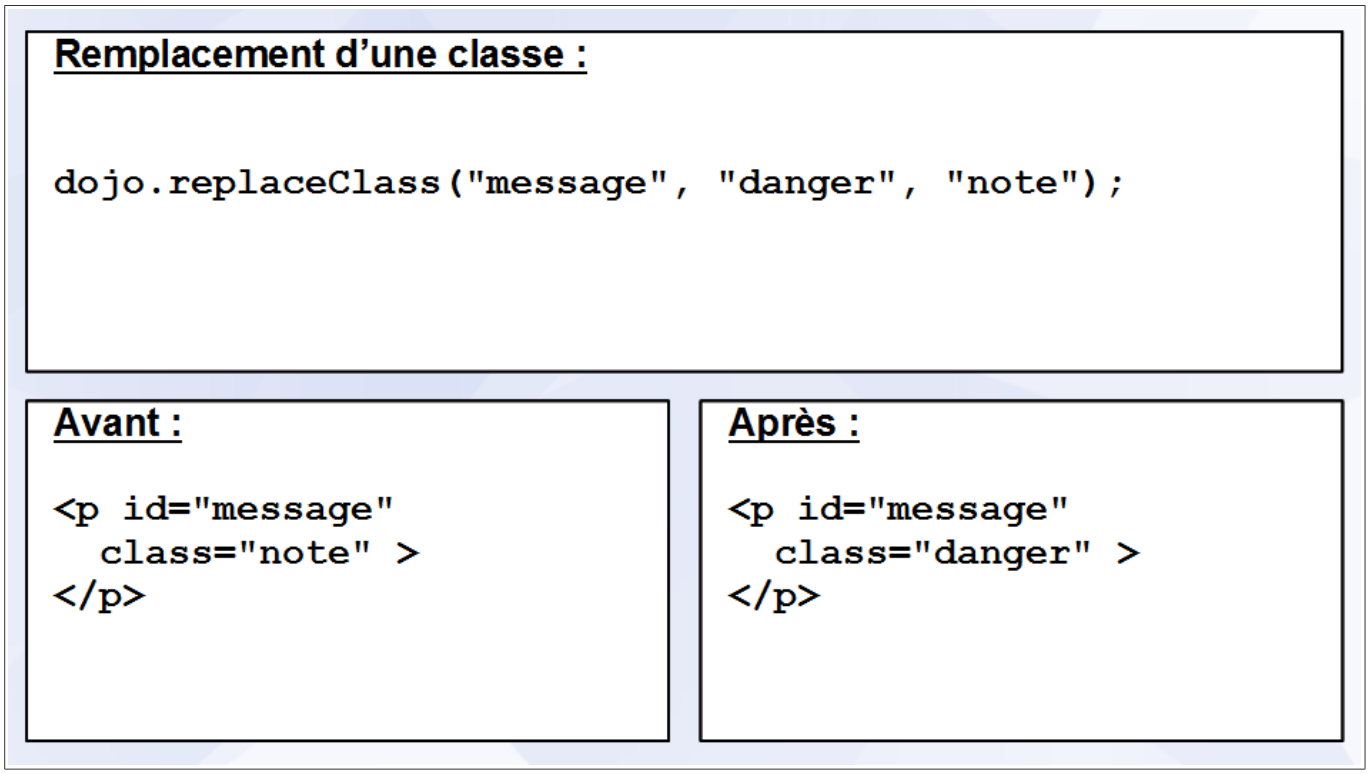
Commutation d'une classe



La fonction `dojo.toggleClass()` ajoute une ou plusieurs classes à un élément pour lequel elles ne sont pas définies et les suppriment dans le cas contraire. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`node`	`String`	Identifiant de l'élément.
`classStr`	`String`	Liste de classes à ajouter ou supprimer.
`condition`	`Boolean`	(optionnel) Si `true` ajoute les classes, si `false` les supprime.

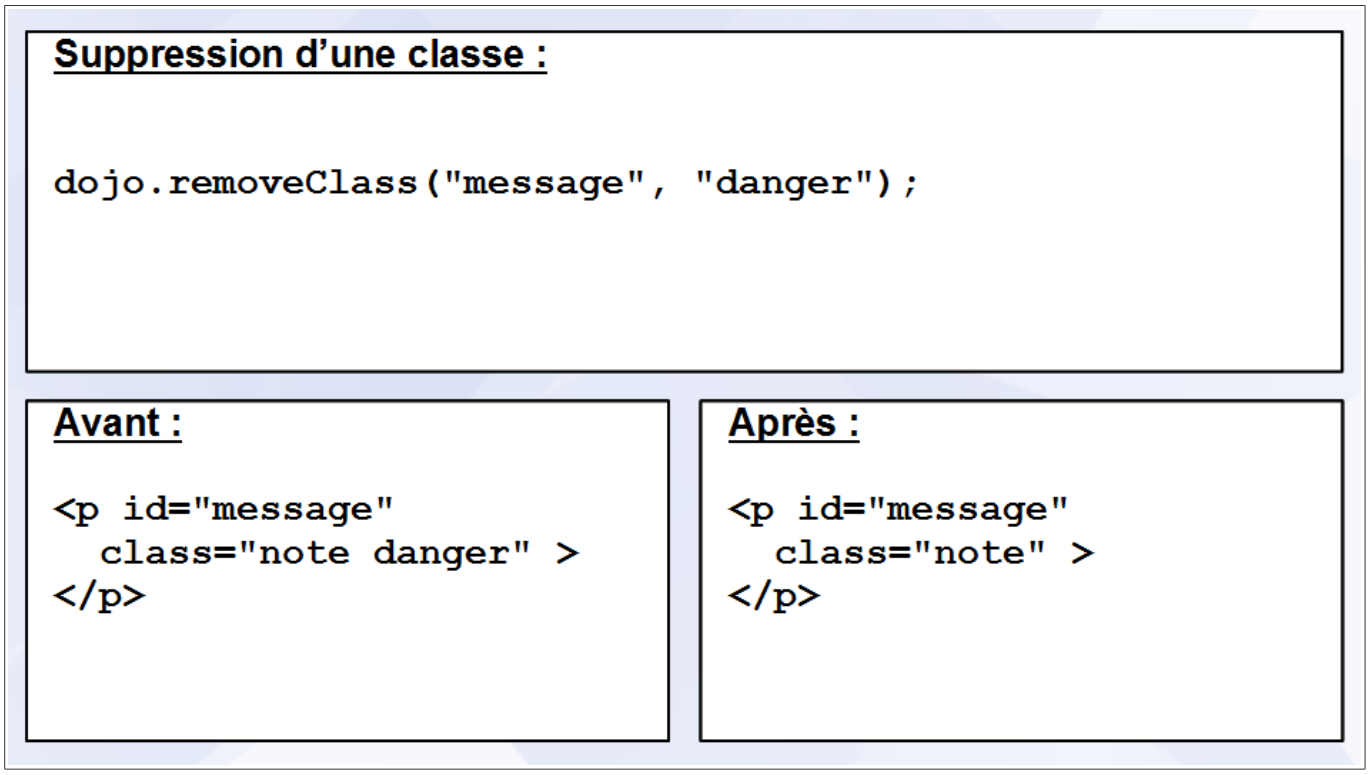
Remplacement d'une classe



La fonction `dojo.replaceClass()` remplace une ou plusieurs classes par une ou plusieurs autres classes. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`node`	`String`	Identifiant de l'élément.
`addClassStr`	`String`	Liste de classes à ajouter.
`removeClassStr`	`String`	Liste de classes à supprimer.

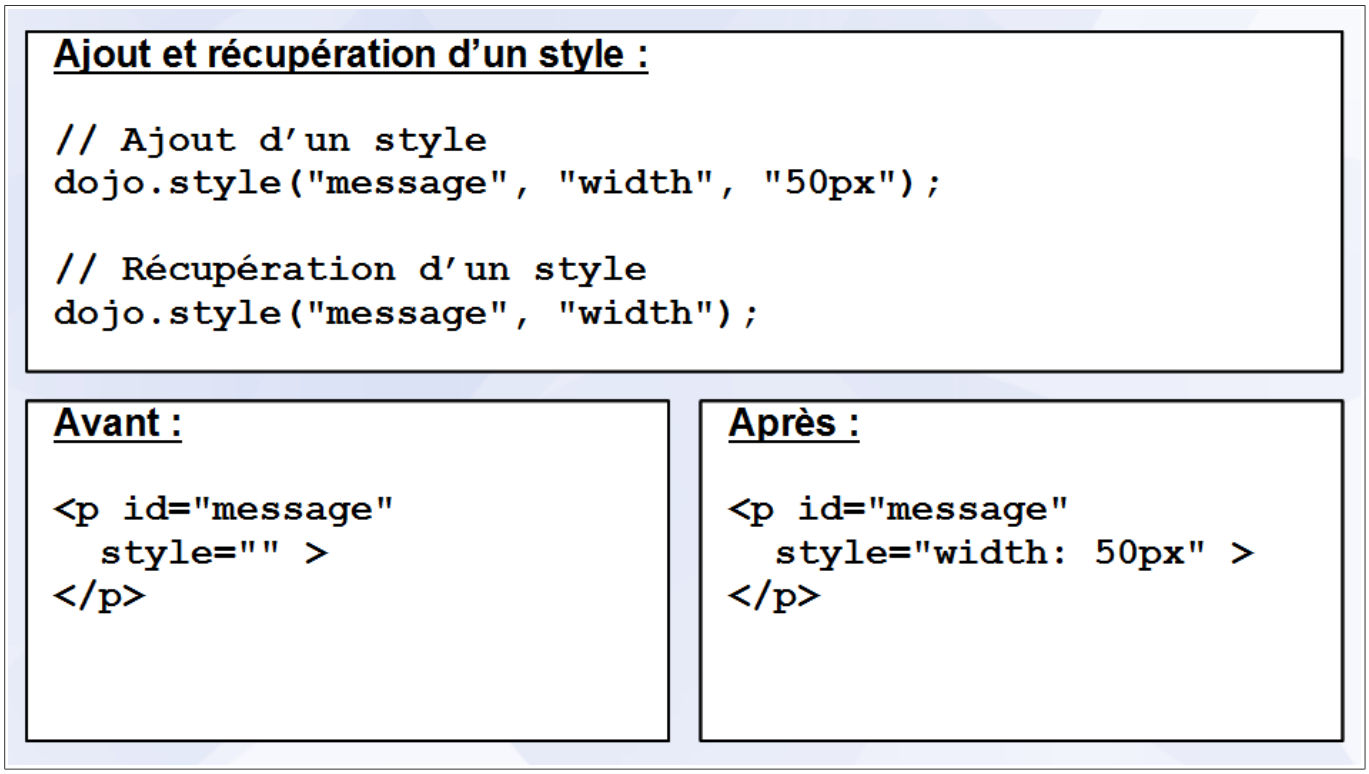
Suppression d'une classe



La fonction `dojo.removeClass()` supprime une ou plusieurs classes d'un élément. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`node`	`String`	Identifiant de l'élément.
`classStr`	`String`	Liste de classes à supprimer.

Manipulation des styles



La fonction `dojo.style()` récupère ou modifie la valeur d'un style. Elle accepte les paramètres suivants :

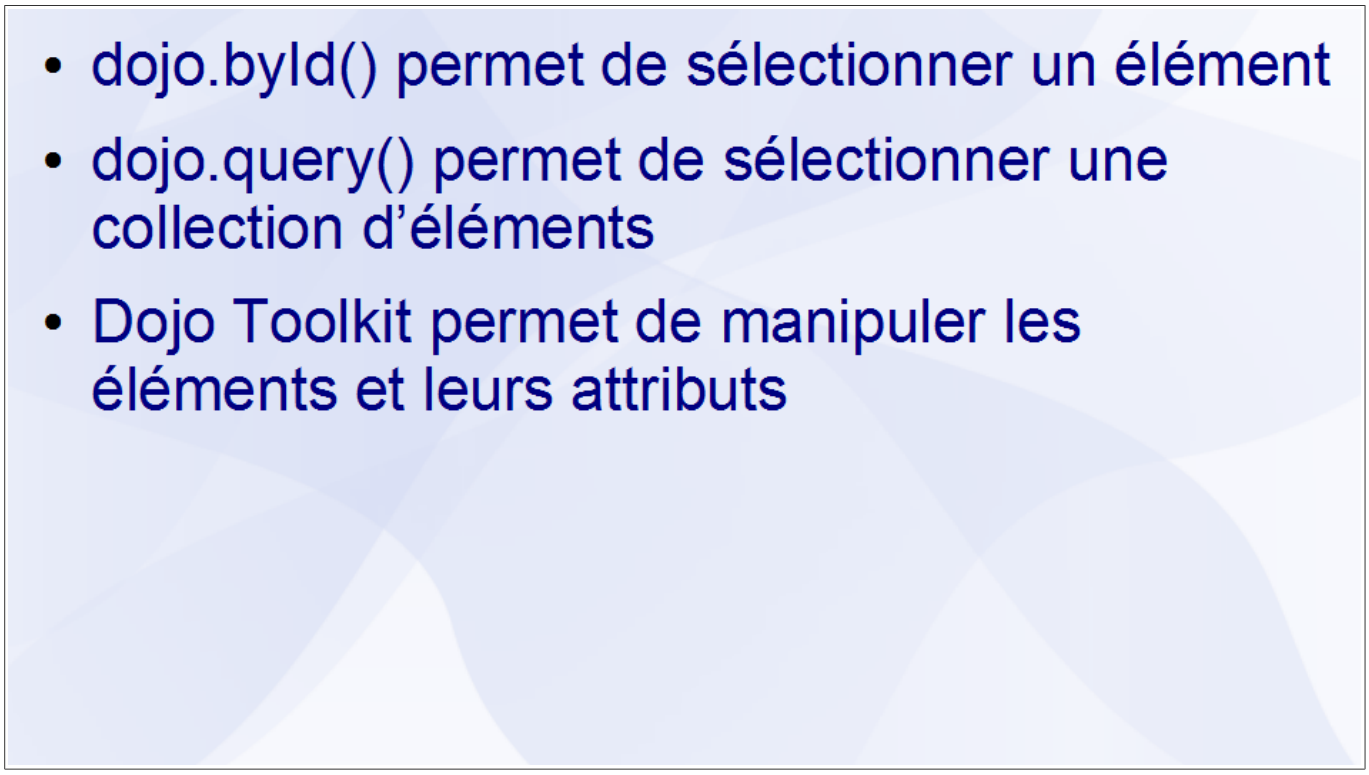
Paramètre	Type	Description
`node`	`String`	Identifiant de l'élément.
`name`	`String`	Nom de la propriété CSS.
`value`	`String`	(optionnel) Valeur de la propriété CSS.

Il est possible de passer un objet en second paramètre pour modifier plusieurs styles à la fois :

```
dojo.style("message", { width: "50px", height: "50px" }) ;
```

Dans la mesure du possible, il est préférable de modifier le style appliqué à un élément en modifiant ses classes et non directement avec `dojo.style()` afin de séparer le code applicatif et le code de présentation.

Conclusion


• dojo.byId() permet de sélectionner un élément
• dojo.query() permet de sélectionner une collection d'éléments
• Dojo Toolkit permet de manipuler les éléments et leurs attributs

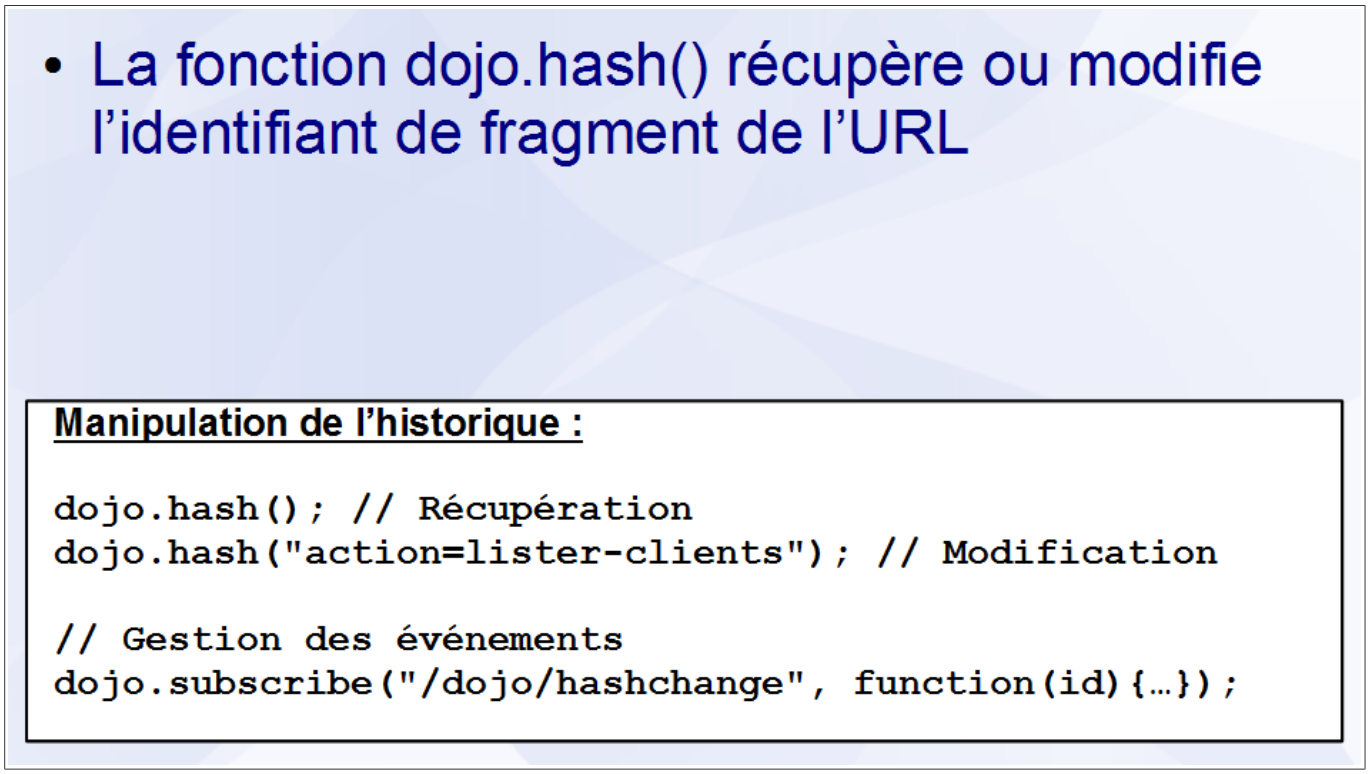
Chapitre 10 – Le navigateur

Sommaire

Introduction

- Le navigateur est le principal environnement d'exécution des applications JavaScript
- Chaque navigateur possède des fonctionnalités qui lui sont propres
- Dojo Toolkit dispose de plusieurs fonctions permettant d'exploiter ces fonctionnalités

Manipulation de l'historique



L'identifiant de fragment est la partie de l'URL située après le symbole # (*hash* en anglais). Normalement, lorsque qu'on indique une nouvelle adresse au navigateur, celui-ci décharge la page affichée avant de charger la nouvelle. La modification de l'identifiant de fragment fait exception car il indique une partie du document en cours d'utilisation ; mais, bien que la page affichée ne change pas, le navigateur ajoute l'adresse modifiée à l'historique. En exploitant cette fonctionnalité, il est possible de sauvegarder l'état de l'application dans l'historique du navigateur et de le restaurer ultérieurement.

On peut ainsi représenter l'écran de modification d'une fiche client avec l'URL ci-dessous :

```
http://www.example.com/#action=modifier-client&id=123456
```

Les fonctions `dojo.objectToQuery()` et `dojo.queryToObject()` sont souvent employées pour convertir un objet en identifiant de fragment et inversement.

La fonction `dojo.hash()` récupère ou modifie l'identifiant de fragment. Elle accepte les paramètres suivants :

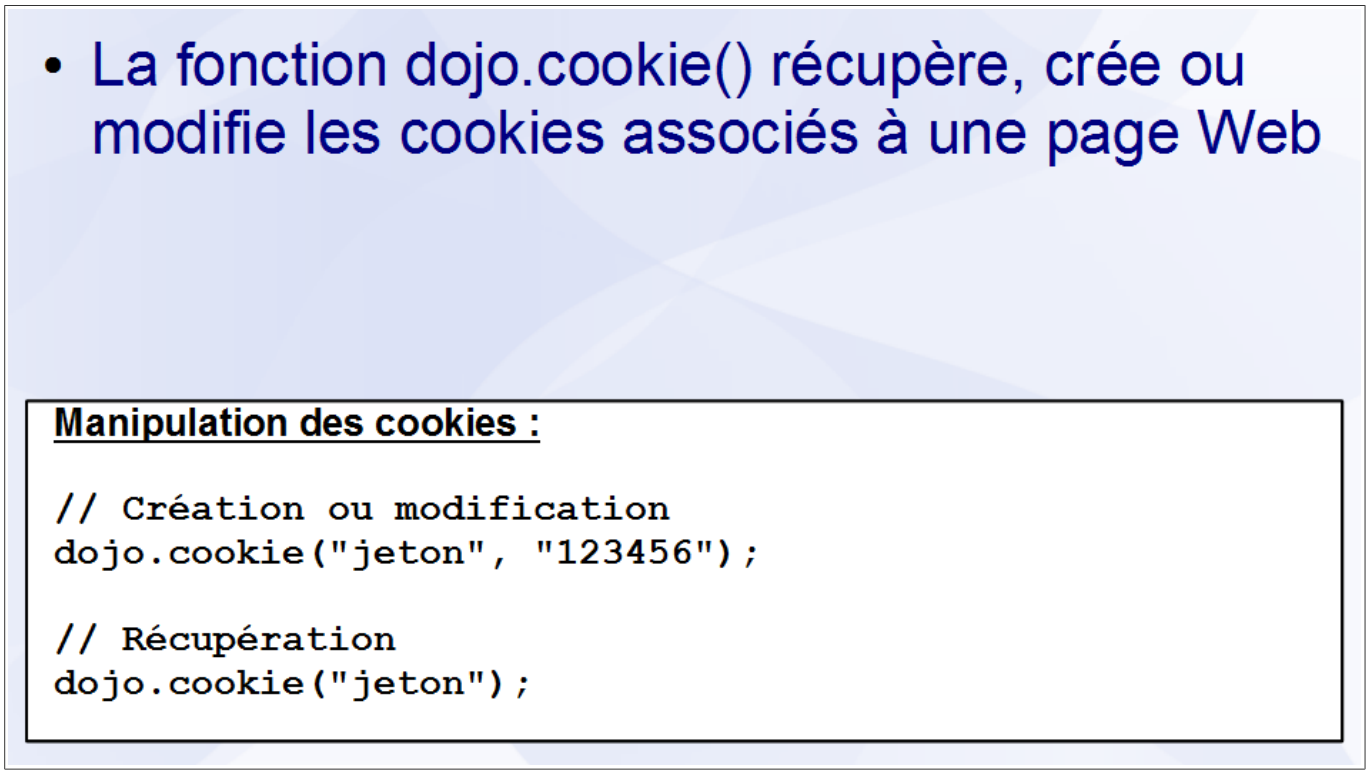
Paramètre	Type	Description
`hash`	`String`	(optionnel) Valeur de l'identifiant de fragment.
`replace`	`Boolean`	(optionnel) Si `true`, met à jour l'entrée de l'historique au lieu d'en créer une nouvelle.

Le module `dojo.hash` doit être importé avant de pouvoir utiliser la fonction `dojo.hash()`.

Lorsque l'identifiant de fragment est modifié, que ce soit ou non par `dojo.hash()`, Dojo Toolkit publie un message dont le sujet est `/dojo/hashchange`. Les abonnés à ce sujet reçoivent en paramètre le nouvel identifiant de fragment.

Le module `dojo.hash` remplace le module `dojo.back`.

Manipulation des cookies



HTTP est un protocole sans état, c'est à dire que le résultat d'une requête ne dépend pas du résultat des requêtes précédentes (elles sont indépendantes). Les cookies sont une extension au protocole HTTP permettant de fournir un contexte aux requêtes effectuées par un même utilisateur. Créés en JavaScript ou par le serveur Web, ils sont stockés, pour le compte de la page Web, par le navigateur et transmis par ce dernier à chaque requête vers le serveur Web d'origine.

Dans le cadre du développement d'une application Web, les cookies peuvent servir à :

- échanger des informations entre le serveur et l'application
- sauvegarder les données de l'application

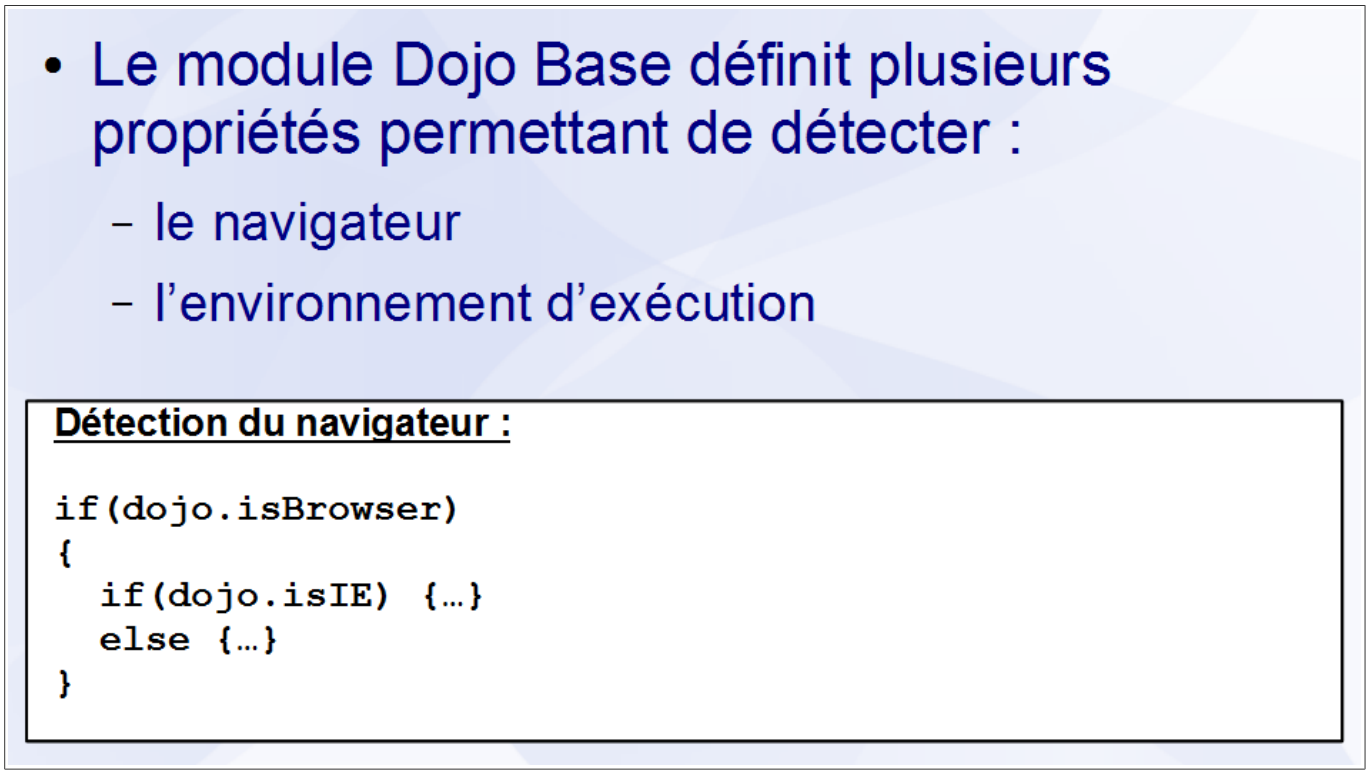
Toutefois, le nombre de cookies pouvant être créés par domaine est limité, tout comme le volume de données pouvant être enregistrées, ce qui limite ce dernier usage.

La fonction `dojo.cookie()` récupère, crée ou modifie les cookies associés à une page Web. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`hash`	`String`	Nom du cookie.
`replace`	`String`	(optionnel) Valeur du cookie.
`props`	`Object`	(optionnel) Tableau associatif des propriétés du cookie : • `expires` – date d'expiration • `path` – chemin où le cookie est valide • `domain` – domaine où le cookie est valide • `secure` – Si `true`, le cookie n'est transmis que pour les connexions sécurisées

Le module expérimental `dojox.storage` supporte des technologies alternatives permettant de sauvegarder un volume de données plus important. Si aucune de ces technologies n'est disponible, la sauvegarde se fait via les cookies.

Détection du navigateur



Développer une application Web n'est pas une chose aisée car le développeur n'a que peu de contrôle sur l'environnement d'exécution. Il doit cependant s'assurer que l'application fournit les mêmes fonctionnalités dans les différents navigateurs supportés tout en utilisant leurs fonctionnalités spécifiques pour améliorer le confort d'utilisation.

Le module Dojo Base définit les propriétés suivantes pour détecter le navigateur utilisé :

Propriété	Type	Description
`dojo.isIE`	`Number`	Numéro de version d'Internet Explorer ou `undefined`.
`dojo.isFF`	`Number`	Numéro de version de Firefox ou `undefined`.
`dojo.isChrome`	`Number`	Numéro de version de Chrome ou `undefined`.
`dojo.isSafari`	`Number`	Numéro de version de Safari ou `undefined`.
`dojo.isOpera`	`Number`	Numéro de version d'Opera ou `undefined`.
`dojo.isKhtml`	`Number`	Numéro de version de Konqueror ou `undefined`.

Le module Dojo Base définit les propriétés suivantes pour détecter l'environnement utilisé :

Propriété	Type	Description
`dojo.isBrowser`	`Boolean`	Si `true`, l'environnement est un navigateur Web.
`dojo.isQuirks`	`Boolean`	Si `true`, l'environnement est un navigateur Web en mode de compatibilité.
`dojo.isAir`	`Boolean`	Si `true`, l'environnement est Adobe Air.
`dojo.isBB`	`Boolean`	Si `true`, l'environnement est un BlackBerry.
`dojo.isMozilla`	`Number`	Numéro de version de Mozilla ou `undefined`.
`dojo.isRhino`	`Boolean`	Si `true`, l'environnement est Mozilla Rhino.
`dojo.isSpidermonkey`	`Boolean`	Si `true`, l'environnement est Mozilla SpiderMonkey.
`dojo.isWebKit`	`Number`	Numéro de version de WebKit ou `undefined`.

Conclusion

- La fonction dojo.hash() permet de manipuler l'historique du navigateur
- La fonction dojo.cookie() permet de manipuler les cookies associés à un document
- Les propriétés de détection du navigateur permettent d'adapter une application aux capacités de son environnement

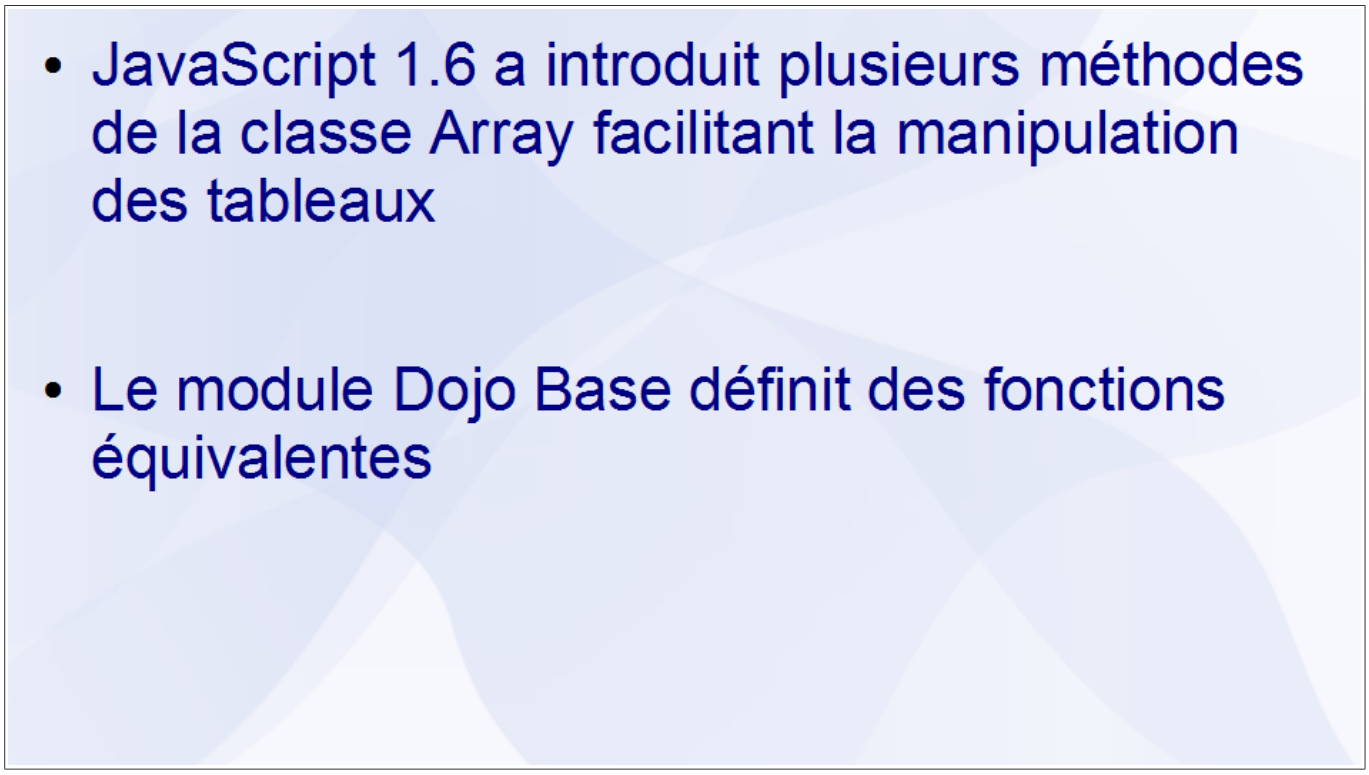
Chapitre 11 – Les fonctions utilitaires

Sommaire

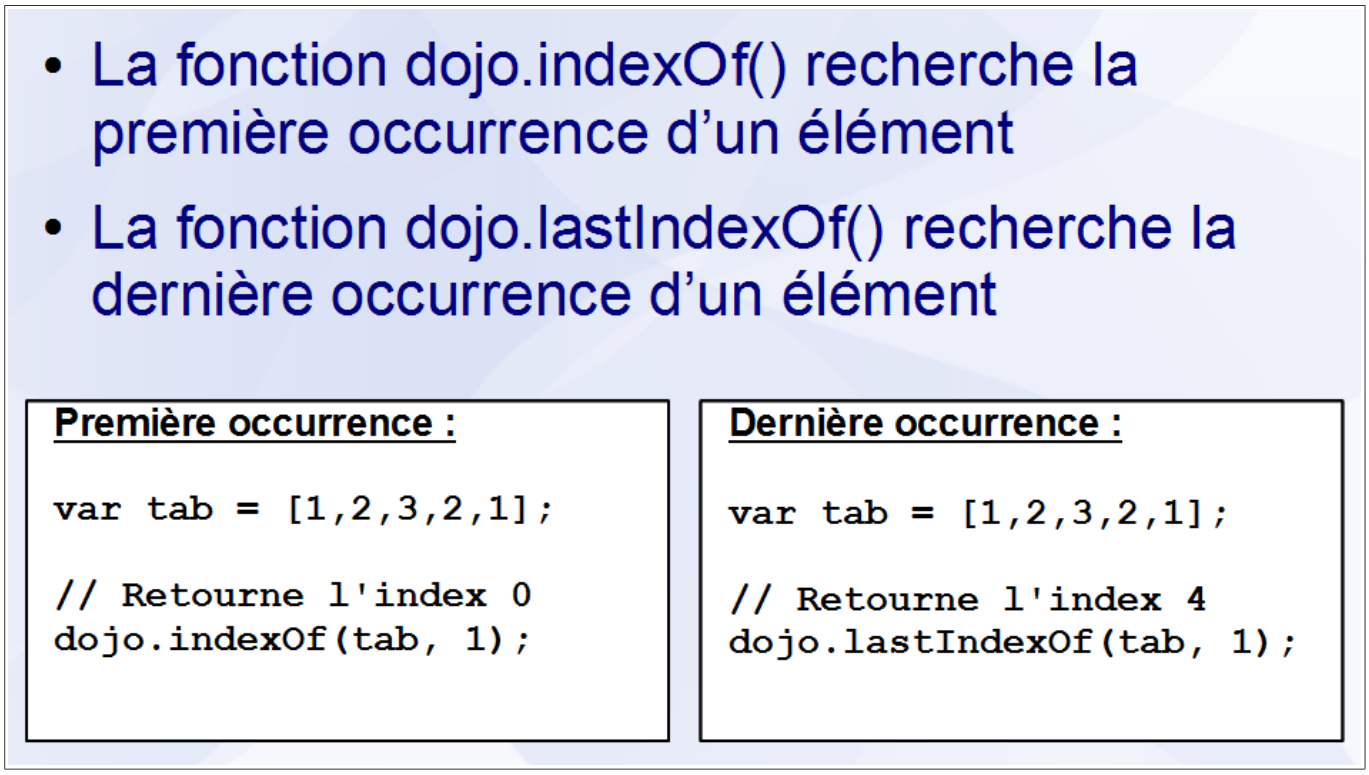
Introduction

- Le langage JavaScript introduit régulièrement de nouvelles fonctionnalités
- Ces nouveautés ne sont disponibles que dans les navigateurs les plus récents
- Dojo Toolkit :
  - permet de les utiliser avec tous les navigateurs
  - introduit des fonctionnalités complémentaires

Manipulation des tableaux



Recherche d'un élément

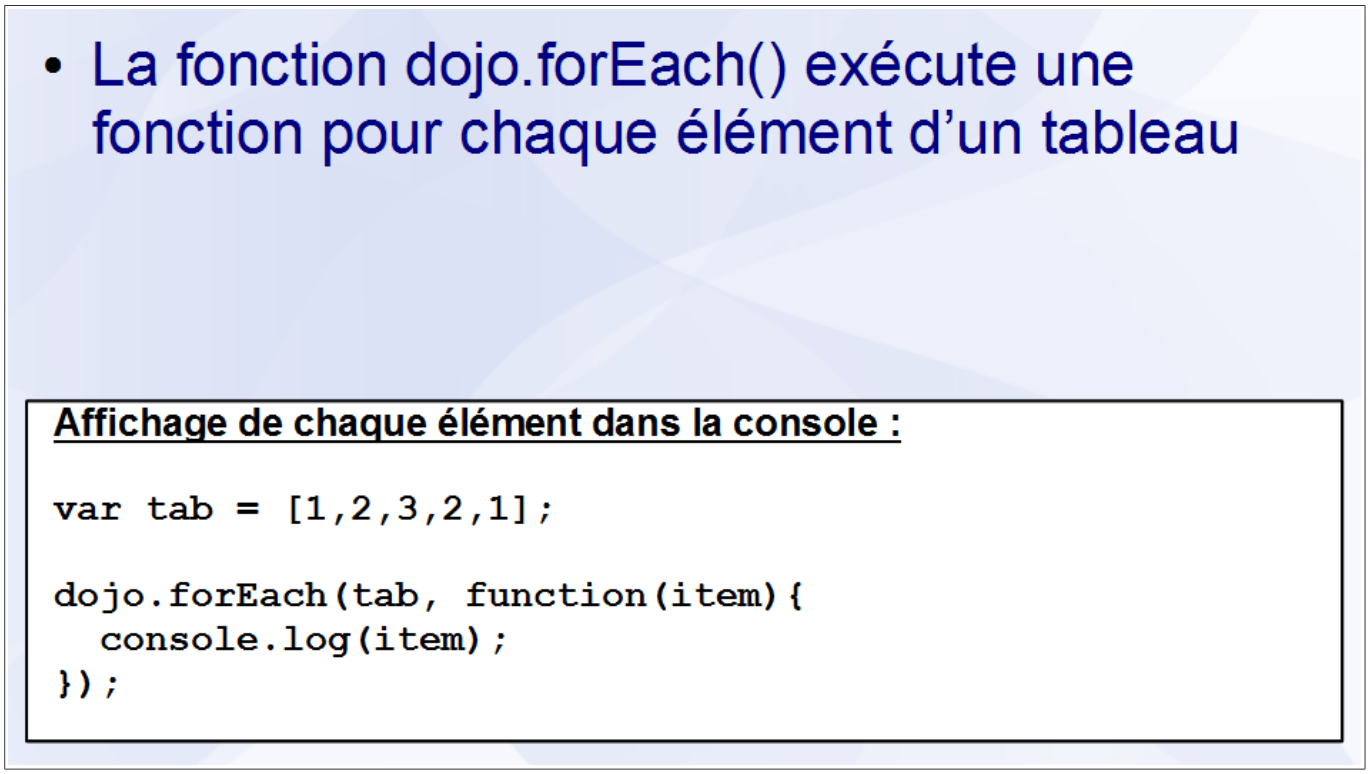


Les fonctions `dojo.indexOf()` et `dojo.lastIndexOf()` recherchent respectivement la première et la dernière occurrence d'un élément dans un tableau. Elles acceptent les paramètres suivants :

Paramètre	Type	Description
`arr`	`Array`	Tableau dans lequel effectuer la recherche.
`value`	`-`	Élément à rechercher.
`fromIndex`	`Number`	(optionnel) Index à partir duquel commencer la recherche.

Les fonctions `dojo.indexOf()` et `dojo.lastIndexOf()` retournent l'index de l'élément s'il est présent dans le tableau ou `-1` dans le cas contraire.

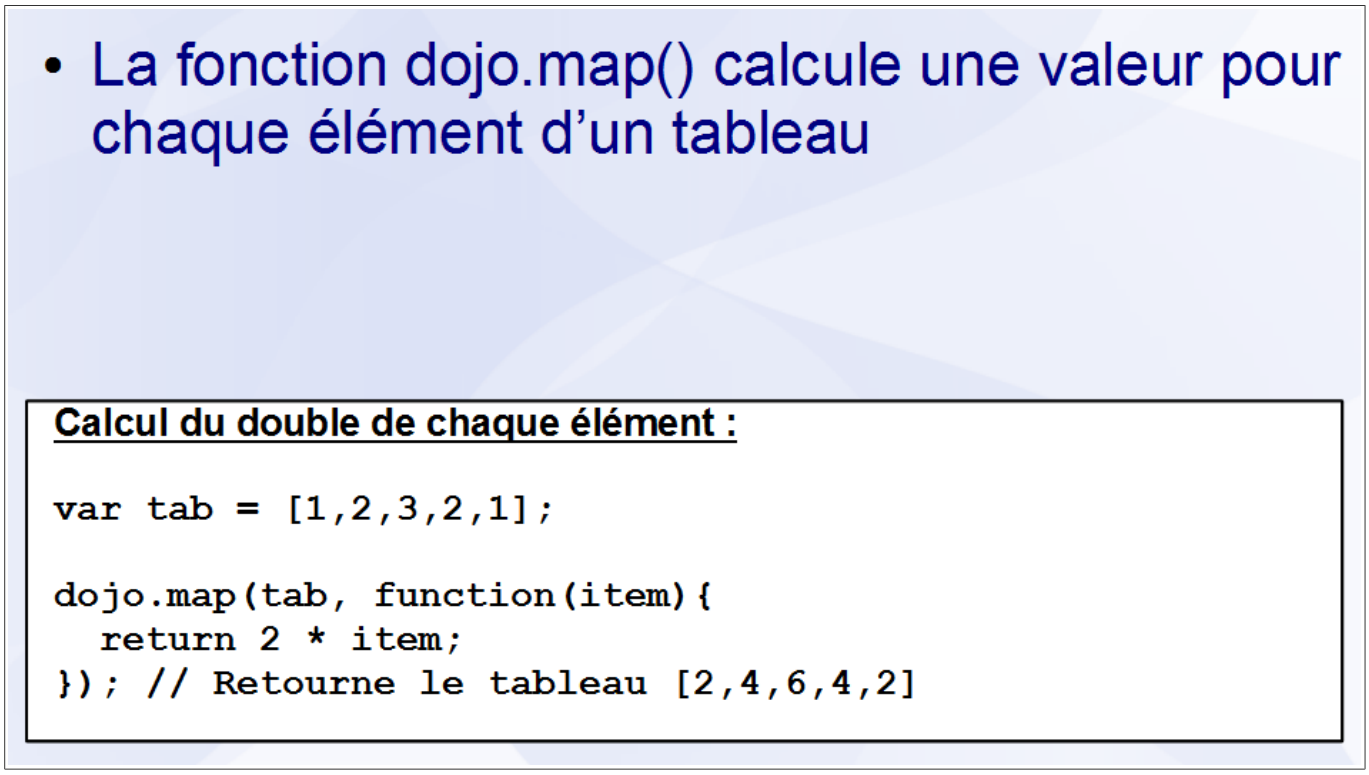
Exécution d'une fonction



La fonction `dojo.forEach()` exécute une fonction pour chaque élément d'un tableau. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`arr`	`Array`	Tableau à traiter.
`callback`	`Function`	Fonction de rappel à appliquer aux éléments du tableau. Elle accepte les paramètres suivants : • `item` – élément du tableau • `index` – index de l'élément • `array` – tableau en cours de traitement
`thisObject`	`Object`	(optionnel) Objet de contexte pour la fonction de rappel.

Calcul d'une valeur

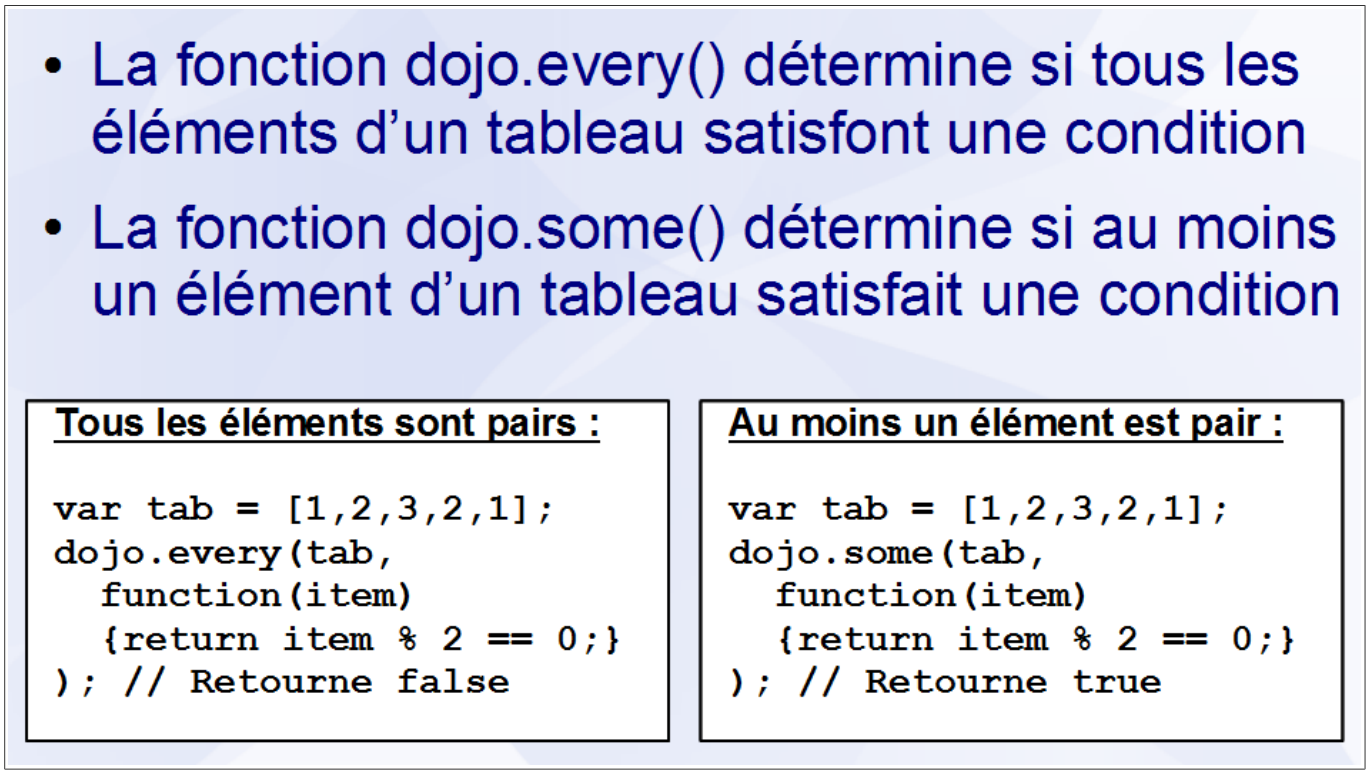


La fonction `dojo.map()` calcule une valeur pour chaque élément d'un tableau. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`arr`	`Array`	Tableau à traiter.
`callback`	`Function`	Fonction de rappel à appliquer aux éléments du tableau. Elle accepte les paramètres suivants : • `item` – élément du tableau • `index` – index de l'élément • `array` – tableau en cours de traitement
`thisObject`	`Object`	(optionnel) Objet de contexte pour la fonction de rappel.

La fonction `dojo.map()` retourne un tableau contenant les valeurs calculées pour chaque élément par la fonction de rappel.

Satisfaction d'une condition

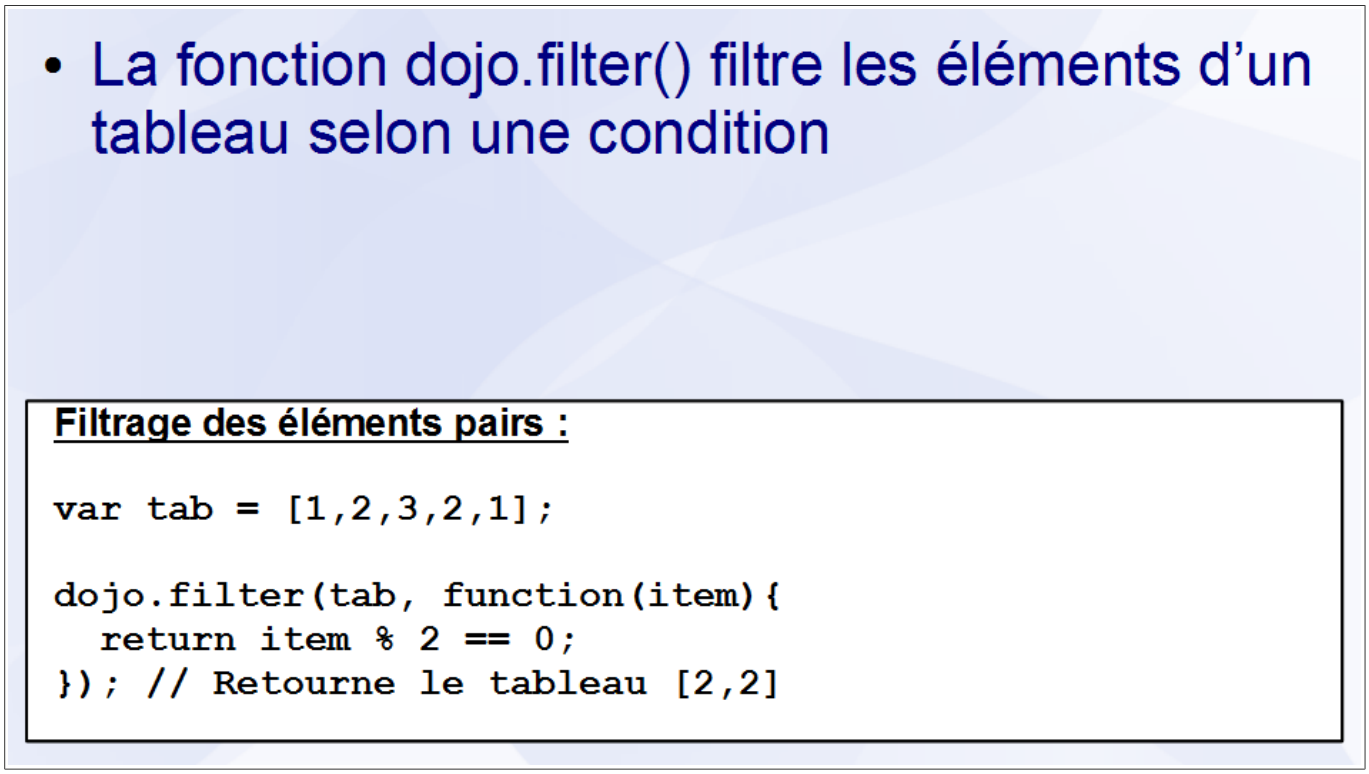


La fonction `dojo.every()` détermine si tous les éléments d'un tableau satisfont une condition tandis que la fonction `dojo.some()` détermine si au moins un élément d'un tableau satisfait une condition. Elles acceptent les paramètres suivants :

Paramètre	Type	Description
`arr`	`Array`	Tableau à traiter.
`callback`	`Function`	Fonction de rappel à appliquer aux éléments du tableau. Elle accepte les paramètres suivants : • `item` – élément du tableau • `index` – index de l'élément • `array` – tableau en cours de traitement
`thisObject`	`Object`	(optionnel) Objet de contexte pour la fonction de rappel.

La fonction `dojo.every()` retourne `true` si la fonction de rappel retourne `true` pour tous les éléments du tableau tandis que la fonction `dojo.some()` retourne `true` si la fonction de rappel retourne `true` pour au moins un élément du tableau.

Filtrage des éléments

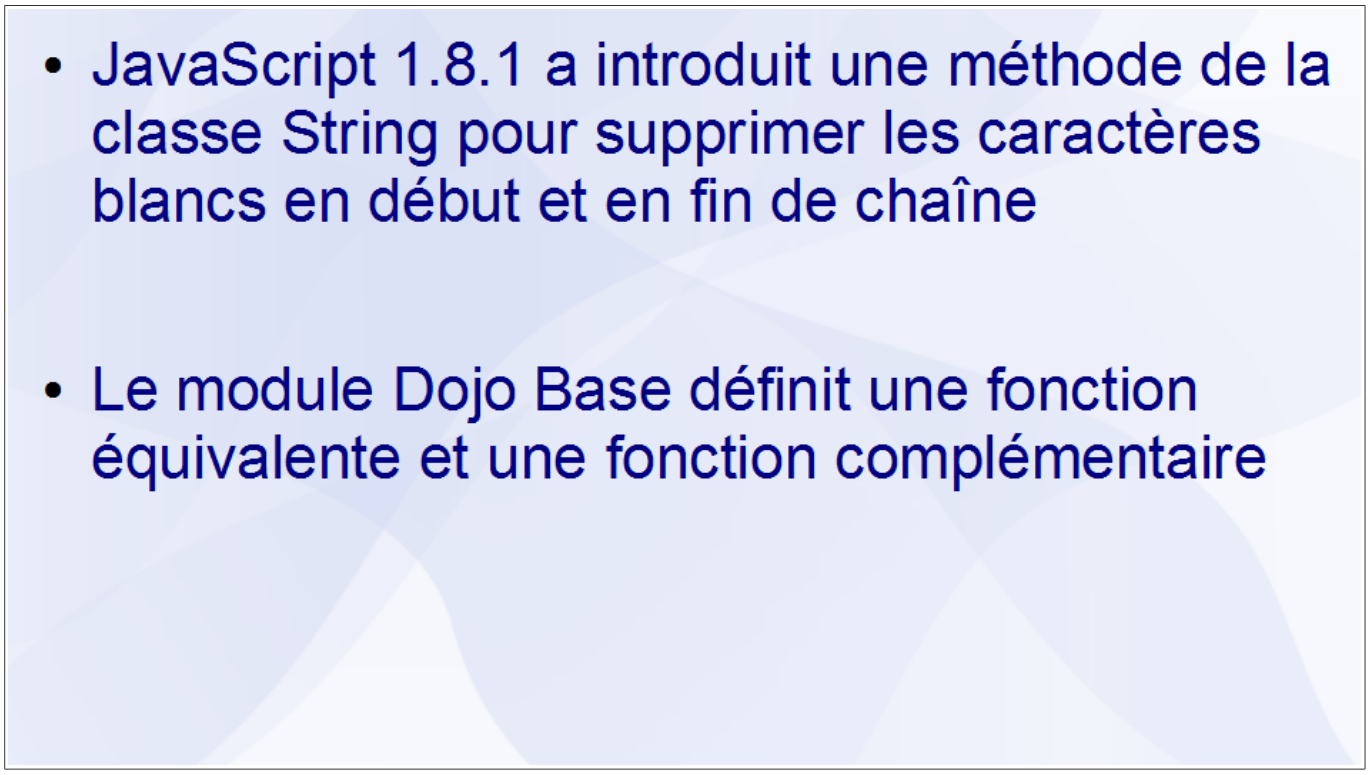


La fonction `dojo.filter()` filtre les éléments d'un tableau selon une condition. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`arr`	`Array`	Tableau à traiter.
`callback`	`Function`	Fonction de rappel à appliquer aux éléments du tableau. Elle accepte les paramètres suivants : • `item` – élément du tableau • `index` – index de l'élément • `array` – tableau en cours de traitement
`thisObject`	`Object`	(optionnel) Objet de contexte pour la fonction de rappel.

La fonction `dojo.filter()` retourne un tableau contenant les éléments pour lesquels la fonction de rappel retourne `true`.

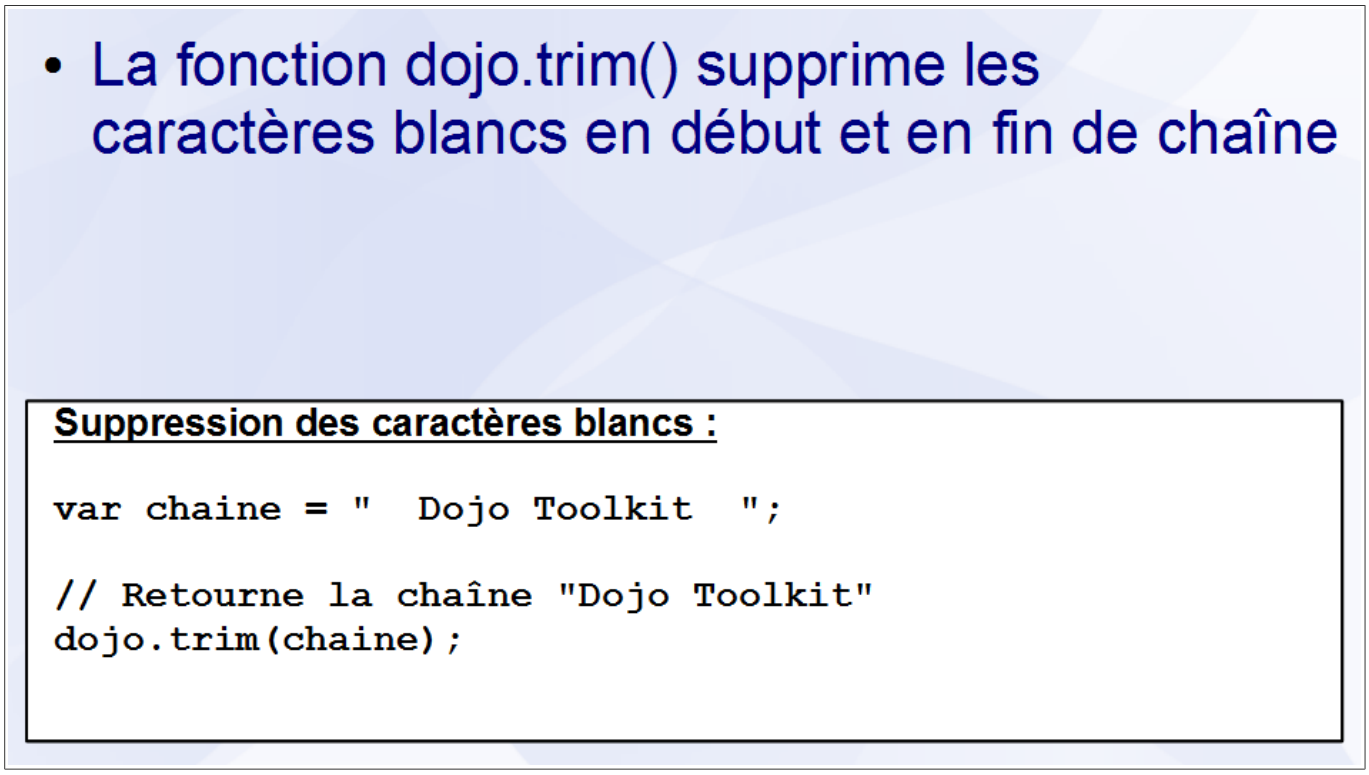
Manipulation des chaînes



Le module `dojo.string` définit les fonctions complémentaires suivantes :

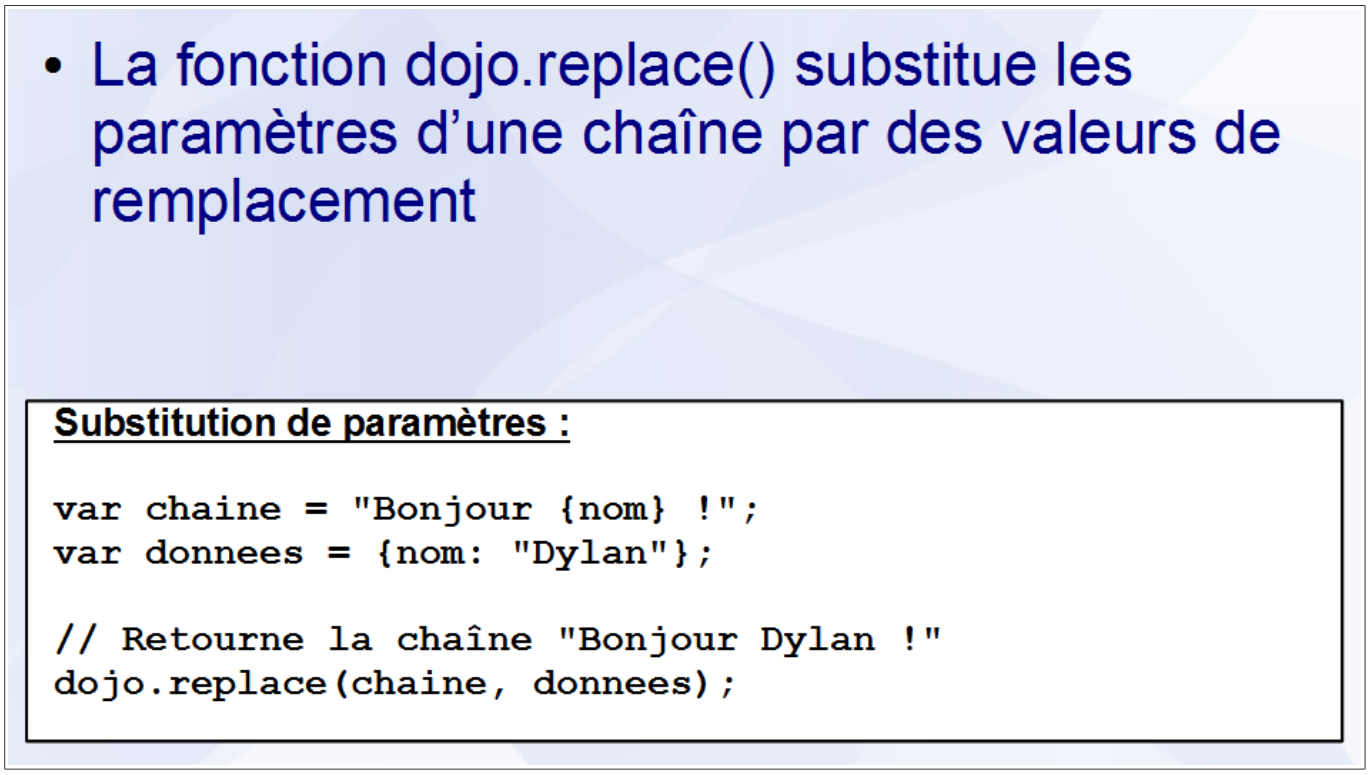
Fonction	Description
`dojo.string.trim()`	Supprime les caractères blancs en début et en fin de chaîne. Cette implémentation offre de meilleurs performances que `dojo.trim()`.
`dojo.string.substitute()`	Substitue les paramètres d'une chaîne par des valeurs de remplacement. Cette implémentation est utilisée dans le système de gabarits des Widgets.
`dojo.string.rep()`	Duplique une chaîne un certain nombre de fois.
`dojo.string.pad()`	Complète une chaîne, si nécessaire, pour qu'elle ait une certaine longueur.

Suppression des caractères blancs



La fonction `dojo.trim()` supprime les caractères blancs en début et en fin de chaîne. Elle accepte en paramètre la chaîne à traiter et retourne une nouvelle chaîne.

Substitution de paramètres



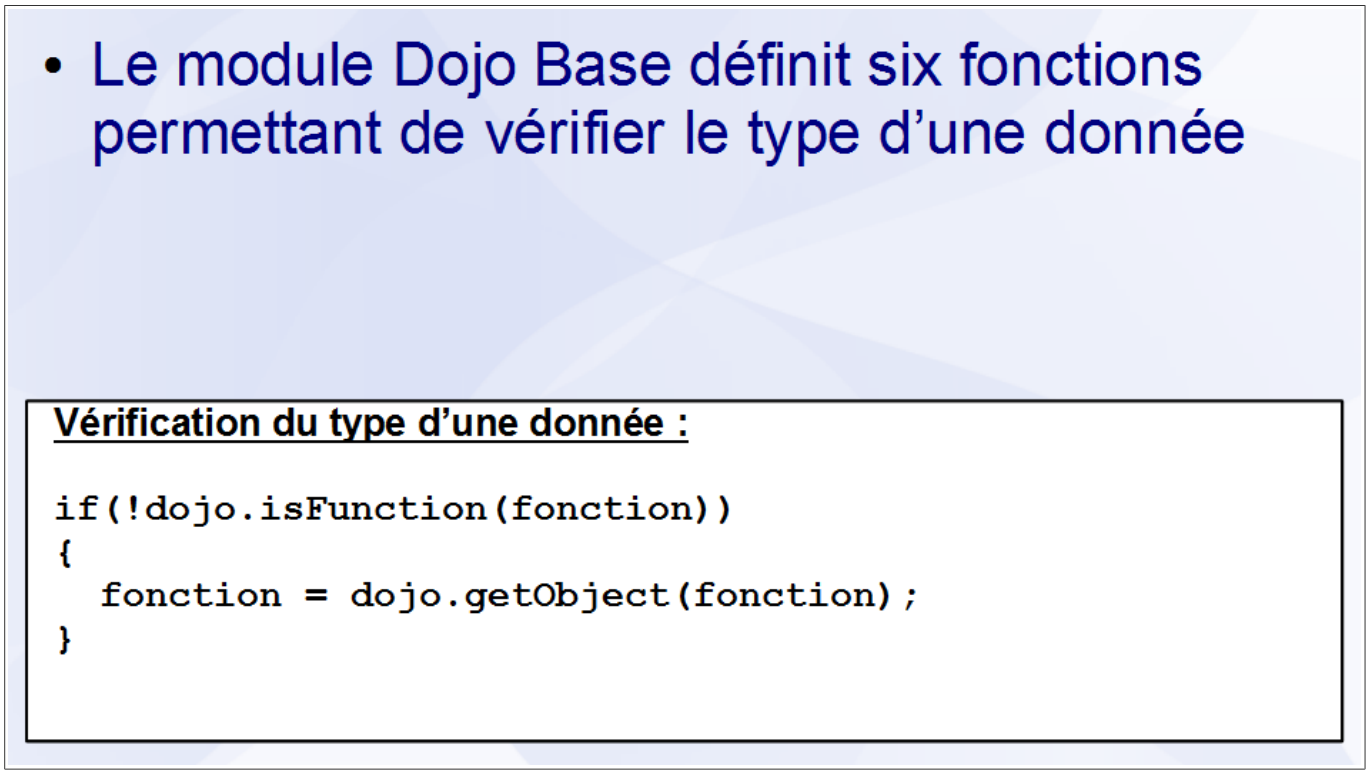
La fonction `dojo.replace()` substitue les paramètres d'une chaîne par des valeurs de remplacement. Elle accepte les paramètres suivants :

Paramètre	Type	Description
`tmpl`	`String`	Chaîne paramétrée.

`map`	`Object`
	`Array`
	`Function`
		Valeurs de remplacement.

La fonction `dojo.replace()` retourne la chaîne obtenue après la substitution des paramètres.

Vérification des types de données

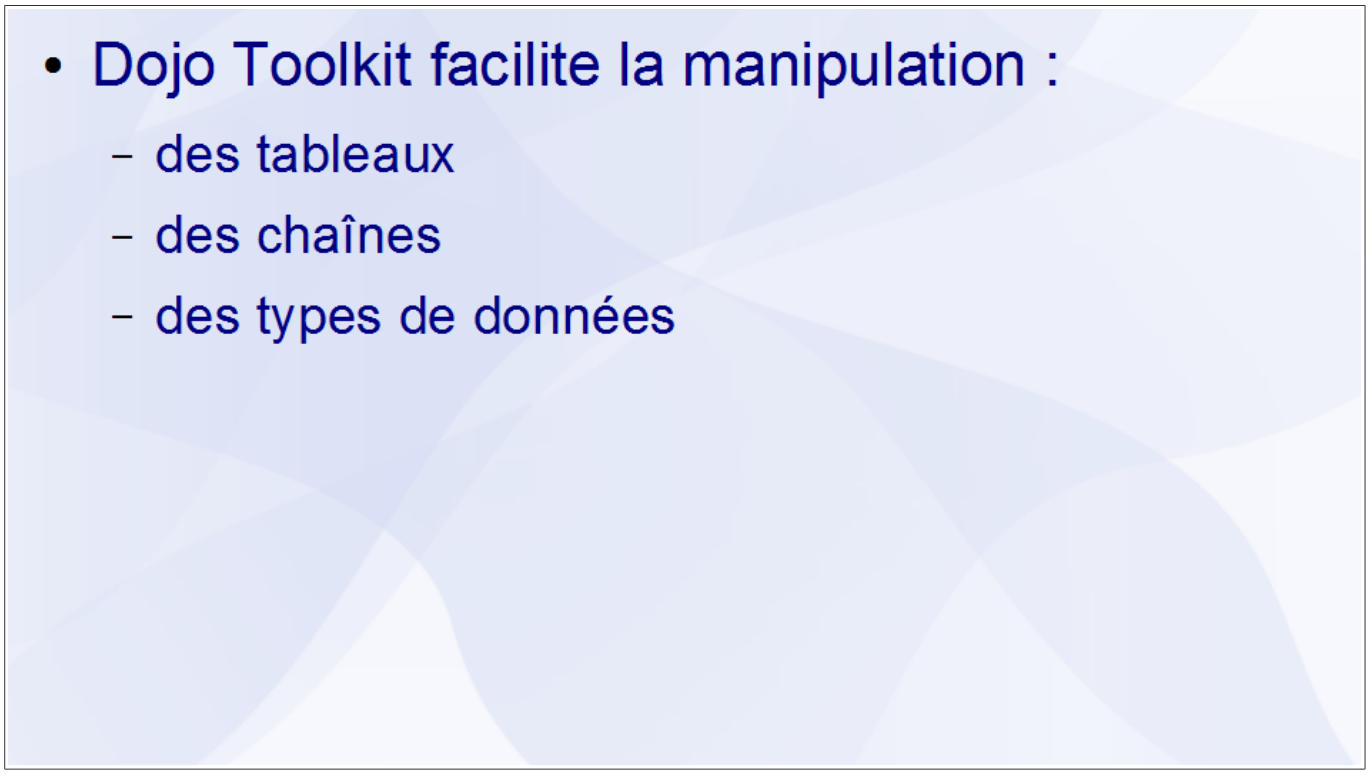


Le JavaScript est un langage à typage dynamique. Il convertit automatiquement les données vers le type approprié pour l'opération effectuée. Cependant, il est parfois nécessaire de s'assurer du type d'une donnée avant son utilisation.

Le module Dojo Base définit les fonctions suivantes pour vérifier le type d'une donnée :

Fonction	Description
`dojo.isString()`	Vérifie que la donnée est une chaîne.
`dojo.isArray()`	Vérifie que la donnée est un tableau.
`dojo.isArrayLike()`	Vérifie que la donnée se comporte comme un tableau.
`dojo.isObject()`	Vérifie que la donnée est un objet.
`dojo.isFunction()`	Vérifie que la donnée est une fonction.
`dojo.isAlien()`	Vérifie que la donnée est une fonction intégrée de l'environnement.

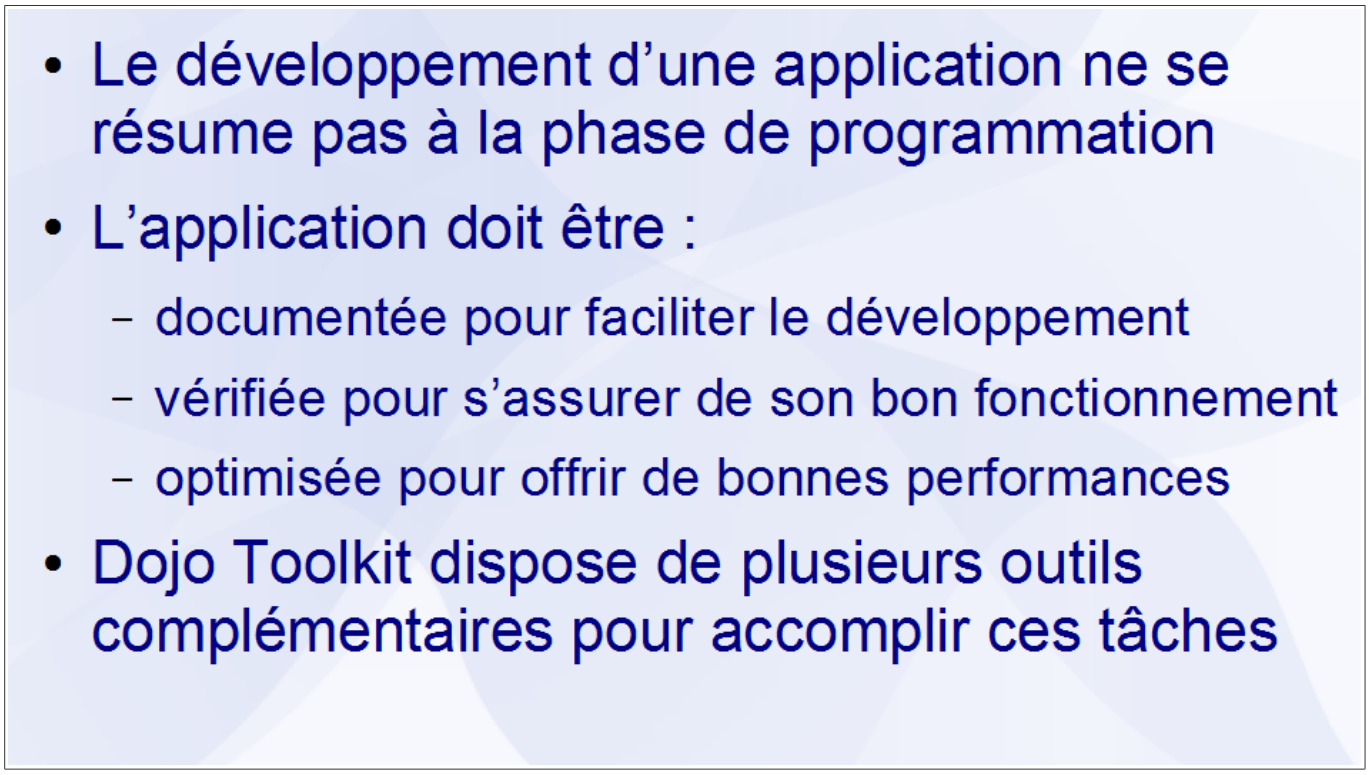
Conclusion


• Dojo Toolkit facilite la manipulation :
– des tableaux
– des chaînes
– des types de données

Chapitre 12 – Les outils complémentaires

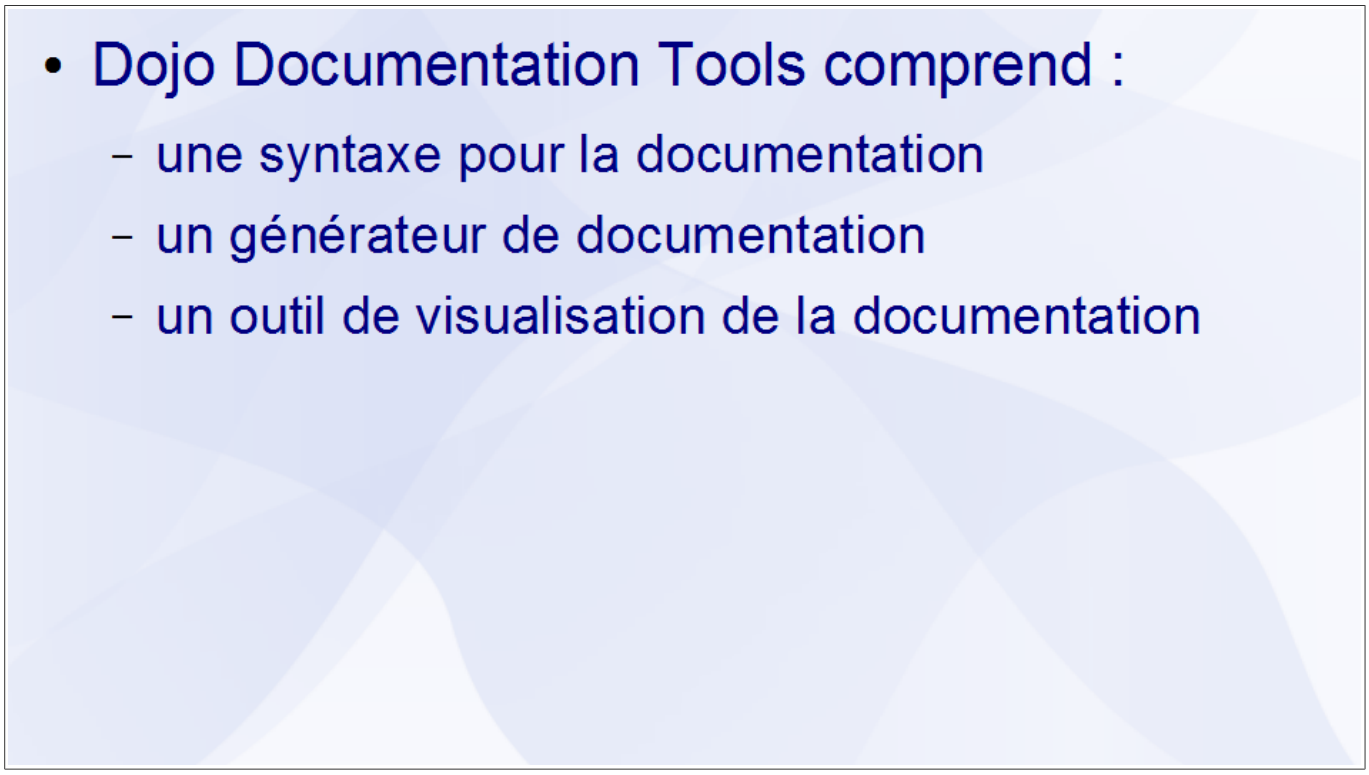
Sommaire

Introduction



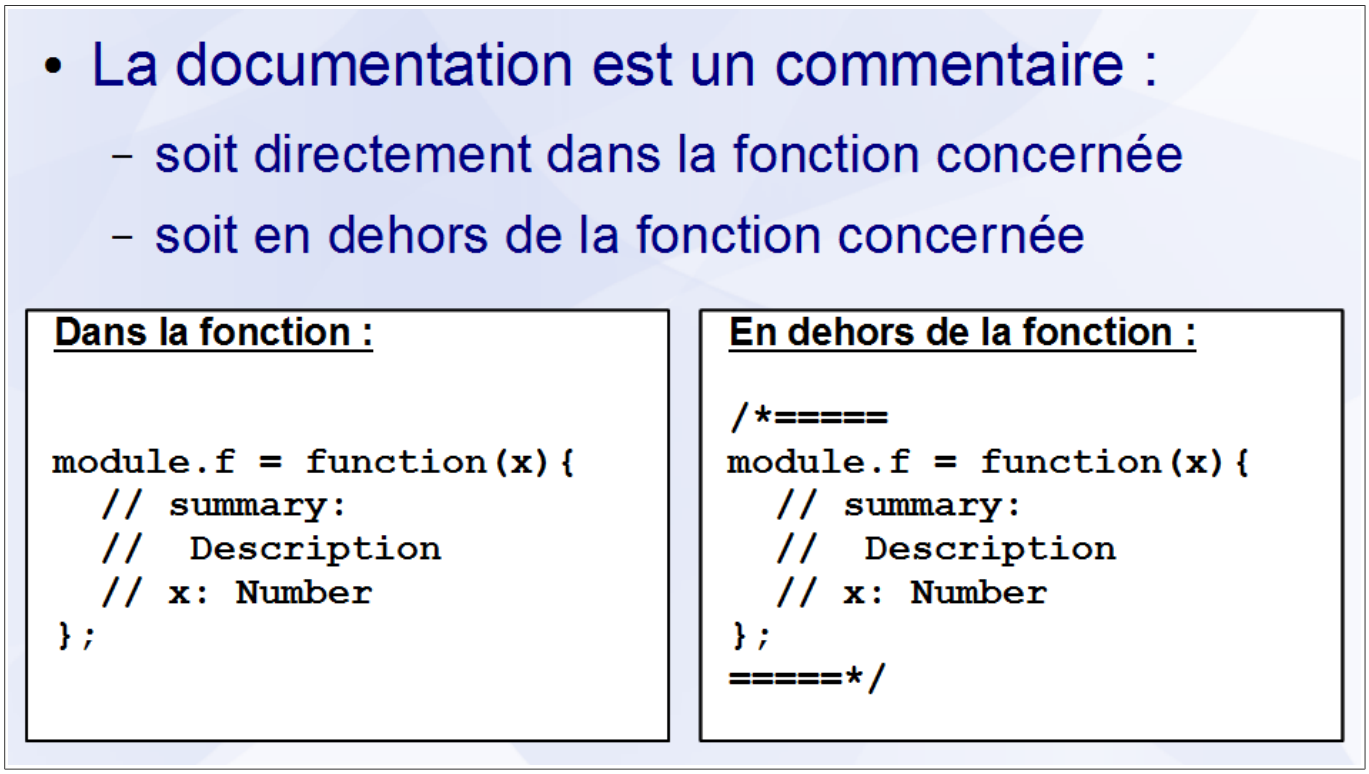
Les outils complémentaires ne sont présents que dans l'édition Dojo Toolkit SDK et sont situés dans le répertoire `util`.

Dojo Documentation Tools



Dojo Documentation Tools est situé dans le sous-répertoire `docscripts` du répertoire `util`.

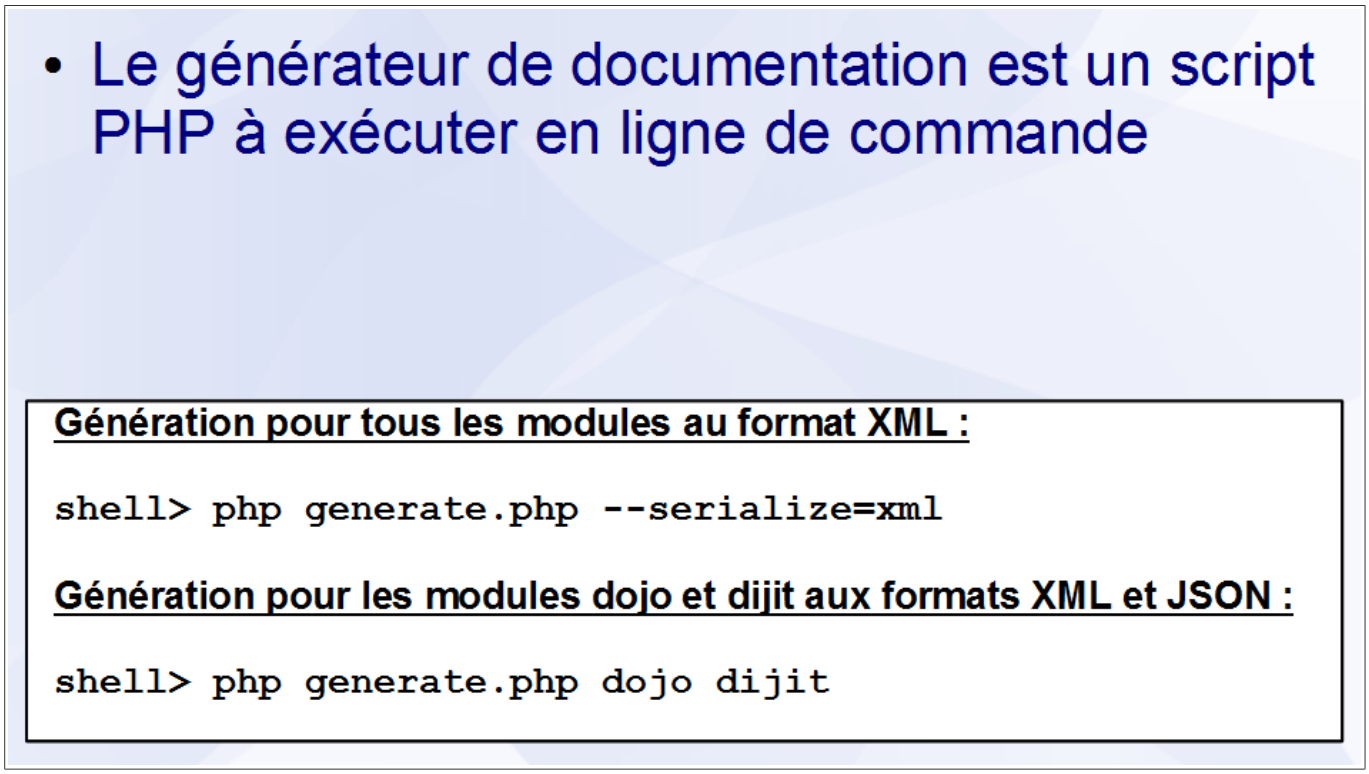
Syntaxe pour la documentation



Le commentaire de documentation peut contenir les sections suivantes :

Section	Description
`summary`	Description courte de la fonction.
`description`	Description longue fonction.
`tags`	Liste de mots séparés par des espaces permettant de classifier la fonction. `public`, `protected`, `private`, `callback` et `extension` sont supportés par l'outil de visualisation.
`this`	Type de `this`.
`returns`	Description de la valeur de retour.
`example`	(multiple) Exemple d'utilisation de la fonction.

Générateur de documentation

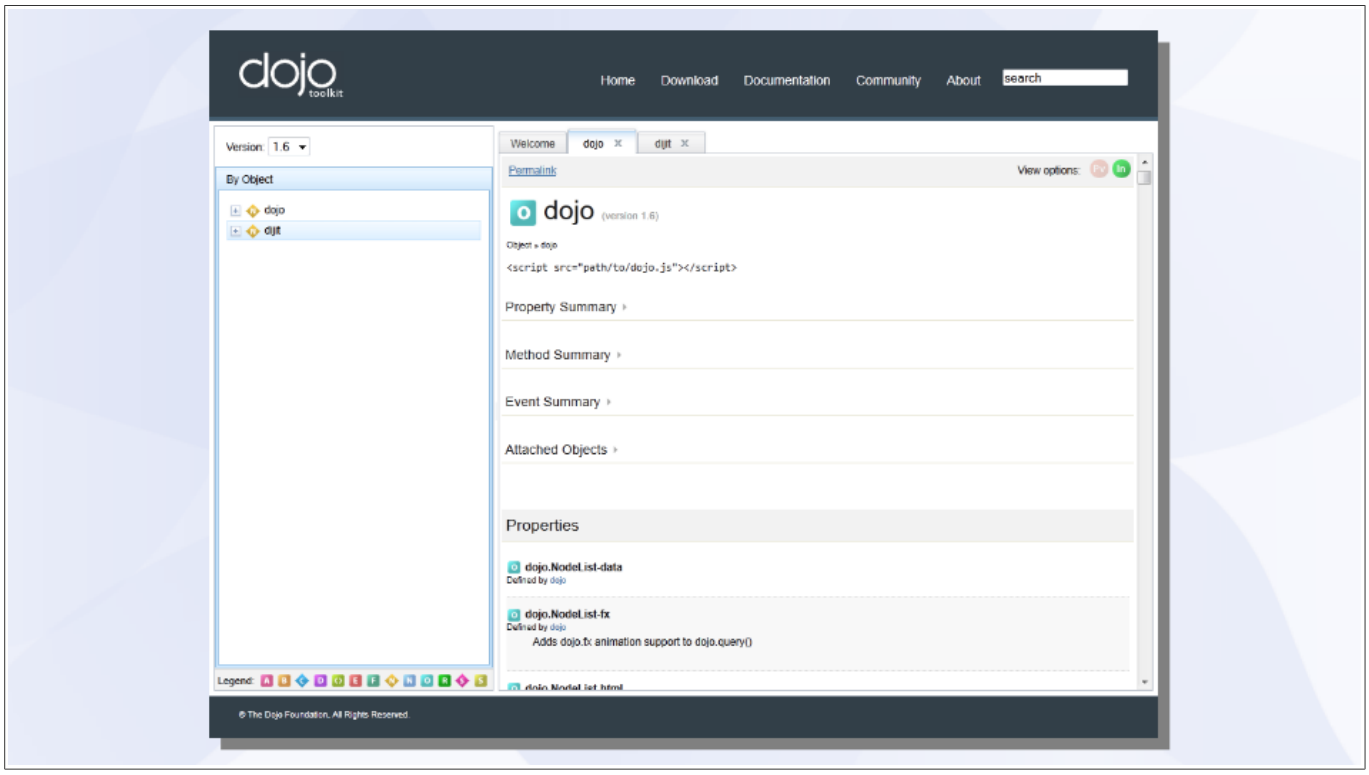


Le script `generate.php` est situé dans le sous-répertoire `docscripts` du répertoire `util`. Pour l'utiliser, il faut s'assurer que PHP est installé et disponible dans la variable d'environnement PATH.

Pour générer la documentation de ses propres modules, il faut créer dans le sous-répertoire `modules` de `docscripts` un fichier nommé *`nomModule`*`.module.properties` contenant :

```
location = chemin/vers/nomModule/
```

Outil de visualisation de la documentation



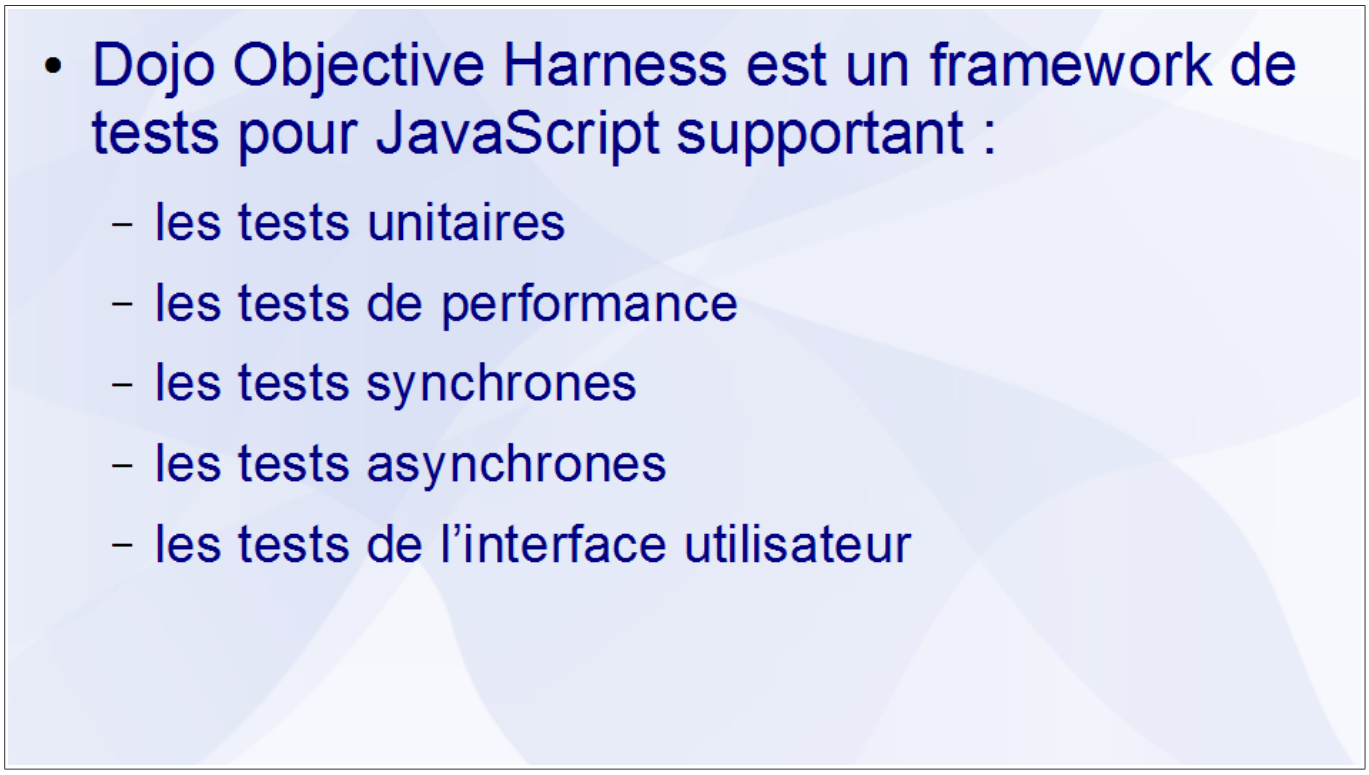
L'outil de visualisation n'est pas disponible directement avec Dojo Toolkit SDK. Pour l'obtenir, il faut récupérer son code directement dans le référentiel Subversion :

```
shell> svn export http://svn.dojotoolkit.org/website/trunk/api
```

L'outil de visualisation est une application développée en PHP et doit être installé sur un serveur Web pour être utilisée. Les instructions d'installation se trouve dans le fichier `README.txt` du répertoire `api`.

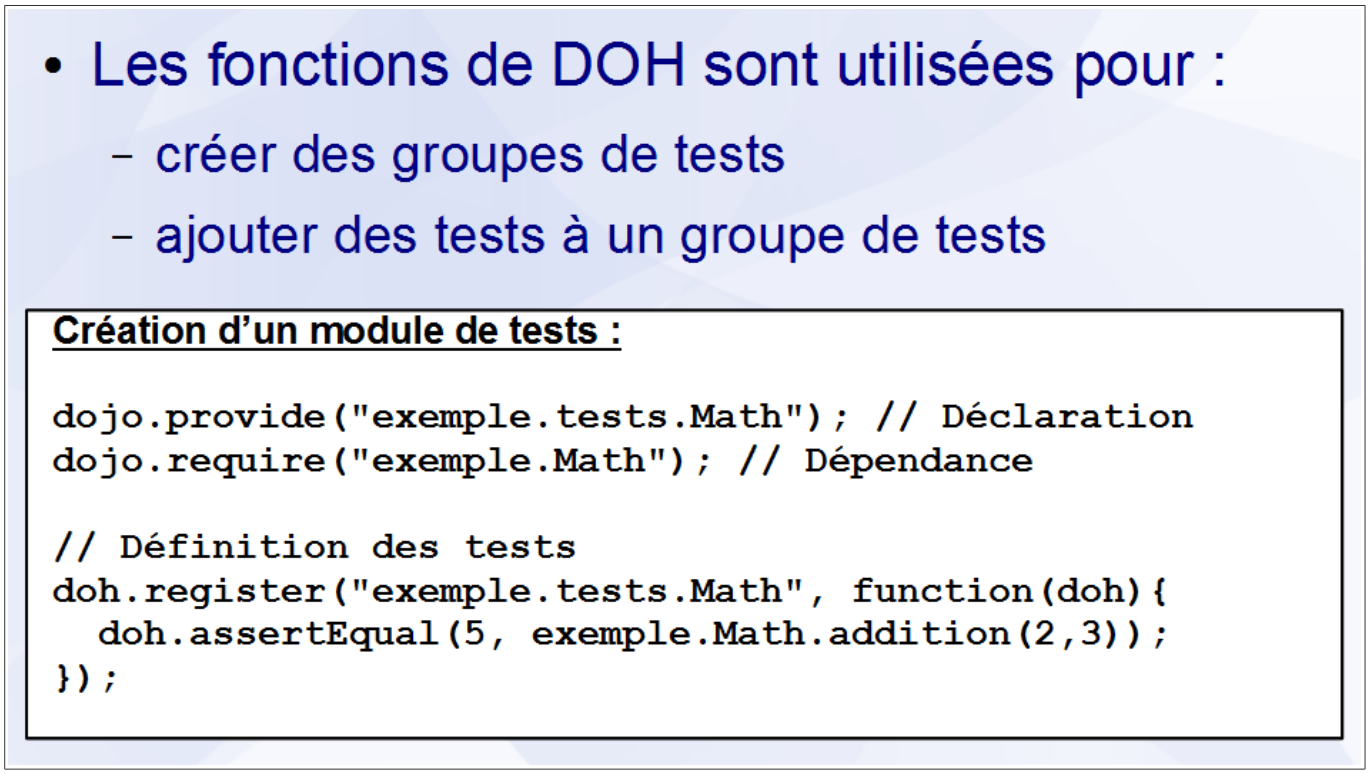
Il est toutefois possible de prévisualiser la documentation, avant même sa génération, avec le script `preview.php` situé dans le répertoire `docscripts`. Il nécessite également d'être installé sur un serveur Web.

Dojo Objective Harness



Dojo Objective Harness (DOH) est situé dans le sous-répertoire `doh` du répertoire `util`. Il peut être utilisé depuis un navigateur (`runner.html`) ou depuis l'invite de commandes (`runner.sh`).

Création d'un module de tests



Un module de tests a la même structure qu'un module standard. Il comprend une déclaration, une liste de dépendances et la définition d'une ou plusieurs fonctions. Ces dernières sont enregistrées auprès de DOH en vue de leur exécution.

Pour exécuter les tests contenus dans un module de tests, il faut lancer le fichier `runner.html` en spécifiant le nom du module à tester :

```
runner.html?testModule=exemple.tests.Math
```

La fonction `doh.register()` enregistre un ou plusieurs tests auprès de DOH. Elle accepte les paramètres suivants :

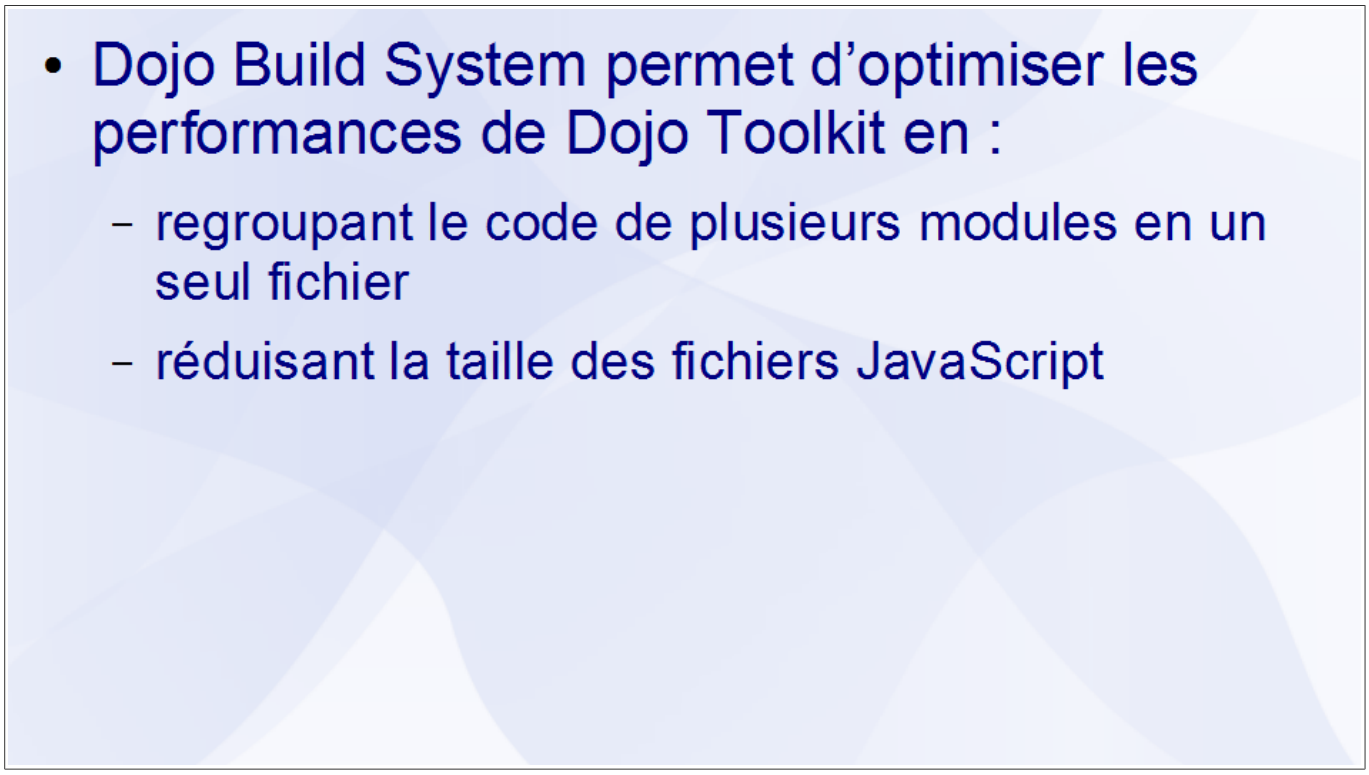
Paramètre	Type	Description
`groupOrNs`	`String`	Nom du groupe de tests.

`testOrNull`	`Object`
	`Array`
	`Function`
		Test ou ensemble de tests à enregistrer.

Les tests utilisent plusieurs fonctions de DOH appelées « assertions ». Une assertion est une condition qui doit toujours être vraie. Le non-respect d'une assertion entraîne l'échec du test. DOH définit les assertions suivantes :

Fonction	Description
`doh.assertTrue()`	Vérifie qu'une valeur est vraie.
`doh.assertFalse()`	Vérifie qu'une valeur est fausse.
`doh.assertError()`	Vérifie qu'une fonction lève une exception.
`doh.assertEqual()`	Vérifie que deux valeurs sont égales.
`doh.assertNotEqual()`	Vérifie que deux valeurs sont différentes.

Dojo Build System



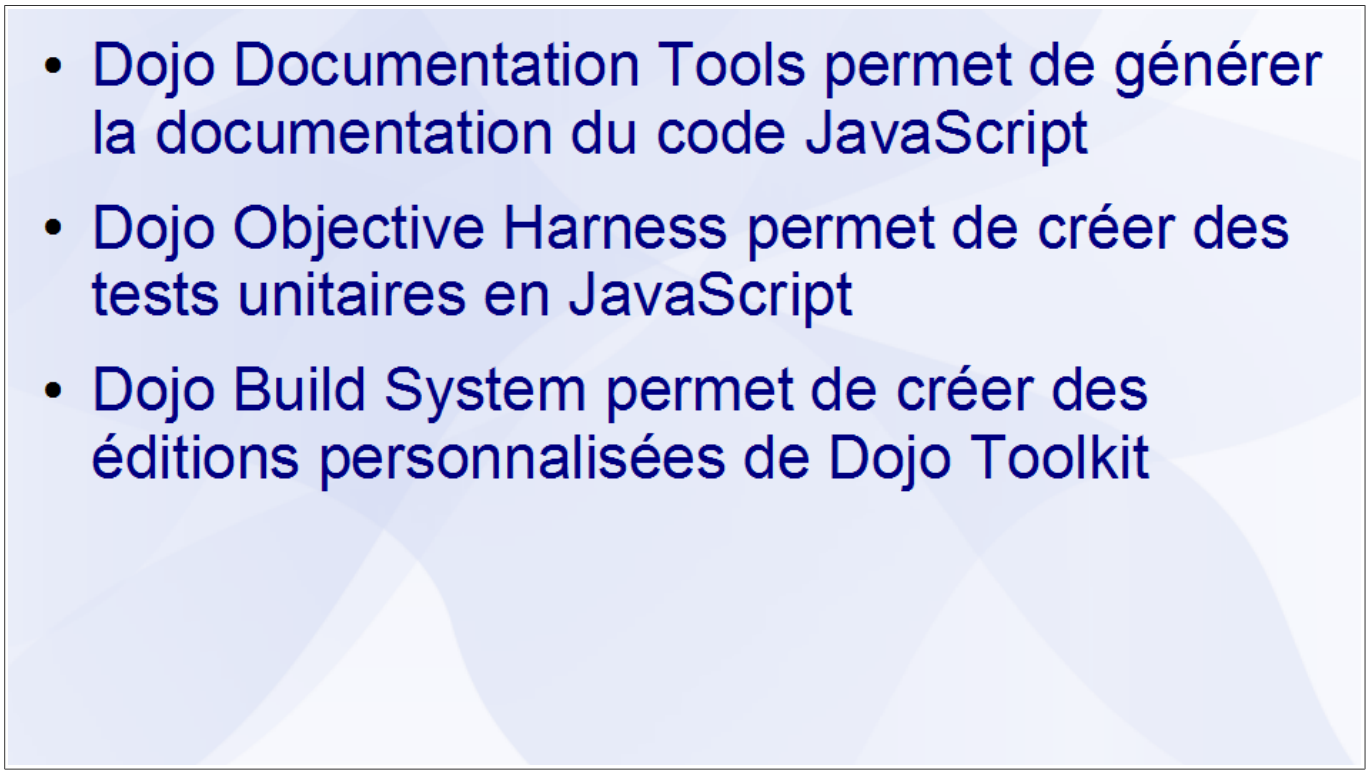
Dojo Build System est situé dans le sous-répertoire `buildscripts` du répertoire `util`. Il doit être utilisé depuis l'invite de commandes : `build.bat` sur Windows ou `build.sh` sur Linux.

```
shell> build action=clean,release profile=nomProfil
```

À la création d'une édition personnalisée de Dojo Toolkit, il faut spécifier un profil. Celui-ci définit quels sont les modules à regrouper au sein d'un même fichier. Le sous-répertoire `profiles` de `buildscripts` contient plusieurs exemples de profils dont on peut s'inspirer.

Il existe également une version Web appelée Dojo Toolkit Web Builder permettant de créer simplement des éditions personnalisées de Dojo Toolkit.

Conclusion


• Dojo Documentation Tools permet de générer la documentation du code JavaScript
• Dojo Objective Harness permet de créer des tests unitaires en JavaScript
• Dojo Build System permet de créer des éditions personnalisées de Dojo Toolkit