EBook Gratuito

APPRENDIMENTO grails

Free unaffiliated eBook created from
Stack Overflow contributors.

#grails

Sommario

Di

Capitolo 1: Iniziare con i graal

Osservazioni

Grails è un potente framework per lo sviluppo rapido di applicazioni (RAD), per la **piattaforma Java che** mira a moltiplicare **la produttività degli sviluppatori** grazie a una convenzione di configurazione, impostazioni predefinite sensibili e API approvate. Si integra perfettamente con la JVM e il **linguaggio Groovy,** consentendo di essere **immediatamente produttivi** offrendo al contempo potenti funzionalità, tra cui ORM (Object-Relational-Mapping) integrato, DSL (Domain-Specific Languages), metadati del runtime e della compilazione in tempo e asincroni programmazione.

La homepage di Grails si trova qui

Versioni

Versione	Osservazioni	Data di rilascio
2.5.5	ultima versione in linea legacy 2.x.	2016/10/27
3.2.2	ultimo dal 30 ottobre 2016	2016/06/24
3.2.3	ultima versione in 3.x	2016/11/10

Examples

Installazione di Grails

Nota: GRAILS richiede l'installazione di un JDK Java (un ambiente di runtime JRE non è sufficiente) sul sistema, prima di configurare Grails. Si prega di fare riferimento a come installare JDK . Al momento della stesura di questo manuale, si consiglia di installare l'ultimo JDK.

Per Mac OSX, Linux, Cygwin, Solaris e FreeBSD:

Il modo più semplice per gestire le versioni di Grails è usare sdkman . Se `sdkman` è installato, puoi quindi installare qualsiasi versione di Grails usando

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

Questo si prenderà cura di tutti i passaggi per farlo bene. Se salti la `version` , verrà installata l'ultima versione di Grails. Per ulteriori informazioni sull'uso di `sdkman` , consultare la pagina di utilizzo di sdkman .

Per Linux:

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

Per Windows:

1. Scarica un JDK Java da Oracle e installa sul tuo computer Windows. Prendi nota della cartella di installazione.
2. Scarica manualmente una versione di Grails dalla pagina Download .
3. Estrai il file Grails, ovunque tu voglia.
4. **Importante:** è necessario impostare 2 nuove variabili di ambiente `JAVA_HOME` e `GRAILS_HOME` (per Windows 10 trovato in * Pannello di controllo \ Sistema e sicurezza \ Sistema \ Impostazioni di sistema avanzate \ scheda Avanzate \ Variabili d'ambiente) *, che punta alle directory estratte, ad es.

   Nome: JAVA_HOME

   Valore: C: \ Programmi \ Java \ jdk1.8.0_31

   Nome: GRAILS_HOME

   Valore: c: \ grails \ grails-3.2.4

5. **Importante:** è necessario estendere la variabile `PATH` Windows per includere sia JAVA_HOME che GRAILS_HOME. La variabile percorso si trova anche nel pannello di controllo (vedere 4), ad esempio aggiungere alla fine:

   ; C: \ Programmi \ Java \ jdk1.8.0_31 \ bin; c: \ graal \ graal-3.2.4; c: \ \ Grails Grails-3.2.4 \ bin

5. Per verificare che l'installazione sia corretta, apri un prompt dei comandi e digita `GRAILS -VERSION` . Dovresti ottenere qualcosa del tipo:

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

Creare un'applicazione

Per creare un'applicazione Grails, usa il comando `grails create-app` . Il seguente comando crea un'applicazione Grails, denominata `myapp` nella directory corrente:

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

Eseguirlo, è semplice come visitare la directory dell'applicazione appena creata:

```
cd fancy-app
```

e poi

```
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
```

Testare un'applicazione

I comandi create- * in Grails creano automaticamente test di unità o di integrazione all'interno della directory src / test / groovy. Spetta naturalmente a voi compilare questi test con una logica di test valida, le cui informazioni possono essere trovate nella sezione sui test di unità e di integrazione.

Per eseguire i test, eseguire il comando test-app come segue:

```
grails test-app
```

Creare un modello

Un modello (vedi: modello Model-View-Controller) in Grails è rappresentato da una cosiddetta **Domain Class** . Le classi di dominio possono definire sia la persistenza che la presentazione di informazioni nei graal. Le classi di dominio possono contenere anche convalide.

Per gestire una flotta di auto nella tua applicazione Grails puoi definire una classe dominio per descrivere, memorizzare e rappresentare varie auto nella tua flotta.

Per creare uno stub per una classe di dominio, eseguire il seguente comando all'interno della cartella dell'applicazione:

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

Quindi, apri il file car.groovy generato e modifica la tua classe di dominio come segue:

```
package org.fleetmanager

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

Infine, genera un controller per il tuo dominio auto e una vista usando il seguente comando Grails:

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

Ora puoi eseguire le tue applicazioni, selezionare il controller dell'auto e gestire la tua flotta.

Leggi Iniziare con i graal online: https://riptutorial.com/it/grails/topic/1435/iniziare-con-i-graal

Capitolo 2: Classi di dominio come risorse REST

introduzione

Il modo più semplice per creare un'API RESTful in Grails è esporre una classe di dominio come risorsa REST. Questo può essere fatto aggiungendo la trasformazione grails.rest.Resource a qualsiasi classe di dominio.

Examples

Semplice API REST con graal

```
import grails.rest.*

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

    String title

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

Semplicemente aggiungendo la trasformazione delle risorse e specificando un URI, la classe del dominio sarà automaticamente disponibile come risorsa REST nei formati XML o JSON. La trasformazione registrerà automaticamente la mappatura URL RESTful necessaria e creerà un controller denominato BookController.

Puoi provarlo aggiungendo alcuni dati di test a BootStrap.groovy:

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

E poi colpendo l'URL `http://localhost:8080/books/1` , che renderà la risposta come:

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

Se cambi l'URL in `http://localhost:8080/books/1.json` otterrai una risposta JSON come:

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

Se si desidera modificare l'impostazione predefinita per restituire JSON anziché XML, è possibile farlo impostando l'attributo formats della trasformazione Resource:

```
import grails.rest.*

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

Mappatura alle risorse REST

Se si preferisce mantenere la dichiarazione della mappatura URL nel file UrlMappings.groovy, rimuovere semplicemente l'attributo uri della trasformazione Risorsa e aggiungere la seguente riga a `UrlMappings.groovy` sarà sufficiente:

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

Estendere la tua API per includere più endpoint diventa quindi banale:

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

L'esempio sopra esporrà l'URI `/books/1/publisher`.

Aggiungi HTTPS al server Grails

I certificati SSL utilizzano qualcosa chiamato crittografia a chiave pubblica. È necessario utilizzare Https anziché Http per mantenere i dati protetti tra i server e migliorare la fiducia dei clienti. Per abilitare questa opzione in Grails, dobbiamo eseguire la nostra app in modo diverso. Il comando qui sotto:

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

Leggi Classi di dominio come risorse REST online: https://riptutorial.com/it/grails/topic/8944/classi-di-dominio-come-risorse-rest

Capitolo 3: Distribuzione

Examples

Jar eseguibile

Uno dei modi più semplici per distribuire Grails 3.x è creare un file jar eseguibile che incorpori un contenitore servlet (Tomcat, Undertow, ecc.) Con l'applicazione.

Modifica `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')
}
```

Costruisci usando `./gradlew assemble`

Il jar risultante ora è un'app completamente eseguibile che puoi avviare:

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

Puoi avviarlo come faresti normalmente per qualsiasi app della riga di comando:

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

Si comporta anche come un servizio di init:

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

La documentazione dettagliata è contenuta nella documentazione di avvio della primavera: http://docs.spring.io/spring-boot/docs/current/reference/html/deployment-install.html

Creazione di file di guerra

Quando scriviamo un'applicazione Web in Grails, per distribuire l'applicazione abbiamo bisogno di un file "war" che deve essere inserito nel contenitore servlet (Tomcat, ecc.).

Per prima cosa vai alla directory del progetto:

```
cd to_project_directory
```

1. Creazione di file di guerra dal prompt dei comandi:

```
grails war
```

2.Si consiglia sempre di pulire la tua applicazione prima della creazione di guerra

Pulizia dell'applicazione dal prompt dei comandi:

```
grails clean
```

La combinazione dei due passaggi precedenti in uno risulterà in

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

Inoltre è possibile specificare l'ambiente in cui si desidera creare il file di guerra.

```
grails [environment] war
```

Dove `[environment]` può assumere i seguenti valori: `dev` , `prod` o `test` per esempio.

A differenza di altri comandi, il comando war viene eseguito nell'ambiente di produzione per impostazione predefinita anziché per lo sviluppo.

Leggi Distribuzione online: https://riptutorial.com/it/grails/topic/3701/distribuzione

Capitolo 4: GSP

Parametri

Variabili e ambiti	Dettagli
applicazione	Istanza ServletContext
applicationContext	Spring istanza ApplicationContext
veloce	L'oggetto flash
grailsApplication	Istanza GrailsApplication
su	autore della risposta per scrivere nel flusso di output
params	oggetto params per il recupero dei parametri della richiesta
richiesta	Istanza HttpServletRequest
risposta	HttpServletResponse instance
sessione	Istanza di HttpSession
webRequest	Istanza GrailsWebRequest

Examples

Nozioni di base

GSP supporta l'utilizzo di `<% %>` %% `<% %>` blocchi di scriptlet per incorporare il codice Groovy (questo è sconsigliato):

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

Puoi anche utilizzare la sintassi `<%= %>` per i valori di output, come in JSP:

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

GSP supporta anche i commenti lato server in stile JSP:

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

espressioni

In GSP la sintassi `<%= %>` viene utilizzata raramente a causa del supporto per le **espressioni GSP** .

Un'espressione GSP è simile a un'espressione **JSP EL** oa **Groovy GString** e assume il formato `${expr}` :

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

Tuttavia, a differenza di JSP EL puoi avere qualsiasi espressione di Groovy all'interno del blocco `${..}` .

Qualsiasi espressione di Groovy può essere interpolata in tutte le stringhe letterali, a parte le stringhe singole e triple quotate. L'interpolazione è l'atto di sostituire un segnaposto nella stringa con il suo valore dopo la valutazione della stringa. Le espressioni segnaposto sono circondate da $ {} o precedute da $ per le espressioni tratteggiate. Il valore dell'espressione all'interno del segnaposto viene valutato alla sua rappresentazione di stringa quando il GString viene passato a un metodo che accetta una stringa come argomento chiamando aString () su quell'espressione.

Tag GSP

Esistono vari tipi di tag gsp che possono essere utilizzati per creare moduli, campi di testo, pulsanti di opzione, caselle di controllo, se-else, per ogni ecc.

<g: se>

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

<g: ogni>

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

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

modulo

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

campo di testo

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

Radio

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

Segui questo link per maggiori informazioni - http://docs.grails.org/latest/guide/theWebLayer.html#tags

Leggi GSP online: https://riptutorial.com/it/grails/topic/4531/gsp

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con i graal	Abdelsalam A. Shahlol, Adeel Ansari, Andrea Zago, bronoman, Burt Beckwith, Community, Farshid Zaker, Graeme Rocher, Jason Bourne, Jesús Iglesias, rahul, saw303
2	Classi di dominio come risorse REST	Jason Bourne
3	Distribuzione	erichelgeson, NachoB, Prakash Thete, saw303
4	GSP	Adeel Ansari, Anshul, audittxl