Exemple de Spring Boot CrudRepository

Par Arvind Rai, 2 décembre 2017

Cette page présente l' `CrudRepository` exemple de Spring Boot . Spring Boot Data active la prise en charge du référentiel JPA par défaut. `CrudRepository` fournit une opération CRUD générique sur un référentiel pour un type spécifique. `CrudRepository` est une interface de données Spring et pour l'utiliser, nous devons créer notre interface en l'étendant `CrudRepository`. Spring fournit `CrudRepository` automatiquement la classe d'implémentation au moment de l'exécution. Il contient des méthodes telles que `save`, `findById`, `delete`, `count` etc. démarrage Spring détecte automatiquement notre référentiel si le package de cette interface référentiel est identique ou sous-ensemble de la classe annotée avec `@SpringBootApplication`.
Spring Boot fournit des configurations de base de données par défaut lorsqu'il analyse Spring Data JPA dans classpath. Spring Boot utilise **Spring-Boot-Starter-Data-JPA** démarreur pour configurer le ressort JPA. Pour la source de données, nous devons configurer les propriétés de la source de données en commençant par `spring.datasource.*` dans **application.properties** . Dans la version Spring Boot 2.0, la technologie de mise en commun des bases de données par défaut est passée de Tomcat Pool à HikariCP. Spring boot préfère HikariCP en première position, puis Tomcat pooling et Commons DBCP2 en fonction de la disponibilité. Ici, sur cette page, nous allons créer un service Web Spring Boot Rest pour le fonctionnement CRUD. L'opération CRUD sera effectuée par `CrudRepository`. Trouvez maintenant l'exemple complet étape par étape.

Contenu

Top Trends

- Angular 2 Radio Button and Checkbox Example
- Angular Select Option Set Selected Dynamically
- Angular 2 Http post() Example
- Angular 2/4 minlength and maxlength Validation Example
- Jackson @JsonIgnore, @JsonIgnoreProperties and @JsonIgnoreType

Popular Post

- Angular 2/4 Pattern Validation Example
- Angular HttpClient post
- Jackson @JsonProperty and @JsonAlias Example
- Angular FormArray setValue() and patchValue()
- Angular valueChanges and statusChanges

Featured Post

- Angular Select Option using Reactive Form
- Jackson Ignore Null and Empty Fields
- Angular 2 Decimal Pipe, Percent Pipe and Currency Pipe Example
- Spring Boot CrudRepository Example
- Angular Material Radio Button

1. Technologies utilisées

Trouvez les technologies utilisées dans notre exemple.
1. Java 9
2. Spring 5.0.5.RELEASE
3. Spring Boot 2.0.1.RELEASE
4. Maven 3.5.2
5. MySQL 5.5
6. Eclipse Oxygen

2. Fichier Maven utilisé dans Project

Trouvez l' `pom.xml` utilisé dans notre exemple.

/

```
<? xml version = "1.0" encoding = "UTF-8" ?> <project xmlns = "http://maven.apache.org/POM/4.0.0" xm

        <name> spring-demo </name> <description> Spring Boot Demo Project </description> <parent> <g

                <groupId> org.springframework.boot </groupId> <artifactId> spring-boot-starter-data

                spring-boot-devtools </artifactId> <optional> true </optional> </dependency> </depe
```

3. CrudRepository Interface

`CrudRepository` est une interface et étend l' `Repository` interface de données Spring . `CrudRepository` fournit une opération CRUD générique sur un référentiel pour un type spécifique. Il a des méthodes génériques pour le fonctionnement CRUD. Pour l'utiliser, `CrudRepository` nous devons créer notre interface et l'étendre `CrudRepository`. Nous n'avons pas besoin d'implémenter notre interface, son implémentation sera créée automatiquement lors de l'exécution. Trouvez quelques `CrudRepository` méthodes.

`<S extends T> S save(S entity)`: Enregistre et met à jour l'entité actuelle et renvoie cette entité.
`Optional<T> findById(ID primaryKey)`: Renvoie l'entité pour l'id donné.

/

`Iterable<T> findAll()`: Renvoie toutes les entités.

`long count()`: Renvoie le nombre.
`void delete(T entity)`: Supprime l'entité donnée.
`boolean existsById(ID primaryKey)`: Vérifie si l'entité pour l'ID donné existe ou non.

`CrudRepository`a une sous-interface comme`PagingAndSortingRepository` qui fournissent des méthodes supplémentaires pour récupérer des entités à l'aide de l'abstraction de pagination et de tri.

4. Étapes pour utiliser CrudRepository

Spring Boot active la prise en charge du référentiel JPA par défaut. Pour l'utiliser `CrudRepository`dans notre application de données Spring, nous devons créer une interface d'implémentation `CrudRepository`et tout est fait pour l'utiliser. Laissez-nous discuter étape par étape comment utiliser `CrudRepository`dans notre application de données Spring.

4.1 Créer une interface étendant CrudRepository

Dans notre exemple, nous effectuerons des opérations CRUD sur les données d'article pour la démonstration. Je vais donc créer une interface pour l'article s'étendant `CrudRepository`comme suit.

```
publicinterfaceArticleRepositoryextendsCrudRepository<Article,Long>{}
```

Nous n'avons pas besoin de créer sa classe d'implémentation. Spring créera automatiquement sa classe d'implémentation lors de l'exécution.

4.2 Détection automatique du référentiel JPA

Spring Boot peut détecter automatiquement notre référentiel si le package de cette interface est le même ou un sous-package de la classe annotée `@SpringBootApplication`et sinon, nous devons utiliser l' `@EnableJpaRepositories`annotation avec `@SpringBootApplication`. Comprenons par exemple.
Supposons que nous ayons une classe annotée avec `@SpringBootApplication`dans le package `com.concretepage`comme indiqué ci-dessous.

```
package com . page concrète ; ------ @SpringBootApplication classe publique MyApplication { ------ }
```

Maintenant, si nous avons un référentiel `ArticleRepository`et qu'il réside dans un package `com.concretepage`ou ses sous-packages tels que `com.concretepage.repository`Spring Boot détectera automatiquement notre référentiel et donc pas besoin d'utiliser d' `@EnableJpaRepositories`annotation.

Si nous choisissons un package pour notre référentiel qui n'est ni le même package ni le sous-package du package de la classe annotée `@SpringBootApplication`, Spring boot ne pourra pas détecter les classes de référentiel par défaut. Dans ce cas, nous devons utiliser l' `@EnableJpaRepositories`annotation avec `@SpringBootApplication`. En utilisant `@EnableJpaRepositories`nous allons configurer le nom du package dans lequel résident nos classes de référentiel. Supposons que le package de nos classes de référentiel soit `com.cp.repository`, nous utiliserons `@EnableJpaRepositories`comme suit.

```
package com . page concrète ; ------ @SpringBootApplication @EnableJpaRepositories ( "com.cp.reposit
```

Si nous voulons configurer des classes spécifiques, nous devons utiliser l' `basePackageClasses`attribut de l' `@EnableJpaRepositories`annotation. Supposons que nous ayons une classe `ArticleRepository`dans le package `com.cp.repository`, alors nous pouvons configurer le référentiel en utilisant `basePackageClasses`comme suit.

```
package com . page concrète ; ----- import com . cp . référentiel . ArticleRepository ; @SpringBootA
```

4.3 Instancier et utiliser CrudRepository

Pour instancier notre `ArticleRepository`extension `CrudRepository`, nous pouvons utiliser l'injection de dépendances.

```
publique classe ArticleService { @Autowired privée ArticleRepository articleRepository ; ------ }
```

Maintenant, nous sommes prêts à utiliser des méthodes de `CrudRepository`. Trouvez l'exemple de certaines de ses méthodes.

une. Créer et mettre à jour :

```
Article savedArticle = articleRepository.save(article);
```

b. Lire :

```
Article obj = articleRepository . findById ( articleId ). get (); Iterable < Article > articles = ar
```

c. Supprimer :

```
articleRepository . supprimer ( article );
```

5. Méthodes de référentiel personnalisées

`CrudRepository`fournit des méthodes pour le fonctionnement CRUD générique et si nous voulons ajouter des méthodes personnalisées dans notre interface qui a été étendue `CrudRepository`, nous pouvons ajouter de la manière suivante.

une. Nous pouvons commencer nos noms de méthode de requête avec `find...By`, `read...By`, `query...By`, `count...By`et `get...By`. Avant de `By`pouvoir ajouter une expression telle que `Distinct`. Après, `By`nous devons ajouter les noms de propriété de notre entité.
b. Pour obtenir des données sur la base de plusieurs propriétés, nous pouvons concaténer les noms de propriété en utilisant `And`et `Or`lors de la création des noms de méthode.
c. Si nous voulons utiliser un nom complètement personnalisé pour notre méthode, nous pouvons utiliser une `@Query`annotation pour écrire une requête.

Recherchez l'extrait de code qui utilise l'exemple de nom de méthode pour les scénarios ci-dessus.

```
public interface ArticleRepository extends CrudRepository<Article, Long>  {
    List<Article> findByTitle(String title);
    List<Article> findDistinctByCategory(String category);
    List<Article> findByTitleAndCategory(String title, String category);

    @Query("SELECT a FROM Article a WHERE a.title =: title et a.category =: category" ) Liste < Arti
```

La classe d'implémentation des méthodes ci-dessus sera créée automatiquement par Spring lors de l'exécution.

6. @Transactional avec CrudRepository

Les méthodes CRUD `CrudRepository`sont transactionnelles par défaut. Ils sont annotés avec une `@Transactional`annotation avec des paramètres par défaut dans la classe d'implémentation au moment de l'exécution. Pour la lecture, l' `readOnly`indicateur d' opération est défini sur `true`. Pour remplacer les paramètres transactionnels par défaut de toutes les `CrudRepository`méthodes, nous devons remplacer cette méthode dans notre interface et annoter en `@Transactional`utilisant les configurations requises. Trouvez l'exemple.

```
publique Interface ArticleRepository étend CrudRepository < article , longue > { @Override @Transact
```

Ici, nous avons configuré `timeout`8 secondes pour exécuter la requête sans `readOnly`indicateur de

Ici, nous avons configuré `timeout`6 secondes pour exécuter la requête sans `readOnly`indicateur de `findAll()`méthode.

7. Configurer les propriétés dans le fichier application.properties

La source de données, les propriétés JPA et la journalisation, etc. doivent être configurées dans un `application.properties`fichier situé dans le chemin de classe de l'application de démarrage Spring. Ces propriétés seront automatiquement lues par Spring Boot.

application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/concretepage
spring.datasource.username=root
spring.datasource.password=cp

spring.datasource.hikari.connection-timeout=20000
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=12
spring.datasource.hikari.idle-timeout=300000
spring.datasource.hikari.max-lifetime=1200000

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
printemps . jpa . propriétés . hiberner . id . new_generator_mappings = faux
ressort . jpa . propriétés . hiberner . format_sql = véritable
journalisation . niveau . org . hiberner . SQL = DEBUG
exploitation forestière . niveau . org . hiberner . type . descripteur . sql .
BasicBinder=TRACE
```

Dans la version Spring Boot 2.0, la technologie de mise en commun des bases de données par défaut est passée de Tomcat Pool à HikariCP. `spring-boot-starter-jdbc`et `spring-boot-starter-data-jpa`résoudre la dépendance HikariCP par défaut et la `spring.datasource.type`propriété a `HikariDataSource`comme valeur par défaut. Les propriétés de la source de données commençant par `spring.datasource.*`seront automatiquement lues par Spring boot JPA. Pour modifier les propriétés Hibernate, nous utiliserons le préfixe `spring.jpa.properties.*`avec le nom de propriété Hibernate. Sur la base d'une URL de source de données donnée, Spring Boot peut identifier automatiquement la classe de pilote de source de données. Nous n'avons donc pas besoin de configurer la classe de plongeur.

Recherchez les propriétés à configurer `JpaBaseConfiguration`et `HibernateJpaAutoConfiguration`dans `application.properties`.

spring.data.jpa.repositories.enabled : il active les référentiels JPA. La valeur par défaut est **true** .

spring.jpa.database : il cible la base de données sur laquelle fonctionner. Par défaut, la base de données intégrée est détectée automatiquement.

spring.jpa.database-platform : Il est utilisé pour fournir le nom de la base de données sur laquelle opérer. Par défaut, il est détecté automatiquement.

spring.jpa.generate-ddl : il est utilisé pour initialiser le schéma au démarrage. Par défaut, la valeur est **false** .

spring.jpa.hibernate.ddl-auto : C'est le mode DDL utilisé pour la base de données intégrée. La valeur par défaut est **create-drop** .

spring.jpa.hibernate.naming.implicit-strategy : Il s'agit du nom qualifié complet de la stratégie de nommage implicite Hibernate 5.

spring.jpa.hibernate.naming.physical-strategy : C'est le nom complet qualifié de la stratégie de nommage physique Hibernate 5.

spring.jpa.hibernate.use-new-id-generator-mappings : Il est utilisé pour Hibernate `IdentifierGenerator`pour AUTO, TABLE et SEQUENCE.

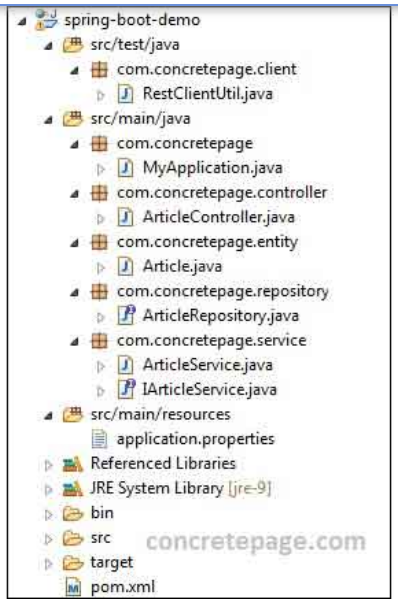
spring.jpa.open-in-view : la valeur par défaut est **true** . Il lie un JPA `EntityManager`au thread pour tout le traitement de la demande.

spring.jpa.properties. * : il définit des propriétés natives supplémentaires à définir sur le fournisseur JPA.

spring.jpa.show-sql : il permet la journalisation des instructions SQL. La valeur par défaut est **false** .

8. Spring Boot REST + Spring Boot Data CrudRepository + JPA + Hibernate + MySQL CRUD Example

Trouvez la structure du projet de notre projet de démonstration.



Trouvez la table de base de données MySQL utilisée dans notre exemple.

Table de base de données

```
CREATE DATABASE IF NOT EXISTS `concretepage`;
USE `concretepage`;

CREATE TABLE IF NOT EXISTS `articles` (
  `article_id` bigint(5) NOT NULL AUTO_INCREMENT,
  `title` varchar(200) NOT NULL,
  `category` varchar(100) NOT NULL,
  PRIMARY KEY (`article_id`)
) ENGINE=InnoDB;

INSERT INTO `articles` (`article_id`, `title` , ` category` ) VALEURS
         ( 1 , 'Java Concurrency' , 'Java' ), ( 2 , 'Spring Boot Getting Started' , 'Spring Boot' ),
```

Trouvez maintenant le code complet.

ArticleRepository.java

```
package com.concretepage.repository;
import java.util.List;
import org.springframework.data.repository.CrudRepository;
import com.concretepage.entity.Article;
public interface ArticleRepository extends CrudRepository<Article, Long>  {
    List<Article> findByTitle(String title);
    List<Article> findDistinctByCategory(String category);
    List<Article> findByTitleAndCategory(String title, String category);
}
```

Article.java

```
package com.concretepage.entity;
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="articles")
public class Article implements Serializable {
        private static final long serialVersionUID = 1L;
        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
```

```
        @Column(name="article_id")
        private long articleId;
        @Column(name="title")
        private String title;
        @Column(name="category")
        private String category;
        public long getArticleId() {
                return articleId;
        }
        public void setArticleId(long articleId) {
                this.articleId = articleId;
        }
        public String getTitle() {
                return title;
        }
        public void setTitle(String title) {
                this.title = title;
        }
        public String getCategory() {
                return category;
        }
        public void setCategory(String category) {
                this.category = category;
        }
}
```

IArticleService.java

```
package com.concretepage.service;
import java.util.List;
import com.concretepage.entity.Article;
public interface IArticleService {
     List<Article> getAllArticles();
     Article getArticleById(long articleId);
     boolean addArticle(Article article);
     void updateArticle(Article article);
     void deleteArticle(int articleId);
}
```

ArticleService.java

```
package com.concretepage.service;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.concretepage.entity.Article;
import com.concretepage.repository.ArticleRepository;
@Service
public class ArticleService implements IArticleService {
        @Autowired
        private ArticleRepository articleRepository;
        @Override
        public Article getArticleById(long articleId) {
                Article obj = articleRepository.findById(articleId).get();
                return obj;
        }
        @Override
        public List<Article> getAllArticles(){
                List<Article> list = new ArrayList<>();
                articleRepository.findAll().forEach(e -> list.add(e));
                return list;
        }
        @Override
        public synchronized boolean addArticle(Article article){
                List<Article> list = articleRepository.findByTitleAndCategory(article.getTitle(), ar
                if (list.size() > 0) {
                   return false;
```

```
                } else {

                        articleRepository.save(article);
                        return true;
                }
        }
        @Override
        public void updateArticle(Article article) {
                articleRepository.save(article);
        }
        @Override
        public void deleteArticle(int articleId) {
                articleRepository.delete(getArticleById(articleId));
        }
}
```

ArticleController.java

```
package com.concretepage.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.util.UriComponentsBuilder;
import com.concretepage.entity.Article;
import com.concretepage.service.IArticleService;
@Controller
@RequestMapping("user")
public class ArticleController {
        @Autowired
        private IArticleService articleService;
        @GetMapping("article/{id}")
        public ResponseEntity<Article> getArticleById(@PathVariable("id") Integer id) {
                Article article = articleService.getArticleById(id);
                return new ResponseEntity<Article>(article, HttpStatus.OK);
        }
        @GetMapping("articles")
        public ResponseEntity<List<Article>> getAllArticles() {
                List<Article> list = articleService.getAllArticles();
                return new ResponseEntity<List<Article>>(list, HttpStatus.OK);
        }
        @PostMapping("article")
        public ResponseEntity<Void> addArticle(@RequestBody Article article, UriComponentsBuilder bu
                boolean flag = articleService.addArticle(article);
                if (flag == false) {
                   return new ResponseEntity<Void>(HttpStatus.CONFLICT);
                }
                HttpHeaders headers = new HttpHeaders();
                headers.setLocation(builder.path("/article/{id}").buildAndExpand(article.getArticleI
                return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
        }
        @PutMapping("article")
        public ResponseEntity<Article> updateArticle(@RequestBody Article article) {
                articleService.updateArticle(article);
                return new ResponseEntity<Article>(article, HttpStatus.OK);
        }
        @DeleteMapping("article/{id}")
        public ResponseEntity<Void> deleteArticle(@PathVariable("id") Integer id) {
```

```
            articleService.deleteArticle(id);
            return new ResponseEntity<Void>(HttpStatus.NO_CONTENT);
        }
}
```

MyApplication.java

```
package com.concretepage;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class MyApplication {
    public static void main(String[] args) {
         SpringApplication.run(MyApplication.class, args);
    }
}
```

9. Client Code with RestTemplate

RestClientUtil.java

```
package com.concretepage.client;
import java.net.URI;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;
import com.concretepage.entity.Article;
public class RestClientUtil {
    public void getArticleByIdDemo() {
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        RestTemplate restTemplate = new RestTemplate();
        String url = "http://localhost:8080/user/article/{id}";
        HttpEntity<String> requestEntity = new HttpEntity<String>(headers);
        ResponseEntity<Article> responseEntity = restTemplate.exchange(url, HttpMethod.GET, requestE
        Article article = responseEntity.getBody();
        System.out.println("Id:"+article.getArticleId()+", Title:"+article.getTitle()
                +", Category:"+article.getCategory());
    }
    public void getAllArticlesDemo() {
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        RestTemplate restTemplate = new RestTemplate();
        String url = "http://localhost:8080/user/articles";
        HttpEntity<String> requestEntity = new HttpEntity<String>(headers);
        ResponseEntity<Article[]> responseEntity = restTemplate.exchange(url, HttpMethod.GET, reques
        Article[] articles = responseEntity.getBody();
        for(Article article : articles) {
              System.out.println("Id:"+article.getArticleId()+", Title:"+article.getTitle()
                      +", Category: "+article.getCategory());
        }
    }
    public void addArticleDemo() {
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        RestTemplate restTemplate = new RestTemplate();
        String url = "http://localhost:8080/user/article";
        Article objArticle = new Article();
        objArticle.setTitle("Spring REST Security using Hibernate");
        objArticle.setCategory("Spring");
        HttpEntity<Article> requestEntity = new HttpEntity<Article>(objArticle, headers);
        URI uri = restTemplate.postForLocation(url, requestEntity);
        System.out.println(uri.getPath());
    }
```

```
    public void updateArticleDemo() {
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        RestTemplate restTemplate = new RestTemplate();
        String url = "http://localhost:8080/user/article";
        Article objArticle = new Article();
        objArticle.setArticleId(1);
        objArticle.setTitle("Update:Java Concurrency");
        objArticle.setCategory("Java");
        HttpEntity<Article> requestEntity = new HttpEntity<Article>(objArticle, headers);
        restTemplate.put(url, requestEntity);
    }
    public void deleteArticleDemo() {
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        RestTemplate restTemplate = new RestTemplate();
        String url = "http://localhost:8080/user/article/{id}";
        HttpEntity<Article> requestEntity = new HttpEntity<Article>(headers);
        restTemplate.exchange(url, HttpMethod . DELETE , requestEntity , Void . classe , 4 ); } publ
        util . getAllArticlesDemo (); } }
```

10. Test Application

Pour tester l'application, créez d'abord une table dans MySQL comme indiqué dans l'exemple. Ensuite, nous pouvons exécuter le service Web REST de la manière suivante.
1. En utilisant Eclipse : Téléchargez le code source du projet en utilisant le lien de téléchargement donné à la fin de l'article. Importez le projet dans eclipse. À l'aide de l'invite de commande, accédez au dossier racine du projet et exécutez.

```
mvn clean eclipse : éclipse
```

puis actualisez le projet dans Eclipse. Exécutez la classe principale `MyApplication`en cliquant sur **Exécuter en tant que -> Application Java** . Le serveur Tomcat va démarrer.

2. À l'aide de la commande Maven : téléchargez le code source du projet. Accédez au dossier racine du projet à l'aide de l'invite de commande et exécutez la commande.

```
mvn spring - boot : run
```

Le serveur Tomcat va démarrer.

3. Utilisation du fichier exécutable JAR : à l'aide de l'invite de commande, accédez au dossier racine du projet et exécutez la commande.

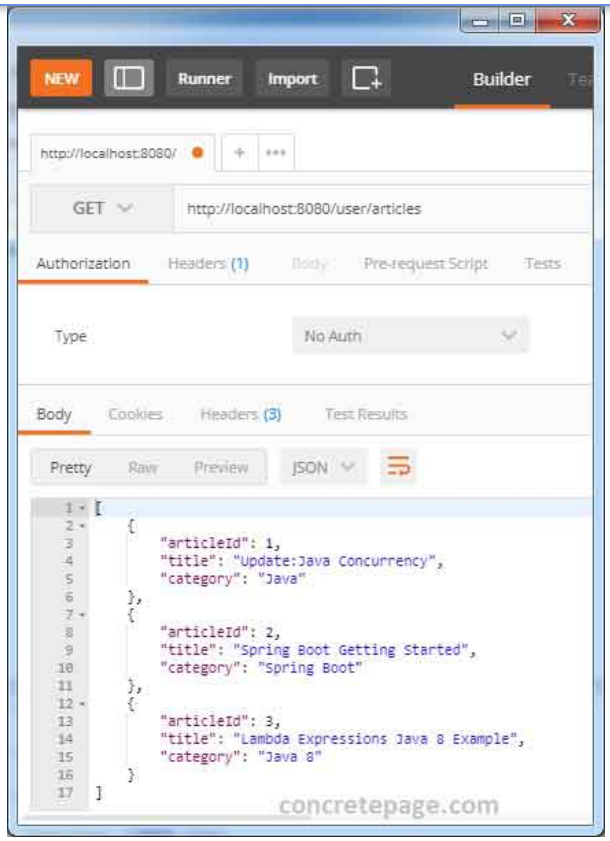
```
mvn clean package
```

Nous obtiendrons l'exécutable JAR **spring-boot-demo-0.0.1-SNAPSHOT.jar** dans le dossier cible. Exécutez ce fichier JAR en tant que

```
java - jar target / spring - boot - demo - 0.0 . 1 - INSTANTANÉ . pot
```

Le serveur Tomcat va démarrer.

Nous sommes maintenant prêts à tester l'application. Pour exécuter le client, accédez à la `RestClientUtil`classe dans eclipse et cliquez sur **Exécuter en tant que -> Application Java** .
Nous pouvons également tester l'application à l'aide de **Postman** . Trouvez l'écran d'impression.



11. Références

Accès aux données avec JPA

Spring Data Exemple CrudRepository

Spring Boot REST + JPA + Hibernate + MySQL Example

12. Download Source Code

spring-boot-crudrepository-example.zip

POSTÉ PAR

ARVIND RAI

Tutoriels populaires: Java 8 | Printemps 4 | Angulaire | Struts 2 | Android

TROUVEZ PLUS DE TUTORILES

SPRING BOOT | HIBERNATE | PRIMEFACES | RESTEASY | FREEMARKER

S'identifier

Laissez votre commentaire...

4 Commentaires Les plus récents

Vongsi Loryongpao 6 mois

Explication claire, merci

Répondre

Cristian Daniel Ortiz Cuellar environ un an

awesome articule best regards from venezuela.

Répondre

Prabath Manjula environ un an

Thanks Aravind.... I'm new to spring boot. Do we need to create two projects to run both client and and service..

Thanks in advance

Répondre

Animesh environ un an

How do we pass the whole request and response objects using ResponseEntity and retrieve their values using spring data jpa and insert response values into dB.
Please provide the example if possible.

Répondre

About Us

We are a group of software developers.
We enjoy learning and sharing technologies.
To improve the site's content,
your valuable suggestions
are most welcome. *Thanks*
Email : **concretepage@gmail.com**

Mobile Apps

SCJP Quiz

ConcretePage.com