Spring boot hello world example – Spring boot REST example

Spring boot est un sous-projet développé par les développeurs de Spring Framework - pour créer une application autonome de qualité production avec une configuration minimale possible. Les applications de démarrage Spring sont généralement regroupées sous forme de fichiers jar fat/uber et peuvent être déployées sur n'importe quelle plate-forme en tant que fichier jar simple. C'est pourquoi les applications de démarrage de spring sont un bon candidat pour créer des microservices en Java.

1. Créez un modèle de projet Spring Boot hello world

Pour créer un modèle pour l'application Spring Boot, je suggérerai d'utiliser http://start.spring.io/ . Ici, vous pouvez sélectionner toutes les dépendances que vous avez actuellement en tête et générer le projet.

J'ai sélectionné des dépendances telles que Jersey , Spring Web , Spring HATEOAS, Spring JPA et Spring Security, etc. Vous pouvez ajouter d'autres dépendances après avoir téléchargé et importé le projet ou à l'avenir lorsque des besoins se présentent.

`Generate Project` bouton va générer un `.zip` fichier. Téléchargez et extrayez le fichier dans votre espace de travail.

3. Configuration automatique du démarrage du spring

Avec Spring Boot, une bonne chose est que lorsque vous ajoutez une dépendance (par exemple *Spring security*), cela fait des hypothèses justes et configure automatiquement certaines valeurs par défaut pour vous. Vous pouvez donc commencer immédiatement.

Spring Boot utilise la convention sur la configuration en analysant les bibliothèques dépendantes disponibles dans le chemin de classe. Pour chaque `spring-boot-starter-*` dépendance dans le fichier POM, Spring Boot exécute une `AutoConfiguration` classe par défaut . `AutoConfiguration` les classes utilisent le `*AutoConfiguration` modèle lexical, où `*`représente la bibliothèque. Par exemple, la configuration automatique de la sécurité des spring-boots se fait via `SecurityAutoConfiguration`.

En même temps, si vous ne souhaitez utiliser la configuration automatique pour aucun projet, cela le rend très simple. Utilisez simplement `exclude = SecurityAutoConfiguration.class` comme ci-dessous.

```
@SpringBootApplication (exclude = SecurityAutoConfiguration.class)
public class SpringBootDemoApplication {
   public static void main(String[] args)
   {
      SpringApplication.run(SpringBootDemoApplication.class, args);
   }
}
```

Il est également possible de remplacer les valeurs de configuration par défaut en utilisant le `application.properties` fichier dans le `src/main/resources` dossier.

4. Annotations de démarrage de spring

Maintenant, regardez l' `@SpringBootApplication` annotation ce qu'elle fait réellement.

4.1. annotation @SpringBootApplication

`SpringBootApplication` est défini comme ci-dessous :

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = @Filter(type = FilterType.CUSTOM,
classes = TypeExcludeFilter.class))
public @interface SpringBootApplication
{
   //more code
}
```

Il ajoute 3 annotations importantes à des fins de configuration de l'application.

1. **@SpringBootConfiguration**

```
@Configuration
public @interface SpringBootConfiguration
{
   //more code
}
```

1. Cette annotation ajoute une `@Configuration` annotation à la classe qui marque la classe comme source de définitions de bean pour le contexte d'application.
2. **@EnableAutoConfiguration**
   Cela indique à Spring Boot de configurer automatiquement les définitions de bean importantes en fonction des dépendances ajoutées en `pom.xml` commençant à ajouter des beans en fonction des paramètres de chemin de classe, d'autres beans et de divers paramètres de propriété.
3. **@ComponentScan**
   Cette annotation indique à Spring Boot d'analyser le package de

base, de rechercher d'autres beans/composants et de les configurer également.

5. Comment vérifier les beans configurés automatiquement par Spring Boot

Si vous voulez savoir ce que tous les beans ont été automatiquement configurés dans votre application Spring boot hello world , utilisez ce code et exécutez-le.

```
import java.util.Arrays;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.boot.autoconfigure.security.SecurityAutoC
onfiguration;
import org.springframework.context.ApplicationContext;

@SpringBootApplication (exclude =
SecurityAutoConfiguration.class)
public class SpringBootDemoApplication {

   public static void main(String[] args)
   {
      ApplicationContext ctx =
SpringApplication.run(SpringBootDemoApplication.class, args);

        String[] beanNames = ctx.getBeanDefinitionNames();

        Arrays.sort(beanNames);

        for (String beanName : beanNames)
        {
            System.out.println(beanName);
```

```
        }
    }
}
```

Avec mon `pom.xml` fichier, il génère les noms de beans suivants ainsi que de nombreuses autres `springframework.boot.autoconfigure` dépendances.

```
simpleControllerHandlerAdapter
sortResolver
spring.datasource-org.springframework.boot.autoconfigure.jdbc.DataSource
Properties
spring.hateoas-org.springframework.boot.autoconfigure.hateoas.HateoasPro
perties
spring.http.encoding-org.springframework.boot.autoconfigure.web.HttpEnco
dingProperties
spring.http.multipart-org.springframework.boot.autoconfigure.web.Multipa
rtProperties
spring.info-org.springframework.boot.autoconfigure.info.ProjectInfoPrope
rties
spring.jackson-org.springframework.boot.autoconfigure.jackson.JacksonPro
perties
spring.jpa-org.springframework.boot.autoconfigure.orm.jpa.JpaProperties
spring.jta-org.springframework.boot.autoconfigure.transaction.jta.JtaPro
perties
spring.mvc-org.springframework.boot.autoconfigure.web.WebMvcProperties
spring.resources-org.springframework.boot.autoconfigure.web.ResourceProp
erties
springBootDemoApplication
standardJacksonObjectMapperBuilderCustomizer
stringHttpMessageConverter
tomcatEmbeddedServletContainerFactory
tomcatPoolDataSourceMetadataProvider
transactionAttributeSource
transactionInterceptor
transactionManager
transactionTemplate
viewControllerHandlerMapping
viewResolver
websocketContainerCustomizer
```

6. Exemple d'API REST de démarrage de spring

Il est maintenant temps d'intégrer n'importe quelle fonctionnalité dans l'application hello world. Vous pouvez ajouter des fonctionnalités selon vos besoins, j'ajoute une API REST .

6.1. Créer un contrôleur REST

Créez un package `com.howtodoinjava.demo.controller` et créez un contrôleur de repos à l'intérieur.

```
import java.util.ArrayList;
import java.util.List;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.howtodoinjava.demo.model.Employee;

@RestController
public class EmployeeController
{
   @RequestMapping("/")
    public List<Employee> getEmployees()
    {
      List<Employee> employeesList = new ArrayList<Employee>();
      employeesList.add(new
Employee(1,"lokesh","gupta","howtodoinjava@gmail.com"));
      return employeesList;
    }
}
```

6.2. Créer un modèle

Créer une classe de modèle `Employee`.

```
public class Employee {

   public Employee() {

   }
   public Employee(Integer id, String firstName, String lastName, String
email) {
      super();
      this.id = id;
      this.firstName = firstName;
      this.lastName = lastName;
      this.email = email;
   }

   private Integer id;
   private String firstName;
   private String lastName;
   private String email;

   //getters and setters

   @Override
   public String toString() {
      return "Employee [id=" + id + ", firstName=" + firstName
            + ", lastName=" + lastName + ", email=" + email + "]";
   }
}
```

7. Démo d'exemple de démarrage de spring hello world

Démarrez maintenant l'application en exécutant la `main()` méthode dans `SpringBootDemoApplication`. Il démarrera le serveur Tomcat intégré sur le port `8080`.

Comme nous avons configuré l'URL de l'API REST de démonstration sur l'URL racine, vous pouvez y accéder par lui- `http://localhost:8080/` même.

Vérifier l'API REST Spring Boot

Vous obtiendrez la réponse ci-dessous dans l'outil de test ou le navigateur.

[{"id":1,"firstName":"lokesh","lastName":"gupta","email":"howtodoinjava@gmail.com"}]

C'est tout pour cet exemple de hello world de repos de démarrage de spring avec un exemple d' API de repos simple .