在 Spring Boot Web 应用程序中使用 JdbcTemplate

韩博孙

2018 年 12 月 3 日 麻省理工学院

在本教程中，我将介绍 Spring JdbcTemplate 在 Spring Boot Web 应用程序中的使用。本教程将展示如何为 JdbcTemplate 创建必要的配置。以及它如何用于数据插入和检索。

下载源 - 8.2 KB

介绍

这将是我 2018 年的最后一个教程。在本教程中，我将讨论一些有趣的事情。数据访问是我最喜欢的主题之一。我曾经讨厌使用数据库，因为我不太擅长使用它们。随着经验的积累，我对数据库的厌恶越来越少。多年来，我从事过 ADO.NET、实体框架、Hibernate、SSRS 报告以及其他一些与数据库相关的工作。我学到的是，即使我讨厌这些东西，也无法逃脱。不妨面对它并学习它。

过去我没有机会使用的 Spring 技术之一是`JdbcTemplate`. 这是 Spring 对 JDBC 的方式。在当今世界，我们有实体框架和用于 ORM 的 Hibernate，为什么 JDBC 很重要？有两个主要原因：

- 使用 JDBC，您可以直接使用查询本身并以任何您想要的方式对其进行优化。使用 ORM，您不会得到这种奢侈。
- 另一个原因是支持遗留代码。有时，您会继承过去所做的事情。并且需要升级，而您是被指定执行此操作的人。所以很多代码都是用JDBC完成的，也许用Spring JDBC能帮上忙？谁知道。

对我来说，JDBC 只是一种非常简单直接的将数据传入和传出数据库的方法。而且我知道它可能比 ORM 快一点。有时，ORM 框架会使编码变得非常笨拙。但是，JDBC 也有它的笨拙之处，一旦我将数据取出，就必须将它们映射到实体对象。

本教程将讨论如何为 设置 Spring DAO `JdbcTemplate`，以及插入数据和检索数据的基础知识。

设置数据库

为了让这个示例应用程序正常工作，我们必须使用一个数据库用户和一个表来设置一个 MySQL 数据库。我准备了两个 SQL 脚本：

- 将创建数据库用户和数据库的一个脚本
- 另一个脚本将在新创建的数据库中创建表

要创建数据库和用户，脚本如下：

SQL 复制代码

```sql
-- user.sql
CREATE DATABASE cardb;
CREATE USER 'cardbuser'@'localhost' IDENTIFIED BY '123test321';
```

```
GRANT ALL PRIVILEGES ON cardb.* TO 'cardbuser'@'localhost';
FLUSH PRIVILEGES;
```

要为示例应用程序创建表，请使用以下脚本：

SQL 复制代码

```
-- tables.sql

use cardb;

DROP TABLE IF EXISTS carinfo;

CREATE TABLE carinfo (
   id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
   yearofmanufacture INT NOT NULL,
   model VARCHAR(64) NOT NULL,
   make VARCHAR(64) NOT NULL,
   suggestedretailprice FLOAT NOT NULL,
   fullprice FLOAT NOT NULL,
   rebateamount FLOAT NOT NULL,
   createdate DATETIME NOT NULL,
   updatedate DATETIME NOT NULL
);
```

现在，让我们看一下*pom.xml*文件。

Maven POM 文件

Maven POM XML 非常简单，它看起来像这样：

XML 缩小▲ 复制代码

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.springframework</groupId>
    <artifactId>hanbo-boot-rest</artifactId>
    <version>1.0.1</version>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.5.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
```

```
            <version>8.0.11</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

总结一下这个 POM 文件中的内容，该项目继承自`spring-boot-starter-parent`. 所需的依赖项是：

- `spring-boot-starter-web`：这是创建基于 Web 的应用程序所必需的。它支持 MVC 和 RESTFul 应用程序。
- `spring-boot-starter-jdbc`：这是使用 Spring DAO JDBC 功能所必需的。
- `mysql-connector-java`: 这是提供特定于 MySQL 数据库的 jdbc 驱动程序所需要的。

当使用 Maven 进行构建时，它将创建一个 jar 文件，该文件可以仅使用 Java 命令运行。我会告诉你最后如何。

主要入口

正如我在第一篇 Spring Boot 教程中所描述的，基于 Spring Boot 的 Web 应用程序不需要应用程序容器来承载它。所以打包成jar文件，就可以作为普通的Java程序运行了。这就是此示例应用程序有一个主要条目的原因。此主要条目的代码如下所示：

爪哇 复制代码

```
package org.hanbo.boot.rest;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

使用 annotation 将该类标记为 Spring 启动应用程序`@SpringBootApplication`。并且主入口方法会将类传递给`SpringApplication.run()`. Spring Boot 的应用程序运行器将创建 IoC 容器，并扫描所有包和子包以查找可以添加到 IoC 容器中的任何类。因为我们添加了 web starter 依赖项，应用程序将作为 web 应用程序启动。Spring Boot 会自动处理配置。

REST API 控制器

接下来，我将向您展示 REST API 的控制器类，这些 API 用于演示`JdbcTemplate`. 这是完整的源代码：

爪哇 缩小▲ 复制代码

```
package org.hanbo.boot.rest.controllers;

import java.util.ArrayList;
import java.util.List;
```

```
import org.hanbo.boot.rest.models.CarModel;
import org.hanbo.boot.rest.models.GenericResponse;
import org.hanbo.boot.rest.repository.CarRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class SampleController
{
   @Autowired
   private CarRepository carRepo;

   @RequestMapping(value="/public/addCar", method = RequestMethod.POST)
   public ResponseEntity<GenericResponse> addCar(
      @RequestBody
      CarModel carToAdd)
   {
      GenericResponse retMsg = new GenericResponse();
      if (carToAdd != null)
      {
         try
         {
            carRepo.addCar(carToAdd);

            retMsg.setSuccess(true);
            retMsg.setStatusMsg("Operation is successful.");
         }
         catch (Exception ex)
         {
            ex.printStackTrace();
            retMsg.setSuccess(false);
            retMsg.setStatusMsg("Exception occurred.");
         }
      }
      else
      {
         retMsg.setSuccess(false);
         retMsg.setStatusMsg("No valid car model object to be added");
      }

      ResponseEntity<GenericResponse> retVal;
      retVal = ResponseEntity.ok(retMsg);
      return retVal;
   }

   @RequestMapping(value="/public/getCars", method = RequestMethod.GET)
   public ResponseEntity<List<CarModel>> getCars(
      @RequestParam("make")
      String make,
      @RequestParam("startYear")
      int startYear,
      @RequestParam("endYear")
      int endYear)
   {
      List<CarModel> foundCars
         = carRepo.findCar(make, startYear, endYear);

      if (foundCars == null) {
         foundCars = new ArrayList<CarModel>();
      }

      ResponseEntity<List<CarModel>> retVal;
      retVal = ResponseEntity.ok(foundCars);
      return retVal;
```

```
    }
}
```

我特意将其作为 REST API 控制器，因为演示我想要展示的功能相对简单。该类被注释，`@RestController`以便 Spring Boot 将识别该类将充当 RESTFul API 控制器。

在类内部，有两种方法。并且两者都用于处理用户的 HTTP 请求。它们基本上是动作方法。第一个用于将汽车模型添加到数据库中。另一种是查询符合三个不同搜索条件的汽车型号列表。两种方法都使用相同的数据访问存储库，该存储库使用`JdbcTemplate`对象。

简单地说，要向数据库表中添加一个新的汽车模型，方法如下：

爪哇 复制代码

```
@Autowired
private CarRepository carRepo;

...

carRepo.addCar(carToAdd);
```

要查询的车型，该方法使用的制造商（如`Honda`，`Mazda`，`Ford`，`Chevrolet`，等），年份范围（一一年开始和结束年），并返回车型符合条件的列表。所以这个代码是：

爪哇 复制代码

```
@RequestMapping(value="/public/getCars", method = RequestMethod.GET)
public ResponseEntity<List<CarModel>> getCars(
   @RequestParam("make")
   String make,
   @RequestParam("startYear")
   int startYear,
   @RequestParam("endYear")
   int endYear)
{
   ...
   List<CarModel> foundCars
      = carRepo.findCar(make, startYear, endYear);
   ...
}
```

为了设计这样的数据访问存储库，必须进行配置。什么配置？像：

- 数据库连接字符串
- 用于连接数据库的用户名和密码
- 交易经理
- 数据库jdbc驱动类

这将在下一节中介绍。

JDBC 连接配置

为了使用`JdbcTemplate`，必须先设置配置。这是执行此操作的类：

爪哇 缩小▲ 复制代码

```
package org.hanbo.boot.rest.config;

import javax.sql.DataSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
```

```
@Configuration
public class JdbcConfiguration
{
   @Bean
   public DataSource dataSource()
   {
      DriverManagerDataSource dataSource = new DriverManagerDataSource();

      dataSource.setDriverClassName("com.mysql.jdbc.Driver");
      dataSource.setUrl("jdbc:mysql://localhost:3306/cardb");
      dataSource.setUsername("cardbuser");
      dataSource.setPassword("123test321");

      return dataSource;
   }

   @Bean
   public NamedParameterJdbcTemplate namedParameterJdbcTemplate()
   {
      NamedParameterJdbcTemplate retBean
          = new NamedParameterJdbcTemplate(dataSource());
       return retBean;
   }

   @Bean
   public DataSourceTransactionManager txnManager()
   {
      DataSourceTransactionManager txnManager
         = new DataSourceTransactionManager(dataSource());
      return txnManager;
   }
}
```

让我们一次一节地查看这门课。首先是用`@Configuration`. 这告诉 Spring Boot 将类视为配置，并在启动期间对其进行引导。

复制代码

```
@Configuration
public class JdbcConfiguration
{
...
}
```

接下来，我需要一个数据源。它是一个 Java 对象，按照惯例，它被称为`dataSource`. 该方法`dataSouce()`也用于配置MySQL连接：

爪哇 复制代码

```
@Bean
public DataSource dataSource()
{
   DriverManagerDataSource dataSource = new DriverManagerDataSource();

   dataSource.setDriverClassName("com.mysql.jdbc.Driver");
   dataSource.setUrl("jdbc:mysql://localhost:3306/cardb");
   dataSource.setUsername("cardbuser");
   dataSource.setPassword("123test321");

   return dataSource;
}
```

此方法`dataSource()`返回类的对象`DriverManagerDatSource`。该方法的其余部分将使用 MySQL jdbc 驱动程序、连接字符串、用户名和密码等配置来设置数据源对象。该代码是不言自明的。

现在，是时候揭开本教程的核心对象，即`JdbcTemplate`对象的创建。我真正喜欢使用的是带有命名参数的查询。为了做到这一点，我必须使用特定类型的`JdbcTemplate`对象 class `NamedParameterJdbcTemplate`。所以这里是：

爪哇 复制代码

```
@Bean
public NamedParameterJdbcTemplate namedParameterJdbcTemplate()
{
   NamedParameterJdbcTemplate retBean
       = new NamedParameterJdbcTemplate(dataSource());
    return retBean;
}
```

这个方法所做的就是创建一个 的对象`NamedParameterJdbcTemplate`。它需要一个数据源对象。

您最不需要的是事务管理器。Spring 事务管理器可用于向存储库方法添加事务注释，以便 CRUD 操作可以包装到 SQL 事务中。这是事务管理器对象：

爪哇 复制代码

```
@Bean
public DataSourceTransactionManager txnManager()
{
   DataSourceTransactionManager txnManager
      = new DataSourceTransactionManager(dataSource());
   return txnManager;
}
```

有了这个事务管理器，就可以在存储库方法中提交和回滚事务。这些是我们让`JdbcTemplate`对象工作所需的所有配置。我们最后需要的是带有 CRUD 操作的存储库类。

存储库类

一个`repository`对象用于与SQL数据库执行CRUD操作。为了简单起见，我直接`repository`在`controller`类中使用了我的对象。现实情况是，最好将 DTO 对象和数据实体对象分开。以及在存储库层之上的服务层。

我的`repository`课是这样的：

爪哇 缩小▲ 复制代码

```
package org.hanbo.boot.rest.repository;

import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.hanbo.boot.rest.models.CarModel;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

@Repository
public class CarRepository
{

   @Autowired
   private NamedParameterJdbcTemplate sqlDao;

   private final String addCar_sql = "INSERT INTO carinfo (yearofmanufacture, model, make,
           suggestedretailprice, fullprice, rebateamount, createdate, updatedate)"
           + " VALUES (:yearOfManufacture, :model, :make, :suggestedRetailPrice, :fullPrice,
           :rebateAmount, :createdDate, :updatedDate)";

   private final String getCars_sql = "SELECT id,"
```

```
        + " yearofmanufacture,"
        + " model,"
        + " make,"
        + " suggestedretailprice,"
        + " fullprice,"
        + " rebateamount,"
        + " createdate,"
        + " updatedate FROM carinfo WHERE make = :make AND
        yearofmanufacture >= :startYear AND yearofmanufacture <= :endYear";


    @Transactional
    public void addCar(CarModel carToAdd)
    {
        if (carToAdd != null)
        {
            Map<String, Object> parameters = new HashMap<String, Object>();

            Date dateNow = new Date();

            parameters.put("yearOfManufacture", carToAdd.getYearOfManufacturing());
            parameters.put("model", carToAdd.getModel());
            parameters.put("make", carToAdd.getMaker());
            parameters.put("suggestedRetailPrice", carToAdd.getSuggestedRetailPrice());
            parameters.put("fullPrice", carToAdd.getFullPrice());
            parameters.put("rebateAmount", carToAdd.getRebateAmount());
            parameters.put("createdDate", dateNow);
            parameters.put("updatedDate", dateNow);

            int retVal = sqlDao.update(addCar_sql, parameters);
            System.out.println("Rows updated: " + retVal);
            throw new RuntimeException("dummy Bad");
        }
        else
        {
            System.out.println("Car to add is invalid. Null Object.");
        }
    }

    @Transactional
    public List<CarModel> findCar(String make, int startYear, int endYear)
    {
        List<CarModel> foundObjs = sqlDao.query(getCars_sql,
            (new MapSqlParameterSource("make", make))
                .addValue("startYear", startYear)
                .addValue("endYear", endYear),
            (rs) -> {
                List<CarModel> retVal = new ArrayList<CarModel>();
                if (rs != null)
                {
                    while(rs.next())
                    {
                        CarModel cm = new CarModel();
                        cm.setYearOfManufacturing(rs.getInt("yearOfManufacture"));
                        cm.setMaker(rs.getString("make"));
                        cm.setModel(rs.getString("model"));
                        cm.setSuggestedRetailPrice(rs.getFloat("suggestedretailprice"));
                        cm.setFullPrice(rs.getFloat("fullprice"));
                        cm.setRebateAmount(rs.getFloat("rebateamount"));
                        retVal.add(cm);
                    }
                }

                return  retVal;
            });

        return foundObjs;
    }
}
```

关于这个类的第一件事是它用`@Repository`. 接下来是`JdbcTemplate`对象的自动装配注入，将用于数据访问 CRUD 操作。

爪哇 复制代码

```
@Autowired
private NamedParameterJdbcTemplate sqlDao;
```

`JdbcTemplate`对象的注入来自配置类。接下来，我必须定义两个查询`string`。一种是用于将数据行插入到数据库表中。另一个是返回`car`模型列表的查询。

爪哇 复制代码

```
private final String addCar_sql = "INSERT INTO carinfo (yearofmanufacture, model, make,
                suggestedretailprice, fullprice, rebateamount, createdate, updatedate)"
   + " VALUES (:yearOfManufacture, :model, :make, :suggestedRetailPrice, :fullPrice,
                :rebateAmount, :createdDate, :updatedDate)";

private final String getCars_sql = "SELECT id,"
   + " yearofmanufacture,"
   + " model,"
   + " make,"
   + " suggestedretailprice,"
   + " fullprice,"
   + " rebateamount,"
   + " createdate,"
   + " updatedate FROM carinfo WHERE make = :make AND
     yearofmanufacture >= :startYear AND yearofmanufacture <= :endYear";
```

还记得我说的命名参数吗？在上面的查询，我用是这样的：`:suggestedRetailPrice`。这是一个命名参数。命名参数是以冒号“`:`”开头，后跟实际参数名称的东西。无论如何，这是添加汽车模型的方法：

爪哇 复制代码

```
@Transactional
public void addCar(CarModel carToAdd)
{
   if (carToAdd != null)
   {
      Map<String, Object> parameters = new HashMap<String, Object>();

      Date dateNow = new Date();

      parameters.put("yearOfManufacture", carToAdd.getYearOfManufacturing());
      parameters.put("model", carToAdd.getModel());
      parameters.put("make", carToAdd.getMaker());
      parameters.put("suggestedRetailPrice", carToAdd.getSuggestedRetailPrice());
      parameters.put("fullPrice", carToAdd.getFullPrice());
      parameters.put("rebateAmount", carToAdd.getRebateAmount());
      parameters.put("createdDate", dateNow);
      parameters.put("updatedDate", dateNow);

      int retVal = sqlDao.update(addCar_sql, parameters);
      System.out.println("Rows updated: " + retVal);
   }
   else
   {
      System.out.println("Car to add is invalid. Null Object.");
   }
}
```

它的工作方式：

- 定义一个`Map`对象，其中键与命名参数匹配，值是将添加到数据库中的命名参数的值。
- 使用`NamedParameterJdbcTemplate`被调用的对象`sqlDao`来调用`update()`，传入`addCar_sql`（它具有插入`car`模型的查询）。
- 对 update 的调用将返回一个整数，指示有多少行受到影响。
- 最后，我打印出整数，看看它是否成功。

查找cars列表的查询由另一个名为 的存储库方法完成findCar()，这是源代码：

爪哇 缩小▲ 复制代码

```
@Transactional
public List<CarModel> findCar(String make, int startYear, int endYear)
{
   List<CarModel> foundObjs = sqlDao.query(getCars_sql,
      (new MapSqlParameterSource("make", make))
         .addValue("startYear", startYear)
         .addValue("endYear", endYear),
      (rs) -> {
         List<CarModel> retVal = new ArrayList<CarModel>();
         if (rs != null)
         {
            while(rs.next())
            {
               CarModel cm = new CarModel();
               cm.setYearOfManufacturing(rs.getInt("yearOfManufacture"));
               cm.setMaker(rs.getString("make"));
               cm.setModel(rs.getString("model"));
               cm.setSuggestedRetailPrice(rs.getFloat("suggestedretailprice"));
               cm.setFullPrice(rs.getFloat("fullprice"));
               cm.setRebateAmount(rs.getFloat("rebateamount"));
               retVal.add(cm);
            }
         }

         return  retVal;
      });

   return foundObjs;
}
```

这个方法有点复杂。下面是它的工作原理：

该方法调用sqlDao的query()方法，传入三个参数：

- 第一个参数是getCars_sql，它是查询字符串。
- 第二个参数是 的对象MapSqlParameterSource，它可能是一个包装Map对象。键匹配命名参数，值是命名参数的值。
- 第三个是用函数式编程语言编写的接口的实现，它基本上将ResultSet对象映射到实体对象列表。

方法调用完成后，CarModel将返回类型对象列表。该方法的函数式编程部分是这样的：

爪哇 复制代码

```
(rs) -> {
   List<CarModel> retVal = new ArrayList<CarModel>();
   if (rs != null)
   {
      while(rs.next())
      {
         CarModel cm = new CarModel();
         cm.setYearOfManufacturing(rs.getInt("yearOfManufacture"));
         cm.setMaker(rs.getString("make"));
         cm.setModel(rs.getString("model"));
         cm.setSuggestedRetailPrice(rs.getFloat("suggestedretailprice"));
         cm.setFullPrice(rs.getFloat("fullprice"));
         cm.setRebateAmount(rs.getFloat("rebateamount"));
         retVal.add(cm);
      }
   }

   return  retVal;
}
```

名为“ rs”的输入参数的类型为ResultSet。箭头“ ->”基本上表示ResultSet对象将由外壳中定义的功能操作{...}。在这个附件中，它首先创建一个CarModel. 然后它检查ResultSet对象是否不是null。它将一次一行地遍历结果集并将该行转换为一

个CarModel对象。该对象将被添加到列表中。最后，列表将返回。该list对象是该findCars()方法将返回的列表。

这就是关于如何设置要使用的示例项目的全部内容JdbcTemplate。是时候测试一下了。

测试应用程序

现在我们拥有了应用程序的所有内容，是时候测试它并查看它是否有效。它有效。在我们进行测试之前，我们需要构建它。然后，我们需要将它作为 Java 应用程序启动。

要构建项目，请cd在示例项目的基目录中使用“ ”，然后使用以下命令：

复制代码

```
mvn clean install
```

要启动应用程序，请在示例项目的基本目录中使用以下命令：

复制代码

```
java -jar target\hanbo-boot-rest-1.0.1.jar
```

命令行控制台会输出很多文本。最后，它会成功并输出如下内容：

复制代码

```
...
2018-12-01 22:56:10.870  INFO 10832 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  :
Tomcat started on port(s): 8080 (http) with context path ''

2018-12-01 22:56:10.877  INFO 10832 --- [           main] org.hanbo.boot.rest.App
:
Started App in 15.656 seconds (JVM running for 18.044)
```

现在，是时候测试它了。我使用了一个名为“ Postman”的程序，它可以向这个 RESTFul 示例应用程序发送 HTTP 请求。要尝试的第一件事是将CarModel对象添加到数据库中。这个网址是：

复制代码

```
http://localhost:8080/public/addCar
```

此 URL 仅接受 HTTP Post 请求。请求正文将是一个string表示CarModel对象的 JSON ，如下所示：

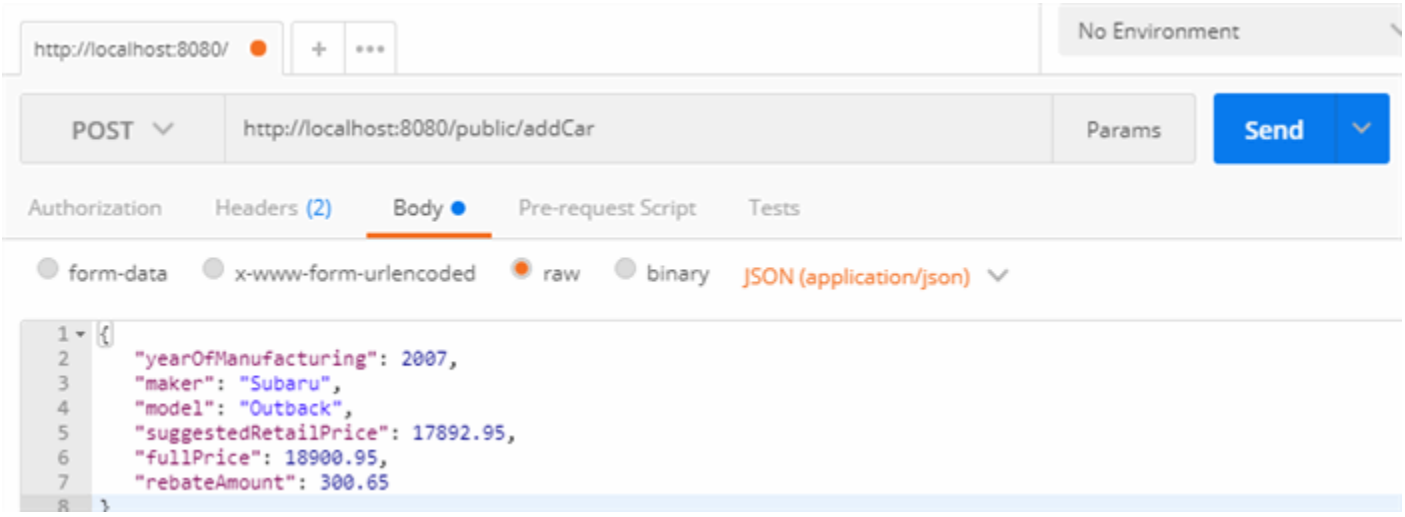
JavaScript 复制代码

```
{
   "yearOfManufacturing": 2007,
   "maker": "Subaru",
   "model": "Outback",
   "suggestedRetailPrice": 17892.95,
   "fullPrice": 18900.95,
   "rebateAmount": 300.65
}
```

要使用“ Postman”发送请求，请执行以下操作：

- 将 URL 设置为*http://localhost:8080/public/addCar*
- 在 URL 文本框的左侧，选择 HTTP 方法为“ POST”。
- 请求正文应包含 JSON string。
- 将内容类型设置为“ application/json”。

这是“ Postman”的屏幕截图：



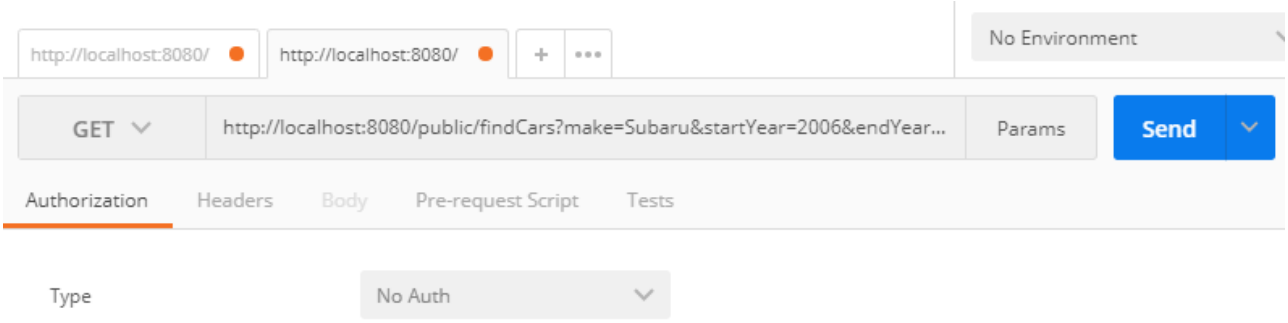
单击“**发送**”按钮。服务器将返回成功响应。这是一个屏幕截图：

现在尝试按制造商“ Subaru”、开始年份（值：）2006和结束年份（值：）搜索汽车2008。请求是通过GET带有请求参数的HTTP完成的。这是请求网址：

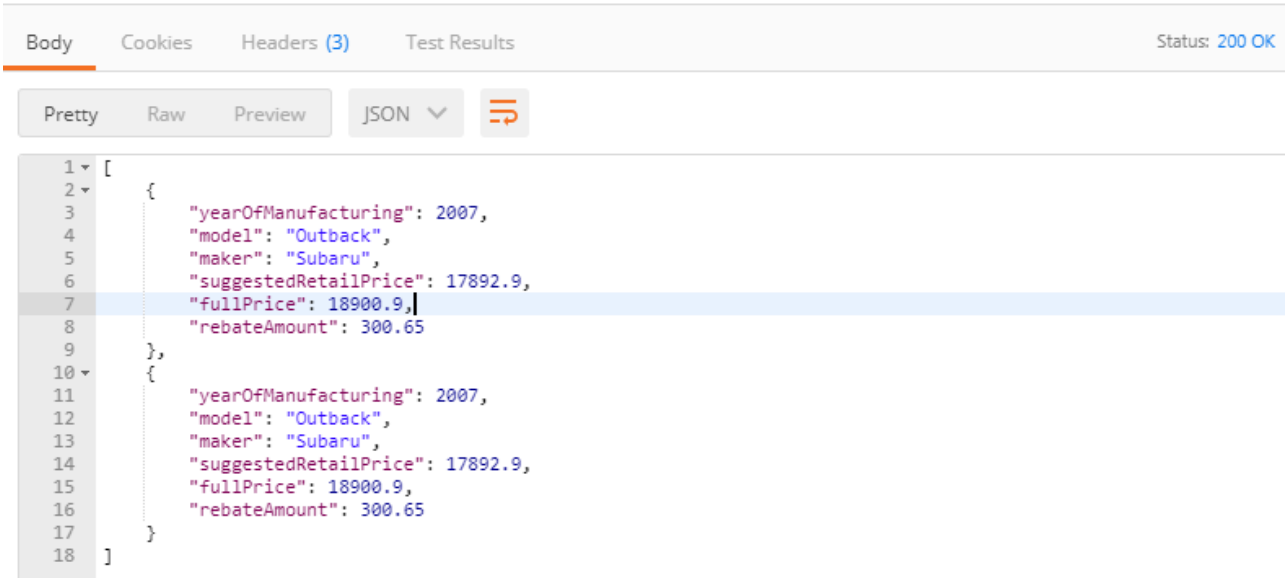
复制代码

```
http://localhost:8080/public/getCars?make=Subaru&startYear=2006&endYear=2008
```

Postman此请求的“ ”屏幕截图如下所示：



单击“**发送**”按钮，请求将成功处理，并使用string表示找到的汽车列表的JSON 。根据您在数据库中添加的汽车，该列表将没有元素、一个或多个元素。这是我两次添加同一辆车的屏幕截图：



就这些。如果要测试事务回滚，可以`addCar()`在调用后的行上添加一个抛出的异常`sqlDao.update()`。然后测试添加汽车场景。将抛出异常并阻止将行添加到数据库中。

概括

这是我为 2018 年发布的最后一个教程。 2017 年 12 月 31 日，我开始在 2018 年写 10 个教程，我在 2018 年 12 月 1 日完成了这一点。一整年，我有推出各种主题的教程。这是我最近的努力。

在本教程中，我讨论了以下主题：

- 基于 JDBC 的数据访问的配置。
- 可以将实体添加到数据库并根据一些简单标准进行检索的存储库对象。
- 使用`JdbcTemplate`, 专门`NamedParameterJdbcTemplate`用于数据插入和检索的对象。
- 以及通过示例应用程序的 RESTFul 接口对数据访问的一些简单测试。

与往常一样，推出这样的教程真是太棒了。我希望你喜欢它。

历史

- 12-01-2018 - 初稿

执照

分享

关于作者

韩博孙
组长 The Judge Group
美国
手表该会员
没有提供传记

评论和讨论

添加评论或问题 | 电子邮件提醒 | Search Comments

-- 本论坛暂无消息 --