

Donner une dimension locale à une application Web avec Google Gears

Google est décidément une entreprise très prolifique en API de toutes sortes pour enrichir nos applications Web de demain. Une de ses dernières œuvres – et qui fait beaucoup parler d'elle – est Gears. De quoi s'agit-il exactement ? En quoi cela peut-il nous être réellement utile ? Explications.

Gears ("Mécanismes" en anglais) est un plug-in qui étend les fonctionnalités de nos navigateurs préférés. Ces fonctionnalités sont accessibles via une API JavaScript. Elles permettent de donner davantage d'autonomie à des applications Ajax. Il y a, pour l'essentiel, deux mécanismes nouveaux apportés par Gears :

- Un mécanisme de cache de ressources diverses
- L'installation d'une base de données embarquée.

Il existe un troisième mécanisme permettant d'exécuter des scripts de façon asynchrone. Ce mécanisme, qui présente de sérieuses limites d'intégration avec les autres ressources du navigateur, est surtout utile et utilisable pour piloter la synchronisation des données manipulées par les deux mécanismes principaux.

Installer Gears

Gears est un plug-in qui s'installe sur Microsoft IE 6 et 7 et sur Firefox 1.5 et plus. Les plates-formes actuellement prises en charge sont Windows XP/Vista, Mac OS, Linux. L'installation de Gears est très simple et il suffit de faire une petite visite chez Google à l'adresse suivante : <http://gears.google.com/>

La page affichée nous propose d'installer Gears. La sélection du bouton d'installation provoque le téléchargement d'un exécutable d'installation qu'il suffit d'invoquer. Le reste se passe sans aucune intervention de notre part, si ce n'est qu'il faut fermer le navigateur et le rouvrir pour constater que Gears est désormais présent.

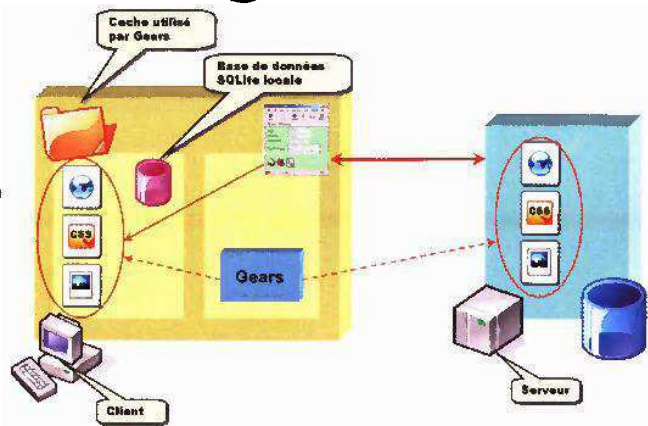
Sans autre intervention, Gears reste quasiment invisible. Il ne se signale que dans la liste des plug-in installés et par une boîte de dialogue de configuration qui indique quels sont les sites pour lesquels l'utilisation de Gears est autorisée (ou interdite). Pour le reste, le comportement de votre navigateur reste identique.

Utiliser Gears avec JavaScript

La richesse de Gears ne s'exprime que dans des applications Ajax. Gears propose une API JavaScript pour accéder aux mécanismes qu'il recèle. Nous allons découvrir cette API au travers de quelques exemples concrets.

Pour activer Gears dans notre page html, la première chose à faire est d'importer le script qui définit les API proposées par Gears :

```
<script type="text/javascript" src="gears_init.js"></script>
```



Mise en œuvre d'un cache

Le premier mécanisme de Gears est l'activation d'un cache de ressources récupérées depuis un serveur Web distant. Pour créer un cache de ressources, il faut inclure la snippet suivante :

```
<script>
var store ;
if (!window.google || !google.gears) {
  alert("Ooops ! Google Gears n'est pas installé.");
} else {
  var localServer = google.gears.factory.create(
    "beta.localserver", "1.1");
  store = localServer.createStore("myStore");
}
</script>
```

Quoique simples, ces quelques lignes méritent quelques explications :

- On commence par vérifier si Gears a été installé. Il suffit pour cela de tester l'existence et la valeur des attributs `window.google` et `google.gears`.
- Si Gears est présent, on construit d'abord un serveur local Gears (sur le navigateur, donc) qui prendra en charge le nouveau cache.
- Cet objet offre une API de manipulation des caches plutôt fournie. La méthode la plus utile reste "createStore" qui ouvre un cache s'il existe ou le crée s'il n'existe pas encore.

La création d'un cache se traduit par la création d'une arborescence dans le système de fichiers du poste client. L'emplacement où cette arborescence est créée dépend du système d'exploitation et du navigateur. Pour Firefox sur Windows XP, ce répertoire est :

C:\Documents and Settings\Henri\Local Settings\Application Data\Mozilla\Firefox\Profiles\s7wjd3h2.default\Google Gears for Firefox

Comme la gestion de ce cache est entièrement prise en charge par Gears, développeurs et utilisateurs peuvent ignorer sa structure interne.



Pour placer une ressource en cache, la méthode " capture " est disponible sur l'objet cache :

```
<script>
function captureCallback(url, success, captureId) {
    alert(url + 'captured: ' + success ? 'success' : 'failure');
}
var captureId = store.capture(
    'http:8080//localhost/myapp/myPage.html', captureCallback);
</script>
```

Cette méthode s'exécute de manière asynchrone, en tâche de fond. Dès que la ressource a été récupérée, elle est conservée localement. Cette opération finie, la fonction réflexe éventuellement associée à la capture est exécutée. A partir de ce moment-là, toute tentative pour accéder à cette ressource sera interceptée par Gears, qui servira la version en cache plutôt que d'aller la rechercher à nouveau sur Internet. Que se passe-t-il, donc, si la ressource sur Internet évolue ? Vous ne le verrez jamais, sauf si vous rappelez la méthode " capture " pour cette même ressource. Aucune synchronisation automatique n'est tentée par Gears. L'utilisation de cette version du cache est donc assez délicate. C'est à vous de créer la tâche de fond, en JavaScript, qui va périodiquement rafraîchir le contenu de votre cache. Mettre en œuvre une solution robuste de synchronisation est un exercice non trivial. Heureusement, Google y a pourvu. Il existe une version plus évoluée de cache Gears, que l'on peut obtenir en utilisant la méthode suivante du serveur local :

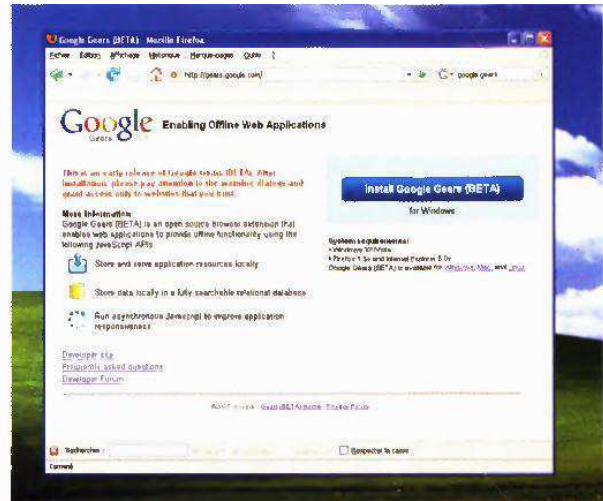
```
var store = localServer.createManagedStore("myStore");
store.manifestUrl = "http:8080//localhost/myapp/filesInCache.json";
store.checkForUpdate();
```

La méthode "createManagedStore" crée un nouveau cache " géré " s'il n'existait pas déjà et l'ouvre simplement s'il existe. Il n'y a pas de méthode " capture " ou équivalente pour cet objet. Ce cache utilise, à la place, un fichier de description du contenu appelé " manifeste ". Il dresse la liste de toutes les ressources qui doivent être téléchargées. Cette description est donnée au format JSON (JavaScript Object Notation).

Un tel fichier a l'aspect suivant :

```
filesInCache.json
{
  "betaManifestVersion": 1,
  "version": "v1.0.1",
  "entries": [
    { "url": "myPage.html" },
    { "url": "myScript.js" },
    { "url": "myImage.png" }
  ]
}
```

La méthode " checkForUpdate " provoque l'import ou le rafraîchissement explicite des ressources correspondant aux URLs. Notez que les ressources à importer peuvent s'exprimer sous la forme d'une URL absolue ou relativement à l'emplacement, sur le serveur, du fichier



manifeste. Si les ressources ont déjà été chargées une première fois localement, elles ne le seront à nouveau que si l'attribut " version " du manifeste évolue. Cette vérification - avec rafraîchissement éventuel - est effectuée régulièrement par Gears (c'est-à-dire sans appel explicite à " checkForUpdate ").

Gears propose donc une API simple et fonctionnelle pour bâtir et utiliser un cache. Il existe d'autres méthodes qui viennent compléter l'ensemble essentiel présenté dans ces lignes.

Mise en œuvre d'une base de données embarquée

Mais Gears propose mieux : une API qui permet de construire et d'exploiter une vraie base de données SQL locale. La base de données embarquée dans le plug-in installé par Gears est SQLite. C'est un moteur léger, performant et concis, implémentant une grande partie de SQL92 dont les transactions, les triggers, les jointures (même externes!). La base de données est matérialisée par un unique fichier (.db) qui sera placé - dans le cas de Gears - à la racine de l'arborescence des caches (voir plus haut).

Aucune installation préalable (création d'une base de données par exemple) n'est nécessaire. Tout se fera simplement à l'aide de l'API JavaScript de Gears.

```
var db = google.gears.factory.create('beta.database', '1.0');
db.open('mydb');
db.execute('create table if not exists DVD' +
    '(Title text, Duration int)');
db.execute('insert into DVD values (?, ?)',
    ['Deadwood Season 1', 640]);
db.execute('insert into DVD values (?, ?)',
    ['My beautiful Laundrette', 97]);
var rs = db.execute('select * from DVD order by Title');

while (rs.isValidRow()) {
    alert('Title : '+rs.field(0) + ' duration : ' + rs.field(1));
    rs.next();
}
rs.close();
```



Et c'est tout ! Toute l'API qu'il est nécessaire de connaître figure dans ces quelques lignes. Quelques explications complémentaires sont à peine nécessaires :

- la première ligne construit un gestionnaire de base de données.
- La seconde ligne ouvre une connexion vers la base de données "mydb.bd" et la crée si nécessaire.
- La création d'une table se fait directement en SQL en utilisant la directive "create table if not exists".
- Une seule méthode est utilisée pour passer tout type de requête : il s'agit de "execute". Elle permet à la fois de changer le schéma de la base de données, d'en enrichir ou modifier le contenu (insert, update, delete) et de rechercher des données déjà présentes (select). Elle permet de passer des paramètres aux requêtes. Ces paramètres sont donnés sous la forme de tableaux.
- Dans le cas où la requête soumise est une sélection, les résultats obtenus sont matérialisés par un objet ResultSet (rs dans l'exemple). Cet objet est un curseur qu'il est possible de parcourir (méthodes "isValidRow" et "next") pour extraire, sur chaque ligne, le contenu des champs retournés (méthode "field").

Il reste, bien entendu, à connaître la syntaxe et les possibilités offertes par SQLite : l'API proposée par Gears, elle, s'acquiert en quelques minutes.

Gears et le Google Web Toolkit

Si vous êtes, comme moi, peu à l'aise avec JavaScript, et que vous avez choisi le Google Web Toolkit pour développer vos applications Ajax, vous pouvez aussi intégrer Gears dans vos applications sans effort supplémentaire : Google propose un module GWT pour piloter Gears.

Ce module est disponible à l'adresse suivante :
<http://code.google.com/p/gwt-google-apis/>

Il s'agit d'un projet dans la mouvance du Google Web Toolkit, qui propose une intégration de différents projets Google. Pour l'instant, il ne propose que Gears. Il devrait s'enrichir dans les mois et les années qui viennent.

Le fichier à télécharger est "gwt-google-apis-x.y.z.zip" (ou tar.gz pour Linux). "x.y.z" est le numéro de version du module, 1.0.0 au moment où cet article est écrit. Décompressez-le dans votre environnement de développement. Sous la racine de l'arborescence décompressée, vous trouverez un fichier "gwt-google-api.jar". C'est lui qui doit être ajouté au CLASSPATH de votre application GWT pour qu'elle devienne "Gears powered" ! Il faut aussi déclarer ce module dans le fichier de configuration de votre application (.gwt.xml) sous la forme d'un héritage :

```
<module>
  <inherits name="com.google.gwt.user.User"/>
  <inherits name="com.google.gwt.gears.Gears"/>
  <entry-point class="com.objetdirect.myapp.client.MyModule"/>
</module>
```

Il ne reste plus qu'à faire appel aux méthodes de Gears dans votre application pour lui donner une nouvelle dimension locale. Voici un exemple d'utilisation du cache équivalent à ce qui a été présenté dans le paragraphe JavaScript :

```
LocalServer ls = new LocalServer();
ResourceStore rt = ls.createResourceStore("myStore");
rt.captureURLs(new String[] {
  "http:8080//localhost/myapp/myPage.html"
}, new URLLCaptureCallback() {
  public void onCaptureFailure(String url, int captureId) {
    Window.alert("unable to capture : "+url);
  }
  public void onCaptureSuccess(String url, int captureId) {
    Window.alert(url+" captured");
  }
});
```

La similitude avec les API JavaScript est frappante. Il en va de même pour la manipulation de base de données embarquée :

```
Database db = new Database("mydb");
db.execute("create table if not exists DVD ((Title text, Duration int));");
db.execute("insert into DVD values (?, ?)",
  new String[] {"Deadwood season 1", "640"});
db.execute("insert into DVD values (?, ?)",
  new String[] {"My beautiful Laundrette", "97"});
ResultSet rs = db.execute("select * from DVD");
for (;rs.isValidRow(); rs.next()) {
  Window.alert("Title : " + rs.getFieldAsString(0) +
    " duration : " + rs.getFieldAsInt(1));
}
```

En conclusion

Avec Gears, Google offre une extension aux applications Ajax qui leur permettent de s'exécuter avec un certain degré d'autonomie. Cette solution est simple, les API présentées dans cet article le démontrent. Gears ne propose pas de mécanisme qui rende l'exécution locale transparente : c'est aux développeurs de décider et d'implémenter la localité, point par point. C'est aussi eux qui doivent se charger des mécanismes de synchronisation avec le serveur. Pour cela, Google et Gears apportent des outils (ManagedResourceStore) et des idées (architectures présentées sur le site Gears).

Le projet Gears en est à ses débuts. La communauté des premiers utilisateurs est aujourd'hui très sollicitée pour donner son avis, faire connaître ses besoins. De substantielles évolutions devraient apparaître dans les mois qui viennent, en particulier sur le troisième service – l'exécution asynchrone de tâches JavaScript – qu'il n'a pas été possible de décrire dans cet article, faute de place.

Gears est donc un complément fort utile pour construire des applications qui doivent disposer d'un véritable environnement individualisé, comme un client mail, un agenda, un carnet d'adresses. Mais toutes les applications Ajax ne relèvent pas de cette logique.



■ Henri DARMET

Directeur Technique - Objet Direct / Homsys Group

Objet Direct, filiale à 100% de Homsys Group est une société de conseil, de services et de formation, spécialisée sur les technologies objet et Web. Conseil en méthodologie, en architecture et en urbanisation du SI, développement applicatif, édition et distribution de logiciels. www.objetdirect.com