

- 1. [Configuration de votre environnement de développement Go](#)
 - 1.1. [Installation](#)
 - 1.2. [\\$GOPATH et workspace](#)
 - 1.3. [Commandes Go](#)
 - 1.4. [Outils de développement Go](#)
 - 1.5. [Résumé](#)
- Appendice A [Références](#)

1 Configuration de votre environnement de développement Go

Bienvenue dans le monde de Go, commençons notre exploration!

Go est un langage de programmation système concurrent à la compilation rapide et disposant d'un ramasse-miettes. Il a les avantages suivants:

- Compilation d'un large projet en quelques secondes.
- Fournit un modèle de développement facile à gérer, évitant la plupart des problèmes liés aux fichiers d'en-tête C.
- C'est un langage statique avec un système de types de données simple, afin que les utilisateurs n'aient pas à perdre du temps à gérer les relations entre types. Il peut-être vu comme un langage orienté-objet simplifié.
- Il possède un ramasse-miettes. Il fournit un support de base pour la concurrence et de la communication.
- Conçu pour les ordinateurs multi-core.

Go est un langage compilé. Il combine l'efficacité de développement des langages interprétés ou dynamiques avec la sécurité de langues statiques. Il va être la langue de choix pour les ordinateurs modernes multi-core inter-connectés. À ces fins, il ya quelques problèmes qui doivent fondamentalement être résolu au niveau de ce langage, comme un système

de types très expressif et léger, un modèle de concurrence natif, et un ramasse-miettes géré strictement. Durant longtemps, aucun paquet ou outil n'est apparu qui avait pour but de résoudre tous ces problèmes de façon pragmatique; ainsi naquit la motivation pour du langage Go.

Dans ce chapitre, je vais vous montrer comment installer et configurer votre propre environnement de développement Go.

Table des matières



Navigation

- [Table des matières](#)
- Section suivante: [Installation](#)

1.1 Installation

3 méthodes d'installation

Il y a plusieurs façons de configurer votre environnement de développement Go sur votre ordinateur, vous pouvez choisir celle qui vous convient le

mieux. Les trois façons les plus courantes sont les suivantes.

- Paquets d'installation officiels:
 - L'équipe de Go propose des paquets d'installation pour Windows, Linux, Mac et encore d'autres systèmes d'exploitation. C'est probablement la méthode la plus simple pour commencer.
- Installation manuelle via les sources
 - Populaire chez les développeurs habitués aux environnements type Unix.
- Utilisation d'outils tiers.
 - Il y a plusieurs outils tiers et gestionnaires pour installer Go, comme apt-get pour Ubuntu et homebrew pour Mac.

Dans le cas où vous voudriez installer plus d'une version de Go sur votre machine, vous devriez jeter un oeil à l'outil [GVM](#). C'est le meilleur outil pour accomplir cette tâche, sinon vous devrez le faire vous-même.

##Installation à partir des sources

Parce-que certaines parties de Go sont écrits en C Plan 9 et en assembleur AT&T, vous devez installer un compilateur C avant de continuer.

Sous Mac, si vous avez Xcode d'installé, vous disposé déjà d'un compilateur.

Sur un système type Unix, vous devez installer gcc ou un compilateur équivalent. Par exemple, en utilisant le gestionnaire de paquets apt-get(livré avec Ubuntu), vous pouvez installer les compilateurs requis comme suit:

```
sudo apt-get install gcc libc6-dev
```

Sous Windows, vous devez installer MinGW pour installer gcc. N'oubliez pas de configurer vos variables d'environnement après avoir finalisé l'installation.

L'équipe Go utilise [Mercurial](#) pour gérer le code source, vous devez commencer par installer celui-ci pour télécharger le code source de Go.

Une fois Mercurial installé, exécutez les commandes suivantes pour cloner le code source de Go et le compiler:

```
hg clone -u release https://code.google.com/p/go
cd go/src
./all.bash
```

Une installation réussie finira en affichant le message "ALL TESTS PASSED."

Sous Windows, vous pouvez arriver à faire la même chose en lançant `all.bat`.

Si vous utilisez Windows, le paquet d'installation va mettre à jour automatiquement vos variables d'environnement. Sous Unix, vous devez configurer celles-ci manuellement comme suit:

```
export GOROOT=$HOME/go
export GOBIN=$GOROOT/bin
export PATH=$PATH:$GOROOT/bin
```

Si vous voyez l'affichage suivant sur votre écran, l'installation s'est terminée avec succès.

```

applematoMacBook-Pro-3:~ apple$ go
Go is a tool for managing Go source code.

Usage:

    go command [arguments]

The commands are:

    build      compile packages and dependencies
    clean      remove object files
    doc        run godoc on package sources
    env        print Go environment information
    fix        run go tool fix on packages
    fmt        run gofmt on package sources
    get        download and install packages and dependencies
    install    compile and install packages and dependencies
    list       list packages
    run        compile and run Go program
    test       test packages
    tool       run specified go tool
    version    print Go version
    vet        run go tool vet on packages

Use "go help [command]" for more information about a command.

Additional help topics:

    gopath     GOPATH environment variable
    packages   description of package lists
    remote     remote import path syntax
    testflag   description of testing flags
    testfunc   description of testing functions

Use "go help [topic]" for more information about that topic.

applematoMacBook-Pro-3:~ apple$ █

```

Figure 1.1 Information après installation

Lorsque vous voyez les informations d'utilisation de Go, cela signifie que vous avez installé Go avec succès sur votre ordinateur. Si votre système affiche 'aucune commande Go', vérifiez que votre variable d'environnement \$PATH contient bien le chemin d'installation de Go.

Utilisation des paquets standards d'installation

Go a des paquets d'installation rapides pour chaque système d'exploitation supporté. Ces paquets installent Go dans `/usr/local/go` (`c:\Go` sous Windows) par défaut. Bien sûr, cela est configurable, mais vous aurez également à changer manuellement les variables d'environnement comme décrit précédemment.

Comme savoir si votre système est 32 ou 64 bits?

L'étape suivante dépend du type d'architecture de votre système, nous devons donc le vérifier avant de télécharger les paquets d'installation.

Si vous utilisez Windows , tapez `Win+R` et lancez l'outil de commandes `cmd` . Lancez la commande `systeminfo` qui va vous afficher des informations très utiles sur votre système. Trouvez la ligne indiquant "type du système" - si vous voyez "x64-based PC" cela signifie que votre système est 64-bit, sinon il est 32-bit.

Je vous recommande fortement de télécharger le paquet 64-bit si vous êtes sous Mac, comme Go ne supporte plus les processeurs 32-bit sous Mac OSX.

Les utilisateurs Linux peuvent exécuter `uname -a` dans un terminal pour afficher les informations système. Un système d'exploitation 64-bit affichera les informations suivantes:

```
<descriptions diverses> x86_64 x86_64 x86_64 GNU/Linux
```

Certaines machines comme Ubuntu 10.04 afficheront simplement:

```
x86_64 GNU/Linux
```

Les systèmes 32-bit afficheront quant à eux:

```
<descriptions diverses> i686 i686 i386 GNU/Linux
```

Mac

Aller sur la [page de téléchargement](#), choisir `go1.4.darwin-386.pkg` pour les systèmes 32-bit et `go1.4.darwin-amd64.pkg` pour les 64-bit. Cliquez "next" jusqu'à la fin du processus, `~/go/bin` sera ajouté au `$PATH` du système à la fin de l'installation. Maintenant ouvrez un terminal et tapez `go`. Vous devriez avoir la même sortie qu'en figure 1.1.

Linux

Aller à la [page de téléchargement](#), choisir `go1.4.linux-386.tar.gz` pour les systèmes 32-bit et `go1.4.linux-amd64.tar.gz` en 64-bit. Supposons que vous voulez installer Go dans `$GO_INSTALL_DIR` path. Décompressez l'archive `tar.gz` dans le dossier désiré de destination en utilisant la commande `tar zxvf go1.4.linux-amd64.tar.gz -C $GO_INSTALL_DIR`. Ensuite définissez votre `$PATH` avec: `export PATH=$PATH:$GO_INSTALL_DIR/go/bin`. Vous pouvez maintenant ouvrir un terminal et tapez `go`. Vous devriez avoir la même sortie qu'en figure 1.1.

Windows

Allez à la [page de téléchargement](#), choisissez `go1.4.windows-386.msi` pour les systèmes 32-bit et `go1.4.windows-amd64.msi` pour les systèmes 64-bit. Cliquez "next" jusqu'à la fin du processus, `c:/go/bin` sera ajouté à `path`. Vous pouvez maintenant ouvrir un terminal et tapez `go`. Vous devriez avoir la même sortie qu'en figure 1.1.

Utilisez des outils tiers

GVM

GVM est un outil de gestion de versions multiples développé par un tiers, comme `rbenv` pour ruby. Il est simple d'utilisation. Installez GVM avec les commandes suivantes dans votre terminal:

```
bash < <(curl -s -S -L https://raw.githubusercontent.com/moovweb/gvm/master/binscripts/gvm-installer)
```

Ensuite nous installons Go avec les commandes suivantes:

```
gvm install go1.4  
gvm use go1.4
```

À la fin du processus, Go est installé.

apt-get

Ubuntu est la version bureau la plus utilisée de Linux. Elle utilise `apt-get` comme gestionnaire de paquets. On peut installer Go avec les commandes suivantes:

```
sudo add-apt-repository ppa:gophers/go  
sudo apt-get update  
sudo apt-get install golang-stable
```

Homebrew

Homebrew est un outil de gestion de logiciels communément utilisé sous Mac pour gérer les paquets. On peut installer Go avec les commandes suivantes:

```
brew install go
```

Navigation

- [Tables des matières](#)

- Section précédente: [Configuration de votre environnement Go](#)
- Section suivante: [\\$GOPATH et workspace](#)

#1.2 \$GOPATH et workspace

\$GOPATH

Les commandes Go reposent sur une variable d'environnement nommée `$GOPATH`¹. Notez que ce n'est pas la variable `$GOROOT` où Go est installé. Cette variable pointe vers votre espace de travail(workspace) Go de votre ordinateur (J'utilise ce chemin sur mon ordinateur; si vous n'utilisez pas la même structure de dossiers, veuillez à le faire correspondre par vous-même).

Dans un système de type Unix, la variable peut-être configurée comme suit:

```
export GOPATH=/home/apple/mygo
```

Pour plus de commodité, vous devriez ajouter une ligne à votre `.bashrc` ou `.zshrc`, ou tout autre fichier de configuration de configuration du shell approprié.

Sous Windows, vous devez créer une variable d'environnement nommée `GOPATH`, et définir sa valeur à `c:\mygo` :

```
GOPATH=c:\mygo
```

Il est possible d'avoir plusieurs répertoires dans votre `$GOPATH`, à séparer par un `:` sous Unix, et `;` sous Windows

Dans votre `$GOPATH` vous devez avoir trois sous-répertoires:

- `src` pour héberger le code source (par exemple: `.go .c .h .s`, etc.)
- `pkg` pour les fichiers compilés (par exemple: `.a`)

- `bin` pour les fichiers exécutable compilés

Dans le reste de ce livre, j'utilise le dossier `Mygo` comme répertoire de mon `$GOPATH`.

Dossier de paquets

Créez les fichiers et dossiers d'un paquet source comme `$GOPATH/src/mymath/sqrt.go` (`mymath` est le nom du paquet)

Chaque fois que vous créez un paquet, vous devriez créer un nouveau dossier dans le répertoire `src`. Les noms de dossiers correspondent généralement à celui du paquet que vous souhaitez utiliser. Vous pouvez avoir plusieurs niveaux de dossiers si vous le désirez. Par exemple si vous créez le dossier `$GOPATH/src/github.com/astaxie/beedb`, alors le chemin du paquet sera `github.com/astaxie/beedb`. Le nom du paquet sera celui du dernier dossier de votre chemin, qui est `beedb` dans notre exemple.

Exécutez les commandes suivantes:

```
cd $GOPATH/src
mkdir mymath
```

Créez un nouveau fichier `sqrt.go`, ajoutez le contenu suivant à celui-ci:

```
// Code source de $GOPATH/src/mymath/sqrt.go
package mymath

func Sqrt(x float64) float64 {
    z := 0.0
    for i := 0; i < 1000; i++ {
        z -= (z*z - x) / (2 * x)
    }
    return z
}
```

Maintenant le dossier de mon paquet a été créé et son code écrit. Je vous recommande d'utiliser le même nom pour vos paquets et leurs dossiers, et que les dossiers contiennent tous les fichiers sources du paquet.

Compilation des paquets

Nous avons créé notre paquet précédemment, mais comment le compiler pour un usage concret? Il y a deux méthodes différentes.

1. Placez-vous dans le dossier contenant votre paquet, puis exécutez la commande `go install`.
2. Exécutez la commande précédente mais en passant un nom de fichier en paramètre, comme `go install mymath`.

Après compilation, on peut ouvrir le dossier suivant:

```
cd $GOPATH/pkg/${GOOS}_${GOARCH}
// On peut y trouver le fichier compilé
mymath.a
```

Le fichier dont le suffixe est `.a` est le fichier binaire de notre paquet. Comment l'utiliser?

Assurément, nous devons créer une application pour l'utiliser.

Créez un nouveau paquet appelé `mathapp`.

```
cd $GOPATH/src
mkdir mathapp
cd mathapp
vim main.go
```

Le code de `main.go`

```
//code source de $GOPATH/src/mathapp/main.go.  
package main  
  
import (  
    "mymath"  
    "fmt"  
)  
  
func main() {  
    fmt.Printf("Hello, world. Sqrt(2) = %v\n", mymath.Sqrt(2))  
}
```

Pour compiler cette application, vous devez vous placer dans le dossier du paquet

```
cd $GOPATH/src/mathapp
```

puis exécutez

```
go install
```

Vous devriez désormais voir un fichier exécutable `mathapp` généré dans le dossier `$GOPATH/bin/`. Pour lancer ce programme, utilisez la commande

```
./mathapp
```

Vous devriez voir le contenu suivant dans votre terminal:

```
Hello world. Sqrt(2) = 1.414213562373095
```

Installer des paquets distants

Go a un outil pour installer des paquets distants, qui est l'outil `go get`. Il supporte la majorité des communautés libres, comme Github, Google Code, BitBucket, et Launchpad.

```
go get github.com/astaxie/beedb
```

Vous pouvez utiliser `go get -u` pour mettre à jour vos paquets distants, cela mettra aussi à jour les dépendances.

Vous obtiendrez le code source des paquets désirés grâce à cette commande, depuis différentes plate-formes, utilisant chacune leur système de gestion de version. Par exemple, `git` pour Github et `hg` pour Google Code. Ainsi vous devrez au préalable installer le client du système de gestion de version approprié avant d'utiliser `go get`.

Après avoir utilisé les commandes précédentes, votre structure de dossier devrait ressembler à celle-ci:

```
$GOPATH
src
| -github.com
|   | -astaxie
|     | -beedb
pkg
| - -${GOOS}_${GOARCH}
|   | -github.com
|     | -astaxie
|       | -beedb.a
```

En fait, `go get` clone le source code dans le dossier `$GOPATH/src` en local, puis exécute `go install`.

Vous pouvez utiliser des paquets distants de manière similaire aux paquets locaux.

```
import "github.com/astaxie/beedb"
```

Arborescence complète de dossier

Si vous avez suivi toutes les étapes précédentes, la structure de votre dossier ressemble désormais à ceci:

```
bin/  
  mathapp  
pkg/  
  ${GOOS}_${GOARCH}, such as darwin_amd64, linux_amd64  
  mymath.a  
  github.com/  
    astaxie/  
      beedb.a  
src/  
  mathapp  
    main.go  
  mymath/  
    sqrt.go  
  github.com/  
    astaxie/  
      beedb/  
        beedb.go  
        util.go
```

Vous pouvez désormais vous rendre compte de manière plus claire de l'arborescence; `bin` contient les exécutables, `pkg` les fichiers compilés et `src` les paquets sources.

[1] Le format des variables d'environnement sous Windows est `%GOPATH%`, pourtant ce livre suit principalement la norme Unix, les utilisateurs Windows devront faire les modifications appropriées.

Navigation

- [Table des matières](#)
- Section précédente: [Installation](#)
- Section suivante: [Go commands](#)

#1.3 Commandes Go

Commandes Go

Le langage Go est disponible par défaut avec un ensemble complet d'outils. Vous pouvez exécuter la ligne de commande `go` pour les afficher:

```
10 C:\>go
11 Go is a tool for managing Go source code.
12
13 Usage:
14
15     go command [arguments]
16
17 The commands are:
18
19     build      compile packages and dependencies
20     clean      remove object files
21     doc        run godoc on package sources
22     env        print Go environment information
23     fix        run go tool fix on packages
24     fmt        run gofmt on package sources
25     get        download and install packages and dependencies
26     install    compile and install packages and dependencies
27     list       list packages
28     run        compile and run Go program
29     test       test packages
30     tool       run specified go tool
31     version    print Go version
32     vet        run go tool vet on packages
33
34 Use "go help [command]" for more information about a command.
35
36 Additional help topics:
37
38     gopath     GOPATH environment variable
39     packages   description of package lists
40     remote     remote import path syntax
41     testflag   description of testing flags
42     testfunc   description of testing functions
43
44 Use "go help [topic]" for more information about that topic.
45
```

Figure 1.3 Informations détaillées affichées par la commande `go`

Elles ont toutes une utilité. Voyons l'utilisation de certaines d'entre elles.

go build

Cette commande sert de tests de compilation. Elle compilera les paquets dépendants au besoin.

- Si le paquet n'est pas le paquet `main`, comme `mymath` en section 1.2, rien ne sera généré après l'exécution de `go build`. Si vous avez besoin de fichier de paquet `.a` dans `$GOPATH/pkg`, utilisez plutôt `go install` à la place.
- Si le paquet est le paquet `main`, la commande générera un exécutable dans le même dossier. Si vous voulez que l'exécutable soit généré dans `$GOPATH/bin`, utilisez `go install` ou `go build -o ${VOTRE_CHEMIN}/a.exe`.
- S'il y a plusieurs fichiers dans le dossier, mais que vous voulez juste compiler un d'entre eux, ajoutez le nom de ce fichier après `go build`. Par exemple, `go build a.go`. `go build` va compiler les fichiers dans ce dossier.
- Vous pouvez également définir le nom du fichier généré. Par exemple, dans le projet `mathapp` (section 1.2), lancer `go build -o astaxie.exe` va générer `astaxie.exe` au lieu de `mathapp.exe`. Le nom par défaut est le nom du dossier (package nom `main`) ou celui du premier fichier source (paquet `main`).

(D'après [Les Spécifications du Langage Go](#), les noms de paquets devraient être ceux suivant la déclaration `package` en première ligne de vos fichiers sources. Cela n'a pas à être le même que celui du dossier, et que le nom de fichier de l'exécutable soit celui du dossier par défaut.)

- `go build` ignore les fichiers dont le nom commence par `_` ou `.`.
- Si vous voulez avoir différents fichiers sources par système d'exploitation, vous pouvez nommer vos fichiers avec pour suffixe le

nom du système. Supposez qu'il y ait plusieurs fichiers pour charger des tableaux. Ces fichiers peuvent être nommés comme suit:

array_linux.go | array_darwin.go | array_windows.go | array_freebsd.go

`go build` choisira celui qui correspondra à votre système d'exploitation. Par exemple, seul sera compilé `array_linux.go` sous Linux, les autres seront ignorés.

go clean

Cette commande sert à supprimer les fichiers générés par le compilateur, comprenant les fichiers suivants:

```
_obj/           // Ancien dossier objet, créé par les Makefiles
_test/         // Ancien dossier test, créé par les Makefiles
_testmain.go   // Ancien dossier de gtest, créé par les Makefiles

test.out       // Ancien fichier de test, créé par les Makefiles
build.out      // Ancien fichier de test, créé par les Makefiles
*.[568ao]      // fichiers objets, créés par les Makefiles

DIR(.exe)      // généré par go build
DIR.test(.exe) // généré par go test -c
MAINFILE(.exe) // généré par go build MAINFILE.go
```

J'utilise généralement cette commande pour supprimer les fichiers avant de mettre à jour mon projet sur Github. Ils sont utiles lors de tests en local, mais inutiles à avoir dans votre système de gestion de version.

go fmt

Les développeurs C/C++ doivent avoir l'habitude des débats sur quelle convention d'écriture de code est la meilleure: le style K&R ou ANSI. Avec Go, il n'existe qu'une convention à appliquer. Par exemple, les parenthèses ouvrantes doivent être en fin de ligne, et ne peuvent être sur une ligne

toutes seules, sinon vous aurez des erreurs à la compilation!

Par chance, vous n'avez pas à retenir toutes ces règles. `go fmt` le fait à votre place. exécutez simplement `go fmt <File name>.go` dans un terminal. Je n'utilise pas souvent cette commande car les IDE l'exécutent normalement automatiquement lorsque vous sauvegardez un fichier. Nous reviendrons sur les IDE dans la section suivante.

On utilise généralement `gofmt -w` au lieu de `go fmt`. Ce dernier ne modifie pas vos fichiers source après formatage. `gofmt -w src` formate tout votre projet.

go get

Cette commande récupère des paquets distants. Sont supportés BitBucket, Github, Google Code et Launchpad. Il se passe en fait deux choses à l'exécution de cette commande. En premier lieu, Go télécharge le code source, puis il exécute `go install`. Avant d'utiliser cette commande, assurez-vous d'installer les outils nécessaires.

```
BitBucket (Mercurial Git)
Github (git)
Google Code (Git, Mercurial, Subversion)
Launchpad (Bazaar)
```

Pour utiliser cette commande, vous devez installer ces outils correctement. N'oubliez pas de définir votre `$PATH`. Notez que `go get` supporte également vos noms de domaine personnalisés. Exécutez `go help remote` pour en connaître les détails.

go install

Cette commande compile tous les paquets et génère les fichiers, puis les déplace dans `$GOPATH/pkg` ou `$GOPATH/bin`.

go test

Cette commande charge tous les fichiers qui ont pour nom `*_test.go` et génère les fichiers de test, puis affiche des informations similaires à celles-ci.

```
ok      archive/tar      0.011s
FAIL    archive/zip      0.022s
ok      compress/gzip    0.033s
...
```

Elle teste tous vos fichiers de test par défaut. Utilisez la commande `go help testflag` pour plus de détails.

go doc

Beaucoup de personnes disent qu'il est inutile d'avoir une quelconque documentation tiers pour programmer en Go (bien qu'en fait j'ai déjà développé [CHM](#)). Go a par défaut un outil très élaboré pour travailler avec la documentation.

Alors comment rechercher des informations sur un paquet dans la documentation? Par exemple, si vous voulez plus d'informations sur le paquet `builtin`, utilisez la commande `go doc builtin`. De la même manière, utilisez la commande `go doc net/http` pour rechercher la documentation du paquet `http`. Si vous voulez plus de détails spécifiques à une fonction, utilisez `godoc fmt Printf`, ou `godoc -src fmt Printf` pour voir le code source.

Exécutez la commande `godoc -http=:8080`, puis ouvrez `127.0.0.1:8080` dans votre navigateur. Vous devriez voir un `golang.org` local. Il peut non seulement afficher les informations des paquets standards, mais aussi les paquets de votre dossier `$GOPATH/pkg`. C'est très bien pour ceux bloqués derrière la Grande Muraille de Chine.

Autres commandes

Go fournit d'autres commandes que celles vues précédemment:

```
go fix      // mise à jour du code d'une version antérieure
            // à go1 vers une version plus récente que go1
go version  // obtenir des informations sur votre version de Go
go env      // afficher vos variables d'environnement Go
go list     // lister tous les paquets installés
go run      // compiler les fichiers temporaires et lancer l'applica
tion
```

Les commandes que l'on a vues proposent également plus d'options que ce que nous avons vu. Vous pouvez utiliser `go help <command>` pour les visualiser.

Navigation

- [Table des matières](#)
- Section précédente: [\\$GOPATH et workspace](#)
- Section suivante: [Outils de développement Go](#)

Outils de développement Go

Dans cette section, je vais vous présenter quelques IDE qui peuvent vous aider à coder plus efficacement, grâce à des fonctionnalités telles la complétion de code et le formatage automatiques. Ils sont tous multi-plate-formes, les manipulations présentées ne devraient donc pas être très éloignées, même si vous n'utilisez pas le même système d'exploitation.

LiteIDE

LiteIDE est un IDE léger et multi-plate-formes dédié uniquement au

développement en Go, développé par visualfc.

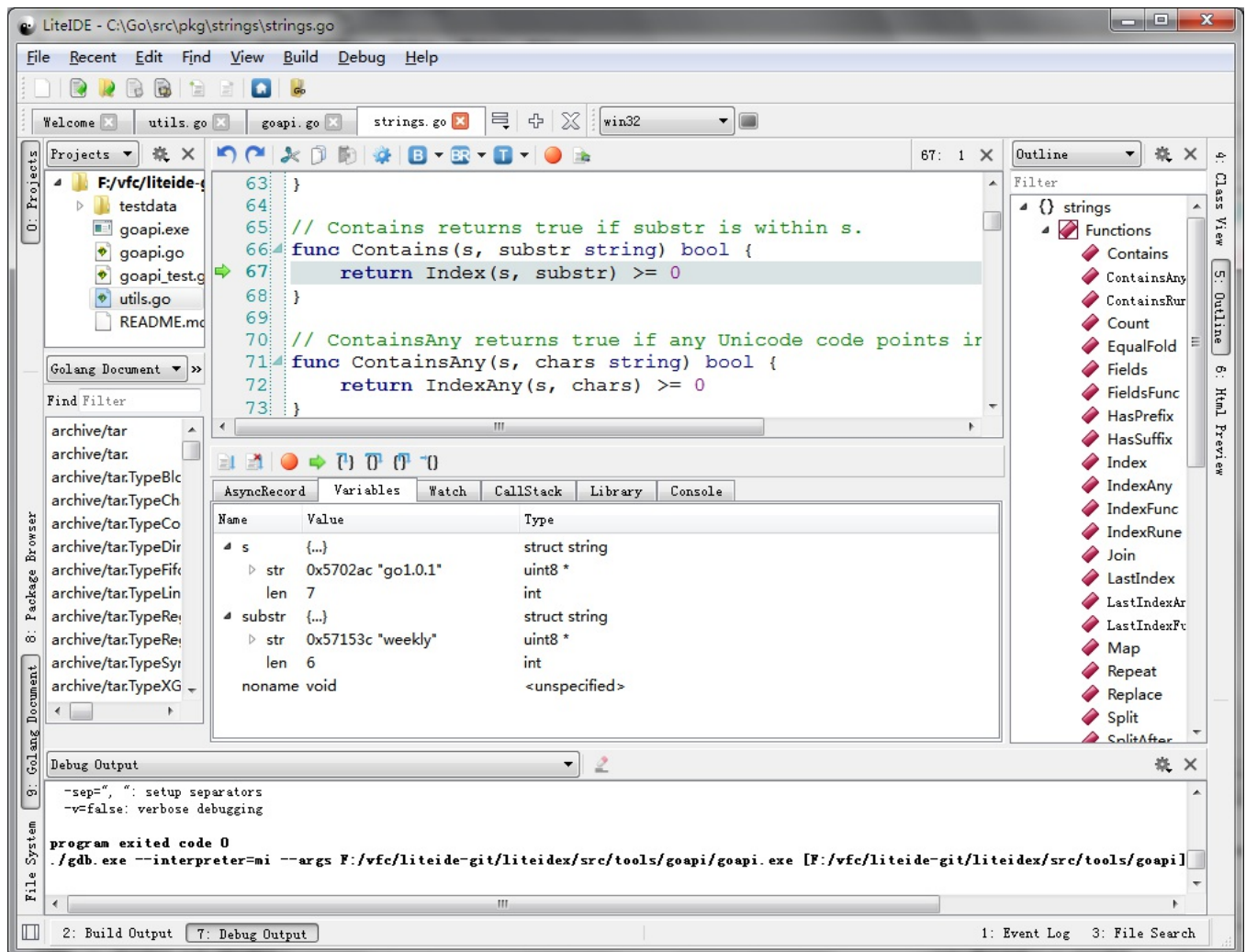


Figure 1.4 Panneau principal de LiteIDE

Fonctionnalités de LiteIDE:

- Multi-plate-formes
 - Windows
 - Linux
 - Mac OS
- Compilation croisée
 - Gestion d'environnements de compilation multiples
 - Support de la compilation croisée avec Go
- Standards en gestion de projet

- Documentation basée sur le \$GOPATH
- Compilation basée sur le \$GOPATH
- Index la documentation de l'API basée sur le \$GOPATH
- Éditeur de code source Go
 - Mise en forme du code
 - Support complet de gocode
 - Gestion de la documentation and et de l'index de l'API de Go
 - Afficher l'expression d'un code via `F1`
 - Naviguer entre déclaration de fonctions via `F2`
 - Support Gdb
 - Auto-formatage via `gofmt`
- Autres
 - Multi-langues
 - Système d'extensions
 - Thèmes d'éditeur de texte
 - Support de la syntaxe basé sur Kate
 - Complétion intelligente basée en mode full-text
 - Personnalisation des raccourcis
 - Support Markdown
 - Prévisualisation en temps réel
 - CSS personnalisé
 - Exportation en HTML et PDF
 - Conversion et fusion en HTML et PDF

Installation de LiteIDE

- Installer LiteIDE
 - [Page de téléchargement](#)
 - [Code source](#)

Vous devez installer Go en premier, puis télécharger la version

adaptée à votre système d'exploitation. Décompressez le paquet pour ensuite l'exécuter directement.

- Installer gocode

Vous devez installer gocode pour utiliser la complétion intelligente

```
go get -u github.com/nsf/gocode
```

- Environment de compilation

Changer de configuration de LiteIDE pour correspondre à votre système d'exploitation Dans Windows avec la version 64-bit de Go, vous devriez utiliser win64 comme configuration de votre environnement dans la barre d'outils. Ensuite, choisissez `Options`, puis `LiteEnv` dans la liste de gauche et sélectionnez le fichier `win64.env` dans la liste de droite.

```
GOROOT=c:\go
GOBIN=
GOARCH=amd64
GOOS=windows
CGO_ENABLED=1

PATH=%GOBIN%;%GOROOT%\bin;%PATH%
```

Modifiez `GOROOT=c:\go` par le chemin de votre installation Go, puis sauvegardez. Si vous avez MinGW64, ajoutez `c:\MinGW64\bin` au chemin de votre variable d'environnement pour le support de `cgo`.

Pour Linux avec Go en version 64-bit, choisissez linux64 comme configuration de votre environnement dans la barre d'outils. Ensuite, choisissez `Options`, puis `LiteEnv` dans la liste de gauche et sélectionnez le fichier `linux64.env` dans la liste de droite.

```
GOROOT=$HOME/go
```

```
GOBIN=  
GOARCH=amd64  
GOOS=linux  
CGO_ENABLED=1  
  
PATH=$GOBIN:$GOROOT/bin:$PATH
```

Modifiez `GOROOT=$HOME/go` par le chemin de votre installation Go.

- `$GOPATH`

`$GOPATH` est le chemin du dossier de vos projets. Ouvrez l'outil commande (`Ctrl+` dans LiteIDE), puis tapez `go help gopath` pour plus de détails. C'est très simple de voir et modifier votre `$GOPATH` dans LiteIDE. Choisissez `View - Setup GOPATH` pour voir et modifier ces valeurs.

Sublime Text

Dans cette partie je vais vous présenter Sublime Text 2 (Sublime tout court) + GoSublime + gocode + MarGo.

- Complétion Intelligente

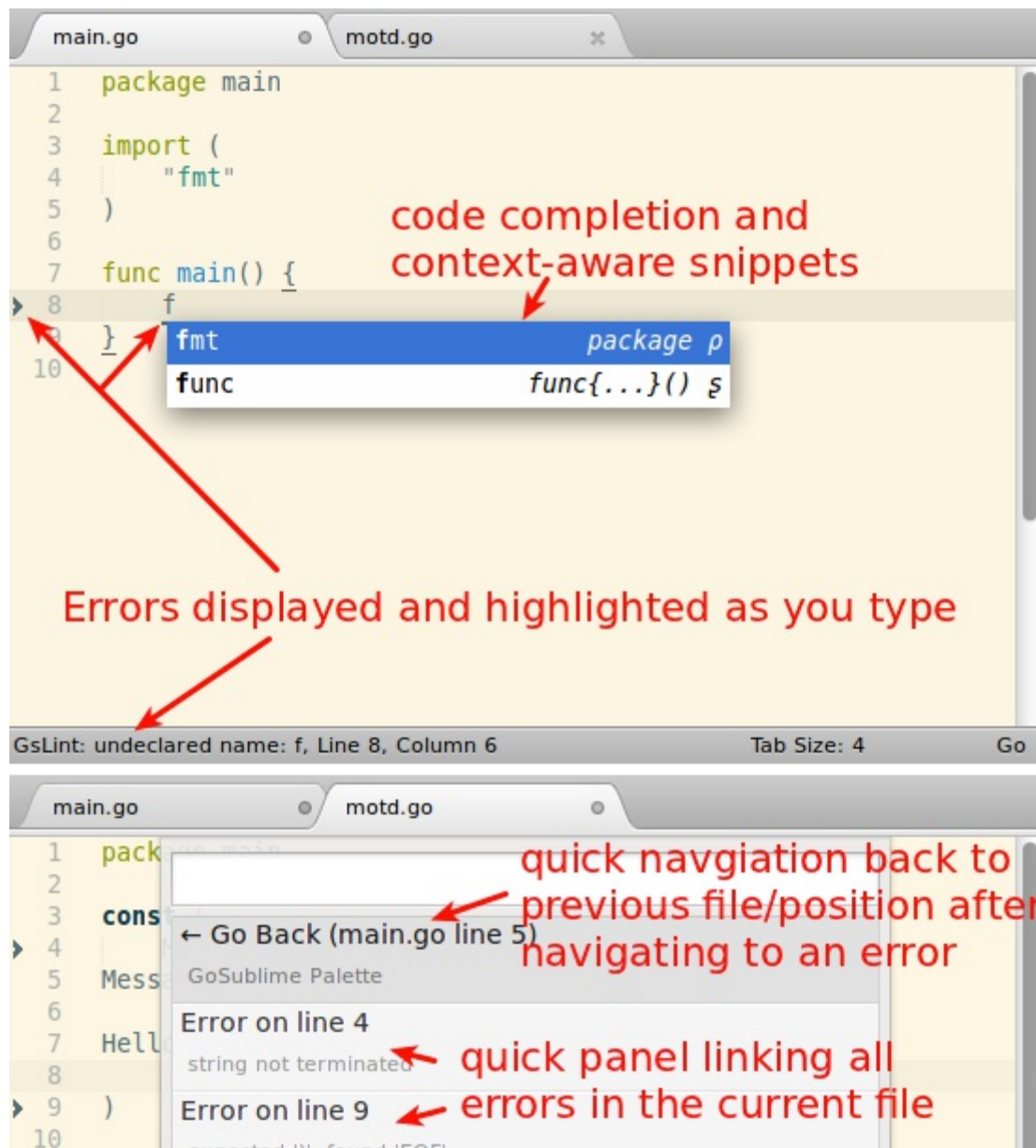


Figure 1.5 Complétion intelligente avec Sublime

- Auto-formatage du code source
- Gestion de projet

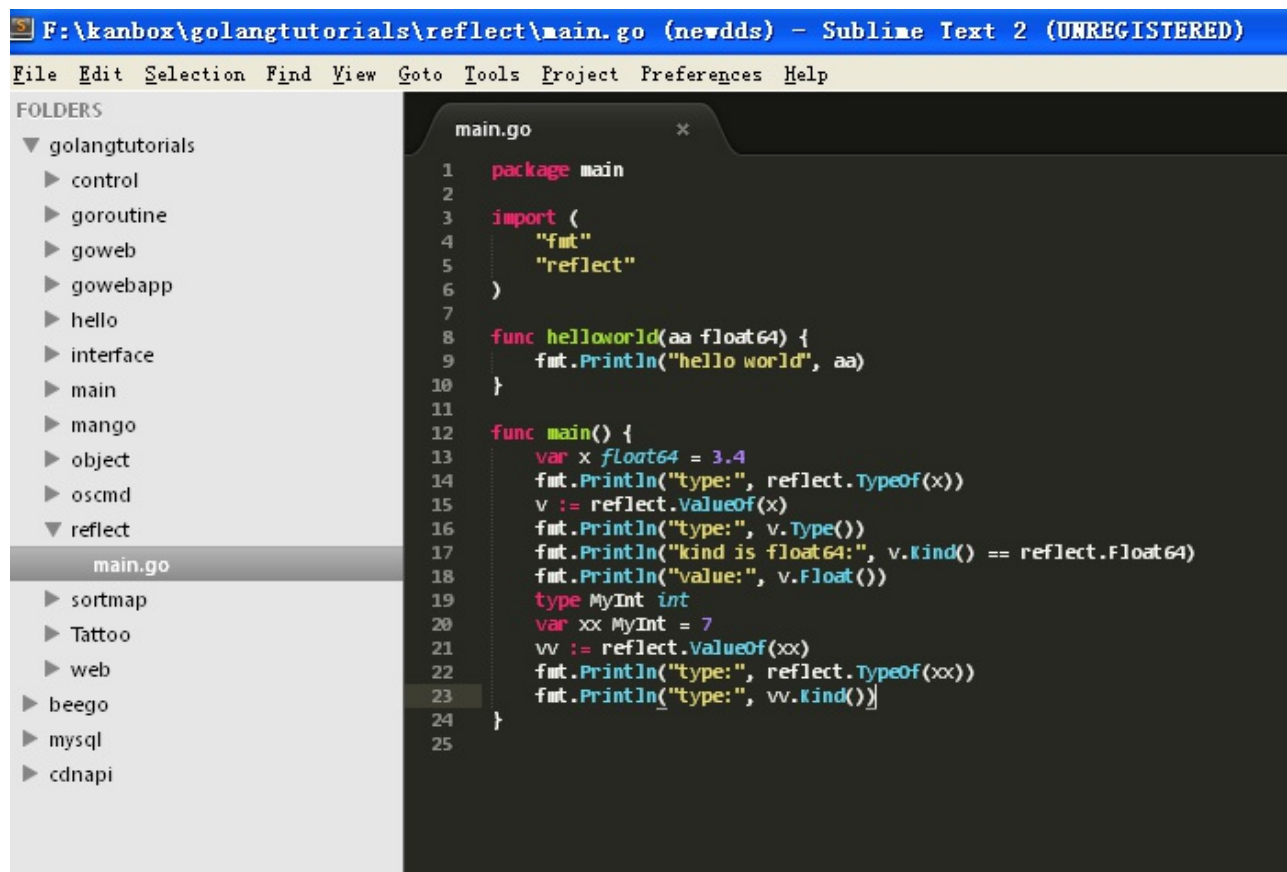


Figure 1.6 Gestion de projet sous Sublime

- Coloration syntaxique
- Version gratuite d'essai permanente sans limitation de fonctionnalités. Vous serez invité de temps en temps à acheter une licence, que vous pourrez refuser si vous voulez. Bien sûr, si vous trouvez que Sublime augmente votre productivité et que vous l'appréciez vraiment, merci d'acheter une licence et de supporter son développement!

Commencez par télécharger une version de [Sublime](#) adaptées à votre système d'exploitation.

1. Tapez `Ctrl+`, ouvrez l'invite de commandes et rentrez les commandes suivantes:.

```
import urllib2,os; pf='Package Control.sublime-package'; ipp=s
ublime.installed_packages_path(); os.makedirs(ipp) if not os.pa
th.exists(ipp) else None; urllib2.install_opener(urllib2.build_
```

```
opener(urllib2.ProxyHandler())); open(os.path.join(ipp,pf), 'wb'
).write(urllib2.urlopen('http://sublime.wbond.net/'+pf.replace(
' ', '%20')).read()); print 'Merci de redémarrer Sublime Text po
ur finaliser l'installation'
```

Redémarrez Sublime text en fin d'installation. Vous devriez trouver une option **Package Control** dans le menu "Preferences".

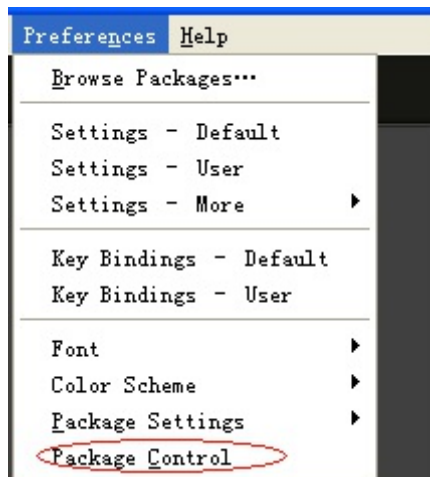


Figure 1.7 Gestion de paquet dans Sublime

2. Pour installer GoSublime, SidebarEnhancements et Go Build, tapez **Ctrl+Shift+p** pour ouvrir Package Control, puis tapez **pcip** (raccourci pour "Package Control: Install Package").

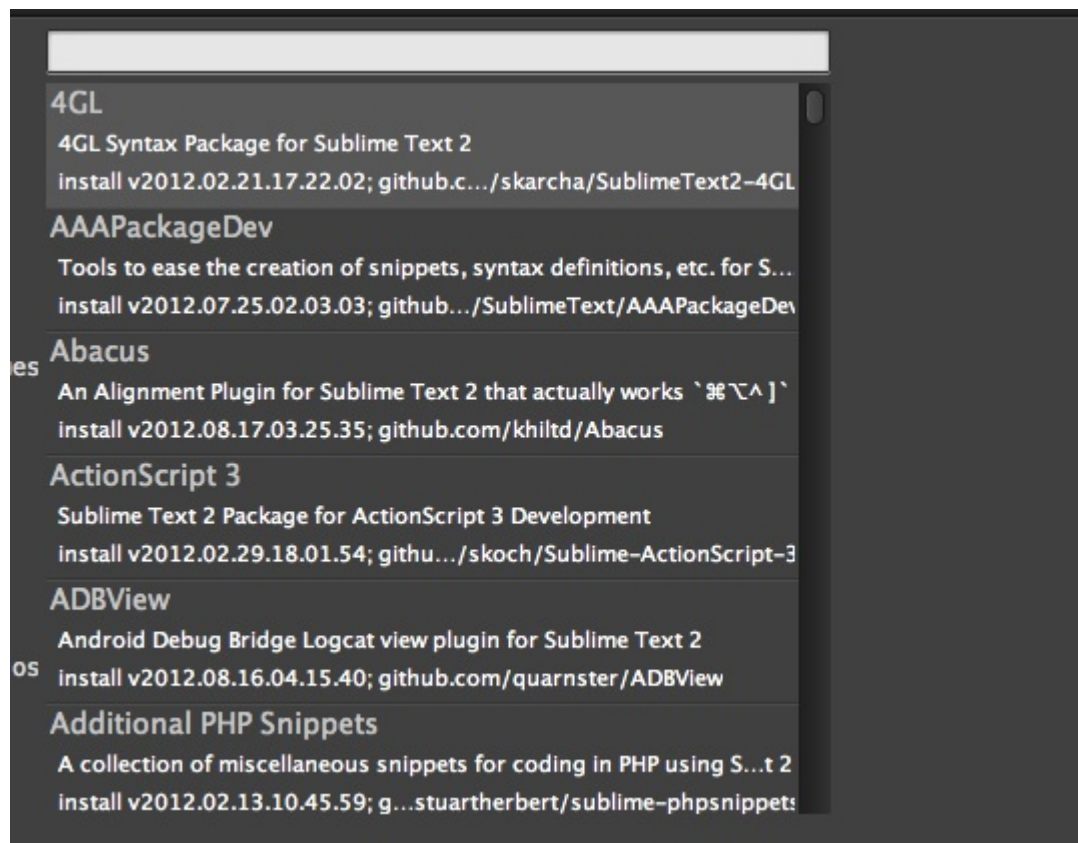


Figure 1.8 Installer des paquets sous Sublime

Puis tapez "GoSublime", entrez OK pour installer le paquet, et répétez ces commandes pour installer SidebarEnhancements et Go Build. À nouveau, redémarrez l'éditeur quand l'installation s'achève.

3. Pour vérifier la réussite de l'installation, ouvrez Sublime, puis ouvrez le fichier `main.go` pour vérifier que la coloration syntaxique est correcte. Tapez `import` pour voir si le prompt de la complétion de code s'affiche. Après avoir tapé `import "fmt"`, tapez `fmt.` juste après `import` pour voir si oui ou non la complétion intelligente de code des fonctions a été activée avec succès.

Si tout fonctionne correctement, vous êtes prêt.

Sinon, vérifiez à nouveau votre \$PATH. Ouvrez un terminal, tapez `gocode`. S'il ne s'exécute pas, votre \$PATH n'est pas configuré correctement.

Vim

Vim est un éditeur très populaire chez les programmeurs, qui est une évolution de son prédécesseur, Vi. Il a des fonctionnalités de complétion intelligente, compilation et navigation entre les erreurs.

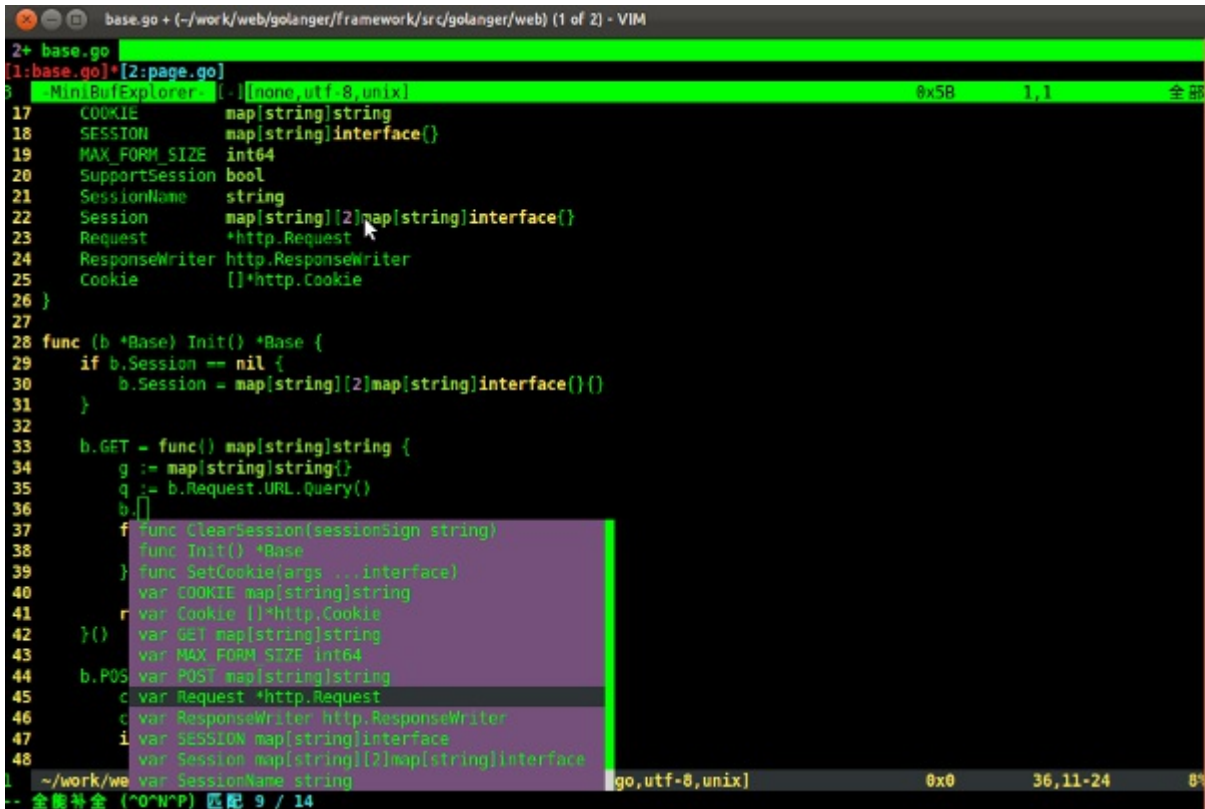


Figure 1.8 Complétion intelligente pour Go sous Vim

1. Coloration syntaxique sous Go

```
cp -r $GOROOT/misc/vim/* ~/.vim/
```

2. Activer la coloration syntaxique

```
filetype plugin indent on  
syntax on
```

3. Installer `gocode`

```
go get -u github.com/nsf/gocode
```

`gocode` sera installé dans `$GOBIN` par défaut

4. Configurer `gocode`

```
~ cd $GOPATH/src/github.com/nsf/gocode/vim
~ ./update.bash
~ gocode set propose-builtins true
propose-builtins true
~ gocode set lib-path "/home/border/gocode/pkg/linux_amd64"
lib-path "/home/border/gocode/pkg/linux_amd64"
~ gocode set
propose-builtins true
lib-path "/home/border/gocode/pkg/linux_amd64"
```

Explications sur la configuration de `gocode`:

`propose-builtins`: active ou non la complétion automatique; faux par défaut.

`lib-path`: `gocode` cherche uniquement les paquets dans `$GOPATH/pkg/$GOOS_$GOARCH` et `$GOROOT/pkg/$GOOS_$GOARCH`. Ce paramètre peut-être utilisé pour ajouter des chemins additionnels.

5. Félicitations! lancez `:e main.go` pour explorer le monde de Go!

Emacs

Emacs est l'arme fatale. Ce n'est pas seulement un éditeur, mais aussi un IDE très puissant.

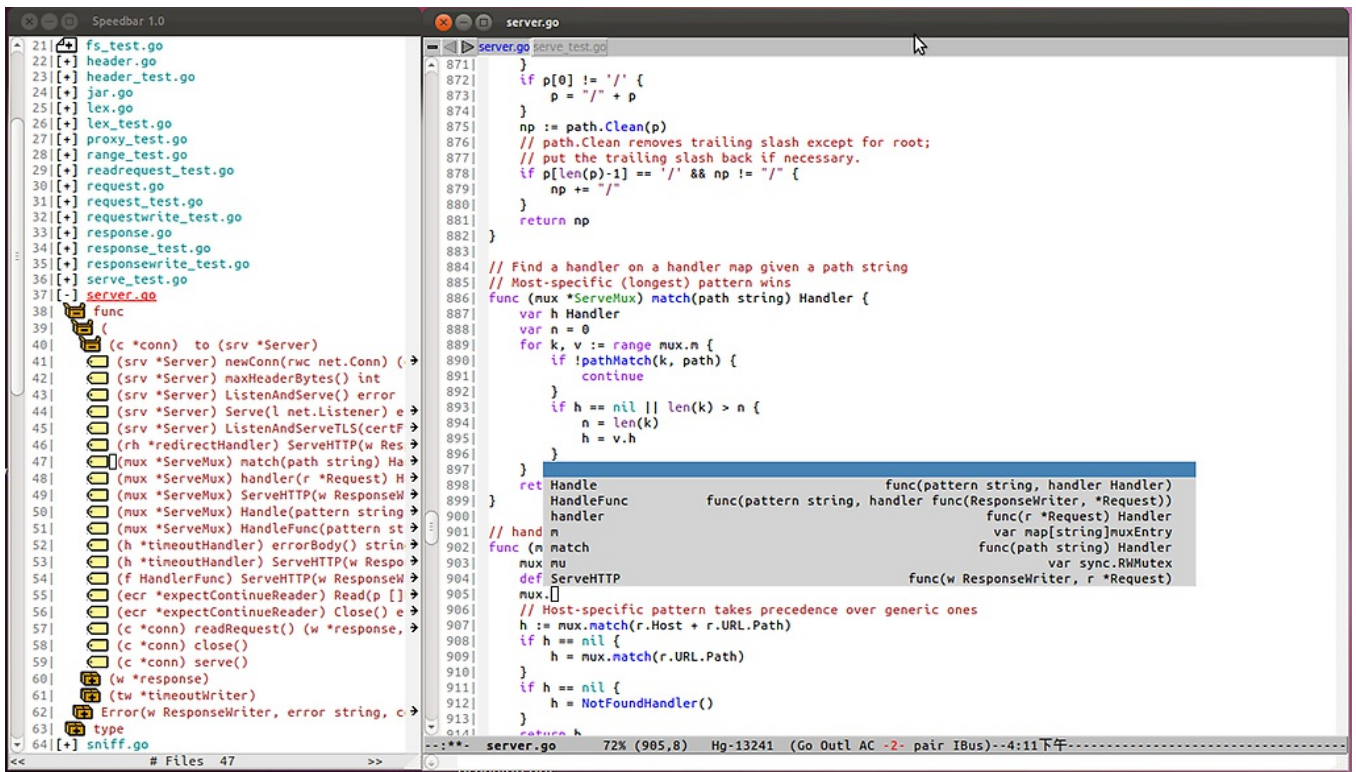


Figure 1.10 Panneau principal de l'éditeur Go d'Emacs

1. Coloration syntaxique

```
cp $GOROOT/misc/emacs/* ~/.emacs.d/
```

2. Installer [gocode](#)

```
go get -u github.com/nsf/gocode
```

gocode sera installé dans `$GOBIN` par défaut

3. Configurer [gocode](#)

```
~ cd $GOPATH/src/github.com/nsf/gocode/vim
~ ./update.bash
~ gocode set propose-builtins true
propose-builtins true
```

```
~ gocode set lib-path "/home/border/gocode/pkg/linux_amd64"
lib-path "/home/border/gocode/pkg/linux_amd64"
~ gocode set
propose-builtins true
lib-path "/home/border/gocode/pkg/linux_amd64"
```

4. Installer l'[Auto Completion](#)

Téléchargement et décompression

```
~ make install DIR=$HOME/.emacs.d/auto-complete
```

Configurer ~/.emacs file

```
;; auto-complete
(require 'auto-complete-config)
(add-to-list 'ac-dictionary-directories "~/.emacs.d/auto-compl
ete/ac-dict")
(ac-config-default)
(local-set-key (kbd "M-./") 'semantic-complete-analyze-inline)
(local-set-key "." 'semantic-complete-self-insert)
(local-set-key ">" 'semantic-complete-self-insert)
```

Cliquez ce [lien](#) pour plus de détails.

5. Configurer .emacs

```
;; golang mode
(require 'go-mode-load)
(require 'go-autocomplete)
;; speedbar
;; (speedbar 1)
(speedbar-add-supported-extension ".go")
(add-hook
'go-mode-hook
'(lambda ()
  ;; gocode
  (auto-complete-mode 1)
```



```

(setq ac-sources '(ac-source-go))
;; Imenu & Speedbar
(setq imenu-generic-expression
  '(("type" "^type *\\([^\t\n\r\f]*\\)" 1)
    ("func" "^func *\\(.*\\)" {" 1"}))
(imenu-add-to-menubar "Index")
;; Outline mode
(make-local-variable 'outline-regexp)
(setq outline-regexp "//\\.\\.\\|//[^\r\n\f][^\r\n\f]\\|pack\\
\\|func\\|impo\\|cons\\|var\\.\\.\\|type\\|\\t\\t*\\.\\.\\.")
(outline-minor-mode 1)
(local-set-key "\M-a" 'outline-previous-visible-heading)
(local-set-key "\M-e" 'outline-next-visible-heading)
;; Menu bar
(require 'easymenu)
(defconst go-hooked-menu
  '("Go tools"
    ["Go run buffer" go t]
    ["Go reformat buffer" go-fmt-buffer t]
    ["Go check buffer" go-fix-buffer t]))
(easy-menu-define
  go-added-menu
  (current-local-map)
  "Go tools"
  go-hooked-menu)

;; Other
(setq show-trailing-whitespace t)
))
;; helper function
(defun go ()
  "run current buffer"
  (interactive)
  (compile (concat "go run " (buffer-file-name))))

;; helper function
(defun go-fmt-buffer ()
  "run gofmt on current buffer"
  (interactive)
  (if buffer-read-only
    (progn
      (ding)
      (message "Buffer is read only"))
    (let ((p (line-number-at-pos))
          (filename (buffer-file-name))
          (old-max-mini-window-height max-mini-window-height))

```

```

        (show-all)
        (if (get-buffer "*Go Reformat Errors*")
            (progn
              (delete-windows-on "*Go Reformat Errors*")
              (kill-buffer "*Go Reformat Errors*"))
            (setq max-mini-window-height 1)
            (if (= 0 (shell-command-on-region (point-min) (point-m
ax) "gofmt" "*Go Reformat Output*" nil "*Go Reformat Errors*" t
))
                (progn
                  (erase-buffer)
                  (insert-buffer-substring "*Go Reformat Output*")
                  (goto-char (point-min))
                  (forward-line (1- p)))
                (with-current-buffer "*Go Reformat Errors*"
                  (progn
                    (goto-char (point-min))
                    (while (re-search-forward "<standard input>" nil t)
                      (replace-match filename))
                    (goto-char (point-min))
                    (compilation-mode))))
                  (setq max-mini-window-height old-max-mini-window-heigh
t)

                  (delete-windows-on "*Go Reformat Output*")
                  (kill-buffer "*Go Reformat Output*")))))
;; helper function
(defun go-fix-buffer ()
  "run gofix on current buffer"
  (interactive)
  (show-all)
  (shell-command-on-region (point-min) (point-max) "go tool
fix -diff"))

```

6. Félicitations, vous avez fini! Speedbar est fermé par défaut - décommentez `;;(speedbar 1)` pour activer cette fonctionnalité, ou vous pouvez l'utiliser via `M-x speedbar`.

Eclipse

Eclipse est aussi un très bon outil de développement. Je vais vous montrer comment l'utiliser pour écrire des programmes Go.

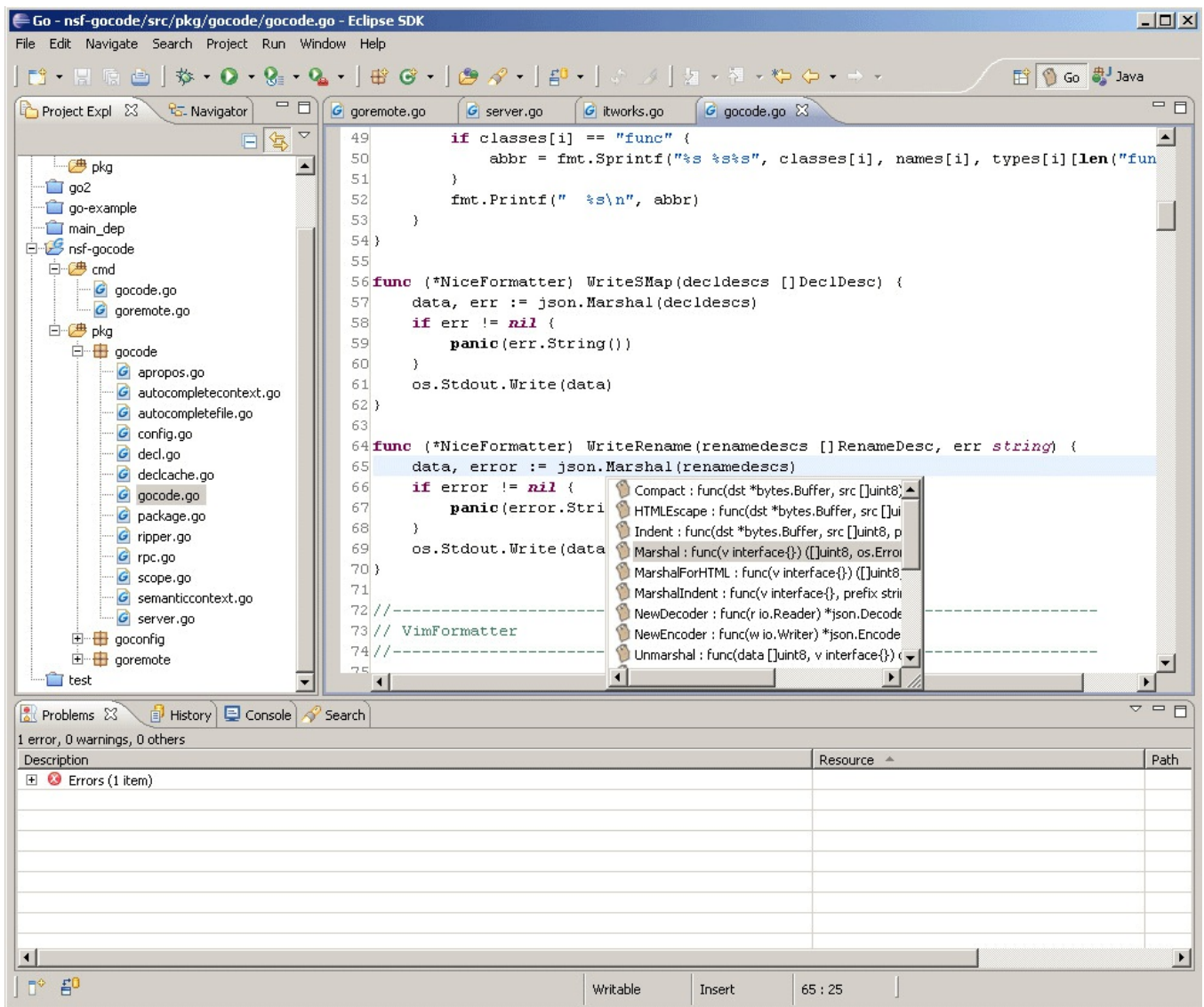


Figure 1.1 Panneau principal d'Eclipse pour développer en Go

1. Téléchargez et installez [Eclipse](#)
2. Téléchargez [goclipse](#)

Instructions d'installation

3. Téléchargez gocode
gocode sur Github.

<https://github.com/nsf/gocode>

Vous devez installer git sous Windows, généralement via [msysgit](#)

Installer gocode en ligne de commandes

```
go get -u github.com/nsf/gocode
```

Vous pouvez l'installer depuis les sources si vous préférez. .

4. Téléchargez et installez [MinGW](#)

5. Configurez les extensions.

Windows->Préférences->Go

(1).Configurez le compilateur Go

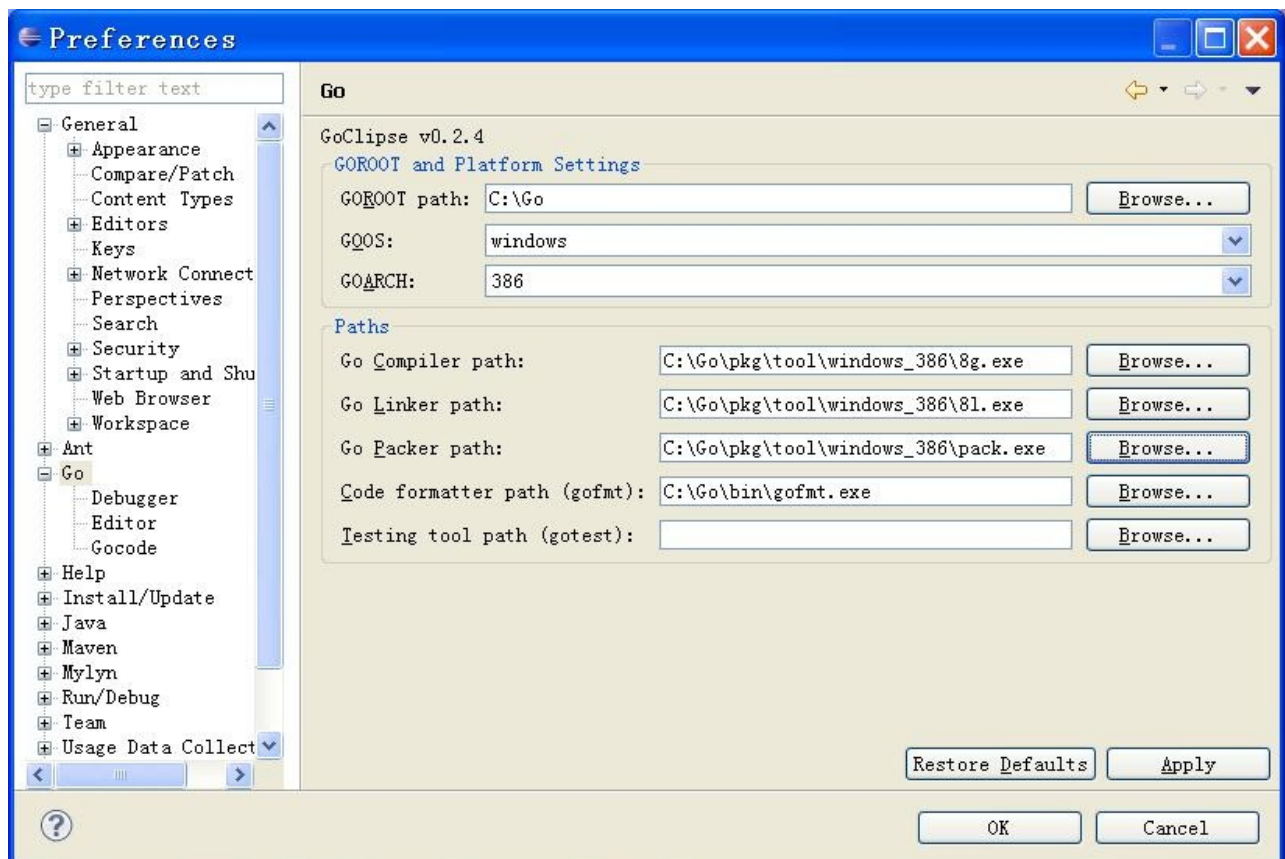


Figure 1.12 Paramètres Go dans Eclipse

(2).Configurez gocode(optionnel), paramétrez le chemin gocode vers l'emplacement de gocode.exe.

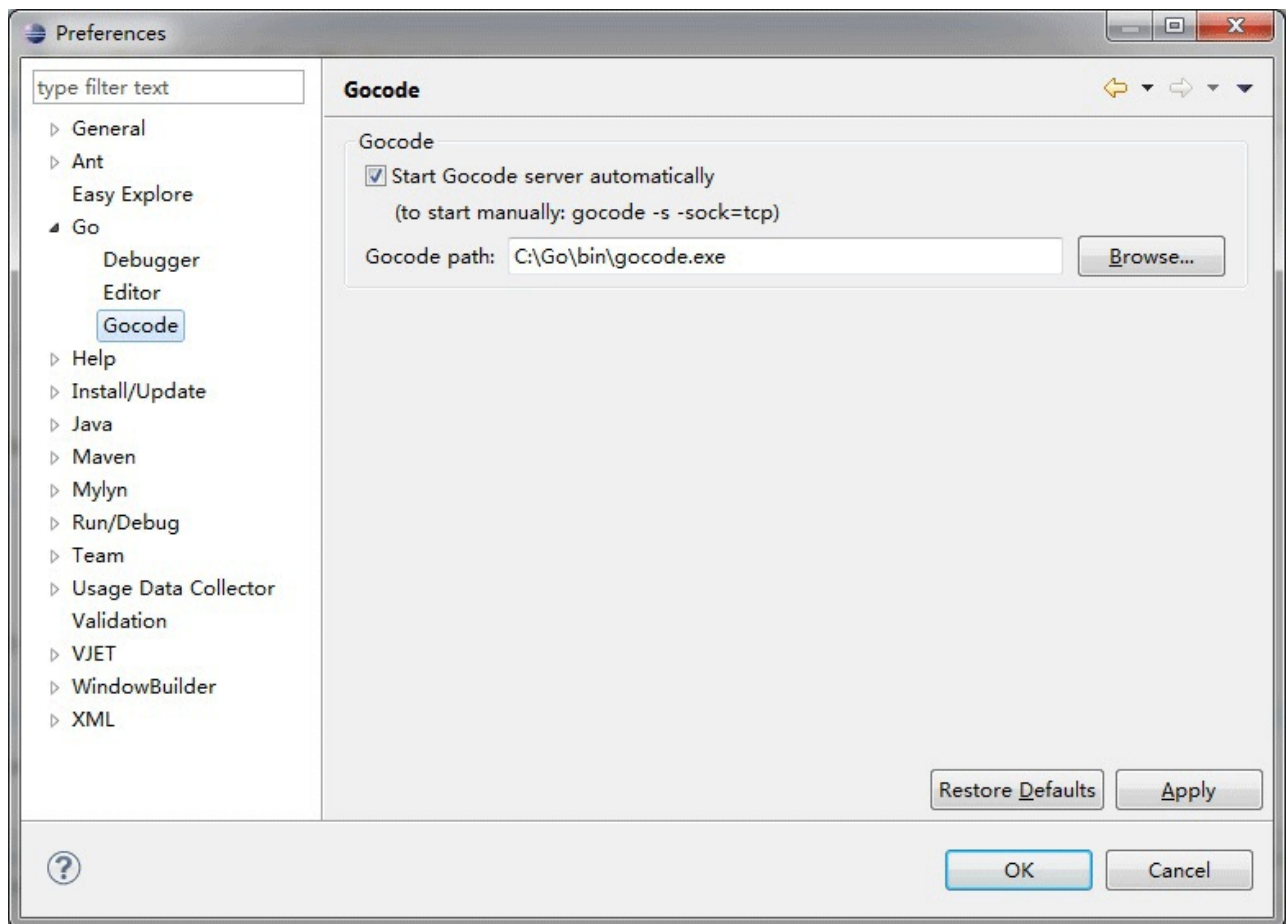


Figure 1.13 Paramètres gocode

(3).Configurez gdb(optionnel), paramétrez le chemin de gdb vers l'emplacement de gdb.exe.

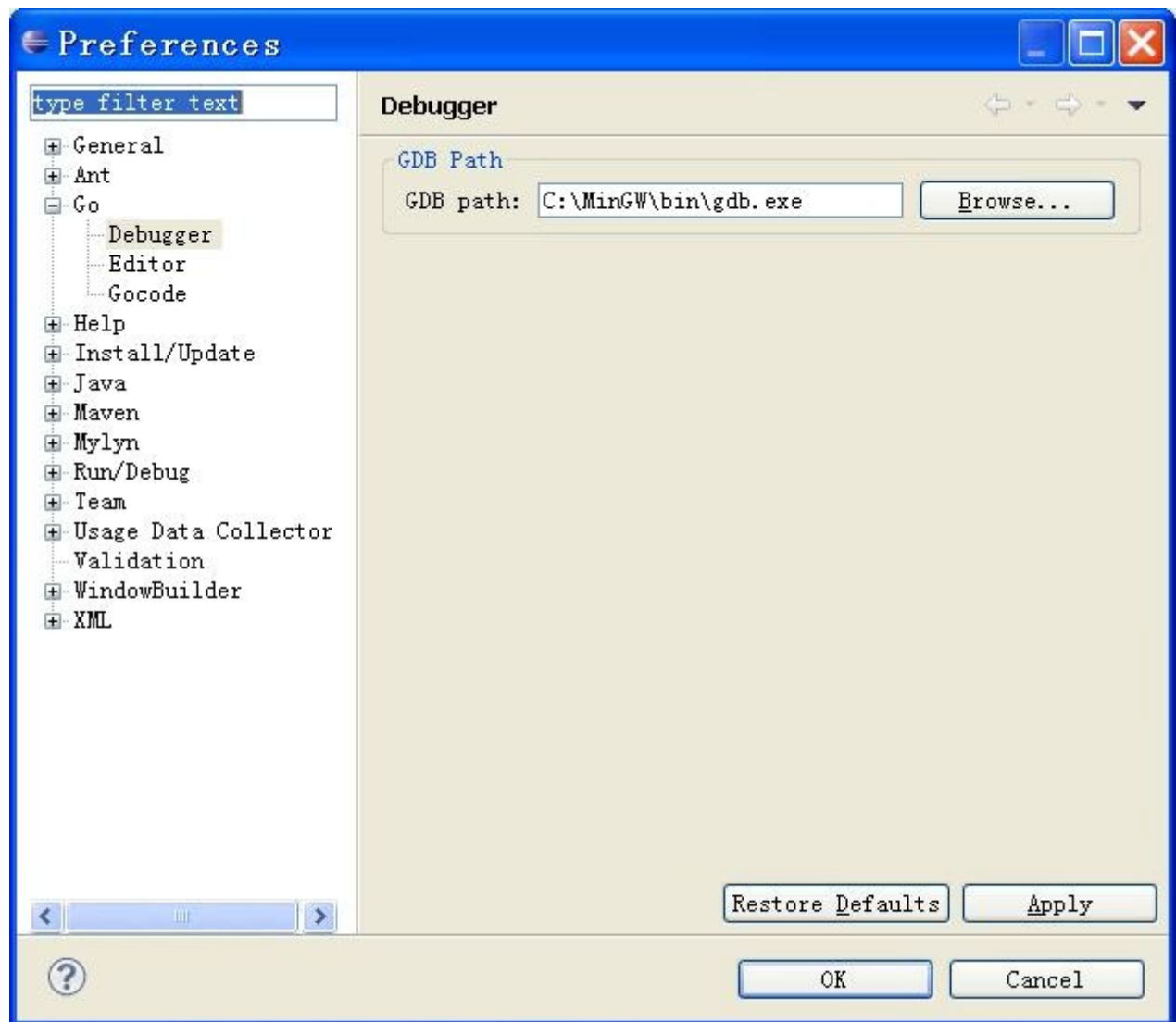


Figure 1.14 Paramètre gdb

6. Vérifiez l'installation

Créez un nouveau projet Go et un fichier hello.go comme suit.

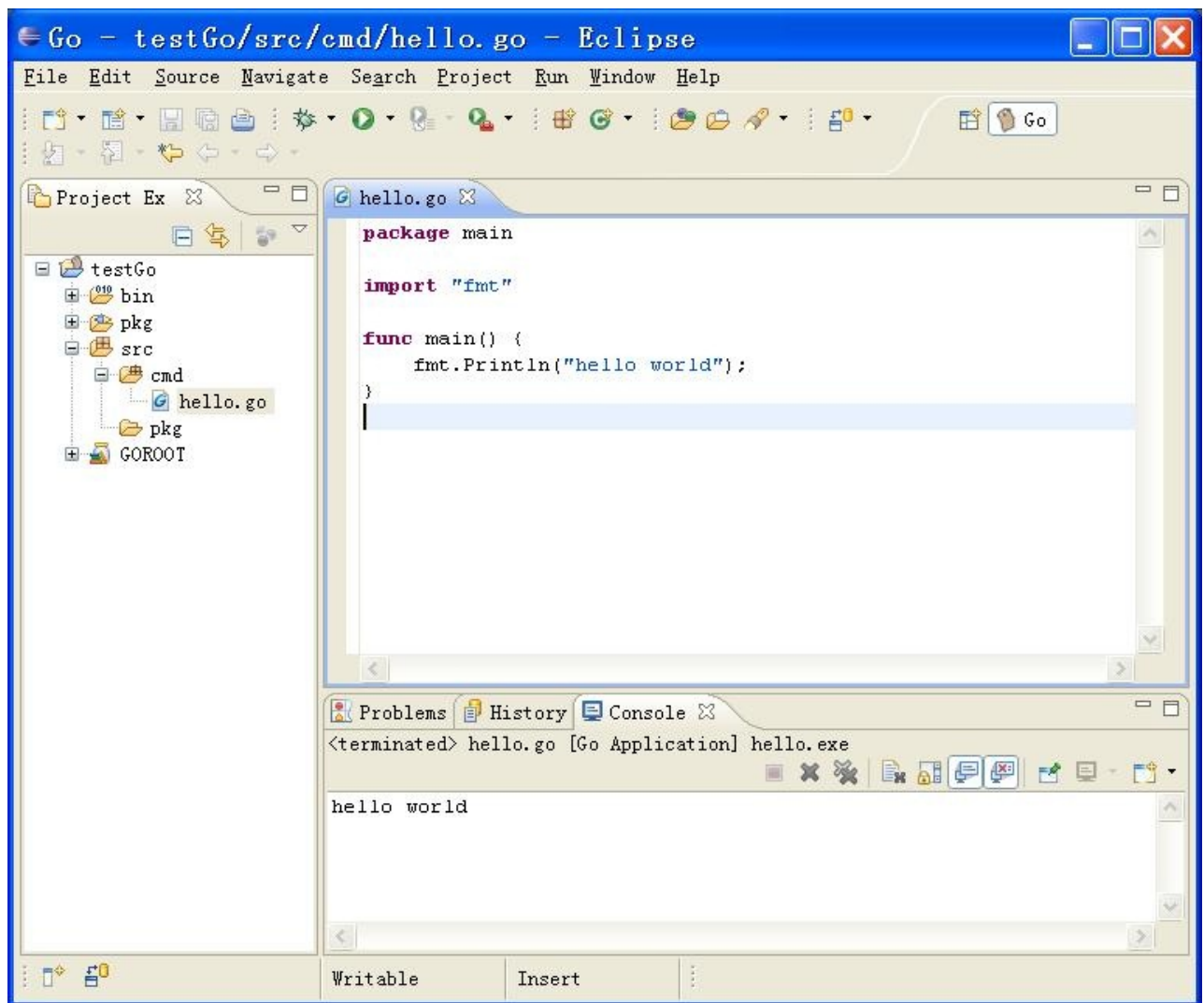


Figure 1.15 Créez un nouveau projet et fichier.

Testez l'installation comme suit (Vous avez besoin de taper la commande dans la console d'Eclipse).

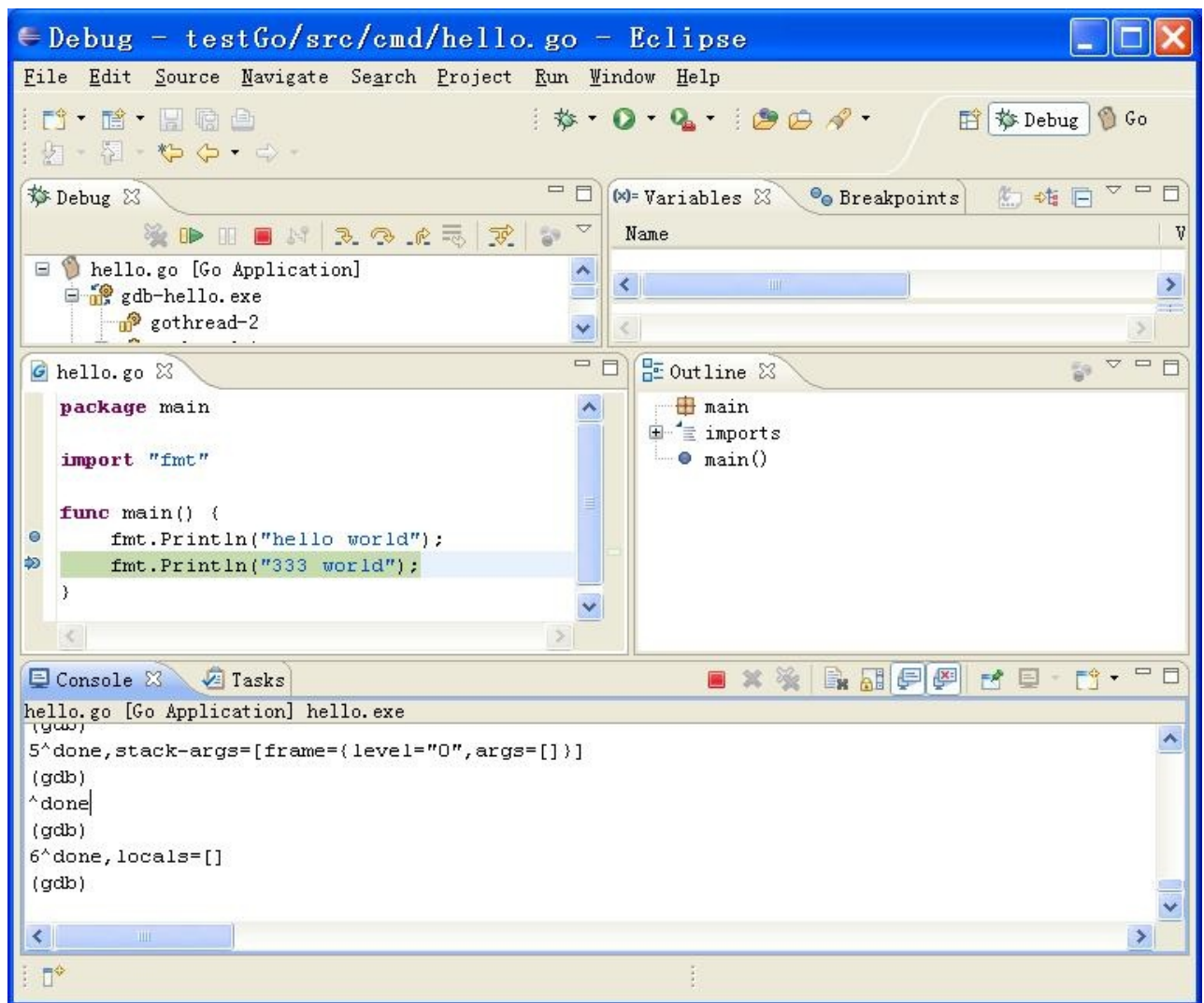


Figure 1.16 Testez Go depuis Eclipse


IntelliJ IDEA

Les développeurs Java sont familiers de cet IDE. Il supporte la coloration syntaxique et la complétion intelligente de code sous Go, via une extension.

1. Téléchargez IDEA, il n'y a aucune différence entre les éditions Ultimate et Community

Download IntelliJ IDEA 12

[Windows](#) [Mac OS X](#) [Linux](#) [See what's new in IntelliJ IDEA 12 »](#)

 Version: 12.0.2 Build: 123.123 Released: January 15, 2013 [System requirements](#) [Installation Instructions](#)

Ultimate Edition Free 30-day trial

Full-featured IDE for **JVM-based** and polyglot development

Java EE, Spring/Hibernate and other technologies support

Deployment and debugging with most application servers

Duplicate code search, dependency structure matrix, etc.

[Download Now](#)

Community Edition FREE

Lightweight IDE for **Java SE**, **Groovy & Scala** development

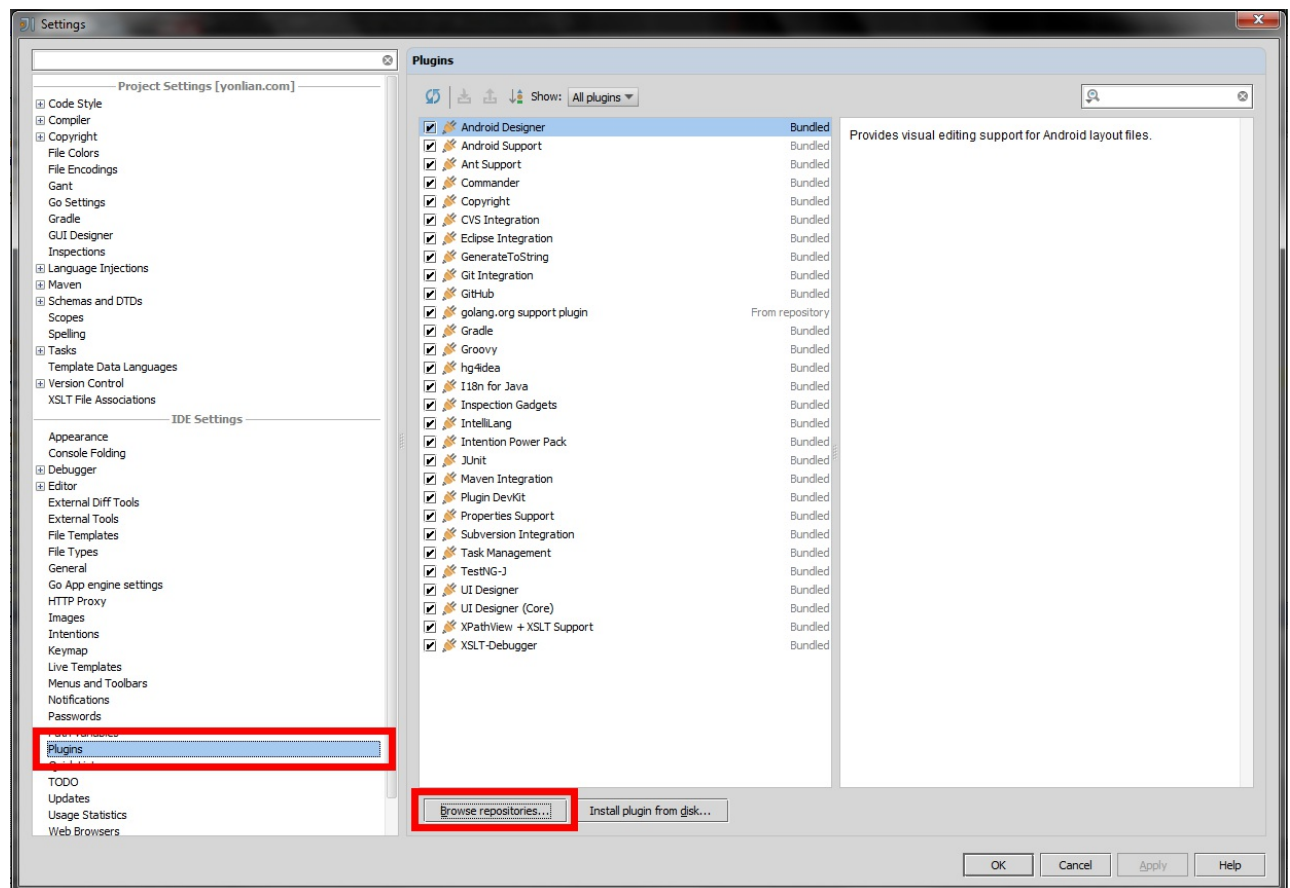
Powerful environment for building **Google Android** apps

Integration with JUnit, TestNG, popular SCMs, Ant & **Maven**

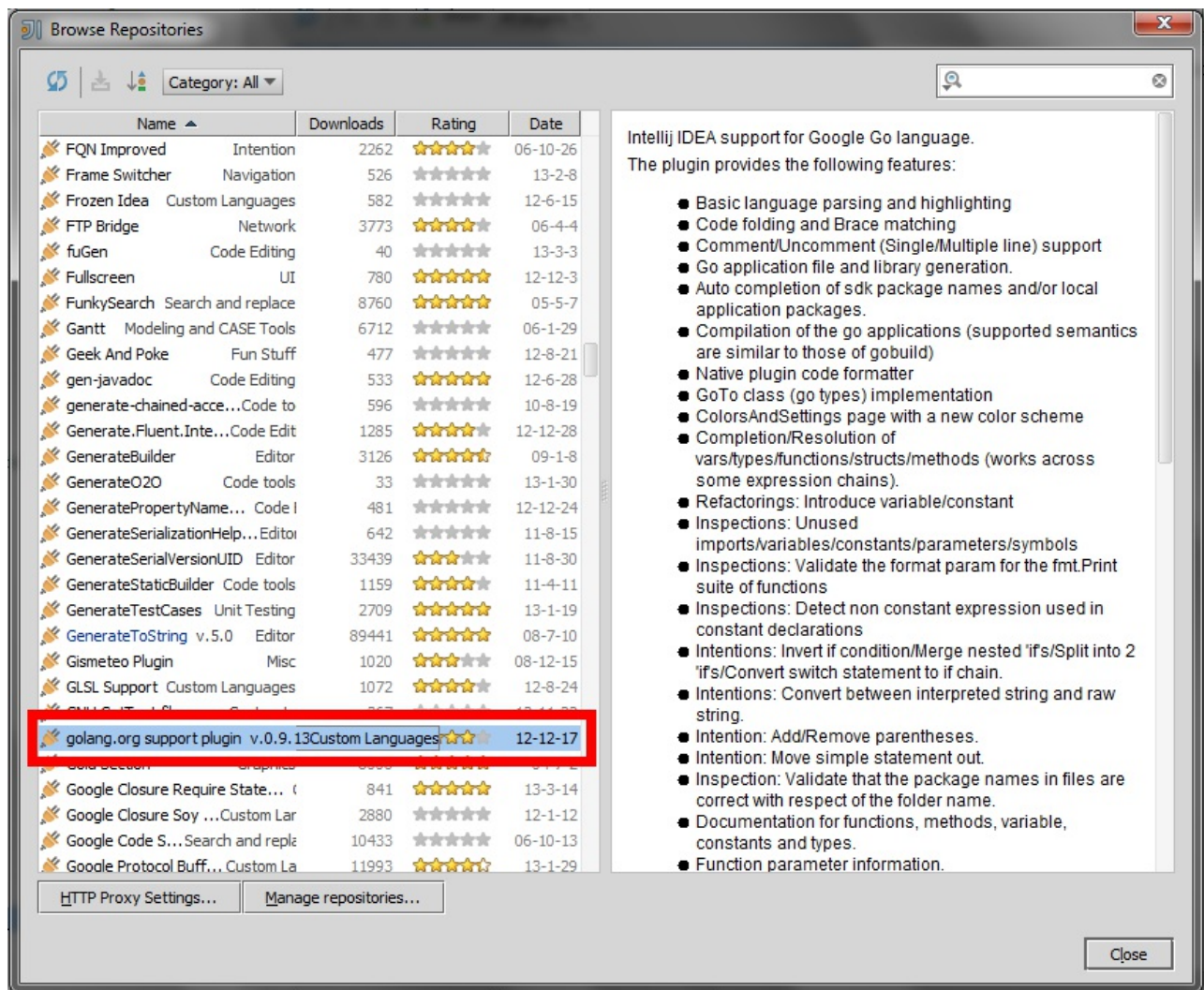
Free and open-source ([get the source code](#))

[Download Now](#)

2. Installer l'extension Go. Choisissez **File - Setting - Plugins**, puis **Browser repo**.

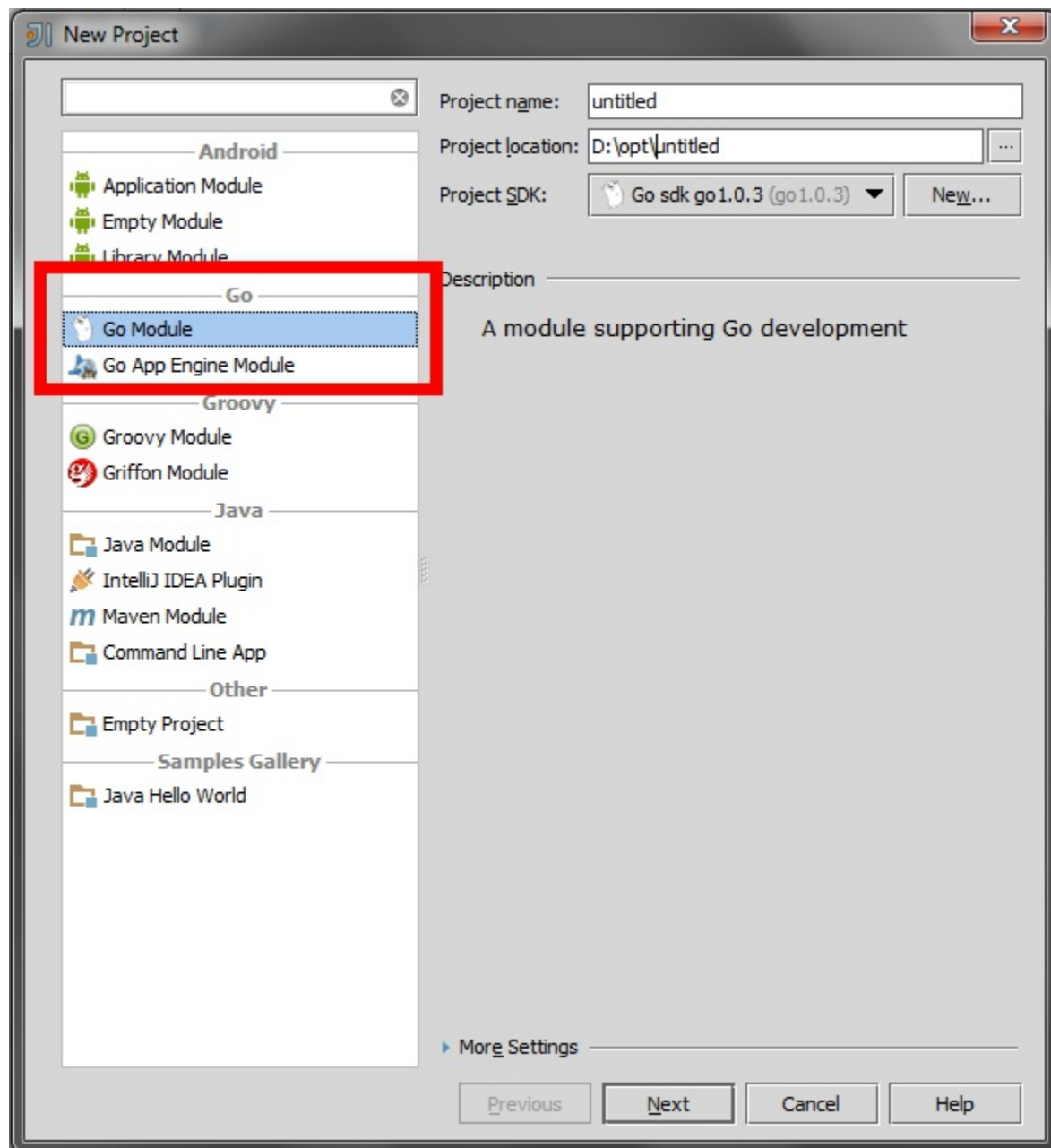


3. Cherchez `golang` , double-cliquez `download and install` et patientez jusqu'à la fin du téléchargement.



Cliquez **Apply** , puis redémarrez.

4. Maintenant vous pouvez créer un projet Go.



Entrez le chemin de votre sdk Go dans l'étape suivante - qui correspond à votre \$GOROOT.

Navigation

- [Table des matières](#)
- Section précédente: [Commandes Go](#)
- Section suivante: [Résumé](#)

1.5 Résumé

Dans ce chapitre, nous avons abordé l'installation de Go suivant trois méthodes différentes comprenant celle par le code source, via le paquet standard et des outils tierce. Ensuite, nous avons vu comment configurer l'environnement de développement Go, couvrant principalement la configuration de votre `$GOPATH`. Après cela, nous avons présenté certaines méthodes de compilation et de déploiement de programmes de Go. Nous avons ensuite couvert des commandes Go, incluant la compilation, l'installation, le formatage et les tests. Enfin, il existe de nombreux outils puissants pour développer des programmes Go tels que LiteIDE, Sublime Text, Vim, Emacs, Eclipse, IntelliJ IDEA, etc. Vous pouvez choisir celui de votre préférence pour découvrir le monde du langage Go.

Navigation

- [Table des matières](#)
- Section précédente: [Outils de développement Go](#)
- Chapitre suivant: [Bases du langage Go](#)

Appendice A Références

Ce livre est un condensé de mon expérience avec Go, certains contenus proviennent de blogs ou sites d'autres gophers. Vous pouvez les remercier!

1. [golang blog](#)
2. [Russ Cox blog](#)
3. [go book](#)
4. [golangtutorials](#)
5. [de](#)
6. [Go Programming Language](#)
7. [Network programming with Go](#)

8. [setup-the-rails-application-for-internationalization](#)
9. [The Cross-Site Scripting \(XSS\) FAQ](#)