

Spring Security 系列 — 带有 H2、JPA、Bootstrap 和 Thymeleaf 的 Spring Boot 应用程序 (第 1 部分)



音乐节

跟随



2019 年 10 月 1 日 · 6分钟阅读

在本系列文章中，我们将深入研究 Spring Security，以演示 Spring Security 用于保护 Web 应用程序的用法。在本文中，我们打算开发一个 **Todo** 应用程序并将其用作整个系列的基础。

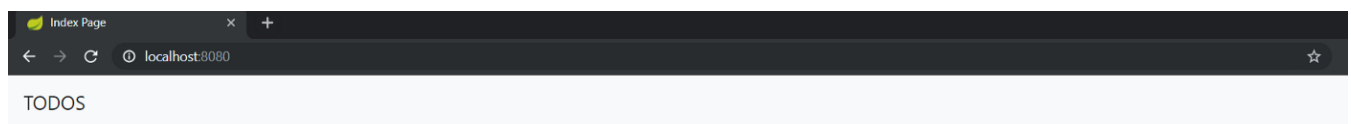


我们在建造什么？

我们正在构建一个 **Todo 管理器**。在我们的应用程序中，用户可以创建、更新、查看和删除各种 Todo 项目。

主页：

以下是我们的应用程序主页： -



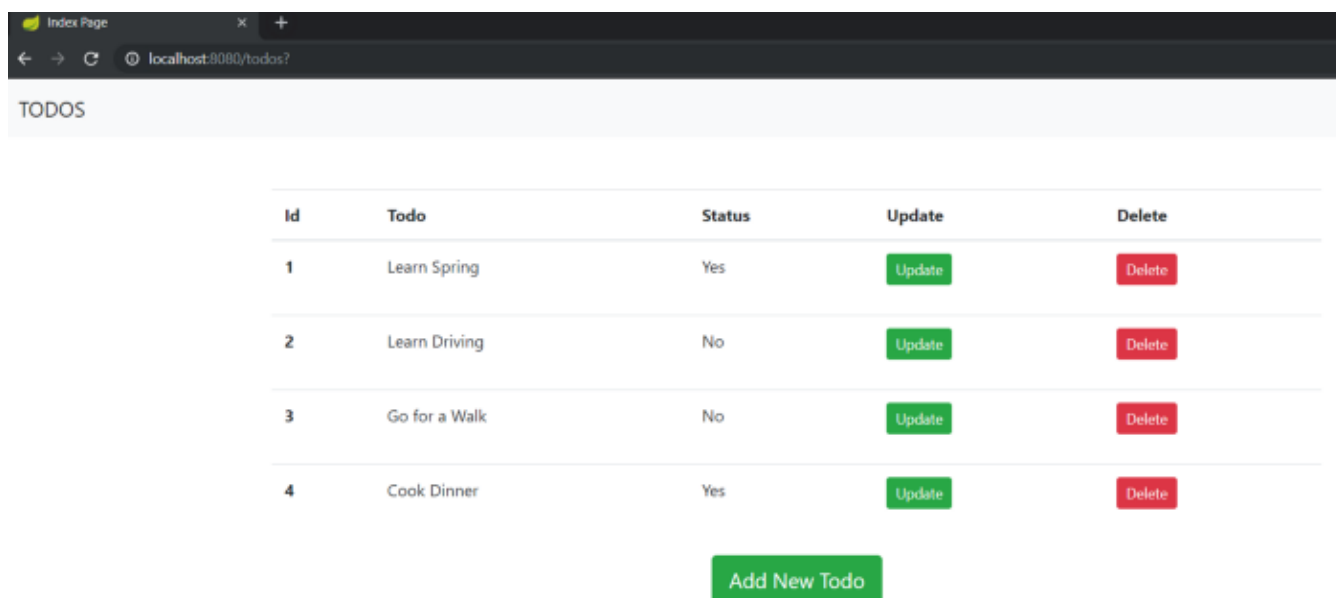
Welcome to Todo Manager

View Todos

Todo 管理器应用程序主页

查看所有待办事项：

应用程序中可用的当前待办事项列表: -



Id	Todo	Status	Update	Delete
1	Learn Spring	Yes	<button>Update</button>	<button>Delete</button>
2	Learn Driving	No	<button>Update</button>	<button>Delete</button>
3	Go for a Walk	No	<button>Update</button>	<button>Delete</button>
4	Cook Dinner	Yes	<button>Update</button>	<button>Delete</button>

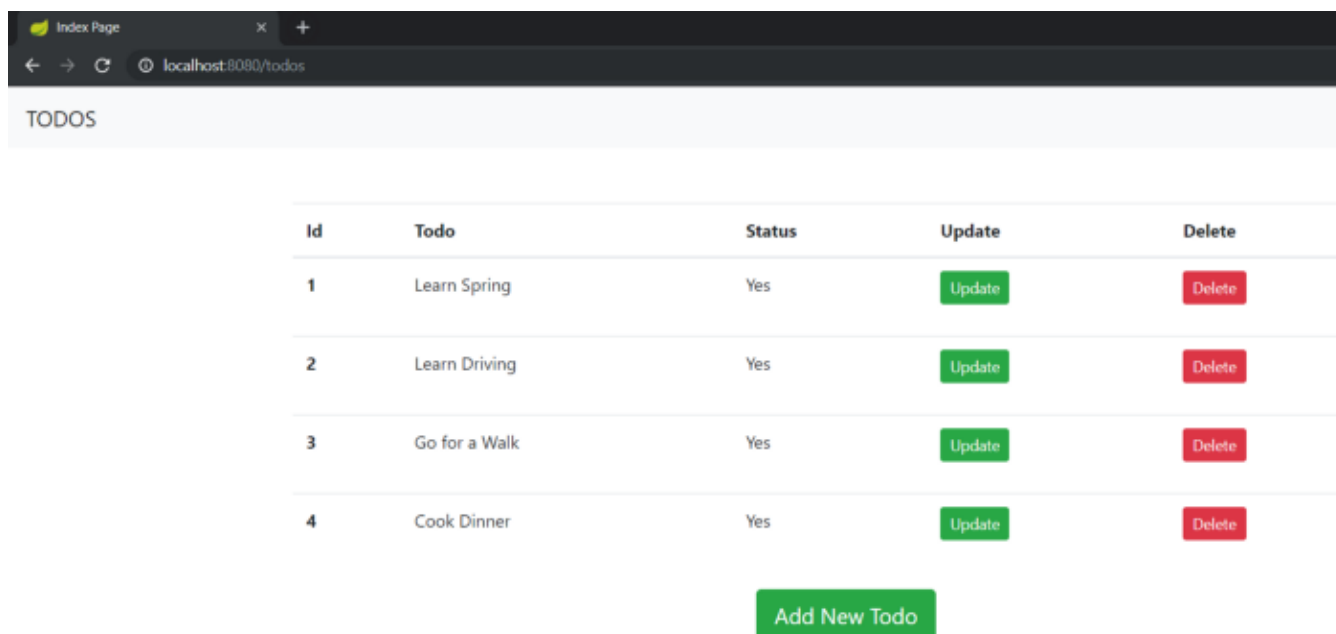
Add New Todo

查看所有待办事项



更新一个 Todo:

在应用程序中, 用户可以切换待办事项状态。为了演示, 我们已将所有 Todos 更新为 **Yes** :-



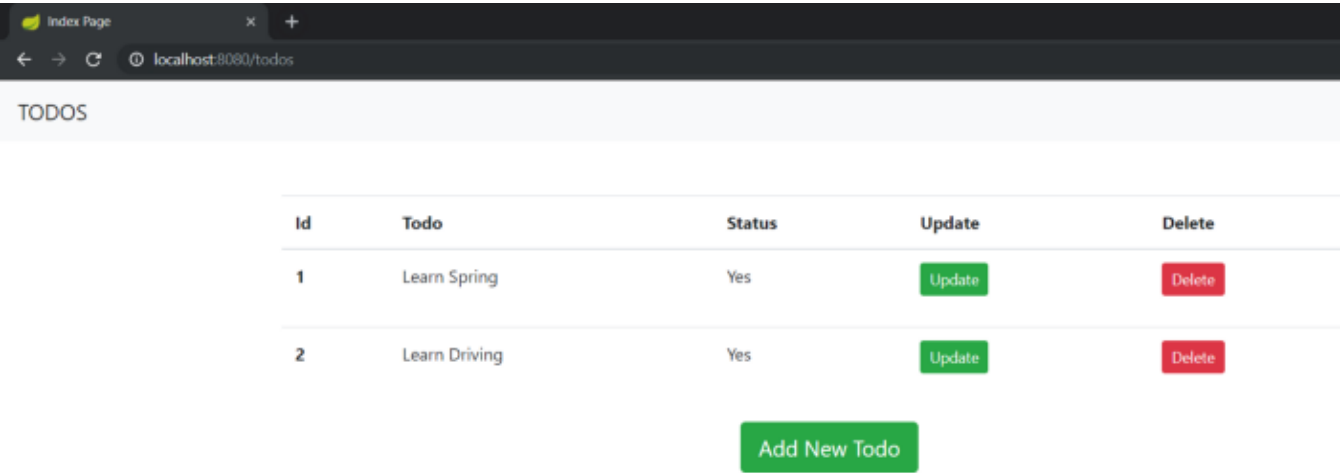
Id	Todo	Status	Update	Delete
1	Learn Spring	Yes	<button>Update</button>	<button>Delete</button>
2	Learn Driving	Yes	<button>Update</button>	<button>Delete</button>
3	Go for a Walk	Yes	<button>Update</button>	<button>Delete</button>
4	Cook Dinner	Yes	<button>Update</button>	<button>Delete</button>

Add New Todo

在应用程序中更新 Todos

删除一个待办事项:

一旦完成并且不再需要，用户可以从应用程序中删除 **Todo** 项目。为了演示用法，我们从应用程序中删除了最后两个 **Todos**： -

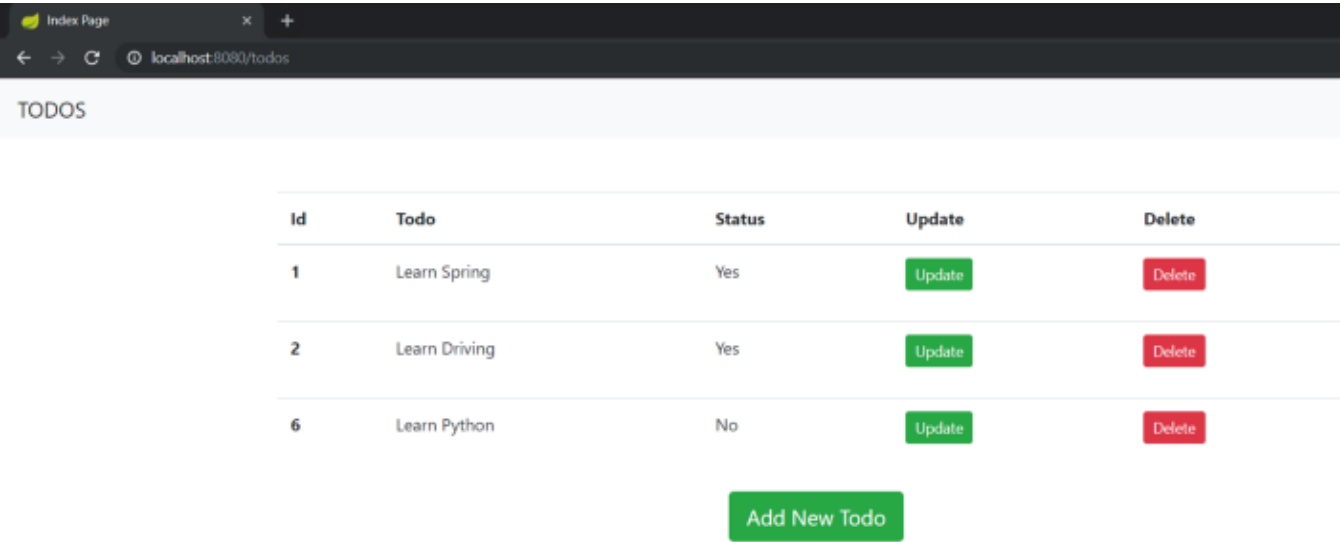


从应用程序中删除 Todos



添加一个新的 **Todo**:

应用程序用户可以轻松地在应用程序中添加新的 **Todo** 项目。我们在我们的应用程序中创建了一个新的 **Todo**，它在 **todo** 列表中可用： -



创建一个新的 Todo

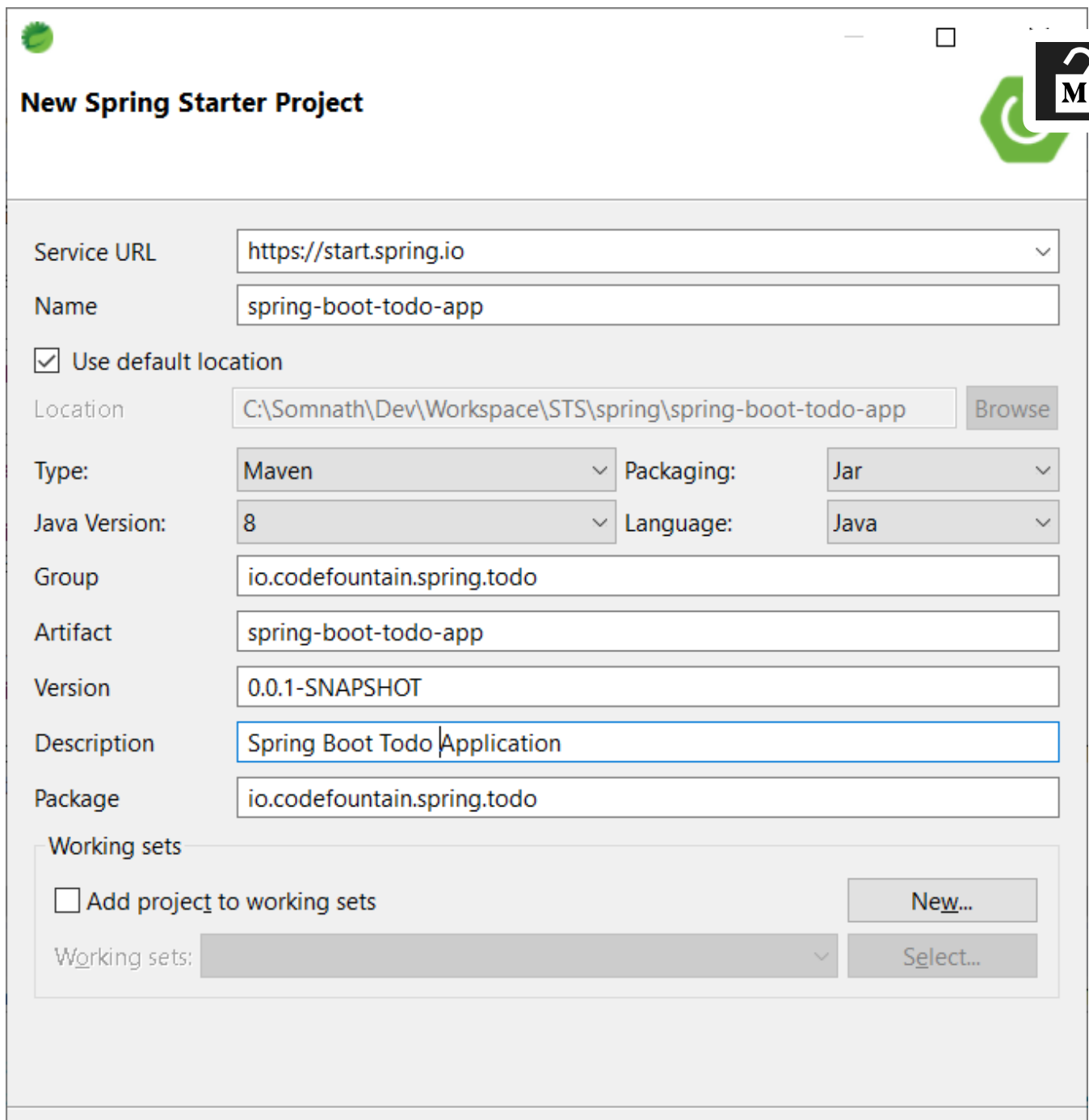
让我们构建应用程序

为了构建我们的 **Todo** 应用程序，我们将使用以下技术/工具： -

1. 弹簧靴
2. H2 内存数据库
3. 弹簧数据 JPA
4. 引导程序
5. 百里香叶
6. 弹簧工具套件 (STS)

创建一个 Spring Boot 项目

打开 STS 并创建一个具有以下依赖项的 Spring Boot 项目： -



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:


Description:

Package:

Working sets

☐ Add project to working sets

Working sets:




< Back
Next >
Finish
Cancel

在 STS 中创建 Todo 应用

添加以下依赖项。我们正在使用：-


1. **Spring Web:** 这将为我们的应用程序提供 web/MVC 特性
2. **Lombok:** 这将帮助我们使用构造函数、getter/setter 和 ToString
3. **H2:** 用作后端内存数据库
4. **Spring Data JPA:** 提供 Java Persistence API 支持
5. **Thymeleaf:** 我们的前端模板引擎。这非常有用，并为我们提供了在 HTML 页面中动态呈现内容的能力
6. **Spring Boot DevTools:** Devtools 在开发环境中非常方便。它会自动刷新应用程序更改，我们不需要总是（有时需要）在开发应用程序时重新启动我们的应用程序





—
□
×

New Spring Starter Project Dependencies



Spring Boot Version: 2.1.8

Frequently Used:

☒ H2 Database
☐ Okta
☐ Spring Security

☒ Lombok
☒ Spring Boot DevTools
☒ Spring Web

☐ OAuth2 Client
☒ Spring Data JPA
☒ Thymeleaf

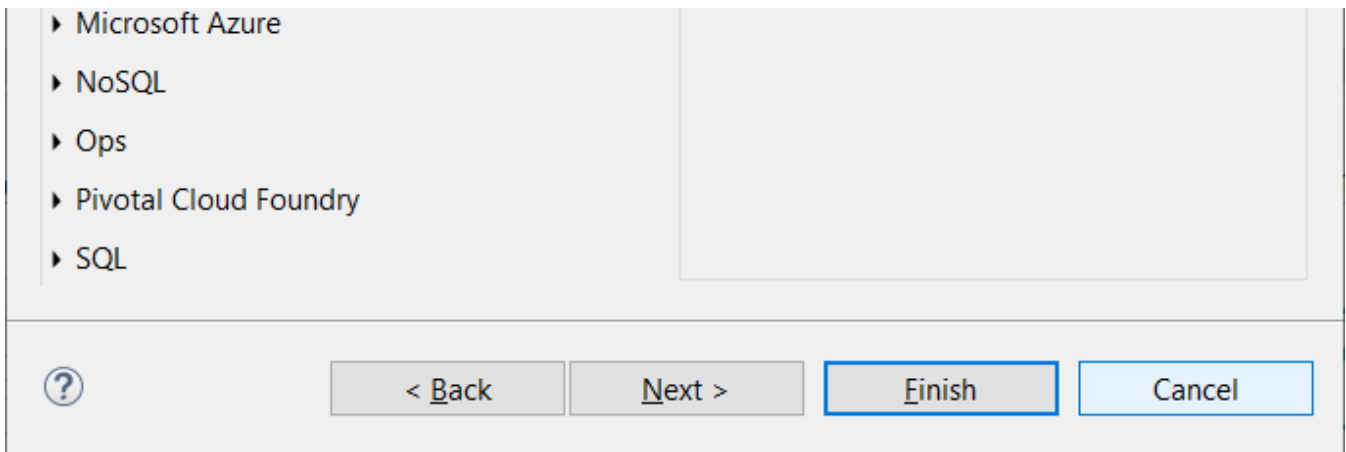
Available:

Type to search dependencies

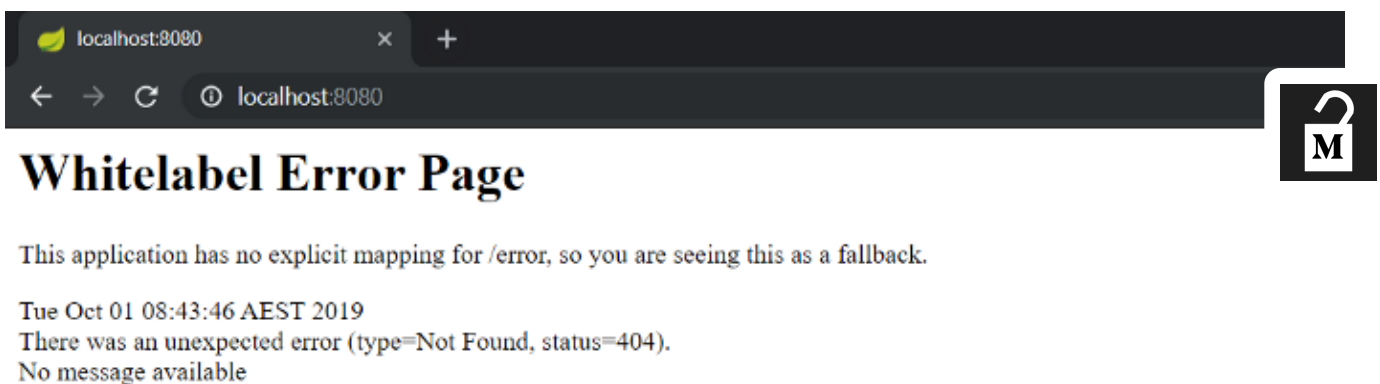
- ▶ Amazon Web Services
- ▶ Developer Tools
- ▶ Google Cloud Platform
- ▶ I/O
- ▶ Messaging

Selected:

- X Spring Boot DevTools
- X Lombok
- X Spring Data JPA
- X H2 Database
- X Thymeleaf
- X Spring Web



单击**完成**并等待下载依赖项。应用程序准备好后，启动它，我们应该会看到以下欢迎屏幕：



默认欢迎页面

就是这样！我们的应用程序正在运行。

我们正在观察此错误消息，因为我们没有索引页面。在下一节中，我们将创建索引和待办事项网页。

创建前端和逻辑

创建欢迎页面 (index.html):

在我们的 `index.html` 页面中，我们添加了 **Bootstrap** 和 **Thymeleaf** 支持。这是我们的默认登陆页面，而且很简单。我们有一个导航栏和一个按钮来查看 **Todos** 列表。这个页面应该放在 `src/main/resources/templates` 文件夹下：

```

1  <!DOCTYPE html>
2  <html lang="en" xmlns:th="http://www.thymeleaf.org">
3  <head>
4  <meta charset="UTF-8">
5  <meta name="viewport" content="width=device-width, initial-scale=1.0">
6  <meta http-equiv="X-UA-Compatible" content="ie=edge">
7  <title>Index Page</title>
8  <style type="text/css">
9  #todocontainer {
10     margin-top: 50px;
11 }
12 </style>
13 <link rel="stylesheet"
14     href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
15 </head>
16 <body>
17     <section id="header">
18         <nav class="navbar navbar-expand-lg navbar-light bg-light">
19             <a class="navbar-brand" href="#">TODOs</a>
20             <button class="navbar-toggler" type="button" data-toggle="collapse"
21                 data-target="#navbarSupportedContent"
22                 aria-controls="navbarSupportedContent" aria-expanded="false"
23                 aria-label="Toggle navigation">
24                 <span class="navbar-toggler-icon"></span>
25             </button>
26         </nav>
27     </section>
28     <section id="todocontainer">
29         <h2 style="padding-top: 30px; padding-bottom: 20px;" class="text-center">Welcom
30         <div class="d-flex justify-content-center">
31             <form th:action="@{/todos}" method="GET" enctype="multipart/form-data">
32                 <div class="form-group">
33                     <button type="submit" class="btn btn-success btn-lg text-white"
34                 </div>
35             </form>
36         </div>
37     </section>
38     <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>
39     <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
40     <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
41     </body>
42 </html>

```



创建 TodoController

现在我们已经创建了我们的索引页面，让我们创建一个控制器来将 `http://localhost:8080` 上的传入请求映射到这个页面： -

```
1  package io.codefountain.spring.todo.controllers;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.GetMapping;
5
6  @Controller
7  public class TodoController {
8
9      @GetMapping
10     public String index() {
11         return "index";
12     }
13 }
```

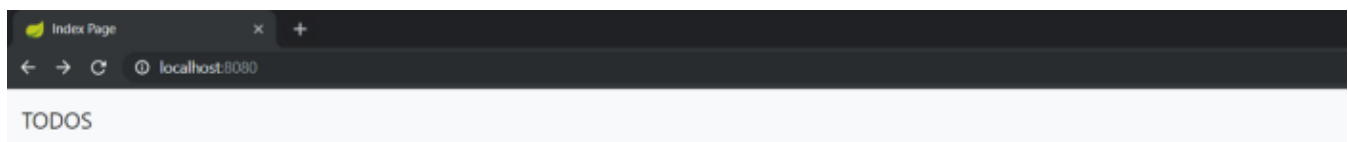


TodoController.java hosted with ❤ by GitHub

[view raw](#)

带有索引页面映射的 Todo 控制器

现在让我们在浏览器中刷新应用程序 URL。请注意，当我们使用 Spring DevTools 时，它将确保重新加载更改，而无需我们完全重新启动应用程序。



Welcome to Todo Manager

[View Todos](#)

Todo 应用主页

创建 Todo 页面

让我们继续并创建 Todo 页面。此页面有 3 个部分。应用程序导航栏，一个显示我们待办事项列表的表格和一个用于创建新待办事项的模式窗口。在显示 Todo 的同时，我们还提供了从应用程序更新和删除 Todo 的选项。

待办事项.html

位于 src/main/resources/templates 文件夹中的 Todo 页面下方: -

```
1  <!DOCTYPE html>
2  <html lang="en" xmlns:th="http://www.thymeleaf.org">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Index Page</title>
8      <style type="text/css">
9          #todocontainer{
10             margin-top: 50px;
11         }
12     </style>
13     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/
14 </head>
15 <body>
16     <section id="header">
17         <nav class="navbar navbar-expand-lg navbar-light bg-light">
18             <a class="navbar-brand" href="#">TODO</a>
19             <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#"
20                 <span class="navbar-toggler-icon"></span>
21             </button>
22             <div class="collapse navbar-collapse" id="navbarSupportedContent">
23                 <ul class="navbar-nav ml-auto">
24                     <li class="nav-item active">
25                         <a class="nav-link" th:href="@{/logout}">Logout</a>
26                     </li>
27                 </ul>
28             </div>
29         </nav>
30     </section>
31     <section id="todocontainer">
32         <div class="row">
33             <div class="col-md-2"></div>
34             <div class="col-md-8">
35                 <table class="table">
36                     <thead>
37                         <tr>
38                             <th scope="col">Id</th>
39                             <th scope="col">Todo</th>
40                             <th scope="col">Status</th>
41                             <th scope="col">Update</th>
42                             <th scope="col">Delete</th>
43                         </tr>
```



```

44         </thead>
45         <tbody>
46             <tr th:each="todo : ${todos}">
47                 <th scope="row" th:text=${todo.id}></th>
48                 <td th:text=${todo.todoItem}></td>
49                 <td th:text=${todo.completed}></td>
50                 <td>
51                     <form th:action="@{/todoUpdate/{id}(id=${todo.id})}" method="POST" enc
52                         <div class="form-group">
53                             <button type="submit" class="btn btn-success btn-sm text-white">Up
54                         </div>
55                     </form>
56                 </td>
57                 <td>
58                     <form th:action="@{/todoDelete/{id}(id=${todo.id})}" method="POST" enc
59                         <div class="form-group">
60                             <button type="submit" class="btn btn-danger btn-sm text-whit
61                         </div>
62                     </form>
63                 </td>
64             </tr>
65         </tbody>
66     </table>
67     <div class="d-flex justify-content-center">
68         <a class="btn btn-success btn-lg text-white" data-toggle="modal" data-targ
69     </div>
70 </div>
71 <div class="col-md-2"></div>
72 </div>
73
74 <!-- View Modal -->
75 <div class="modal fade" id="viewModal" tabindex="-1" role="dialog" aria-labelledby="examplele
76     <div class="modal-dialog modal-dialog-centered" role="document">
77         <div class="modal-content">
78             <div class="modal-header">
79                 <h5 class="modal-title" id="exampleModalLabel">TODO</h5>
80                 <button type="button" class="close" data-dismiss="modal" aria-label="Close">
81                     <span aria-hidden="true">&times;</span>
82                 </button>
83             </div>
84             <div class="modal-body">
85                 <form th:action="@{/todoNew}" method="POST" enctype="multipart/form-data">
86                     <div class="form-group">
87                         <label for="exampleInputEmail1">Todo</label>
88                         <input type="text" class="form-control" name="todoItem" aria-describedby="en
89                     </div>
90                     <div class="form-group">

```



```
91         <label for="exampleFormControlSelect1">Status</label>
92         <select class="form-control" name="status">
93             <option>Yes</option>
94             <option>No</option>
95         </select>
96     </div>
97     <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
98     <button type="submit" class="btn btn-primary">Add Todo</button>
99 </form>
100 </div>
101 </div>
102 </div>
103 </div>
104 </section>
105 <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>
106 <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"></script>
107 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
108 </body>
109 </html>
```



增强 TodoController 以支持其他任务

让我们增强 *TodoController* 以支持渲染可用的 *todo*、更新、删除和创建新的 *Todo* 项。

我们将有以下端点： -

1. **/todo**: 显示应用程序中的所有**待办事项**。这是一个 HTTP GET 请求映射
2. **/todoNew**: 创建一个新的 *Todo*。这是一个 HTTP POST 请求映射
3. **/todoDelete/{id}**: 这个端点删除一个特定的 *Todo*。这也是一个 HTTP POST 请求映射
4. **/todoUpdate/{id}**: 此端点切换特定的 *Todo* 状态。这也是一个 HTTP POST 请求映射

在继续之前，让我们创建我们的域对象 *Todo*。这是一个带有 *id*、*todoItem* 和 *已完成* 字段的 JPA 实体： -

待办事项

请注意，`@NoArgsConstructor`、`@AllArgsConstructor`和`@Data`注释来自 Lombok。这些注释确保为我们生成所有 `getter/setter`、全参数构造函数和无参数构造函数。如果遇到困难，可以用`Todo`类中的经典 `getter/setter` 和构造函数替换这些。

```
1  package io.codefountain.spring.todo.domain;
2
3  import javax.persistence.Entity;
4  import javax.persistence.GeneratedValue;
5  import javax.persistence.GenerationType;
6  import javax.persistence.Id;
7
8  import lombok.AllArgsConstructor;
9  import lombok.Data;
10 import lombok.NoArgsConstructor;
11
12 @Entity
13 @NoArgsConstructor
14 @AllArgsConstructor
15 @Data
16 public class Todo {
17
18     @Id
19     @GeneratedValue(strategy = GenerationType.AUTO)
20     private long id;
21     private String todoItem;
22     private String completed;
23
24     public Todo(String todoItem, String completed) {
25         super();
26         this.todoItem = todoItem;
27         this.completed = completed;
28     }
29 }
```

Todo.java hosted with ❤ by GitHub

[view raw](#)

去做

TodoRepository.java

现在让我们创建一个`TodoRepository`接口来在我们的应用程序中添加存储库服务。

```
1  package io.codefountain.spring.todo.repository;
2
3  import org.springframework.data.repository.PagingAndSortingRepository;
4  import org.springframework.stereotype.Repository;
5
6  import java.util.List;
```

```
5
6  import io.codefountain.spring.todo.domain.Todo;
7
8  @Repository
9  public interface TodoRepository extends PagingAndSortingRepository<Todo, Long>{
10
11  }
```

TodoRepository.java hosted with ❤ by GitHub

[view raw](#)

TodoRepository

由于 *TodoRepository* 接口扩展了 *PagingAndSortingRepository*，它将支持基本的 CRUD 操作。

TodoController.java

是时候增强 *TodoController* 以添加我们在本节前面讨论过的 Rest 端点了。这就是 *TodoController* 的样子： -



```
1  package io.codefountain.spring.todo.controllers;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.stereotype.Controller;
5  import org.springframework.ui.Model;
6  import org.springframework.web.bind.annotation.GetMapping;
7  import org.springframework.web.bind.annotation.PathVariable;
8  import org.springframework.web.bind.annotation.PostMapping;
9  import org.springframework.web.bind.annotation.RequestParam;
10
11  import io.codefountain.spring.todo.domain.Todo;
12  import io.codefountain.spring.todo.repository.TodoRepository;
13
14  @Controller
15  public class TodoController {
16
17      @Autowired
18      TodoRepository todoRepository;
19
20      @GetMapping
21      public String index() {
22          return "index";
23      }
24
25      @GetMapping("/todos")
26      public String todos(Model model) {
27          model.addAttribute("todos", todoRepository.findAll());
28      }
29  }
```

```

28         return "todos";
29     }
30
31     @PostMapping("/todoNew")
32     public String add(@RequestParam String todoItem, @RequestParam String status, Model model) {
33         Todo todo = new Todo();
34         todo.setTodoItem(todoItem);
35         todo.setCompleted(status);
36         todoRepository.save(todo);
37         model.addAttribute("todos", todoRepository.findAll());
38         return "redirect:/todos";
39     }
40
41     @PostMapping("/todoDelete/{id}")
42     public String delete(@PathVariable long id, Model model) {
43         todoRepository.deleteById(id);
44         model.addAttribute("todos", todoRepository.findAll());
45         return "redirect:/todos";
46     }
47
48     @PostMapping("/todoUpdate/{id}")
49     public String update(@PathVariable long id, Model model) {
50         Todo todo = todoRepository.findById(id).get();
51         if("Yes".equals(todo.getCompleted())) {
52             todo.setCompleted("No");
53         }
54         else {
55             todo.setCompleted("Yes");
56         }
57         todoRepository.save(todo);
58         model.addAttribute("todos", todoRepository.findAll());
59         return "redirect:/todos";
60     }
61 }

```

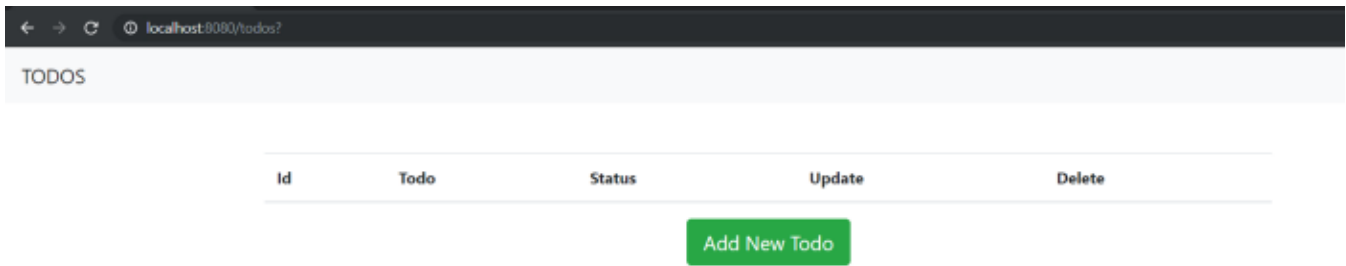


10/12/21, 11:34 PM

我们已将 *TodoRepository* 自动连接到 *TodoController* 以用于存储库服务，并提供所需端点的实现。

测试应用程序

现在是时候测试我们的应用程序了。让我们转到浏览器并点击刷新。出现应用程序主页后，单击“查看待办事项”按钮。我们应该看到以下屏幕： -



由于应用程序中没有可用的待办事项，我们的表格显示为空。我们将使用很少的 **Todo** 项来引导我们的应用程序。为此，让我们在 *SpringBootTestAppApplication* 类中进行以下更改。这是我们应用程序的起点。

```

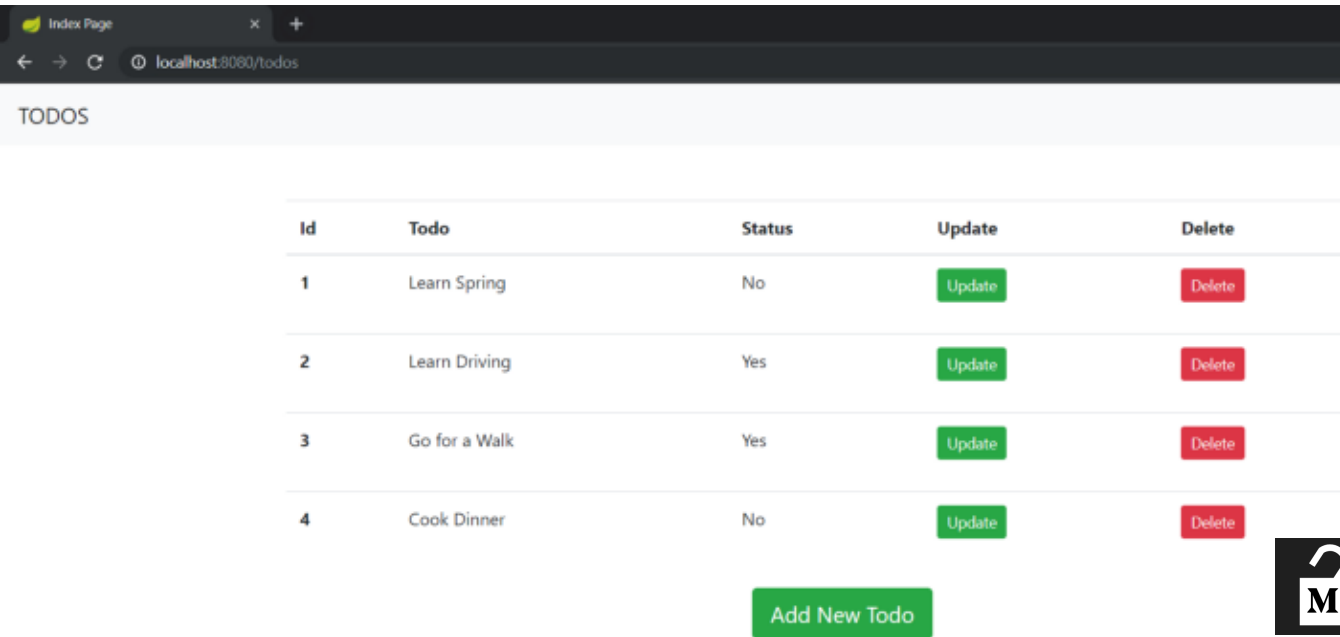
1  import java.util.Arrays;
2  import java.util.Collection;
3
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.boot.CommandLineRunner;
6  import org.springframework.boot.SpringApplication;
7  import org.springframework.boot.autoconfigure.SpringBootApplication;
8
9  import io.codefountain.spring.todo.domain.Todo;
10 import io.codefountain.spring.todo.repository.TodoRepository;
11
12 @SpringBootApplication
13 public class SpringBootTestAppApplication implements CommandLineRunner {
14
15     @Autowired
16     public TodoRepository todoRepository;
17
18     public static void main(String[] args) {
19         SpringApplication.run(SpringBootTestAppApplication.class, args);
20     }
21
22     @Override
23     public void run(String... args) throws Exception {
24
25         Collection<Todo> todos = Arrays.asList(new Todo("Learn Spring", "Yes"), new Todo("Learn Spring", "No"));
26         todos.forEach(todoRepository::save);
27
28     }
29 }

```



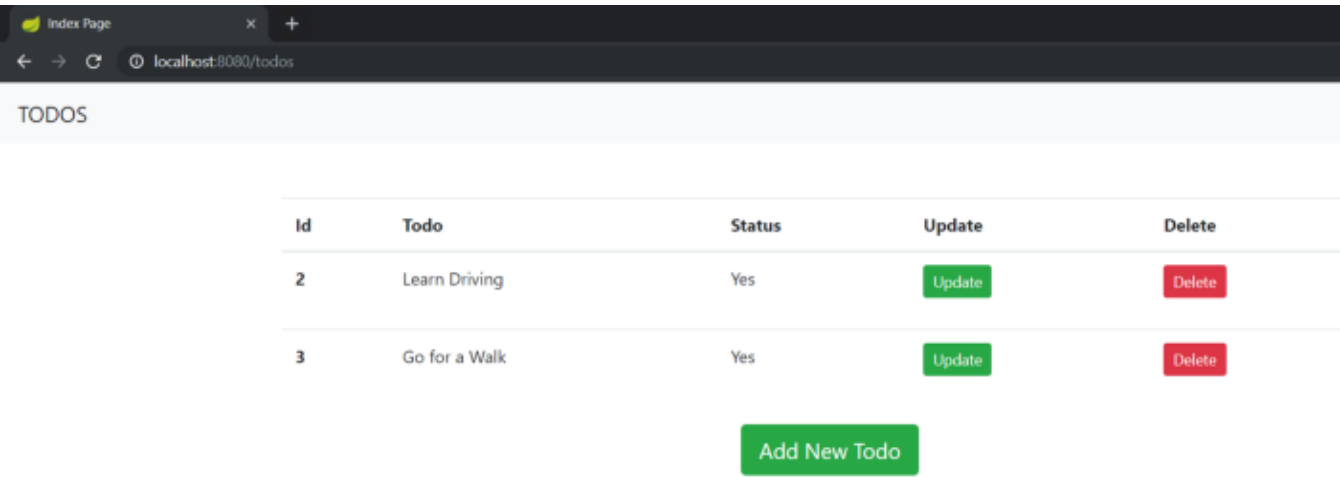
刷新浏览器 URL，我们应该会看到一个 Todos 列表。

更新待办事项：



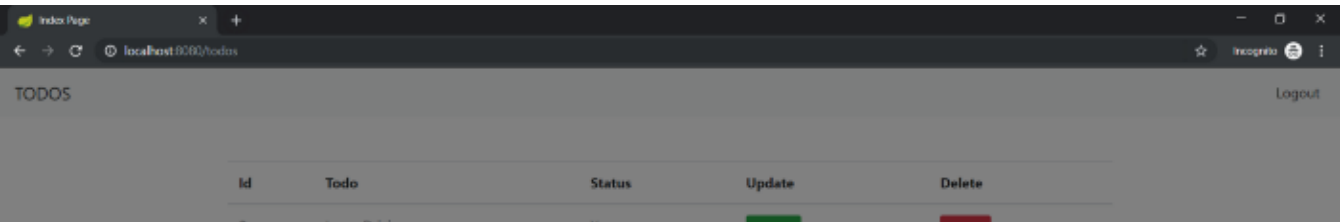
切换待办事项

删除待办事项：



删除的待办事项

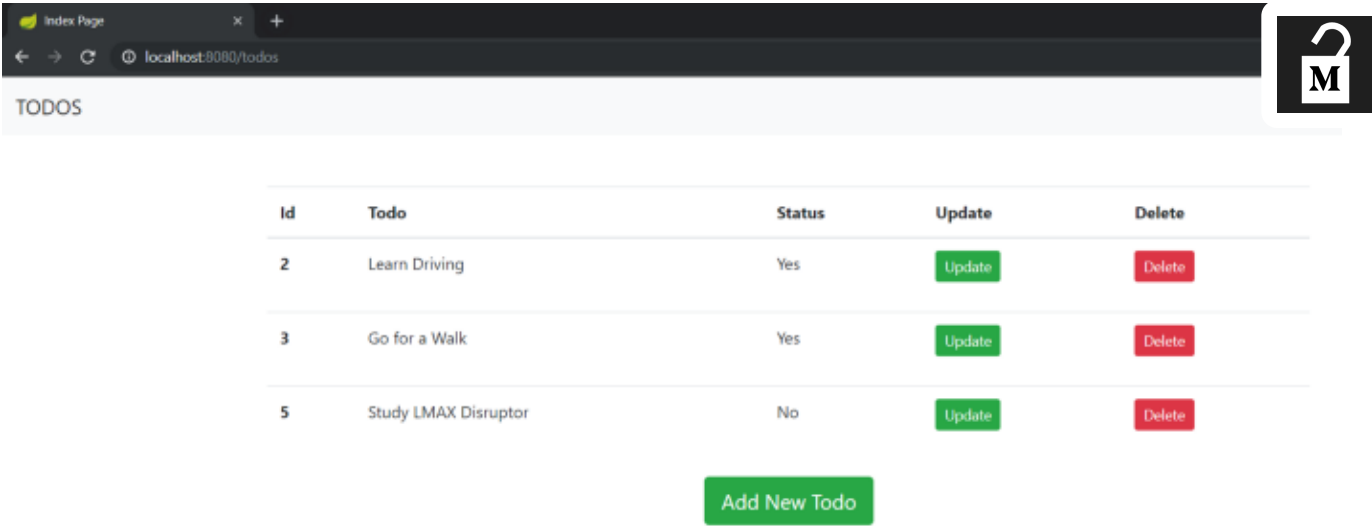
创建一个新的待办事项：





添加一个新的 Todo 项目

添加了新的待办事项: -



添加了待办事项

结论

在本文中，我们创建了一个具有准系统 CRUD 功能的全功能 TODO 管理应用程序。在下一篇文章中，我们将介绍 **Spring** 安全性并保护我们的应用程序访问。我已经在 [Github repository](#)上传了完整的源代码。

在[此处](#)关注本系列的下一部分。

每当 **Somnath Musib** 发布时，都会收到一封电子邮件。

订阅

电子邮件将发送至aa2797559967@gmail.com 。
不是你?

弹簧靴

百里香叶

引导程序

H2

春天

关于

写

帮助

合法的

Get the Medium app

