**eBook Gratuit**

# APPRENEZ grails

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#grails

# Table des matières

# À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: grails

It is an unofficial and free grails ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official grails.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Chapitre 1: Commencer avec les grails

## Remarques

Grails est un framework de développement d'applications (RAD) très performant pour la **plate-forme Java** visant à multiplier **la productivité des développeurs** grâce à une Convention-over-Configuration, des valeurs par défaut et des API fiables. Il s'intègre parfaitement au **langage** JVM et au **langage Groovy,** ce qui vous permet d'être **immédiatement productif** tout en fournissant des fonctionnalités puissantes, notamment le mappage objet-relationnel (ORM), les langages spécifiques au domaine (DSL), l'exécution et la méta-programmation de compilation. la programmation.

La page d'accueil de Grails se trouve ici

## Versions

| Version | Remarques | Date de sortie |
|---|---|---|
| 2.5.5 | dernière version en ligne héritée 2.x | 2016-10-27 |
| 3.2.2 | au plus tard le 30 octobre 2016 | 2016-06-24 |
| 3.2.3 | dernière version en 3.x | 2016-11-10 |

## Examples

### Installation du Grails

Remarque: GRAILS nécessite l'installation d'un JDK Java (un environnement d'exécution JRE n'est pas suffisant) sur votre système avant de configurer Grails. S'il vous plaît se référer à, comment installer JDK . Au moment d'écrire ces lignes, il est recommandé d'installer le dernier JDK.

---

**Pour Mac OSX, Linux, Cygwin, Solaris et FreeBSD:**

Le moyen le plus simple de gérer les versions de Grails est d'utiliser sdkman . Si `sdkman` est installé, vous pouvez installer n'importe quelle version de Grails en utilisant

```
sdk install grails [version]
```

Cela prendra soin de toutes les étapes pour bien faire les choses. Si vous ignorez la `version` , la dernière version de grails sera installée. Pour plus d'informations sur l'utilisation de `sdkman` , reportez-vous à la page d'utilisation de sdkman .

---

**Pour Linux:**

```
GRAILS_HOME=$HOME/bin/grails/current
# abbreviating it using "..." for brevity
PATH=$GRAILS_HOME/bin:$JAVA_HOME/bin: ... :$PATH
```

---

**Pour les fenêtres:**

1. Téléchargez un JDK Java d' Oracle et installez-le sur votre machine Windows. Prenez note du dossier d'installation.
2. Téléchargez une version de Grails manuellement à partir de la page Téléchargements .
3. Extrayez le fichier Grails où vous voulez.
4. **Important:** vous devez configurer 2 nouvelles variables d'environnement `JAVA_HOME` et `GRAILS_HOME` (pour Windows 10 sous * Panneau de configuration \ Système et sécurité \ System \ Paramètres système avancés \ Onglet Avancé \ Variables d'environnement) *, pointant vers les répertoires extraits, par exemple

   Nom: JAVA_HOME

   Valeur: C: \ Programmes \ Java \ jdk1.8.0_31

   Nom: GRAILS_HOME

   Valeur: c: \ grails \ grails-3.2.4

5. **Important:** Vous devez étendre la variable Windows `PATH` pour inclure à la fois JAVA_HOME et GRAILS_HOME. La variable de chemin se trouve également dans le panneau de contrôle (voir 4), par exemple, ajoutez ce qui suit à la fin:

   ; C: \ Programmes \ Java \ jdk1.8.0_31 \ bin; c: \ grails \ grails-3.2.4; c: \ grails \ grails-3.2.4 \ bin

5. Pour vérifier que votre installation est correcte, ouvrez une invite de commandes et tapez `GRAILS -VERSION` . Vous devriez obtenir quelque chose comme:

```
| Grails Version: 3.2.4
| Groovy Version: 2.4.6
| JVM Version: 1.8.0_65
```

## Créer une application

Pour créer une application Grails, utilisez la commande `grails create-app` . La commande suivante crée une application Grails, nommée `myapp` dans le répertoire actuel:

```
grails create-app fancy-app
```

Le lancer, c'est aussi simple que de visiter le répertoire d'application nouvellement créé:

```
cd fancy-app
```

et alors

```
grails run-app
// in order to run the app on a different port, e.g. 8888, use this instead
grails run-app -port 8888
// in order to run the app with a secure communication
grails run-app -https
```

## Test d'une application

Les commandes create- * dans Grails créent automatiquement des tests unitaires ou d'intégration pour vous dans le répertoire src / test / groovy. Il vous appartient bien sûr de remplir ces tests avec une logique de test valide, informations qui peuvent être trouvées dans la section sur les tests unitaires et d'intégration.

Pour exécuter des tests, exécutez la commande test-app comme suit:

```
grails test-app
```

## Créer un modèle

Un modèle (voir: Modèle Model-View-Controller) dans Grails est représenté par une **classe de domaine** . Les classes de domaine peuvent définir à la fois la persistance et la présentation des informations dans les grails. Les classes de domaine peuvent également contenir des validations.

Pour gérer une flotte de voitures dans votre application Grails, vous pouvez définir une classe de domaine pour décrire, stocker et représenter différentes voitures de votre flotte.

Pour créer un stub pour une classe de domaine, exécutez la commande suivante dans votre dossier d'application:

```
grails create-domain-class org.fleetmanager.Car
```

Ensuite, ouvrez le fichier car.groovy généré et modifiez votre classe de domaine comme suit:

```
package org.fleetmanager

class Car {
   String      manufacturer
   String      model
   String      color
   Integer     year
   Date        acquisitionDate
   Boolean     isElectric
}
```

Enfin, générez un contrôleur pour votre domaine automobile et une vue en utilisant la commande Grails suivante:

```
grails generate-all org.fleetmanager.Car
```

Maintenant, vous pouvez exécuter vos applications, sélectionner le contrôleur de voiture et gérer votre flotte.

Lire Commencer avec les grails en ligne: https://riptutorial.com/fr/grails/topic/1435/commencer-avec-les-grails

# Chapitre 2: Classes de domaine en tant que ressources REST

## Introduction

La méthode la plus simple pour créer une API RESTful dans Grails consiste à exposer une classe de domaine en tant que ressource REST. Cela peut être fait en ajoutant la transformation grails.rest.Resource à n'importe quelle classe de domaine.

## Examples

### API REST simple avec grails

```
import grails.rest.*

@Resource(uri='/books')
class Book {

    String title

    static constraints = {
        title blank:false
    }
}
```

En ajoutant simplement la transformation de ressource et en spécifiant un URI, votre classe de domaine sera automatiquement disponible en tant que ressource REST aux formats XML ou JSON. La transformation enregistre automatiquement le mappage d'URL RESTful nécessaire et crée un contrôleur appelé BookController.

Vous pouvez l'essayer en ajoutant des données de test à BootStrap.groovy:

```
def init = { servletContext ->
    new Book(title:"The Stand").save()
    new Book(title:"The Shining").save()
}
```

Et puis en cliquant sur l'URL `http://localhost:8080/books/1` , la réponse sera rendue comme suit:

```
<?xml version="1.0" encoding="UTF-8"?>
<book id="1">
    <title>The Stand</title>
</book>
```

Si vous changez l'URL en `http://localhost:8080/books/1.json` vous obtiendrez une réponse JSON telle que:

```
{"id":1,"title":"The Stand"}
```

Si vous souhaitez modifier la valeur par défaut pour renvoyer JSON au lieu de XML, vous pouvez le faire en définissant l'attribut formats de la transformation Resource:

```
import grails.rest.*

@Resource(uri='/books', formats=['json', 'xml'])
class Book {
    ...
}
```

## Correspondance avec les ressources REST

Si vous préférez conserver la déclaration du mappage d'URL dans votre fichier UrlMappings.groovy, il suffit de supprimer l'attribut uri de la transformation Resource et d'ajouter la ligne suivante à `UrlMappings.groovy` :

```
"/books"(resources:"book")
```

L'extension de votre API pour inclure plus de points d'extrémité devient alors triviale:

```
"/books"(resources:"book") {
    "/publisher"(controller:"publisher", method:"GET")
}
```

L'exemple ci-dessus exposera l'URI `/books/1/publisher`.

## Ajouter HTTPS au serveur Grails

Les certificats SSL utilisent quelque chose appelé cryptographie à clé publique. Nous devons utiliser des Https au lieu de HTTP en raison de la sécurité des données entre les serveurs et de l'amélioration de la confiance des clients. Pour activer cette option dans grails, nous devons exécuter notre application différemment. La commande ci-dessous:

```
grails run-app -https
```

Lire Classes de domaine en tant que ressources REST en ligne:
https://riptutorial.com/fr/grails/topic/8944/classes-de-domaine-en-tant-que-ressources-rest

# Chapitre 3: Déploiement

## Examples

### Jar exécutable

L'un des moyens les plus simples de déployer Grails 3.x consiste à créer un fichier JAR exécutable incorporant un conteneur de servlets (Tomcat, Undertow, etc.) avec l'application.

Modifier `build.gradle` :

```
// Remove or comment out the war plugin:
// apply plugin:"war"

// Enable the executable jar:
springBoot {
    executable = true
}

// Optional: Customize the jar properties:
//  https://docs.gradle.org/current/dsl/org.gradle.api.tasks.bundling.Jar.html
jar {
    archiveName('myapp.jar')
}
```

Construire en utilisant `./gradlew assemble`

Le jar résultant est maintenant une application entièrement exécutable que vous pouvez démarrer:

```
$ head build/libs/myapp.jar
#!/bin/bash
#
#     .   ____          _            __ _ _
#    /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
#   ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
#    \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
#     '  |____| .__|_| |_|_| |_\__, | / / / /
#    =========|_|==============|___/=/_/_/_/
#    :: Spring Boot Startup Script ::
#
```

Vous pouvez le démarrer comme vous le feriez normalement pour toute application de ligne de commande:

```
$ ./build/libs/myapp.jar
Grails application running at http://localhost:8080 in environment: production
```

Il se comporte aussi comme un service init:

```
$ ln -s /opt/myapp/myapp.jar /etc/init.d/myapp
$ service myapp [start|stop|status|restart]
```

La documentation détaillée se trouve dans les documents print-boot: http://docs.spring.io/spring-boot/docs/current/reference/html/deployment-install.html

## Création de fichier de guerre

Lorsque nous écrivons une application Web dans Grails, nous avons besoin d'un fichier "war" qui doit être placé dans le conteneur de servlet (Tomcat, etc.) pour déployer l'application.

D'abord dans le répertoire du projet:

```
cd to_project_directory
```

1. War création de fichier à partir de l'invite de commande:

```
grails war
```

2.Il est toujours recommandé de nettoyer votre application avant la création de la guerre

Application de nettoyage à partir de l'invite de commande:

```
grails clean
```

En combinant les deux étapes ci-dessus, on aboutira à

```
grails clean && grails war
```

Vous pouvez également spécifier l'environnement dans lequel vous souhaitez créer le fichier war.

```
grails [environment] war
```

Où `[environment]` peut prendre les valeurs suivantes: `dev` , `prod` ou `test` par exemple.

Contrairement à d'autres commandes, la commande war s'exécute par défaut dans l'environnement de production au lieu du développement.

Lire Déploiement en ligne: https://riptutorial.com/fr/grails/topic/3701/deploiement

# Chapitre 4: SPG

## Paramètres

| Variables et étendues | Détails |
|---|---|
| application | Instance ServletContext |
| applicationContext | Instance Spring ApplicationContext |
| flash | L'objet flash |
| grailsApplication | Instance GrailsApplication |
| en dehors | rédacteur de réponse pour l'écriture dans le flux de sortie |
| params | objet params pour récupérer les paramètres de la requête |
| demande | Instance HttpServletRequest |
| réponse | Instance HttpServletResponse |
| session | Instance HttpSession |
| webRequest | Instance GrailsWebRequest |

## Examples

### Les bases

GSP prend en charge l'utilisation de blocs de scriptlet `<% %>` %% `<% %>` pour incorporer du code Groovy (cela est déconseillé):

```
<html>
   <body>
     <% out << "Hello GSP!" %>
   </body>
</html>
```

Vous pouvez également utiliser la syntaxe `<%= %>` pour générer des valeurs, comme dans JSP:

```
<html>
   <body>
     <%="Hello GSP!" %>
   </body>
</html>
```

GSP prend également en charge les commentaires côté serveur de style JSP:

```
<html>
   <body>
     <%-- This is my comment --%>
     <%="Hello GSP!" %>
   </body>
</html>
```

## Expressions

Dans GSP, la syntaxe `<%= %>` est rarement utilisée en raison de la prise en charge des **expressions GSP** .

Une expression GSP est similaire à une expression **JSP EL** ou à une **Groovy GString** et prend la forme `${expr}` :

```
<html>
  <body>
    Hello ${params.name}
  </body>
</html>
```

Cependant, contrairement à JSP EL, vous pouvez avoir une expression Groovy dans le bloc `${..}` .

Toute expression Groovy peut être interpolée dans tous les littéraux de chaîne, à l'exception des chaînes simples et triples. L'interpolation consiste à remplacer un espace réservé dans la chaîne par sa valeur lors de l'évaluation de la chaîne. Les expressions d'espace réservé sont entourées par $ {} ou préfixées par $ pour les expressions en pointillés. La valeur de l'expression à l'intérieur de l'espace réservé est évaluée par sa représentation sous forme de chaîne lorsque GString est transmise à une méthode prenant un argument String en appelant toString () sur cette expression.

## Tags GSP

Il existe une variété de balises gsp disponibles qui peuvent être utilisées pour créer des formulaires, des champs de texte, des boutons radio, des cases à cocher, sinon, pour chacun, etc.

**<g: si>**

```
<g:if test="${session.role == 'admin'}">
   <%-- show administrative functions --%>
</g:if>
<g:else>
   <%-- show basic functions --%>
</g:else>
```

**<g: chacun>**

```
<g:each in="${[1,2,3]}" var="num">
  <p>Number ${num}</p>
```

```
</g:each>
```

### forme

```
<g:form name="myForm" url="[controller:'book',action:'list']">...</g:form>
```

### champ de texte

```
<g:textField name="myField" value="${myValue}" />
```

### radio

```
<g:radio name="myGroup" value="1"/>
```

Suivez ce lien pour plus d'informations - http://docs.grails.org/latest/guide/theWebLayer.html#tags

Lire SPG en ligne: https://riptutorial.com/fr/grails/topic/4531/spg

# Crédits

| S. No | Chapitres | Contributeurs |
|---|---|---|
| 1 | Commencer avec les grails | Abdelsalam A. Shahlol, Adeel Ansari, Andrea Zago, bronoman, Burt Beckwith, Community, Farshid Zaker, Graeme Rocher, Jason Bourne, Jesús Iglesias, rahul, saw303 |
| 2 | Classes de domaine en tant que ressources REST | Jason Bourne |
| 3 | Déploiement | erichelgeson, NachoB, Prakash Thete, saw303 |
| 4 | SPG | Adeel Ansari, Anshul, audittxl |