CodeJava
Coding Your Passion

Learn Spring framework:

- Learn Spring on YouTube
- Understand the core of Spring
- Understand Spring MVC
- Understand Spring AOP
- Understand Spring Data JPA
- Spring Dependency Injection (XML)
- Spring Dependency Injection (Annotations)
- Spring Dependency Injection (Java config)
- Spring MVC beginner tutorial
- Spring MVC Exception Handling
- Spring MVC and log4j
- Spring MVC Send email
- Spring MVC File Upload
- Spring MVC Form Handling

- Spring MVC Form Validation
- Spring MVC File Download
- Spring MVC JdbcTemplate
- Spring MVC CSV view
- Spring MVC Excel View
- Spring MVC PDF View
- Spring MVC XstlView
- Spring MVC + Spring Data JPA + Hibernate - CRUD
- Spring MVC Security (XML)
- Spring MVC Security (Java config)
- Spring & Hibernate Integration (XML)
- Spring & Hibernate Integration (Java config)
- Spring & Struts Integration (XML)
- Spring & Struts Integration (Java config)
- 14 Tips for Writing Spring MVC Controller

Spring Boot Export Data to CSV Example

Written by Nam Ha Minh

Last Updated on 04 September 2020 | Print Email

In this tutorial, I will guide you how to implement CSV export function in a Spring Boot application that uses Spring Data JPA and Hibernate for the data access layer, Thymeleaf as template engine in the view layer, and MySQL database. The CSV export function allows the user to download data from a table in the database to a file in CSV (comma-separated values) format.

The code examples below demonstrate export information about users from database to CSV files.

1. Code for the Entity Classes and Repositories

Suppose that we have the `User` entity class as follows:

```
package net.codejava;

import java.util.*;
import javax.persistence.*;

@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String email;

    private String password;

    @Column(name = "full_name")
    private String fullName;

    private boolean enabled;

    @ManyToMany(cascade = CascadeType.PERSIST, fetch = FetchType.EAGER
    @JoinTable(
            name = "users_roles",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id")
            )
    private Set<Role> roles = new HashSet<>();

    // constructors, getter and setters are not shown for brevity
}
```

And the `Role` entity class:

```java
package net.codejava;

import javax.persistence.*;

@Entity
@Table(name = "roles")
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String name;

    private String description;

    // constructors, getter and setters are not shown for brevity
}
```

So the fields we want to include in the generated CSV file are: User ID, E-mail, Full Name, Roles and Enabled. And nothing special about the repositories, as shown below:

```java
package net.codejava;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Integer> {

}


public interface RoleRepository extends CrudRepository<Role, Integer>

}
```

2. Declare Dependency for CSV Library

Though CSV is a simple file format (values are separated by commas), it's still much better to use a dedicated CSV library. In this guide, I'm using SuperCSV – a free and open-source CSV library for Java. So declare the following dependency in the `pom.xml` file:

```xml
<dependency>
    <groupId>net.sf.supercsv</groupId>
    <artifactId>super-csv</artifactId>
    <version>2.4.0</version>
</dependency>
```

3. Code for the Service Class

We have the `UserServices` class that implements the `listAll()` method that retrieves all users from the database, as follows:

```
1  package net.codejava;
2
3  import java.util.List;
4
5  import javax.transaction.Transactional;
6
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.data.domain.Sort;
9  import org.springframework.stereotype.Service;
10
11 @Service
12 @Transactional
13 public class UserServices {
14
15     @Autowired
16     private UserRepository repo;
17
18     public List<User> listAll() {
19         return repo.findAll(Sort.by("email").ascending());
20     }
21
22 }
```

The `findAll()` method in the `UserRepository` interface is implemented by Spring Data JPA (extended from `JpaRepository`). Here I just pass a `Sort` object to sort the result list by email of the users, in ascending order.

4. Code Export to CSV in the Controller Class

We're going to implement the CSV export function for an existing Spring Boot web application, so we write the code that allows the users to download a CSV file in a handler method of a controller class, as shown below:

```
package net.codejava;

import java.io.IOException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

import javax.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

import org.supercsv.io.CsvBeanWriter;
import org.supercsv.io.ICsvBeanWriter;
import org.supercsv.prefs.CsvPreference;

@Controller
public class UserController {

    @Autowired
    private UserServices service;


    @GetMapping("/users/export")
    public void exportToCSV(HttpServletResponse response) throws IOExc
        response.setContentType("text/csv");
        DateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-dd_HH
        String currentDateTime = dateFormatter.format(new Date());

        String headerKey = "Content-Disposition";
        String headerValue = "attachment; filename=users_" + currentDa
        response.setHeader(headerKey, headerValue);

        List<User> listUsers = service.listAll();

        ICsvBeanWriter csvWriter = new CsvBeanWriter(response.getWrite
        String[] csvHeader = {"User ID", "E-mail", "Full Name", "Roles
        String[] nameMapping = {"id", "email", "fullName", "roles", "e

        csvWriter.writeHeader(csvHeader);

        for (User user : listUsers) {
            csvWriter.write(user, nameMapping);
        }

        csvWriter.close();

    }

}
```

Let me explain this code. To send data to the users as file download, we need to set the header “Content-Disposition” for the response as below:

```
String headerKey = "Content-Disposition";
String headerValue = "attachment; filename=users_" + currentDateTime +

response.setContentType("text/csv");
response.setHeader(headerKey, headerValue);
```

The content type is set to text/csv so the browser will know and handle it properly. And the CSV file name is generated based on the current date time:

```
DateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-dd_HH-mm-ss");
String currentDateTime = dateFormatter.format(new Date());
```

So each time the user downloads a CSV file, its name is different – with datetime appended to the end of file name.

And the rest of the code uses SuperCSV library to generate the CSV file based on the data (list users) returned from the `UserServices` class.

Note that to write CSV data to the response, the response's writer is passed to the `CsvBeanWriter`:

```
ICsvBeanWriter csvWriter = new CsvBeanWriter(response.getWriter(), CsvP
```

And to map the columns in the CSV file with field names in the entity class, we use an array of String like this:

```
String[] nameMapping = {"id", "email", "fullName", "roles", "enabled"};
```

So make sure to use this name mapping so the CSV writer can read field names from the entity class properly.

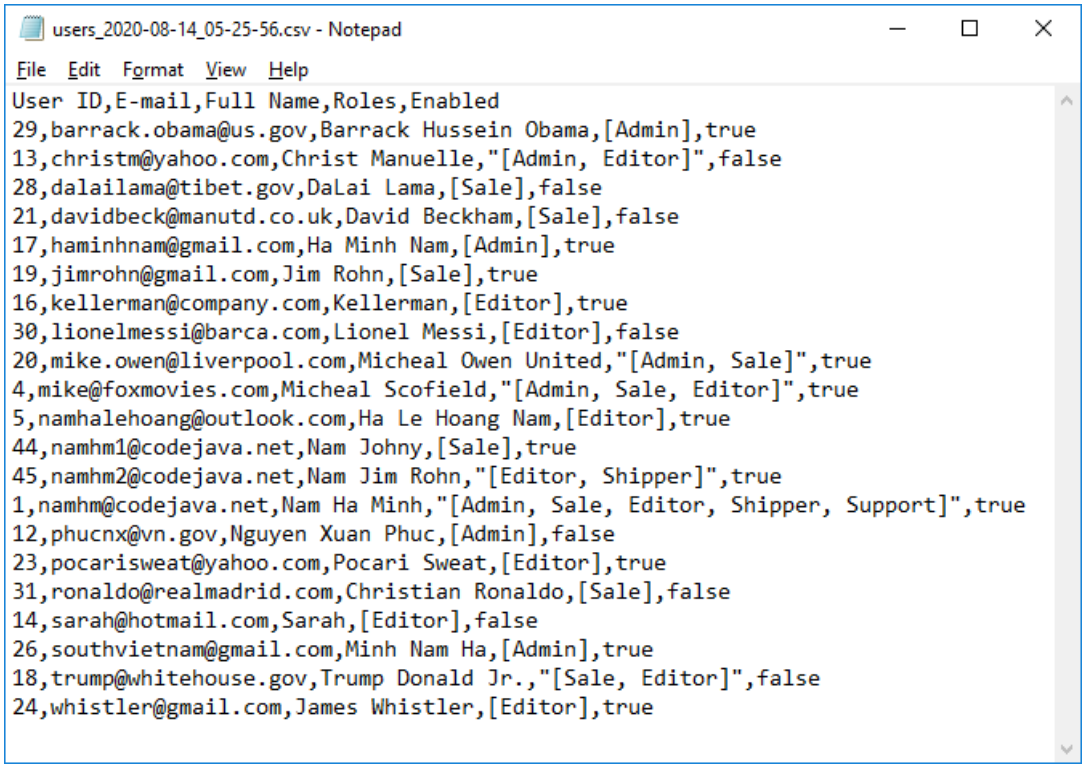
5. Add Export CSV Link in the View Page

We use HTML and Thymeleaf to create a hyperlink that allows the user to click to export data to CSV as follows:

```
<a th:href="/@{/users/export}">Export to CSV</a>
```

6. Test Export and Download CSV file

Click the hyperlink Export to CSV, the Spring Boot application will generate a CSV file and the browser will automatically download that file. The file name is something like this: `users_2020-08-14_05-25-56.csv`. Open this file using a text editor like Notepad, you will see it is actually a CSV file:



Conclusion

So far you have learned how to code CSV export function for a Spring Boot web application. You see, Spring Data JPA makes it easy to get data from the database, and SuperCSV makes it easy to generate CSV files.

For video version of this tutorial, watch the video below:

Related Tutorials:

- Spring MVC with CSV File Download Example
- Java Reading CSV File Example with Super CSV
- Java Export to CSV File Example
- Java code example to export from database to CSV file
- Java code example to insert data from CSV to database

Other Spring Boot Tutorials:

- Spring Boot Export Data to Excel Example
- Spring Boot Export Data to PDF Example
- Spring Boot Hello World Example
- Spring Boot automatic restart using Spring Boot DevTools
- Spring Boot Form Handling Tutorial with Spring Form Tags and JSP
- How to create a Spring Boot Web Application (Spring MVC with JSP/ThymeLeaf)
- Spring Boot - Spring Data JPA - MySQL Example
- Spring Boot Hello World RESTful Web Services Tutorial
- How to use JDBC with Spring Boot
- Spring Boot CRUD Web Application with JDBC - Thymeleaf - Oracle
- Spring Boot RESTful CRUD API Examples with MySQL database
- How to package Spring Boot application to JAR and WAR
- Spring Boot Security Authentication with JPA, Hibernate and MySQL
- Spring Data JPA Paging and Sorting Examples
- Spring Boot Error Handling Guide

About the Author:

Nam Ha Minh is certified Java programmer (SCJP and SCWCD). He started programming with Java in the time of Java 1.4 and has been falling in love with Java since then. Make friend with him on Facebook and watch his Java videos you YouTube.

Add comment

Name E-mail

comment

Notify me of follow-up comments

I'm not a robot reCAPTCHA Privacy - Terms

Send

Comments

#5**Luis** 2021-03-25 04:15
Hello, is there any way to change the separator? European countries use ';' instead of ',' as the later is their digit separator.

Quote

#4**Digaant Garg** 2021-02-01 03:19
Hi, if I want to schedule a task which generates a CSV file everyday at 16:00 , how should I go about it, as this method includes parameters, and scheduling

doesn't allow any parameters.

Quote

#3**Hninwai** 2020-11-28 10:07

Thank you so much for tutorial. It is very useful for me and thank you.

Quote

#2**Keval** 2020-10-01 00:07

Thanks a lot for the tutorial, Although can you do the same for Json List of Objects

Quote

#1**Surya** 2020-09-02 20:37

Thank you for the useful tutorial, it demonstrates the concept clearly.

Quote

Refresh comments list

See All Java Tutorials