

# Programmation GWT 2

Développer des applications  
HTML5/JavaScript en Java  
avec Google Web Toolkit

2<sup>e</sup> édition

**Sami Jaber**

© Groupe Eyrolles, 2012, ISBN : 978-2-212-13478-0

**EYROLLES**



# Avant-propos

---

## Pourquoi ce livre ?

Lorsque la plate-forme GWT est sortie en 2006, nous n'étions qu'une petite poignée à parier sur le potentiel de cette technologie. Les composants graphiques étaient austères, l'environnement de développement balbutiant et les performances plutôt médiocres.

C'est un peu par hasard que j'ai découvert la valeur réelle de GWT : en parcourant un jour le code source d'une classe égarée sur mon disque dur chargée de traduire le code Java en JavaScript ([JavaToJavaScriptCompiler](#)).

Dès lors, GWT ne m'a plus quitté. Il était clair pour moi que les jours des frameworks web Java tels que Struts ou JSF étaient comptés. Le concept de la compilation façon GCC adapté à Java et reléguant JavaScript au rang de vulgaire assembleur, la voie était toute tracée.

Les deux années qui ont suivi ont vu GWT gagner en maturité. Malgré les défauts de jeunesse de la version 1.x, les premiers projets ont été couronnés de succès. Jour après jour, les développeurs GWT gagnaient en productivité.

Au même rythme, les premiers ouvrages anglophones sont arrivés. Il en existe aujourd'hui pléthore couvrant la version 1.0. Reproduire en français de tels livres n'aurait eu aucun intérêt. On n'écrit pas un livre pour copier mais pour créer quelque chose. Sans compter qu'il ne faut jamais, c'est bien connu, se précipiter sur les premières versions d'un framework.

GWT ne fait pas exception à cette règle. La version 1.x a prouvé la viabilité du concept, mais a également mis en avant quelques limites. Lorsque j'ai découvert les premiers travaux autour de la version 2.0, j'ai tout de suite compris que GWT 2 ne ressemblerait plus jamais à GWT 1.x.

Il aura fallu quasiment deux ans à l'équipe des contributeurs GWT pour finaliser la version 2. Si les concepts des premiers jours restent d'actualité, de nouvelles fondations ont été bâties pour préparer les outils de demain. Avec GWT 2, les littératures francophones et anglophones actuelles devenaient de facto obsolètes. Une case se libérait dans l'amoncellement de livres dédiés à GWT. Plus de doute possible, il en fallait un sur le sujet !

À l'heure de la publication de cet ouvrage, il n'existait aucun équivalent anglais ou français.

## Dans quelles conditions a été écrit ce livre ?

Pendant plus de six mois, j'ai vécu (quelques soirs et week-ends) en immersion totale avec l'équipe GWT, un peu comme quelqu'un qui regarderait par dessus l'épaule d'un développeur pour formaliser les moindres lignes de code qu'il écrit. Au fur et à mesure que les choix de conception s'opéraient, j'écrivais. Et comme un développeur rature souvent, altère ou supprime son code, des chapitres entiers de ce livre ont été modifiés ou supprimés après leur finalisation. J'aurais sûrement pu écrire un second ouvrage avec toutes les pages supprimées ou modifiées.

Pendant six mois, je n'ai pu m'appuyer sur aucune documentation technique. Il n'y en avait simplement pas. Les quelques pages de wiki qui ornaient le site du projet étaient à moitié bâclées et souvent publiées trop en avance.

Et puis, j'ai appris à décrypter les cas de tests publiés au jour le jour sur le tronc SVN. J'ai appris à solliciter les développeurs, Joel, Bob, Bruce, John et Fred. Ils m'ont toujours aidé avec l'humilité et la modestie qui les caractérise tant.

J'ai aussi appris à critiquer le code au fur et à mesure qu'il se construisait, je me suis fait une opinion sur les développeurs et sur la qualité du code. Dans ce chemin semé d'embûches, j'ai dû faire face à d'innombrables bogues de jeunesse qui me handicapaient toujours plus dans la quête d'une finalisation qui s'éternisait au fil des évolutions du code source. Qu'importe, j'avais également appris à modifier le code source de GWT. Dans un premier temps, je le faisais pour les besoins du livre... puis, je l'ai fait pour la communauté, en soumettant quelques correctifs.

Lors de sa sortie, ce livre a été le premier au monde à traiter de GWT 2. Des livres en anglais existent aujourd'hui, mais celui-ci restera le premier livre sur GWT 2 avec ses qualités, mais aussi ses défauts.

J'espère que vous prendrez autant de plaisir à le lire que j'en ai eu à l'écrire.

## À qui s'adresse ce livre ?

Ce livre s'adresse à tous les développeurs, architectes, chefs de projet ou testeurs souhaitant comprendre les bénéfices de GWT.

Il n'est nul besoin d'être un expert en programmation ou un gourou des langages HTML ou JavaScript. Il suffit de connaître un minimum la syntaxe Java et la structure d'une page HTML. Si vous avez un profil plutôt débutant en Java, il faudra tout de même vous armer d'un peu de courage. Ce livre n'est pas à prendre comme un tutoriel dans lequel on explique pas à pas l'utilisation des concepts. C'est un ouvrage qui se veut le plus exhaustif possible et qui occulte volontairement certains éléments de mise en place.

Le chapitre 8 se focalise sur l'intégration avec les serveurs d'application existants, notamment Spring et les EJB. Si vous ne connaissez pas du tout cette partie de Java, n'hésitez pas à vous aider des nombreux tutoriels disponibles sur le Web.

## Structure de l'ouvrage

Le point essentiel de ce livre est qu'il n'y a aucun passage obligé. Excepté les trois premiers chapitres, qui posent les bases du framework et présentent les outils, tous les autres chapitres peuvent être lus dans le désordre.

Les **chapitres 1, 2 et 3** abordent la partie visible de GWT, les composants, les outils, l'environnement de développement et le nouveau modèle de placement CSS.

Le **chapitre 4** est un tour d'horizon des différentes bibliothèques du marché.

L'intégration JavaScript (**chapitre 5**) est traitée comme préalable à l'implémentation de composants personnalisés (**chapitre 6**).

Les deux chapitres sur l'architecture des services RPC (**chapitre 7**) vous montrent comment exploiter efficacement la communication avec le serveur et intégrer des services existants Spring, EJB3 (**chapitre 8**)...

Le chargement à la demande (**chapitre 9**) est une des grosses nouveautés de GWT 2. Ce chapitre en illustre le principe à travers un exemple concret et vous présente un design pattern permettant de se prémunir contre ses pièges.

Le **chapitre 10** sur la liaison différée est complexe à comprendre, mais important pour bien saisir les fondements de GWT et la manière dont les spécificités des différents navigateurs sont couvertes. Les débutants pourront faire l'impasse sur ce chapitre et y revenir dès qu'ils seront plus aguerris.

Les chapitres suivants, que ce soit la gestion des ressources (**chapitre 11**), l'internationalisation (**chapitre 13**) ou les tests (**chapitre 14**), peuvent être parcourus dans n'importe quel ordre.

Le **chapitre 15** sur les design patterns traite de sécurité, de gestion des sessions et de bonnes pratiques de conception. Quant au **chapitre 12**, il met en lumière les innombrables vertus du compilateur et des optimisations GWT. Ces deux chapitres sont une source d'informations précieuses, même pour ceux ayant déjà une première expérience GWT.

Le livre se poursuit sur le **chapitre 16** qui décrit UIBinder, l'une des grandes nouveautés de GWT 2, qui s'appuie intensivement sur le mécanisme des ressources, l'internationalisation et la liaison différée.

C'est le **chapitre 17** qui est dédié au plug-in Eclipse de GWT. Cela est essentiellement lié au fait que nous souhaitons vous présenter en priorité... tout ce qu'il est censé vous masquer.

Le **chapitre 18** traite de l'API CellWidget, le nouveau modèle de composant introduit dans GWT 2.1. Le **chapitre 19** couvre un sujet brûlant de GWT : le framework « Activity and Places », plus connu sous le terme « MVP ». Précédant le **chapitre 21** sur le framework Editors, le **chapitre 20** aborde la nouvelle approche d'exposition de services appelée « RequestFactory ».

#### FORMATION GWT proposée par DNG Consulting

Une très grande partie du contenu technique de ce livre s'appuie sur la formation GWT 2 proposée par DNG Consulting. Si vous souhaitez creuser le sujet de manière plus interactive, nous vous encourageons vivement à faire partie des centaines de stagiaires formés depuis nos débuts avec GWT.

## Remerciements

Il est coutume de dire que l'écriture est un exercice solitaire, c'est vrai. Néanmoins, ce livre n'aurait jamais pu voir le jour sans l'aide précieuse des quelques personnes que je souhaite remercier ici. Tout d'abord, l'équipe éditoriale Eyrolles pour son professionnalisme et pour avoir accepté de me publier dans des conditions parfois difficiles : Muriel, Sophie, Anne-Lise, Pascale, Gaël et Éric.

Mes remerciements vont aussi à mes deux amis et précieux relecteurs, Romain Hochedez, consultant pour DNG Consulting et animateur de la formation GWT, et Jan Vorwerk, développeur GWT. Grâce à son œil de lynx, Romain m'a aidé à améliorer la qualité de certains chapitres, à éviter certains pièges. Jan m'a apporté ses pré-

cieux conseils techniques et pédagogiques. Il faut être plus qu'un vrai passionné pour accepter de passer soirées et week-ends à relire des manuscrits parfois indigestes comme ce fut le cas avec mes brouillons.

Il est difficile de ne pas remercier ceux qui ont créé GWT alors qu'ils n'étaient pas encore employés chez Google : Bruce Johnson et son équipe. La qualité principale de Bruce est d'avoir su instaurer autour de ce projet une convivialité quasi familiale. Que ce soit Joel, Ray Ryan, John, Fred ou Bob, tous m'ont aidé à corriger les bogues qui m'empêchaient d'avancer correctement dans ce livre. S'ils ne sont pour la plupart plus chez Google aujourd'hui, tous gardent un œil averti sur l'évolution de GWT.

Je remercie également les clients de DNG Consulting qui nous font confiance depuis plus de six ans pour les accompagner dans leurs projets GWT. Je dois l'expérience de ce livre à toutes les formidables applications que DNG a créé durant cette période.

Le dernier mot, enfin, est pour ma famille. Je ne souhaite à personne d'avoir à rédiger un livre de plus de 500 pages en trois mois tout en assurant la direction quotidienne d'un cabinet de conseil. Anatole France disait : « Il n'est pas d'amour qui résiste à l'absence ». Stéphanie, Yanis et Alicia m'ont prouvé le contraire. Sans eux, vous n'auriez jamais eu ce livre entre les mains.

Sami Jaber

# Table des matières

---

<b>Introduction à GWT .....</b>	<b>1</b>
GWT et HTML 5 .....	3
Dix lignes de code GWT pour convaincre .....	4
Masquer la complexité du Web .....	4
Coder en Java .....	5
<i>Typage statique</i> .....	5
<i>Débogage</i> .....	6
<i>Refactoring</i> .....	6
<i>Tests unitaires</i> .....	7
<i>Compétences largement disponibles</i> .....	7
Support multinavigateur .....	8
Performances .....	9
Qu'est-ce qu'Ajax ? .....	11
La navigation en mode SPI .....	13
L'architecture RPC .....	14
Modèle 1.0 versus modèle 2.0 .....	14
GWT face aux autres frameworks Ajax .....	16
Quelle place pour GWT face à ses concurrents ? .....	17
Un projet collaboratif .....	18
Une communauté active .....	19
Des navigateurs de plus en plus performants .....	20
L'évolution et les nouveautés de GWT 2 .....	20
CHAPITRE 1	
<b>L'environnement de développement .....</b>	<b>23</b>
Télécharger et installer GWT .....	23
Contenu du répertoire d'installation .....	23
L'ensemble logiciel GWT .....	24
Création du premier projet GWT .....	24
Exécuter l'application .....	26
Notion de module .....	27

<b>Structure d'un projet GWT</b> .....	28
Le package client .....	29
Le package serveur .....	30
Les fichiers de configuration .....	30
La structure du répertoire war .....	31
La page HTML hôte .....	31
<b>Le mode développement</b> .....	33
<b>Le shell</b> .....	36
Le conteneur de servlets Jetty .....	37
<b>Le mode production</b> .....	38
La structure d'un site compilé .....	39
<b>Les types Java émülés par GWT</b> .....	40
<b>Le déploiement</b> .....	42
Fichier Ant .....	42
Plug-in Maven .....	43
La fonctionnalité « Super DevMode » dans GWT 2.5 .....	45

## CHAPITRE 2

### **Les contrôles** ..... **47**

<b>Les classes UIObject et Widget</b> .....	47
<b>Les feuilles de styles CSS</b> .....	49
La syntaxe .....	50
Les styles dépendants .....	51
Les styles prédéfinis .....	52
<b>La gestion des événements</b> .....	54
<b>Tour d'horizon des widgets</b> .....	56
Les composants de formulaires .....	56
SuggestBox .....	59
Les bundles d'images .....	60
Les hyperliens .....	63
<b>Les conteneurs et gestionnaires de placement</b> .....	64
Les conteneurs simples (Panels) .....	64
<i>FormPanel</i> .....	65
<i>LazyPanel</i> .....	66
Les conteneurs complexes .....	67
<i>Exemple d'utilisation</i> .....	68
<i>HTMLPanel</i> .....	70
Synthèse des conteneurs GWT .....	72



## CHAPITRE 3

**Le modèle de placement CSS..... 75**

Pourquoi un modèle de placement ? .....	75
Les limites du modèle GWT 1.x .....	76
Une solution basée sur le standard CSS .....	79
Les API .....	82
Les nouveaux conteneurs de GWT 2 .....	84
Composant StackLayoutPanel .....	85
Le widget TabLayoutPanel .....	86
Sous le capot .....	86

## CHAPITRE 4

**Les bibliothèques tierces..... 89**

L'écosystème GWT .....	89
Les bibliothèques de composants graphiques .....	90
L'incubateur GWT .....	90
Sencha Ext-GWT (GXT v3) .....	90
SmartGWT .....	96
Glisser-déplacer avec GWT-DnD .....	100
Les courbes et graphiques .....	102
<i>GChart</i> .....	103
<i>GWT HighCharts</i> .....	104
Les frameworks complémentaires .....	105
Vaadin .....	105
La gestion des traces .....	106
Manipuler les services Google avec GWT .....	107
Gwt-google-apis .....	107
<i>GWT-GData</i> .....	109
Conclusion .....	113

## CHAPITRE 5

**L'intégration de code JavaScript..... 115**

Comprendre JSNI .....	115
Mise en pratique .....	116
Intégration d'un fichier JavaScript externe .....	118
Invoquer une méthode Java en JavaScript .....	119
Accéder à des attributs Java en JavaScript .....	121
Correspondance des types entre Java et JavaScript .....	122
Instancier un type Java en JavaScript .....	123
Le type JavaScriptObject (JSO) .....	124

Undefined vs null .....	126
Gestion des exceptions JSNI .....	127
Appeler une méthode Java à partir d'un code JavaScript externe .....	128
Les types Overlay .....	128
Un peu d'histoire .....	129
Mise en pratique des Overlay .....	131
Intégration Overlay et JSON .....	134
Sous le capot .....	135
L'implémentation unique du type JSO .....	138
Les contraintes associées à un JSO .....	140
Effet de JSNI sur le framework GWT .....	140
La magie interne de JSNI .....	141

## CHAPITRE 6

### La création de composants personnalisés..... 143

Quelques mots sur le DOM .....	143
La mécanique des événements .....	146
Pourquoi JavaScript fuit-il ? .....	146
Propagation par bouillonnement et capture .....	147
Expando et fuite mémoire .....	149
Créer un composant dérivé de Widget .....	152
Aller plus loin avec l'API événementielle .....	157
Dériver de la classe Composite .....	158
Dériver de la classe UIObject .....	161
Attachement dans un conteneur .....	161

## CHAPITRE 7

### Les services RPC..... 163

L'architecture RPC .....	164
Les étapes de la construction d'un service .....	165
Créer les deux interfaces de service .....	166
Créer les objets d'échange .....	168
Coder l'implémentation .....	169
Coder et configurer le client .....	170
La sérialisation .....	172
La gestion des exceptions .....	173
Exceptions non vérifiées .....	175
Accès à la requête HTTP .....	176
Bonnes pratiques et mode asynchrone .....	177
Déploiement .....	178

Déploiement des classes .....	179
Configuration du fichier web.xml .....	179
Configuration dans un réseau sécurisé avec un frontal web .....	179
<b>Classe RequestBuilder et services REST .....</b>	<b>181</b>
Appel d'URL basique .....	181
Architecture REST .....	186
<b>CHAPITRE 8</b>	
<b>L'intégration J2EE.....</b>	<b>187</b>
Le modèle par délégation .....	187
Le modèle d'extensibilité .....	189
L'option <code>-noserver</code> .....	193
Intégration avec les EJB 3 et JPA .....	193
Le problème des classes instrumentées .....	202
Intégration des protocoles RMI, Corba et Soap .....	207
<b>CHAPITRE 9</b>	
<b>Le chargement à la demande .....</b>	<b>209</b>
Principe général .....	209
Les types de fragments .....	213
Positionner efficacement les points de rupture .....	218
Le design pattern Async package .....	223
Forcer le chargement des fragments .....	225
CodeSplitting V2 .....	227
Sous le capot .....	229
Conclusion .....	231
<b>CHAPITRE 10</b>	
<b>La liaison différée .....</b>	<b>233</b>
Principe général .....	233
Mise en pratique .....	236
Le script de sélection .....	239
Propriétés, règles et conditions .....	239
Propriétés .....	240
Propriétés de configuration .....	242
Les propriétés conditionnelles .....	243
Règles de liaison .....	244
Générateurs de code .....	246
Dans la pratique .....	247
Déboguer .....	251

Conditions de liaison .....	252
Conclusion .....	252

## CHAPITRE 11

### **La gestion des ressources ..... 253**

La problématique des ressources .....	253
Installation et configuration .....	255
Les différents types de ressources .....	256
Les ressources textuelles (TextResource) .....	257
Les ressources textuelles asynchrones .....	259
Les ressources binaires externes .....	263
Les ressources images .....	264
Les options de la liaison différée .....	267
L'injection dynamique CSS .....	267
L'injection différée .....	269
Les constantes .....	270
La substitution à l'exécution .....	271
Les fonctions de valeur .....	272
Les directives conditionnelles .....	274
Les préfixes de style .....	276
Les sprites d'images .....	277

## CHAPITRE 12

### **Sous le capot de GWT ..... 279**

Introduction au compilateur .....	280
Vive les fonctions JavaScript ! .....	280
Les étapes du compilateur .....	283
Lecture des informations de configuration .....	284
Création de l'arbre syntaxique GWT .....	284
La création de code JavaScript et les optimisations .....	285
<i>La réduction de code (pruning)</i> .....	288
<i>La finalisation de méthodes et de classes</i> .....	290
<i>La substitution par appels statiques.</i> .....	290
<i>La réduction de type</i> .....	291
<i>L'élimination de code mort</i> .....	292
<i>L'inlining</i> .....	292
Tracer les optimisations .....	293
Les options du compilateur .....	295
Accélérer les temps de compilation .....	299
Les linkers .....	299

La pile d'erreurs en production .....	305
Table des symboles .....	309

## CHAPITRE 13

### **L'internationalisation ..... 311**

La problématique .....	311
Paramétrer et définir la locale courante .....	312
L'API i18n .....	313
Les dictionnaires à constantes statiques .....	314
Dictionnaire par recherche dynamique de constantes .....	316
Les messages .....	317
Notion de langue par défaut .....	318
Signification, exemple et description .....	319
Les formes plurielles .....	320
Conversion des types .....	322
Formats monétaires .....	322
Date et formats horaires .....	323
Création automatique de dictionnaires .....	325
Bénéfices de l'internationalisation statique .....	326
Externalisation dynamique .....	327
L'outillage .....	328
i18nCreator .....	328
I18nSync .....	329

## CHAPITRE 14

### **L'environnement de tests ..... 331**

GWT et la problématique des tests .....	331
La mixité des tests .....	332
Créer un test unitaire .....	332
Les suites de tests .....	335
Une architecture modulaire et extensible .....	336
Le style HtmlUnit – moteur de test par défaut .....	338
Le style manuel ou interactif .....	340
Le style Selenium .....	340
Le style distant .....	342
Le style externe .....	342
Synthèse des différentes options et annotations .....	343
Tests de charge avec la classe Benchmark .....	343
Les compteurs intégrés de performance .....	346
Tests fonctionnels robotisés : scénarios joués .....	348

Selenium IDE .....	349
<i>Le module WebDriver</i> .....	354
Les stratégies de test par bouchon (mocking) .....	356
Quel est l'atelier de tests idéal ? .....	359

## CHAPITRE 15

### Les design patterns GWT..... 361

Pourquoi des bonnes pratiques ? .....	361
Gestion de la session (cliente et serveur) .....	362
Limiter les besoins mémoire de la session cliente .....	363
La gestion côté serveur .....	364
Session et onglets des navigateurs .....	365
Gestion de l'historique .....	367
Que signifie le contexte précédent avec Ajax ? .....	370
Les traitements longs .....	372
La classe Timer .....	373
La classe Scheduler .....	374
<i>Les traitements différés</i> .....	374
<i>Les traitements incrémentaux</i> .....	375
Séparer présentation et traitement .....	378
Le pattern Commande .....	378
L'approche Modèle Vue Contrôleur .....	381
<i>MVC par l'exemple avec le framework PureMVC</i> .....	382
<i>Le pattern MVP</i> .....	384
Les failles de sécurité .....	386
Injection SQL .....	386
Cross-site Scripting (XSS) .....	387
CSRF (Cross-Site Request Forgery) .....	390
Les autres attaques .....	392
L'authentification .....	392
Authentification Basic et Digest .....	392
Authentification par formulaire .....	393
Les limites de la session HTTP par cookies .....	395

## CHAPITRE 16

### La création d'interfaces avec UIBinder ..... 397

Présentation .....	398
Styles et ressources .....	402
Incorporation d'images .....	408
Intégration des ressources de type données .....	409

Gestionnaires d'événements .....	410
Intégration d'un flux HTML standard .....	413
Internationalisation .....	414
Les emplacements .....	416
<i>Cas des balises imbriquées</i> .....	418
Traduire les attributs .....	419
Liaison avec des beans externes .....	420
Modèles composites et constructeurs .....	424
CHAPITRE 17	
<b>Le plug-in Eclipse pour GWT .....</b>	<b>429</b>
Le cas AppEngine .....	429
Le plug-in GWT .....	430
Création d'un projet GWT .....	430
Les assistants de création .....	433
Création d'un point d'entrée .....	433
Création d'un nouveau module .....	433
Création d'une page HTML hôte .....	434
Création d'un squelette ClientBundle .....	435
Création d'un squelette UIBinder .....	435
Aide à la saisie de code JSNI .....	437
Assistants RPC .....	438
CHAPITRE 18	
<b>Les composants CellWidget .....</b>	<b>439</b>
Philosophie de ce nouveau modèle de composants .....	439
Utilisation du modèle de données .....	443
Le pattern Appearance pour le rendu graphique .....	445
Modèle de présentation : SafeHtmlTemplate .....	448
Les autres composants CellWidget .....	450
DataGrid et CellTable .....	451
Chargement des données de manière asynchrone .....	453
Mise à jour des données et gestion des événements cellule .....	454
Gestion de la sélection .....	458
CHAPITRE 19	
<b>Activités et places .....</b>	<b>459</b>
Objectif et philosophie .....	459
Les notions .....	460
Les emplacements .....	460

Les activités .....	460
Les vues .....	461
Le contrôleur d'emplacement (PlaceController) .....	461
Le dictionnaire d'activités (ActivityMapper) .....	462
Le bus d'événements (EventBus) .....	462
Activity and Places par l'exemple .....	462
L'application Gestion des courriers .....	464
Communication entre vues et bus d'événements .....	475
À ne pas mettre entre toutes les mains .....	476

## CHAPITRE 20

### **L'API Request Factory..... 479**

Objectifs .....	479
La requête et son contexte côté client .....	480
Premiers pas avec l'API .....	480
Les entités .....	481
Les classes proxies entités .....	483
<i>Les proxies de valeur</i> .....	484
RequestFactory et l'interface des requêtes .....	484
Intégration des services et localisateurs .....	485
Utilisation côté client et receveurs .....	489
Création et modification d'un objet .....	492
Validation et JSR 303 .....	492
Les pièges à éviter .....	494
L'API AutoBean .....	495
Autres fonctionnalités avancées .....	497

## CHAPITRE 21

### **L'API Editors..... 499**

Objectifs et concepts .....	499
Modèle de fonctionnement .....	500
Les délégués .....	503
La gestion des collections d'objets .....	505
Gestion de la validation .....	509

### **Index..... 511**



# 1

## L'environnement de développement

---

L'environnement de développement GWT constitue l'une des originalités de ce framework et le moteur principal de la productivité du développeur.

Ce chapitre traite des deux modes développement et production tout en s'attachant à décrire les différentes étapes de l'installation et de la configuration d'un projet GWT.

### Télécharger et installer GWT

GWT est fourni sous la forme d'un simple fichier ZIP téléchargeable en ligne à l'adresse suivante : <http://code.google.com/intl/fr/webtoolkit/download.html>

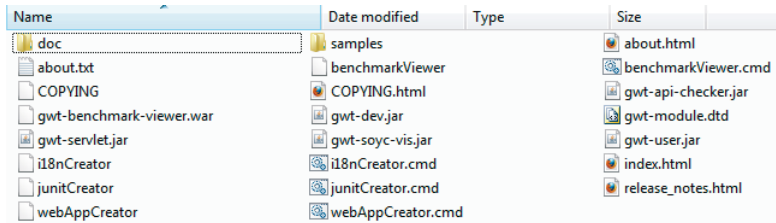
GWT est proposé sous la forme d'une distribution portable sur toutes les plateformes. Seuls les plug-ins sont dépendants des navigateurs et systèmes d'exploitation utilisés.

### Contenu du répertoire d'installation

Une fois téléchargé et décompressé, GWT se présente sous la forme d'un simple répertoire illustré dans la copie d'écran suivante. Ce qui frappe au premier abord est la simplicité de la structure. On trouve la documentation javadoc, les exemples d'uti-

lisation, quelques scripts shell et les différentes bibliothèques qui composent le framework. Par la suite, nous nous appuyerons sur le script `webAppCreator.cmd` pour la création de notre premier projet.

**Figure 1-1**  
Structure du répertoire GWT



Name	Date modified	Type	Size
doc			
about.txt			
COPYING			
gwt-benchmark-viewer.war			
gwt-servlet.jar			
i18nCreator			
junitCreator			
webAppCreator			
samples			
benchmarkViewer			
COPYING.html			
gwt-dev.jar			
gwt-soyc-vis.jar			
i18nCreator.cmd			
junitCreator.cmd			
webAppCreator.cmd			
about.html			
benchmarkViewer.cmd			
gwt-api-checker.jar			
gwt-module.dtd			
gwt-user.jar			
index.html			
release_notes.html			

## L'ensemble logiciel GWT

GWT est constitué de deux fichiers JAR principaux, auxquels s'ajoutent quelques JAR annexes :

- `gwt-dev.jar`
- `gwt-user.jar`

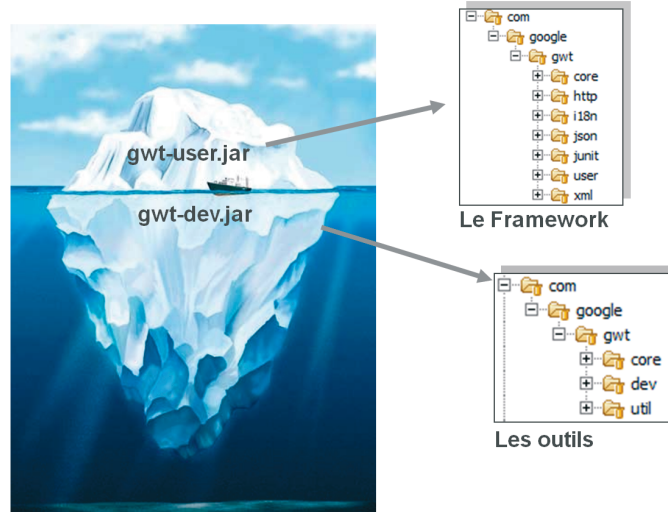
Nous reviendrons par la suite sur le contenu de ces fichiers mais à ce stade de l'ouvrage, il faut simplement savoir que `gwt-user` contient toute la partie framework de GWT (les *widgets*, les classes utilitaires, etc.) nécessaire en phase de développement. Ce JAR ne contenant aucune bibliothèque spécifique à un système d'exploitation (`.dll`, `.so`, ...), il en existe une version pour toutes les plates-formes.

`gwt-dev` contient quant à lui l'ensemble des outils utilisés par GWT, du compilateur Java vers JavaScript à l'émulation des tests, en passant par l'environnement de développement. Le schéma suivant illustre ces deux JAR sous l'angle d'un iceberg. La partie visible est `gwt-user`, la partie cachée à l'utilisateur est `gwt-dev`.

## Création du premier projet GWT

`WebAppCreator` est un script proposé par GWT et qui a pour objectif de créer un squelette de projet prêt à l'emploi. En fonction du type d'utilisation (sous Eclipse, Maven, etc.), `webAppCreator` propose l'option `-templates` définissant le modèle du projet créé. Les modèles par défaut sont `sample`, `ant`, `eclipse` et `readme`. Ce squelette comprend un exemple de code, des fichiers de construction Ant, un fichier d'extension `.launch` et un module (`sample`) déjà paramétré (nous reviendrons sur la notion de module un peu plus loin dans ce chapitre).

**Figure 1-2**  
Les deux archives gwt-user.jar  
et gwt-dev.jar



**REMARQUE** Que contient le fichier gwt-servlet.jar ?

L'archive `gwt-servlet.jar` contient les API des services RPC. C'est une sorte de package modulaire destiné à être copié dans le répertoire `WEB-INF/lib`.

Pour créer un projet Eclipse, nous lançons `WebAppCreator` avec les options indiquées dans la figure suivante (bien veiller à ce que le nom du module corresponde au répertoire de destination utilisé comme projet Eclipse).

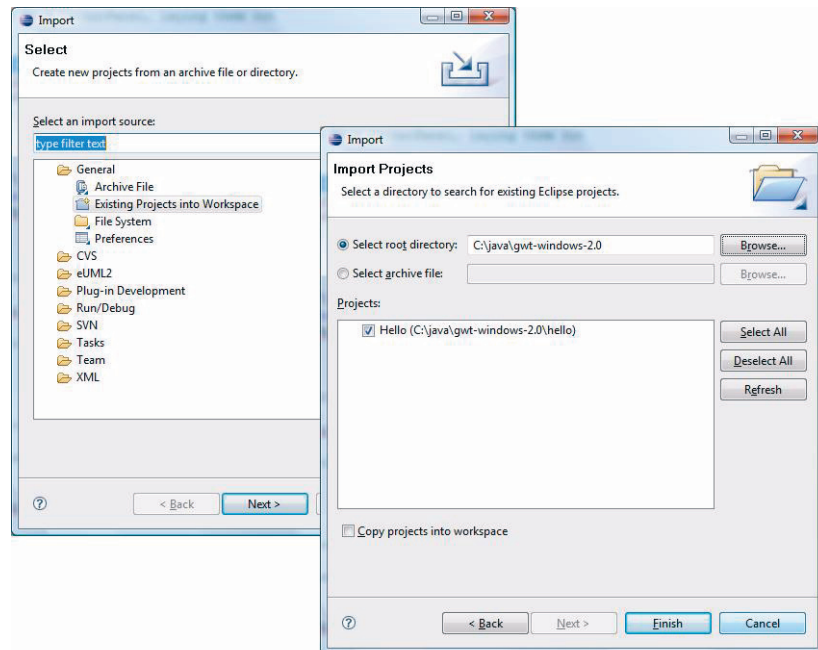
**Figure 1-3**  
L'outil webAppCreator

```
Administrator: C:\Windows\system32\cmd.exe
d:\java\gwt-2.6\gwt-2.6.D>webAppCreator
Missing required argument 'moduleName'
Google Web Toolkit 2.6.0
WebAppCreator [-overwrite] [-ignore] [-templates template1,template2,...] [-out dir] [-junit pathToJUnitJar] [-maven] [-noant] moduleName
where
  -overwrite Overwrite any existing files
  -ignore Ignore any existing files; do not overwrite
  -templates Specifies the template(s) to use (comma separated). Defaults to 'sample,ant,eclipse,readme'
  -out The directory to write output files into (defaults to current)
  -junit Specifies the path to your junit.jar (optional)
  -maven Deprecated. Create a maven2 project structure and pom file (default disabled). Equivalent to specifying 'maven' in the list of
  -noant Deprecated. Do not create an ant configuration file. Equivalent to not specifying 'ant' in the list of templates.
  and
  moduleName The name of the module to create (e.g. com.example.myapplication)
d:\java\gwt-2.6\gwt-2.6.D>webAppCreator -out c:\hello com.dngconsulting.hello.Hello
Generating from templates: [sample, eclipse, readme, ant]
Not creating tests because -junit argument was not specified.
Created directory c:\hello
Created directory c:\hello\src
Created directory c:\hello\src\com\dngconsulting\hello
Created directory c:\hello\src\com\dngconsulting\hello\client
Created directory c:\hello\src\com\dngconsulting\hello\server
Created directory c:\hello\src\com\dngconsulting\hello\shared
Created directory c:\hello\test
Created directory c:\hello\test\com\dngconsulting\hello
Created directory c:\hello\war
Created directory c:\hello\war\WEB-INF
Created file c:\hello\src\com\dngconsulting\hello\Hello.gwt.xml
Created file c:\hello\src\com\dngconsulting\hello\client\GreetingService.java
Created file c:\hello\src\com\dngconsulting\hello\client\GreetingServiceSync.java
Created file c:\hello\src\com\dngconsulting\hello\client\Hello.java
Created file c:\hello\src\com\dngconsulting\hello\server\GreetingServiceImpl.java
Created file c:\hello\src\com\dngconsulting\hello\shared\IntegerFieldVerifier.java
Created file c:\hello\war\WEB-INF\web.xml
Created file c:\hello\war\Hello.css
Created file c:\hello\war\Hello.html
Created file c:\hello\classespath
Created file c:\hello\project
Created file c:\hello\hello.launch
Created file c:\hello\HelloTest.txt
Created file c:\hello\build.xml
d:\java\gwt-2.6\gwt-2.6.D>
```

Le nom du module GWT est pleinement qualifié et contient un nom de package complet. Le nom du projet Eclipse créé par [WebAppCreator](#) reprend celui du module. La notion de module est abordée dans le paragraphe suivant.

Pour exécuter ce projet GWT sous Eclipse, il suffit simplement d'importer un projet existant dans l'espace de travail comme illustré par la figure suivante.

**Figure 1-4**  
Import d'un projet GWT



## Exécuter l'application

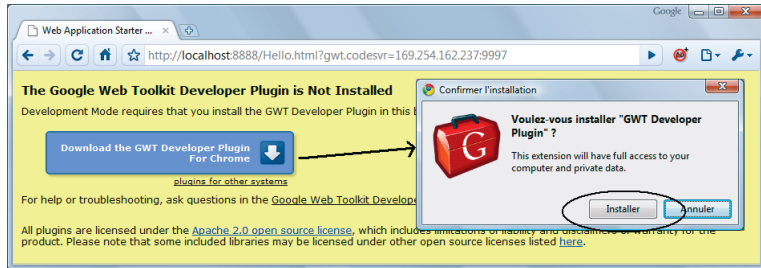
Nous reviendrons un peu plus loin sur la structure projet créée par [webAppCreator](#). Avant cela, nous allons lancer notre première application GWT. Il suffit de se positionner sur le fichier d'extension `.launch` et de sélectionner `run` ou `exécuter` en fonction de votre version d'Eclipse. Une fenêtre apparaît et un message nous incite à lancer un navigateur pointant vers une URL donnée :

```
http://localhost:8888/Hello.html?gwt.codesvr=169.254.162.237:9997
```

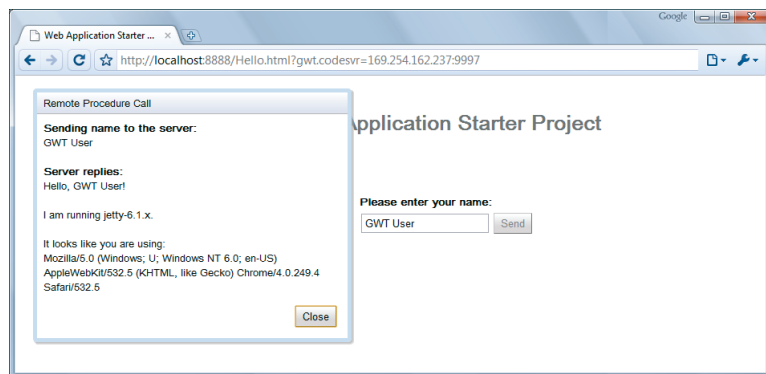
Comme c'est la première fois, aucun plug-in GWT n'est installé sur notre ordinateur. Le message suivant apparaît sous Chrome (identique quel que soit le navigateur).

Nous installons le plug-in puis rafraîchissons la fenêtre du navigateur. Notre fameuse application d'exemple s'exécute sous nos yeux ébahis.

**Figure 1-5**  
Installation du plug-in GWT  
pour Google Chrome



**Figure 1-6**  
Première application GWT



Essayons maintenant de comprendre comment tout cela s'articule.

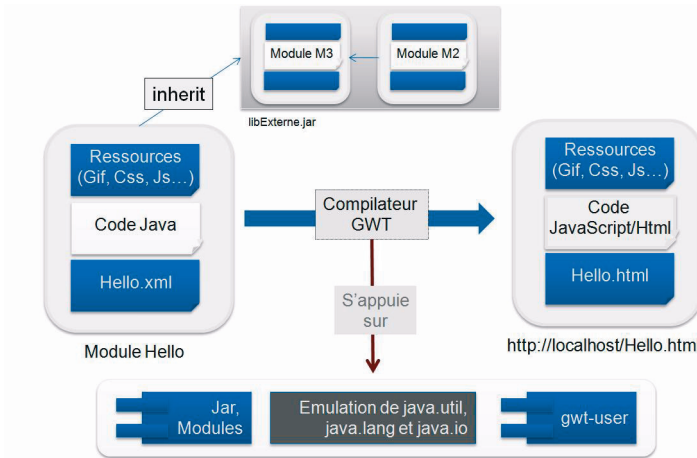
## Notion de module

Il est primordial de maîtriser la notion de module dans GWT. Sur le même principe qu'un fichier JAR dans le monde Java, un module est un élément primaire de configuration dans GWT.

Contrairement à une idée reçue, un module n'est pas nécessairement un projet Eclipse, ni forcément un fichier JAR. Un module est identifié par le nom complètement qualifié du package dans lequel il se trouve, associé au nom du module.

Il peut exister plusieurs modules par projet Eclipse, mais également plusieurs modules par archive JAR. À titre d'exemple, le fichier `gwt-user.jar` constituant le framework GWT de base contient lui-même plus d'une vingtaine de modules.

**Figure 1-7**  
Cycle de compilation



Maintenant que la notion de module a été abordée, voyons la structure d'un projet GWT telle que le crée l'outil [WebAppCreator](#).

## Structure d'un projet GWT

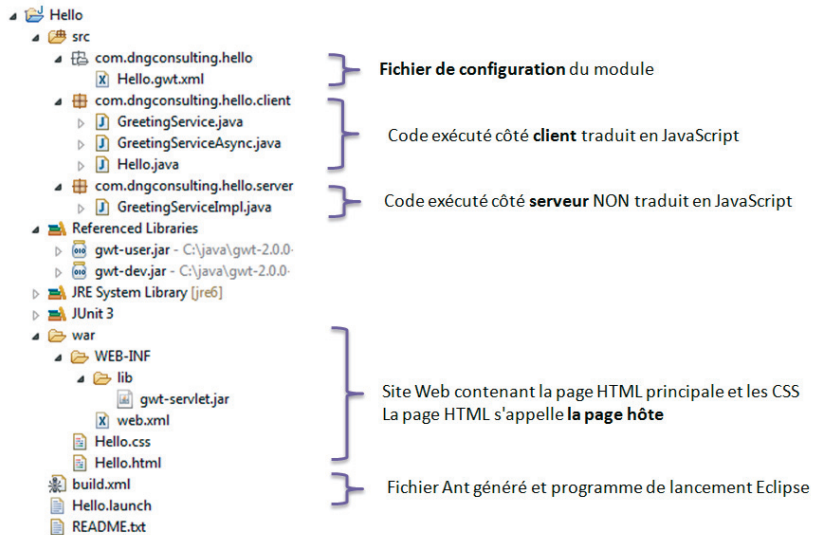
Même s'il est coutume d'affirmer qu'on développe avec GWT comme en Java, un projet GWT, contrairement à un projet Java classique, possède une structure très particulière qui reprend celle du module.

Cela est d'autant plus compréhensible qu'une partie du code est destinée à passer sous la moulinette d'un compilateur Java vers JavaScript. Il faut donc délimiter les portions du projet de nature à être traduites en JavaScript de celles qui resteront sur le serveur.

On peut globalement identifier quatre délimitations logiques dans un projet GWT :

- le code client (ou partagé entre client et serveur) ;
- le code serveur ;
- les fichiers de configuration de module ;
- le répertoire `WAR`.

**Figure 1–8**  
La structure  
d'un projet GWT



## Le package client

Par défaut, le compilateur GWT part du principe que toutes les classes présentes dans un package contenant le mot-clé « client » (ex : `com.dng.projet.client.XX`) sont traduites en JavaScript. Notez qu'il est possible de redéfinir ce nom dans le fichier de configuration du module. Lorsqu'on souhaite partager ce code avec le serveur, il est de coutume de l'appeler « shared » ou « common ». Ce package intervient juste en dessous du nom du module (ex : `com.dng.projet`).

Le fait d'écrire du code dans le package client (ou `shared`) impose un certain nombre de contraintes. Tout d'abord, celles sur le type des classes utilisées côté client. Le compilateur GWT ne sait traduire que certaines classes Java du JDK en JavaScript. Ces classes sont un sous-ensemble de celles présentes dans les packages `java.lang`, `java.lang.annotation`, `java.util`, `java.io` et `java.sql`.

Pour plus d'informations, n'hésitez pas à vous référer à la javadoc GWT qui dénombre tous les types reconnus.

On pourrait penser que cette contrainte est pénalisante, mais dans la pratique un développeur web n'aura pas nécessairement besoin des sept mille classes du JDK. L'émulation des classes du JRE réalisée par GWT correspond généralement à des traitements fréquents effectués dans une couche d'interface graphique. Cela va de la création de listes ou collections à l'utilisation de types primitifs ou complexes (`Char`, `String`, `int`, `Date`, etc.). Les traitements plus évolués ou consommateurs en ressources sont généralement dévolus au serveur.

Par ailleurs, un interpréteur JavaScript, contrairement à une machine virtuelle Java, possède de nombreuses limitations. La première est la nature mono-thread d'un navigateur. En GWT, la notion de thread n'a pas de sens et l'utilisation des ordres de synchronisation ou de rendez-vous se solderont par un échec lors de la compilation JavaScript.

La seconde contrainte est l'impossibilité de réaliser des traitements dits dynamiques, c'est-à-dire faisant intervenir de nouveaux types à l'exécution. De par sa nature, le compilateur opère des optimisations du code fourni et ne permet pas de tirer parti de la réflexivité et de l'introspection Java.

Le piège habituel du débutant GWT est de penser qu'un code compilé sans erreur sous Eclipse le sera également sous GWT.

## Le package serveur

Tout le code ne faisant pas partie du package client sort de la responsabilité du compilateur GWT. Le package serveur contient toute la logique de services. Ces classes sont compilées normalement avec le compilateur d'Eclipse et n'ont aucune existence côté client ; libre au développeur d'y opérer des traitements tels qu'un accès à une base de données ou d'utiliser des outils Java fournis par des bibliothèques externes.

## Les fichiers de configuration

Toute la configuration d'un module GWT est localisée à cet endroit. Le fichier `module.gwt.xml` contient des propriétés de configuration telles que le point d'entrée du module (l'équivalent de la méthode `main()` dans un programme Java classique), les dépendances vers d'autres modules, les ressources externes (CSS, JavaScript...) ou la liste des différents navigateurs gérés par l'application. Voici un exemple de fichier de configuration.

### Exemple de fichier de configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='hello'>
  <!-- Correspond aux classes du Framework GWT -->
  <inherits name='com.google.gwt.user.User' />
  <inherits name='com.google.gwt.user.theme.standard.Standard' />
  <inherits name='com.google.gwt.rpc.RPC' />

  <!-- Correspond au point d'entrée de l'application -->
  <entry-point class='com.dng.hello.client.Hello' />

  <!-- packages indiquant au compilateur les classes à traduire en JavaScript -->
  <source path='client' />
  <source path='shared' />
```



```
<!-- feuille de styles injectée dans la page HTML lors de la compilation -->
<stylesheet src="Hello.css" />

</module>
```

Le fichier de configuration est un élément clé d'un module : il structure les classes constituant le module. Avant toute exécution de code GWT, le compilateur analyse ce fichier de manière récursive pour tous les modules dépendants.

Dans cet exemple, le mot-clé `inherit` définit une dépendance entre deux modules (en l'occurrence ici avec le framework GWT et la partie RPC). `entry-point` précise la classe contenant le point d'entrée identifié par la méthode `onModuleLoad()` (dérivée elle-même de l'interface `EntryPoint`). Le mot-clé `source` définit le package contenant les classes clientes. Quant à `stylesheet`, il énumère le ou les fichier(s) CSS à intégrer à la page HTML de rendu. Le paragraphe suivant montre qu'il est également possible d'ajouter le CSS directement dans le fichier HTML hôte.

## La structure du répertoire war

La structure d'un projet GWT a énormément évolué entre les versions 1.5 et 1.6. Auparavant, les ressources (la page HTML, les images et les fichiers CSS ou JS externes) étaient intégrées aux sources du projet. À partir de la version 1.6 et à la demande des utilisateurs qui trouvaient la structure peu adaptée à un déploiement en mode WAR, un nouveau répertoire a fait son apparition à l'extérieur du répertoire des sources. Il contient tous les fichiers et répertoires spécifiés par la norme Servlet et suit scrupuleusement le format WAR.

Le répertoire `war\WEB-INF` contient le fichier de configuration des servlets `web.xml`. Les répertoires `lib` et `classes` contiennent les bibliothèques utilisées par le projet.

Il est à noter que `war` contient également la page HTML de l'application GWT accompagnée de ses ressources (CSS, JS ou images).

## La page HTML hôte

Contrairement à une application web JSP traditionnelle constituée de plusieurs pages, une application GWT n'en contient qu'une seule appelée *host page* ou page hôte. Cette page contiendra toute la logique d'affichage du site, et ce, au rythme des ajouts et suppression d'éléments du DOM (*Document Object Model*).

C'est d'ailleurs pour cela qu'en termes de contenu, une page hôte ne contient généralement que les balises `<body>` et `</body>` ainsi que certaines balises JavaScript. Cet aspect est assez déroutant la première fois car un développeur a souvent tendance à demander l'affichage du code source de la page à des fins de débogage. Ici en l'occur-

rence, il faudra faire appel à des plug-ins plus évolués tels que Firebug pour inspecter les éléments de la page.

### Exemple de page hôte

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">

    <link type="text/css" rel="stylesheet" href="Hello.css">

    <title>Exemple d'application GWT</title>

    <!-- -->
    <!-- Ce script charge le module JavaScript compilé -->
    <!-- Il est possible d'ajouter des méta-tags GWT -->

    <script type="text/javascript" language="javascript"
            src="hello/hello.nocache.js"></script>
  </head>

  <!-- -->
  <!-- Le body peut contenir des balises HTML arbitraires, ou rien du tout -->
  <!-- Cette page sera construite dynamiquement au rythme des ajouts dans le
  RootPanel -->

  <body>

    <!-- OPTIONAL: include this if you want history support -->
    <iframe src="javascript:''" id="gwt_historyFrame" tabIndex='-1'
            style="position:absolute;width:0;height:0;border:0"></iframe>

  </body>
</html>
```

Le script suffixé `nocache.js` contient le sélecteur GWT, qui est le premier fichier JavaScript chargé par la page hôte pour identifier la bonne permutation à charger (une permutation est également un fichier JavaScript) en fonction, entre autres, du navigateur cible. Contrairement à la page hôte, le sélecteur ne doit jamais être mis en cache, car nous ne verrions pas les éventuelles modifications apportées au site lors de compilations successives.

L'élément `iframe` paramétré avec l'identifiant `gwt_historyFrame` sert à gérer l'historique. Il est facultatif, nous l'aborderons plus tard.

Notez que les éventuelles feuilles de styles, images et scripts externes peuvent également être insérés directement dans cette page (ou dans le fichier de configuration du module).

## Le mode développement

Le mode développement est sans conteste l'élément de GWT qui illustre le mieux l'originalité et, on peut le dire, le génie de ce framework. Le mode développement (ou *dev mode*) est un environnement puissant qui aide le développeur à réduire la lourdeur des étapes de codage, test et débogage. Il reproduit fidèlement un environnement web.

Il existe plusieurs manières de lancer le mode développement lorsqu'un projet a été créé avec [WebAppCreator](#) :

- Soit en ligne de commande à l'aide de la classe `DevMode` en lui passant l'URL de la page hôte et le nom pleinement qualifié du module de la manière suivante :

### Lancement du mode développement

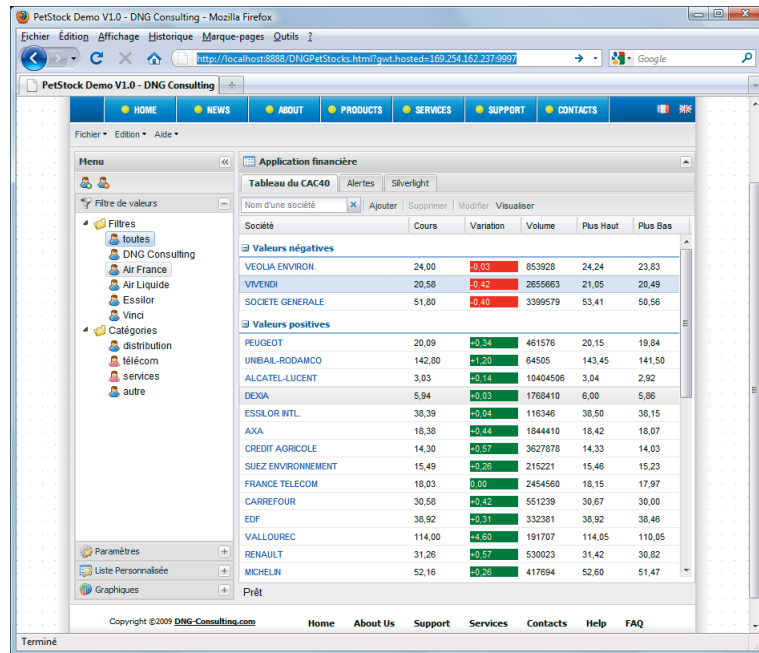
```
c:\projects\hello>java -Xmx256M com.google.gwt.dev.DevMode -startupUrl
                        Hello.html com.dngconsulting.hello.Hello
Using a browser with the GWT Development Plugin, please browse to
the following URL:
http://localhost:8888/Hello.html?gwt.codesvr=127.0.0.1:9997
```

- Soit en passant par Eclipse et le fichier `.launch`. Pour ce faire, nous nous positionnons sur le fichier `Hello.launch` et sélectionnons la commande `Run As` dans le menu contextuel.
- Il existe évidemment d'autres manières en fonction de votre environnement de développement (plug-in GWT Eclipse ou Maven), mais les méthodes précédentes sont les plus simples pour débiter.

Du point de vue de l'outillage, le mode développement est un navigateur qui exécute une version particulière de notre application. Doté d'un plug-in intelligent, ce navigateur possède la capacité de déboguer et charger dynamiquement du bytecode Java.

Il faut bien comprendre qu'en mode normal, un site web bâti avec JavaScript et HTML est impossible à déboguer sous Eclipse (ou tout autre IDE). En supposant que nous utilisions une technologie telle que JSP, la seule chose que l'on serait capable de faire serait un pas à pas des ordres `out.println("<div>...</div>")` ou le contenu de balises personnalisées. Autant dire qu'en pratique, la plus-value de ce type de débogage est assez faible pour le développeur qui n'a qu'une vision technique orientée servlet de son code. Même en JavaScript avec des outils tels que Dojo, jQuery ou Prototype, le débogage se résume souvent à l'utilisation de plug-ins plutôt lourds et incapables de retranscrire une pile d'appels détaillée comme le ferait Eclipse. Cela est également lié à la nature dynamique de JavaScript.

**Figure 1-9**  
Démonstration du PetStocks  
de DNGConsulting



Avec GWT, l'application est déboguée comme une application Swing/SWT ou Windows Forms à la manière d'un client lourd, et ce, à l'intérieur d'un navigateur tout à fait normal ! Mais comment diable est-ce possible ?

Bien souvent, les concepts les plus géniaux tirent leur origine d'idées complètement farfelues. Le mode développement en est une.

Le procédé consiste à étendre le comportement par défaut d'un navigateur pour en faire une sorte de boîte à exécuter du JavaScript et du HTML.

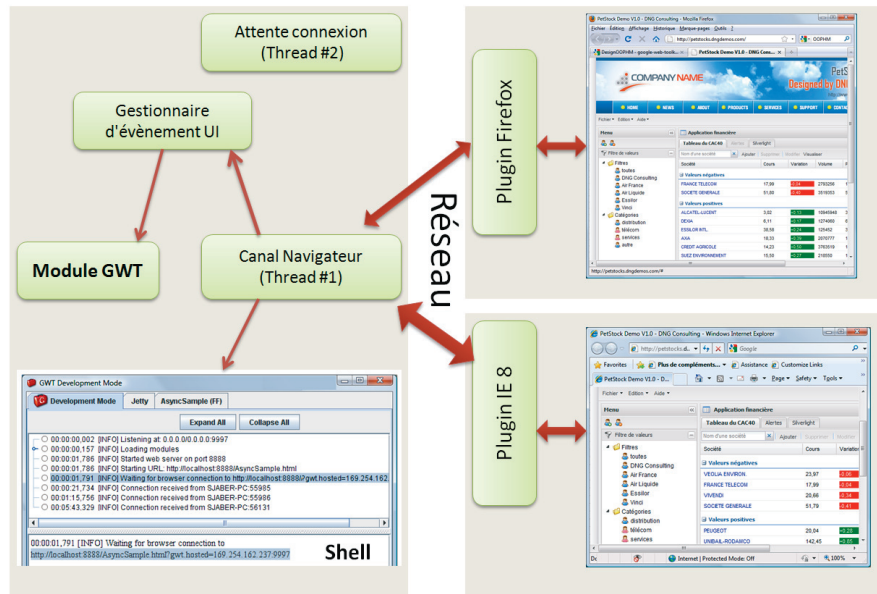
À partir d'une application Java s'exécutant dans une JVM tout à fait normale, un module côté client, appelé également le shell (abordé plus loin), pilote à distance le navigateur. Dans ce mode, l'application Java exécute le code compilé (en bytecode Java) de notre site et communique par socket avec le plug-in installé dans le navigateur. À son tour, ce plug-in est chargé de piloter le navigateur et de renvoyer tous les événements (tels que les clics souris...) à l'application Java.

En mode développement, le navigateur n'a pour seul rôle que d'afficher le rendu de l'application GWT. Toute la partie événementielle (la plus complexe à déboguer) est court-circuitée par le plug-in qui pilote le navigateur. Lorsqu'on clique sur un bouton auquel un gestionnaire d'événement Java est rattaché, c'est en réalité le plug-in qui reçoit l'événement et qui le redirige ensuite à l'application hébergée dans la JVM distante.

En fin de compte, l'ensemble de l'application GWT en mode développement ressemble à n'importe quelle application Java classique client lourd (d'où la possibilité de déboguer), excepté qu'au milieu de la chaîne s'interface un navigateur piloté par un plug-in.

Pour résumer, nous avons donc côté JVM du code 100 % Java communiquant via le réseau avec des plug-ins intégrés à leur navigateur d'origine. Il faut donc autant de plug-ins qu'il existe de navigateurs du marché.

**Figure 1-10**  
Architecture  
des plug-ins



Dans le schéma précédent, il faut noter l'absence de contraintes d'installation particulières sur les machines hébergeant les navigateurs. En d'autres termes, n'importe quelle machine dotée du bon plug-in (donc sans aucun framework GWT) peut prétendre à afficher et tester un site GWT en phase de développement.

Les plug-ins disponibles pour GWT sont proposés en téléchargement sur le site de Google la première fois qu'un utilisateur accède à une application GWT en mode développement. En fonction du navigateur, l'utilisateur configure ou double-clique sur les fichiers proposés.

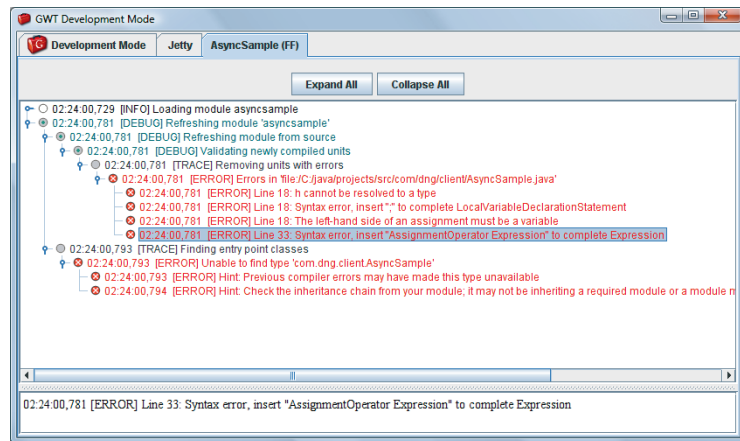
L'intérêt principal du mode développement est de laisser la liberté à l'utilisateur d'installer et configurer n'importe quel plug-in additionnel sur son navigateur. Le plug-in Firebug permet d'espionner et déboguer des fragments de DOM tout en modifiant à la volée le contenu de la page.

D'un point de vue fonctionnel, le mode développement est le plus productif : lorsque le développeur modifie du code Java, il le sauvegarde (ce qui a pour effet de lancer automatiquement une compilation sous Eclipse) et active le bouton *Rafraîchir* de son navigateur. Les modifications sont instantanément prises en compte et il est possible à tout moment de positionner un point d'arrêt et de procéder au pas à pas en mode débogage. Cette productivité était inimaginable il y a encore quelques mois pour le développement d'applications web.

## Le shell

Le shell est la fenêtre hiérarchique dans laquelle s'affichent tous les messages d'erreur en provenance de GWT, qu'il s'agisse d'erreurs de compilation, d'exceptions personnalisées ou de problèmes internes.

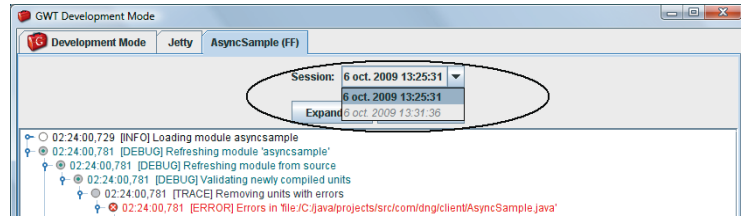
**Figure 1-11**  
La fenêtre du shell



Lorsqu'une erreur de compilation survient dans le shell, une pile d'appels et un message pointant la ligne incriminée s'affichent.

Les onglets affichés dans le shell correspondent aux différents modules en cours d'exécution. Nous avons vu dans le schéma d'architecture précédent qu'il était possible de déboguer un module en ciblant à un instant  $t$  plusieurs navigateurs. Pour cela, chaque navigateur correspond à une session. Les sessions sont affichées sous la forme d'une liste, les navigateurs sous la forme d'onglets. Lorsque le navigateur est fermé, la communication réseau s'interrompt et la session présente un état inactif (grisé et en italique).

**Figure 1-12**  
Les sessions GWT du shell



#### REMARQUE Affichage des traces

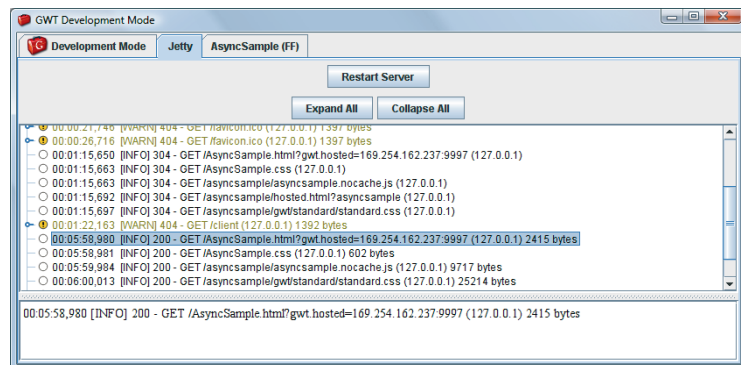
Le shell sert également de console d'affichage pour tous les messages retournés dans le code Java à l'aide de la fonction `GWT.log(message, exception)`.

## Le conteneur de servlets Jetty

En phase de développement, GWT fournit un socle serveur représenté par un conteneur de servlets. Le rôle de ce conteneur est de simuler un environnement d'exécution minimal permettant au développeur de déboguer d'éventuels services distants. Le choix s'est porté sur l'outil Jetty pour sa capacité à démarrer rapidement (il n'est pas rare d'avoir à arrêter et relancer le serveur fréquemment) mais également sa prédisposition à être hébergé dans une JVM et piloté via des API.

Le shell propose un onglet identifié Jetty retraçant toutes les requêtes intervenant entre le client et le serveur.

**Figure 1-13**  
Les traces Jetty



Le mode développement couvre également d'autres scénarios complexes d'utilisation, notamment lorsqu'il existe déjà au sein du réseau (ou en local sur le poste) un serveur d'applications hébergeant des services distants (EJB, Spring, etc.).

L'option `-noserver` permet de lancer le shell sans conteneur de servlets.

## Le mode production

Dans le mode développement, toute l'application est émulée via une JVM. Or, une fois en production, plus question d'appeler du bytecode Java : il nous faut un vrai site web. C'est le rôle du mode production.

Il correspond au contexte dans lequel l'application GWT finale est réellement exécutée, c'est-à-dire au travers d'un navigateur et avec le site entièrement traduit en JavaScript. Ce mode fait suite à une phase de compilation qui peut dans certains cas prendre de quelques secondes à plusieurs minutes.

Créer le code en JavaScript consiste à faire appel au compilateur GWT. Tout comme le mode développement, cette opération dépend de votre environnement de développement. Vous pouvez simplement double-cliquer sur le fichier `build.xml` et lancer la tâche par défaut ou passer par la ligne de commande avec Ant ou Maven comme sur la figure suivante. Le compilateur GWT est lui-même écrit en Java et se trouve dans l'archive `gwt-dev.jar`.

**Figure 1-14**  
Les options du compilateur  
GWT

```

Administrator C:\Windows\system32\cmd.exe
c:\guthack\trunk\build\dist\gwt-2.0>java -cp gwt-dev.jar com.google.gwt.dev.Compiler
Missing required argument: module[s]
Google web toolkit 0.0.0
Compiler [-logLevel] [-warDir] [-gen dir] [-style style] [-no] [-XshardPrecompile] [-XdisableClassMetadata] [-XdisableCastChecking]
[-validateOnly] [-draftCompile] [-compileReport] [-localWorkers count] [-war dir] [-extra dir] module[s]

where
-logLevel      The level of logging detail: ERROR, WARN, INFO, TRACE, DEBUG, SPAN, or ALL
-warDir        The compiler's working directory for internal use (must be writable; defaults to a system temp dir)
-gen           Debugging: causes normally-transient generated types to be saved in the specified directory
-style        Script output style: ORF([SAR]), PRETTY, or DETAILED (defaults to ORF)
-ea           Debugging: causes the compiled output to check assert statements
-XshardPrecompile  Enables running generators on CompilePerms shards
-XdisableClassMetadata  EXPERIMENTAL: Disables some Java lang class methods (e.g. getName())
-XdisableCastChecking  EXPERIMENTAL: Disables run-time checking of cast operations
-validateOnly  Validates all source code, but do not compile
-draftCompile  Enable faster, but less-optimized, compilations
-compileReport  Create a compile report that tells the story of Your Compile
-localWorkers  The number of local workers to use when compiling permutations
-extra         The directory into which deployable output files will be written (defaults to 'war')
              The directory into which extra files, not intended for deployment, will be written
-module[s]    Specifies the name(s) of the module(s) to compile

c:\guthack\trunk\build\dist\gwt-2.0>java -cp gwt-dev.jar com.google.gwt.dev.Compiler
  
```

Les options de ce compilateur sont nombreuses et sont abordées dans le détail au chapitre 12, « Sous le capot de GWT ».

Après avoir placé `gwt-user.jar` et `gwt-servlet.jar` dans le `classpath`, nous compilons le module ainsi :

```
c:\projects\hello>java -Xmx256M com.google.gwt.dev.Compiler
```

```
com.dngconsulting.hello.Hello
```

```
Compiling module com.dngconsulting.hello.Hello
```

```
Compiling 1 permutations
```

```
Worker permutation 0 of 1
```

```
Creating split point map file for the compile report
```

```
Done
```

```
Linking into war
```

```
Link succeeded
```

```
Compilation succeeded -- 12,957s
```



## La structure d'un site compilé

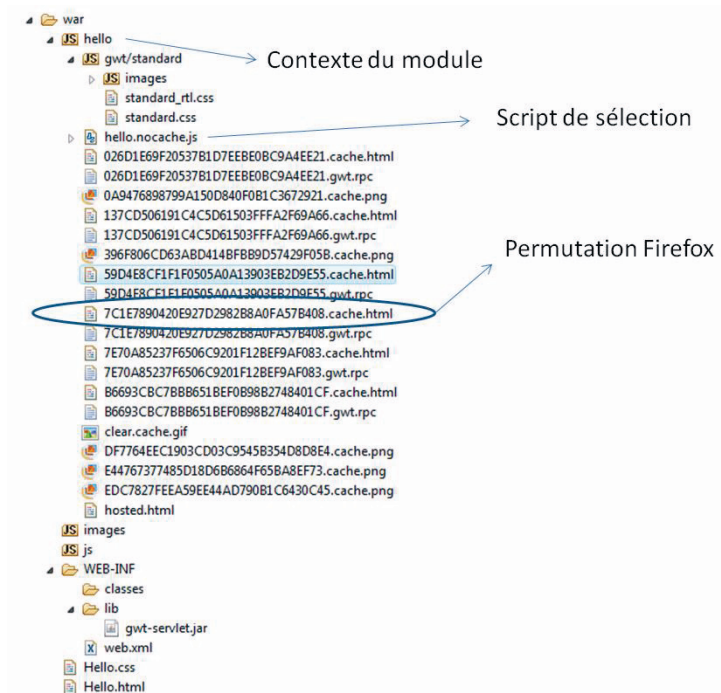
Au premier abord, la structure d'un site GWT compilé n'a rien de très compréhensible pour le commun des mortels.

Le compilateur crée un fichier de permutation par contexte d'utilisation (type de navigateur, langues, chargement à la demande, etc.) et donne à ces permutations un nom unique. Ce dernier est le résultat d'un algorithme de hachage (MD5) reconnu universellement pour calculer des clés uniques. Malgré les apparences et le suffixe `.html`, une permutation est un fichier JavaScript.

Gardez à l'esprit que chaque permutation est mise en cache par le navigateur *ad vitam aeternam*. Cette convention de nommage assure l'unicité des permutations et garantit que toute modification ultérieure du site suivie d'une compilation provoquera une nouvelle mise en cache.

Voici à quoi ressemble un répertoire `war` une fois passé sous le grill du compilateur (voir figure 1-15).

**Figure 1-15**  
Structure du site compilé



Tout le contenu du sous-répertoire `hello` n'est produit que lorsque le développeur demande explicitement la compilation. En mode développement, rappelez-vous, GWT s'appuie sur le bytecode Java de l'application.

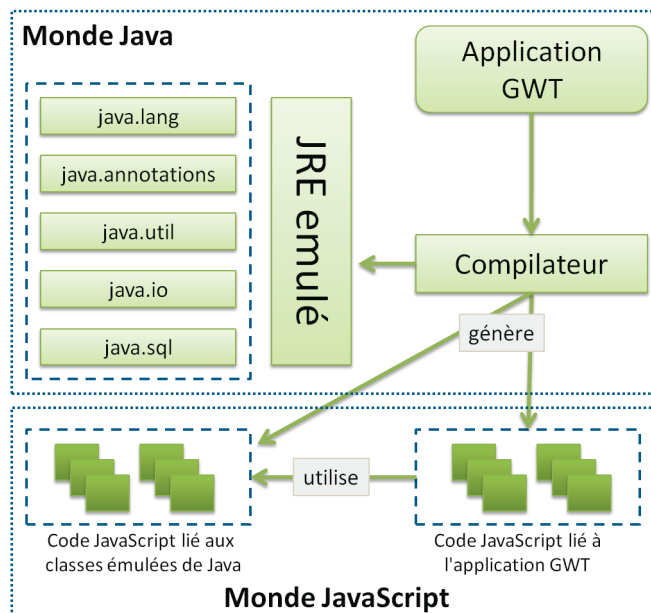
Les classes et dépendances du serveur sont compilées normalement et créées dans le répertoire `WEB-INF/classes` ou sous la forme d'une archive d'extension `.jar` dans `WEB-INF/lib`

Notez qu'il est possible de déployer séparément la partie cliente, constituée de pages et de scripts statiques, et la partie serveur.

## Les types Java émulés par GWT

GWT étant une technologie s'appuyant sur JavaScript, quasiment aucune des classes du JDK Java ne peut prétendre à être convertie par une simple baguette magique. Pour arriver à ses fins, GWT réalise en interne une sorte d'émulation, c'est-à-dire une réécriture d'une partie des classes nécessaires au framework client de certains packages du JDK, pour les rendre compatibles avec JavaScript. Le plus atypique est que cette implémentation est elle-même codée en Java, mais dans un Java suffisamment simple et performant pour prétendre à être converti facilement en JavaScript.

**Figure 1-16**  
Émulation du JRE Java



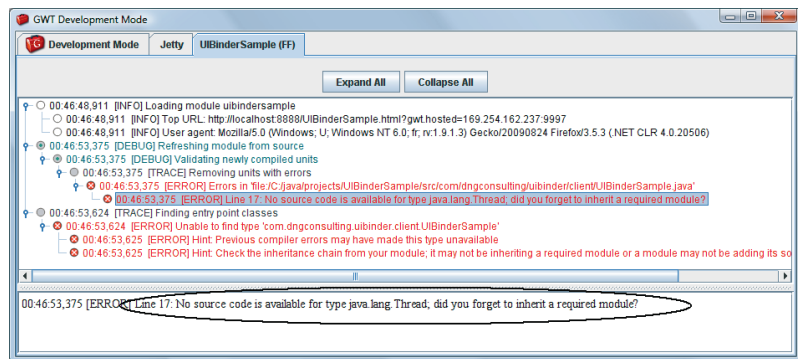
Cette émulation est une des raisons pour lesquelles la compilation sous Eclipse (ou n'importe quel IDE Java) ne suffit pas à valider que l'application GWT fonctionne réellement. Il est indispensable d'effectuer le test sous le compilateur GWT. Celui-ci vérifiera de manière effective la compatibilité des classes Java utilisées dans notre projet avec le JRE émulé.

#### REMARQUE JavaScript est mono-thread

JavaScript étant un environnement mono-thread, toutes les procédures de synchronisation et de gestion des threads en Java sont à proscrire dans le monde GWT. Cette règle prévaut également pour les méthodes de la classe `Object` (`notify()`, `wait()`, `waitAll()`). Le mot-clé `synchronized` est ignoré silencieusement.

Voici le message renvoyé par le compilateur GWT lorsqu'il tombe sur un type compatible Java mais non compatible GWT.

**Figure 1-17**  
Type Java non compatible  
avec JavaScript



On pourrait penser que ce sous-ensemble supporté du JRE est un handicap lorsqu'il s'agit de développer en GWT. Pourtant, il faut garder à l'esprit que nous développons des interfaces graphiques avec GWT. Généralement, ce type d'application n'a besoin que de collections, de types primitifs et de quelques classes essentielles de `java.io` et `java.lang`. Pour le reste (accès en bases, calculs complexes, multi-threading...), il est indispensable d'effectuer les traitements côté serveur en utilisant des services RPC.

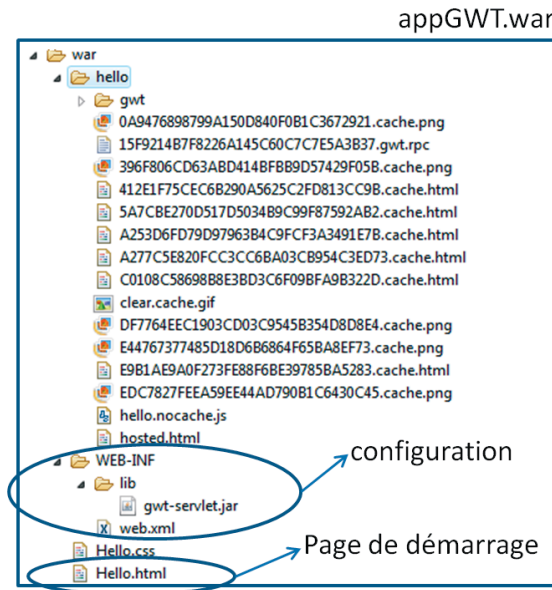
## Le déploiement

Le processus de déploiement d'une application GWT est relativement trivial dans la mesure où le développeur s'appuie à la base sur une structure déjà formatée `war`.

Cette étape consiste généralement à compresser le répertoire `war` et à le déposer dans un quelconque conteneur de servlets.

Parfois certaines spécificités pourront venir compliquer ce déploiement. C'est notamment le cas lorsque les services RPC ne s'exécutent pas sur le même serveur Web que celui ayant fourni les permutations ou lorsqu'une zone DMZ (sécurisée) impose des contraintes de type *reverse proxies*. Nous reviendrons sur ces spécificités dans le chapitre 7 dédié à RPC.

**Figure 1-18**  
Contenu d'un fichier WAR



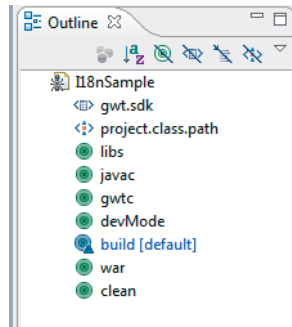
## Fichier Ant

Lors de la création du squelette de l'application, le script `WebAppCreator` fournit par défaut un fichier de construction Ant situé à la racine du projet et proposant toutes les étapes du cycle de vie d'une application GWT :

- compilation ;
- copie des fichiers du `classpath` dans le répertoire `WEB-INF/libs` ;
- lancement du mode production ;

- lancement du mode développement ;
- construction du projet et appel du compilateur ;
- génération d'un fichier WAR.

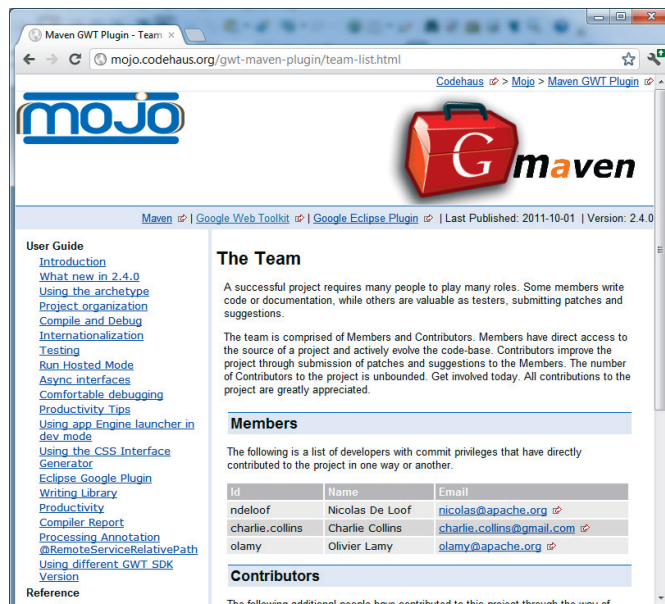
**Figure 1-19**  
Fichier Ant



## Plug-in Maven

Maven est un outil pour la construction et l'automatisation de projets Java. L'objectif recherché est comparable à Make ou Ant : construire un projet à partir de ses sources tout en facilitant la résolution des dépendances binaires. Maven introduit un cycle de vie pour la construction des projets (tests, compilation, production de la documentation, pré-intégration, etc.) et possède de plus en plus d'adeptes à travers le monde.

**Figure 1-20**  
Plugin Maven



Il existe un plug-in Maven pour GWT disponible à l'adresse <http://mojo.codehaus.org/gwt-maven-plugin/>. Il permet entre autres de créer automatiquement les services RPC, les fichiers d'internationalisation et l'édition des rapports de compilation. Par ailleurs, l'option `-maven` utilisée avec `WebAppCreator` produira un fichier d'extension `.pom` contenant les différentes dépendances nécessaires au fonctionnement d'une application minimale.

Voici un exemple de fichier de configuration Maven pour GWT. Il est fort probable que ce plug-in évolue au rythme des modifications de GWT.

```
<project>
  <properties>
    <gwt.version>here your preferred gwt sdk version</gwt.version>
  </properties>
  [...]
  <build>
    <plugins>
      [...]
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>gwt-maven-plugin</artifactId>
        <version>2.4.0</version>
        <dependencies>
          <dependency>
            <groupId>com.google.gwt</groupId>
            <artifactId>gwt-user</artifactId>
            <version>${gwt.version}</version>
          </dependency>
          <dependency>
            <groupId>com.google.gwt</groupId>
            <artifactId>gwt-dev</artifactId>
            <version>${gwt.version}</version>
          </dependency>
        </dependencies>
      </plugin>
      [...]
    </plugins>
  </build>
  [...]
</project>

<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>gwt-maven-plugin</artifactId>
      <configuration>
        <gwtHome>${gwtHome}</gwtHome>
      </configuration>
    </plugin>
  </plugins>
</build>
```

```
        <disableCastChecking>true</disableCastChecking>
        <disableClassMetadata>true</disableClassMetadata>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>generateAsync</goal>
                <goal>compile</goal>
            </goals>
        </execution>
    </executions>
</plugin>
</plugins>
</build>
```

## La fonctionnalité « Super DevMode » dans GWT 2.5

Introduit dans GWT 2.5 en version expérimentale, le Super DevMode a été conçu pour créer un environnement qui puisse s'affranchir de tout plug-in en phase de développement (de plus en plus coûteux à maintenir au rythme des évolutions des navigateurs). L'idée sous-jacente consiste à compiler en quelques secondes à chaque modification le code Java en JavaScript en conservant les fonctionnalités de débogage. Ce n'est donc plus le code Java qui est exécuté mais le code JavaScript final. Pour bien comprendre ce concept, il faut savoir que l'étape la plus longue lorsque le compilateur génère du Java en JavaScript est la phase d'optimisation et de réduction de code. Si l'on part du principe qu'un développeur en local sur sa machine peut sans trop de latence charger un script de plusieurs mégaoctets, il est possible de réduire les délais de compilation à quelques secondes *via* une option particulière du compilateur.

N'hésitez pas à vous référer au chapitre « Sous le capot de GWT » pour plus de détail sur l'option `-draft-Compile`.

Reste ensuite à résoudre la problématique du débogage. Comment déboguer du JavaScript en sachant que le code initial a été développé en Java ? La réponse est dans un outil nommé SourceMaps. Créé initialement par Google et aujourd'hui supporté par plusieurs navigateurs dont Firefox, SourceMaps permet de retrouver le code Java initial ayant servi à générer ou optimiser du JavaScript à partir d'un dictionnaire maintenant des correspondances entre fonctions du source original et code compilé.

Avec ce nouveau SuperDevMode, il devient désormais possible de travailler en mode développement dans les mêmes conditions qu'en production (exceptées les optimisations et réductions évidemment). Le code débogué au final est en JavaScript, contrairement à l'actuel DevMode (qui s'appuie sur du bytecode Java).

Il est fort probable qu'à l'avenir, le SuperDevMode vienne compléter (ou remplacer à plus longue échéance) toute la panoplie d'outils déjà disponibles en GWT pour la phase de développement.

Pour plus d'informations, n'hésitez pas à vous référer à la documentation officielle de GWT.