使用 MySQL 和 Spring Boot 进行会话管理

韩博孙

2021 年 4 月 29 日 麻省理工学院

本教程将讨论使用 MySQL 数据库作为会话管理的高级配置。

本教程将为使用 MySQL 数据库作为会话管理提供不同的解决方案。这是一种比在 Google 结果页面上复制的解决方案更实用的解决方案。

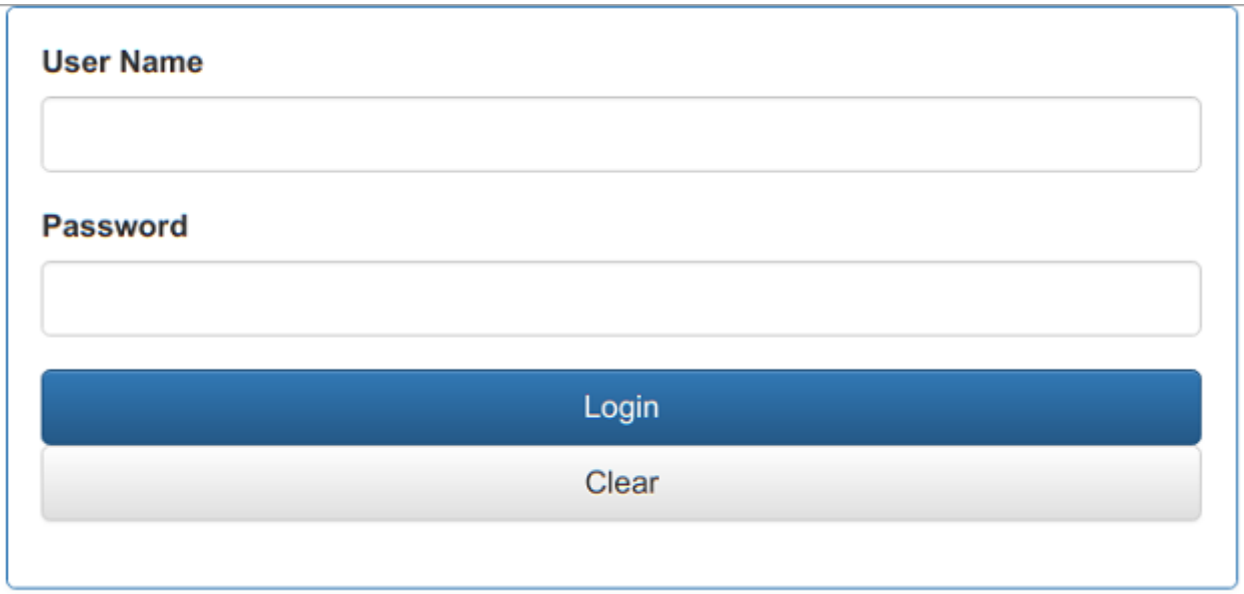
下载演示项目 - 403.1 KB

介绍

使用表单身份验证的安全 Web 应用程序需要会话。对于我自己的项目，我一直使用 Spring Boot 的默认会话。它要么将会话存储在内存中，要么存储在一些临时文件中。这是一个非常有限的解决方案。只有当我在单个服务器上托管应用程序时，它才能很好地工作。如果我需要使其更具可扩展性，比如我想要三个 Web 服务器来共享相同的会话数据，我需要使用比默认配置更好的东西。

根据我使用 ASP.NET 的经验，我知道有一种方法可以将会话数据存储在数据库中。使用 Spring Boot、Spring MVC，我想我可以做类似的事情。Spring Framework 与 ASP.NET 非常相似，当且仅当您能看清这些差异时。困难的部分是弄清楚配置是如何工作的。

我使用关键字“ **Spring Session MySQL** ”在网上搜索，列表显示相同的教程出现在多个搜索结果页面中。我将在本教程中复制相同的方法。但是，在仔细检查配置后，我决定不这样做。你要请客了。本教程将提供不同的解决方案。我认为这比在 Google 结果页面上复制的更实用。您可以决定是否是这种情况。



应用架构

在本教程中，我打算创建一个简单的示例应用程序来说明这一点。对我来说，会话数据应该保存在一个单独的数据库中，而不是应用程序特定的数据。您可以将两者混合在同一个数据库中，但它们有两个不同的用途，将它们放在同一个数据库中似乎是错误的。我们不知道有多少数据会进出会话，也不知道有多少数据会为应用程序进出。我们不知道这两组数据需要什么级别的安全或保护。关于这两组数据还有其他考虑。重点是，最好把它们分到两个不同的数据库中。

当我搜索如何使用 MySQL DB 进行会话存储时，出现的教程使用了一个数据源。一切都在配置文件 application.properties 中完成。我进行了更深入的研究，但找不到更好的方法来拥有两个数据源。更糟糕的是，多个教程使用了完全相同的代码集。我进行了更多研究，很幸运找到了解决方案。我在这里分享，让每个人都能受益。

如前所述，示例应用程序具有登录名，一旦用户成功登录，索引页面将加载应用程序数据库中表的所有行。用户成功登录后，会话已创建并可在会话数据库中查询。我正在重用上一教程中的相同数据库设计（Web 应用程序的 UTF-8 编码）. 要显示的数据将是 UUID 作为行 ID、标题（短字符串）和内容（长字符串）。这里的想法是，一旦用户可以登录，就成功创建会话。会话数据将包含有关用户何时登录以及用户何时将被强制注销的数据。当用户登录时，数据可以从不同的数据库加载，并显示在索引页面上。这非常简单，重点是我想演示如何完成配置以使其有效。

数据库设计

让我从数据库设计开始。对于此示例应用程序，有两个数据库，一个用于会话数据，另一个用于应用程序特定数据。应用数据的数据库设计非常简单，和上一个教程完全一样。更重要的一个是会话数据库。在应用程序执行之前创建数据库至关重要。当我们进入应用程序配置代码时，我将解释原因。

我们来看看会话数据库的设计。为了创建这个数据库，我必须：

1. 创建数据库模式
2. 创建数据库表

第一步是我们有一些自由。我可以定义数据库模式名称、可以访问它的用户以及用户的密码。因为数据是以字母数字字符存储的，所以我不需要对其使用 UTF-8 编码。这是我定义此数据库的 SQL 脚本：

SQL 复制代码

```
DROP DATABASE IF EXISTS `sessiondb`;
DROP USER IF EXISTS 'sndbuser'@'localhost';
CREATE DATABASE `sessiondb`;
CREATE USER 'sndbuser'@'localhost' IDENTIFIED BY '123$Test$321';
GRANT ALL PRIVILEGES ON `sessiondb`.* TO 'sndbuser'@'localhost';
FLUSH PRIVILEGES;
```

这是一个简单的脚本，前两行用于删除数据库模式和与该数据库关联的用户。然后接下来的两行将创建数据库模式和用户。第五行将数据库的所有权限授予用户。最后一行将刷新更改，以便它们立即生效。

接下来，我需要创建实际存储会话数据的表。这些表必须是具有特定列的特定名称。这是 Spring Framework 提供的东西。您可以从特定的 jar 文件中获取脚本。对于这个示例应用程序，我使用了 2.3.1 版本的 Spring Boot 和相关的 Spring 依赖项。所以我必须提取的脚本来自*spring-session-jdbc-2.3.1.RELEASE.jar*。如果您需要不同的版本，只需查找*spring-session-jdbc-<your target version>.RELEASE.jar*。你可以使用归档管理器来浏览这个jar的内部结构，你会发现很多数据库脚本，它们的名字是" *schema-<数据库类型>.sql* "。我需要*的那个*叫做" *schema-mysql.sql*"。我可以只提取文件并在 MySQL 中针对我的数据库方案执行以创建表。它是这样的：

SQL 复制代码

```
CREATE TABLE SPRING_SESSION (
    PRIMARY_ID CHAR(36) NOT NULL,
    SESSION_ID CHAR(36) NOT NULL,
    CREATION_TIME BIGINT NOT NULL,
    LAST_ACCESS_TIME BIGINT NOT NULL,
    MAX_INACTIVE_INTERVAL INT NOT NULL,
    EXPIRY_TIME BIGINT NOT NULL,
    PRINCIPAL_NAME VARCHAR(100),
    CONSTRAINT SPRING_SESSION_PK PRIMARY KEY (PRIMARY_ID)
) ENGINE=InnoDB ROW_FORMAT=DYNAMIC;

CREATE UNIQUE INDEX SPRING_SESSION_IX1 ON SPRING_SESSION (SESSION_ID);
```

```
CREATE INDEX SPRING_SESSION_IX2 ON SPRING_SESSION (EXPIRY_TIME);
CREATE INDEX SPRING_SESSION_IX3 ON SPRING_SESSION (PRINCIPAL_NAME);

CREATE TABLE SPRING_SESSION_ATTRIBUTES (
    SESSION_PRIMARY_ID CHAR(36) NOT NULL,
    ATTRIBUTE_NAME VARCHAR(200) NOT NULL,
    ATTRIBUTE_BYTES BLOB NOT NULL,
    CONSTRAINT SPRING_SESSION_ATTRIBUTES_PK PRIMARY KEY (SESSION_PRIMARY_ID, ATTRIBUTE_NAME),
    CONSTRAINT SPRING_SESSION_ATTRIBUTES_FK FOREIGN KEY (SESSION_PRIMARY_ID) _
                REFERENCES SPRING_SESSION(PRIMARY_ID) ON DELETE CASCADE
) ENGINE=InnoDB ROW_FORMAT=DYNAMIC;
```

为了创建这些表，我必须将以下内容添加到上述脚本的顶部：

复制代码

```
use `sessiondb`;
```

这是我的应用程序数据的数据库架构。正如我之前提到的，我重用了之前教程中的相同数据库：

SQL 复制代码

```
DROP DATABASE IF EXISTS `utf8testdb`;
DROP USER IF EXISTS 'utf8tdbuser'@'localhost';
CREATE DATABASE `utf8testdb`;
CREATE USER 'utf8tdbuser'@'localhost' IDENTIFIED BY '123$Test$321';
GRANT ALL PRIVILEGES ON `utf8testdb`.* TO 'utf8tdbuser'@'localhost';
FLUSH PRIVILEGES;
ALTER DATABASE `utf8testdb` CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

这是我的应用程序数据表：

SQL 复制代码

```
USE `utf8testdb`;

DROP TABLE IF EXISTS `testcontent`;

CREATE TABLE `testcontent` (
   `id` VARCHAR(34) NOT NULL PRIMARY KEY,
   `subject` VARCHAR(512) NOT NULL,
   `content` MEDIUMTEXT NOT NULL
);
```

下一部分将是应用程序的配置代码。我将解释为什么我必须首先创建会话数据库。

应用配置

在本节中，我将向您展示如何为启动配置应用程序，以便它可以正确使用会话和应用程序数据库。让我们从应用程序的主条目开始：

爪哇 复制代码

```
package org.hanbo.boot.app;

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);
   }
}
```

这是 Spring Boot 应用程序的标准主条目。接下来，我需要为表单身份验证设置安全配置。我从关于 Spring Security 的早期 ThymeLeaf 教程之一中获取了这一点。以下是此配置的外观：

爪哇 缩小▲ 复制代码

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

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.
                           builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.
                           EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.
                           WebSecurityConfigurerAdapter;
import org.springframework.security.web.access.AccessDeniedHandler;
import org.springframework.security.web.authentication.
                           SavedRequestAwareAuthenticationSuccessHandler;
import org.hanbo.boot.app.security.UserAuthenticationService;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true)
public class WebAppSecurityConfig extends WebSecurityConfigurerAdapter
{
   @Autowired
   private UserAuthenticationService authenticationProvider;

   @Autowired
   private AccessDeniedHandler accessDeniedHandler;

   @Override
   protected void configure(HttpSecurity http) throws Exception
   {
      http
      .authorizeRequests()
          .antMatchers("/assets/**", "/public/**").permitAll()
          .anyRequest().authenticated()
      .and()
      .formLogin()
          .loginPage("/login")
          .permitAll()
          .usernameParameter("username")
          .passwordParameter("userpass")
          .successHandler(new SavedRequestAwareAuthenticationSuccessHandler())
          .defaultSuccessUrl("/secure/index", true).failureUrl("/public/authFailed")
          .and()
      .logout().logoutSuccessUrl("/public/logout")
          .permitAll()
          .and()
      .exceptionHandling().accessDeniedHandler(accessDeniedHandler);
   }

   @Override
   protected void configure(AuthenticationManagerBuilder authMgrBuilder)
      throws Exception
   {
      authMgrBuilder.authenticationProvider(authenticationProvider);
   }
}
```

我可以通过多种方式演示会话生命周期，但使用 Spring Security 来实现它是最实用的。如果你能成功运行这个应用程序，然后用test用户登录，这意味着会话配置是正确的。您甚至可以查询会话数据库以查看其中存储的数据。

以下是我如何设置测试用户以访问安全索引页面：

爪哇 缩小▲ 复制代码

```
package org.hanbo.boot.app.security;

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

import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.stereotype.Service;

@Service
public class UserAuthenticationService
implements AuthenticationProvider
{
   @Override
   public Authentication authenticate(Authentication auth) throws AuthenticationException
   {
      Authentication retVal = null;
      List<GrantedAuthority> grantedAuths = new ArrayList<GrantedAuthority>();

      if (auth != null)
      {
         String name = auth.getName();
         String password = auth.getCredentials().toString();
         System.out.println("name: " + name);
         System.out.println("password: " + password);

         if (name.equals("user1") && password.equals("user12345"))
         {
            grantedAuths.add(new SimpleGrantedAuthority("ROLE_USER"));

            retVal = new UsernamePasswordAuthenticationToken(
               name, "", grantedAuths
            );
            System.out.println("grant User");
         }
      }
      else
      {
         System.out.println("invalid login");
         retVal = new UsernamePasswordAuthenticationToken(
            null, null, grantedAuths
         );
         System.out.println("bad Login");
      }

      return retVal;
   }

   @Override
   public boolean supports(Class<?> tokenType)
   {
      return tokenType.equals(UsernamePasswordAuthenticationToken.class);
   }
}
```

到目前为止，我还没有展示会话设置或数据库连接配置。设置数据源的配置是最重要的部分。它们可以放在一起，如下所示：

爪哇 缩小▲ 复制代码

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

import javax.sql.DataSource;

import org.apache.commons.dbcp2.BasicDataSource;
```

```
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.session.jdbc.config.annotation.SpringSessionDataSource;
import org.springframework.session.jdbc.config.annotation.web.http.EnableJdbcHttpSession;

@EnableJdbcHttpSession
@Configuration
public class DataAccessConfiguration
{
   @Value("${db.jdbc.driver}")
   private String dbJdbcDriver;

   @Value("${db.conn.string}")
   private String dbConnString;

   @Value("${db.access.username}")
   private String dbAccessUserName;

   @Value("${db.access.password}")
   private String dbAccessPassword;

   @Value("${db.access.validity.query}")
   private String dbAccessValityQuery;

   @Value("${db.session.conn.string}")
   private String dbSesnConnString;

   @Value("${db.session.access.username}")
   private String dbSesnAccessUserName;

   @Value("${db.session.access.password}")
   private String dbSesnAccessPassword;

   @Value("${db.session.access.validity.query}")
   private String dbSesnAccessValityQuery;

   @Bean
   public DataSource contentDataSource()
   {
      BasicDataSource dataSource = new BasicDataSource();

      dataSource.setDriverClassName(dbJdbcDriver);
      dataSource.setUrl(dbConnString);
      dataSource.setUsername(dbAccessUserName);
      dataSource.setPassword(dbAccessPassword);
      dataSource.setMaxIdle(4);
      dataSource.setMaxTotal(20);
      dataSource.setInitialSize(4);
      dataSource.setMaxWaitMillis(900000);
      dataSource.setTestOnBorrow(true);
      dataSource.setValidationQuery(dbAccessValityQuery);

      return dataSource;
   }

   @SpringSessionDataSource
   @Bean
   public DataSource sessionDataSource()
   {
      BasicDataSource dataSource = new BasicDataSource();

      dataSource.setDriverClassName(dbJdbcDriver);
      dataSource.setUrl(dbSesnConnString);
      dataSource.setUsername(dbSesnAccessUserName);
      dataSource.setPassword(dbSesnAccessPassword);
      dataSource.setMaxIdle(4);
```

```
        dataSource.setMaxTotal(20);
        dataSource.setInitialSize(4);
        dataSource.setMaxWaitMillis(900000);
        dataSource.setTestOnBorrow(true);
        dataSource.setValidationQuery(dbSesnAccessValityQuery);

        return dataSource;
    }

    @Bean
    public NamedParameterJdbcTemplate namedParameterJdbcTemplate()
    {
        NamedParameterJdbcTemplate retVal
            = new NamedParameterJdbcTemplate(contentDataSource());
         return retVal;
    }

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

对于这个类，我使用了两个注释，一个是@EnableJdbcHttpSession. 另一个是@Configuration。当这个类被加载时，它会在 Spring IOC 容器中注册为一个提供配置的类，它也会使应用程序能够使用数据库和 JDBC 来管理应用程序会话：

爪哇 复制代码

```
...
@EnableJdbcHttpSession
@Configuration
public class DataAccessConfiguration
{
...
}
```

我需要这个应用程序的两个数据库，我必须以某种方式提供两个不同的数据源（我将展示如何进一步定义数据源）。这就是我必须使用注释@EnableJdbcHttpSession而不是应用程序属性文件来配置会话管理的数据库使用的原因。这也是我必须手动设置会话数据库，而不是使用应用程序属性文件来处理数据库创建的原因。我告诉过你我会解释为什么我必须事先创建数据库。

如您所见，有很多私有字符串属性。这些是从应用程序属性文件注入的属性。它们代表两种不同的数据库连接：

爪哇 缩小▲ 复制代码

```
// application database connection properties
@Value("${db.jdbc.driver}")
private String dbJdbcDriver;

@Value("${db.conn.string}")
private String dbConnString;

@Value("${db.access.username}")
private String dbAccessUserName;

@Value("${db.access.password}")
private String dbAccessPassword;

@Value("${db.access.validity.query}")
private String dbAccessValityQuery;

// session database connection properties
@Value("${db.session.conn.string}")
private String dbSesnConnString;

@Value("${db.session.access.username}")
```

```
private String dbSesnAccessUserName;

@Value("${db.session.access.password}")
private String dbSesnAccessPassword;

@Value("${db.session.access.validity.query}")
private String dbSesnAccessValityQuery;
```

这是*application.properties*文件的样子，您可以看到这些值是如何注入到这些`private`属性中的：

复制代码

```
db.jdbc.driver=com.mysql.cj.jdbc.Driver
db.conn.string=jdbc:mysql://localhost:3306/utf8testdb?useUnicode=true&
useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&
serverTimezone=UTC&allowPublicKeyRetrieval=true&useSSL=false
db.access.username=utf8tdbuser
db.access.password=123$Test$321
db.access.validity.query=SELECT 1

db.session.conn.string=jdbc:mysql://localhost:3306/sessiondb?useUnicode=true&
useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC&
allowPublicKeyRetrieval=true&useSSL=false
db.session.access.username=sndbuser
db.session.access.password=123$Test$321
db.session.access.validity.query=SELECT 1
```

接下来，我要定义数据源。数据源对象包含连接信息，可用于创建到数据库的 SQL 连接。这就是我为应用程序数据库定义的方式：

爪哇 复制代码

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

   dataSource.setDriverClassName(dbJdbcDriver);
   dataSource.setUrl(dbConnString);
   dataSource.setUsername(dbAccessUserName);
   dataSource.setPassword(dbAccessPassword);
   dataSource.setMaxIdle(4);
   dataSource.setMaxTotal(20);
   dataSource.setInitialSize(4);
   dataSource.setMaxWaitMillis(900000);
   dataSource.setTestOnBorrow(true);
   dataSource.setValidationQuery(dbAccessValityQuery);

   return dataSource;
}
```

这是应用程序最重要的部分，为会话数据库定义另一个特殊数据源：

爪哇 复制代码

```
@SpringSessionDataSource
@Bean
public DataSource sessionDataSource()
{
   BasicDataSource dataSource = new BasicDataSource();

   dataSource.setDriverClassName(dbJdbcDriver);
   dataSource.setUrl(dbSesnConnString);
   dataSource.setUsername(dbSesnAccessUserName);
   dataSource.setPassword(dbSesnAccessPassword);
   dataSource.setMaxIdle(4);
   dataSource.setMaxTotal(20);
   dataSource.setInitialSize(4);
   dataSource.setMaxWaitMillis(900000);
```

```
   dataSource.setTestOnBorrow(true);
   dataSource.setValidationQuery(dbSesnAccessValityQuery);

   return dataSource;
}
```

你现在一定已经明白了。注释@SpringSessionDataSource设置它仅由会话管理用于连接到数据库。这就是我定义两个数据源的方式。我在 Spring Boot 官方文档站点中找到了这个。这是一个幸运的发现。再说一次，我几乎可以找到我需要的任何东西。一旦我弄清楚了这一点，所有的技术问题都解决了。最后两个方法是定义一个 JDBC 模板对象和一个事务管理器对象：

爪哇 复制代码

```
@Bean
public NamedParameterJdbcTemplate namedParameterJdbcTemplate()
{
   NamedParameterJdbcTemplate retVal
       = new NamedParameterJdbcTemplate(contentDataSource());
    return retVal;
}

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

至此，会话配置完成。我需要一个控制器和一个请求处理程序来演示 session 和 Spring Security 的使用：

爪哇 缩小▲ 复制代码

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

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

import org.hanbo.boot.app.models.PostDataModel;
import org.hanbo.boot.app.services.PostDataService;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class SecuredPageController
{
   private PostDataService _dataSvc;

   public SecuredPageController(
      PostDataService svc)
   {
      _dataSvc = svc;
   }

   @PreAuthorize("hasRole('USER')")
   @RequestMapping(value="/secure/index", method = RequestMethod.GET)
   public ModelAndView index1()
   {
      List<PostDataModel> allPosts = _dataSvc.getAllPostData();
      if (allPosts == null)
      {
         allPosts = new ArrayList<PostDataModel>();
      }
      System.out.println(allPosts.size());

      ModelAndView retVal = new ModelAndView();
```

```
        retVal.setViewName("indexPage");
        retVal.addObject("allPosts", allPosts);

        return retVal;
    }
}
```

这是一个非常简单的MVC控制器。此类中唯一的方法处理将显示安全索引页面的请求。它将使用注入的服务对象从应用程序数据库加载数据并将加载的数据列表返回到索引页面。请注意，该方法标有 Spring Security 注释 `@PreAuthorize("hasRole('USER')")`。这意味着具有“USER”角色的任何用户都可以访问此页面。

为了使上面的MVC控制器工作，我定义了一个`model`类和`service`类。我不会展示这`model`门课，因为它很简单。这是 的实现类`service`：

爪哇 缩小▲ 复制代码

```
package org.hanbo.boot.app.services;

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

import org.hanbo.boot.app.models.PostDataModel;
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.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class PostDataServiceImpl implements PostDataService
{
   private final String sql_insertPostData =
   "INSERT INTO `testcontent` (id, subject, content) VALUES (:id, :title, :content);";

   private final String sql_queryAllPosts =
           "SELECT id, subject, content FROM `testcontent` LIMIT 1000;";

   @Autowired
   private NamedParameterJdbcTemplate sqlDao;

   @Override
   @Transactional
   public String savePostData(PostDataModel dataToSave)
   {
      if (dataToSave != null)
      {
         String title = dataToSave.getTitle();
         if (title == null || title.isEmpty())
         {
            throw new RuntimeException("Title is NULL or empty");
         }

         String content = dataToSave.getContent();
         if (content == null || content.isEmpty())
         {
            throw new RuntimeException("Content is NULL or empty");
         }

         Map<String, Object> parameters = new HashMap<String, Object>();

         String postId = generateId();
         parameters.put("id", postId);
         parameters.put("title", dataToSave.getTitle());
         parameters.put("content", dataToSave.getContent());
```

```
            int updateCount = sqlDao.update(sql_insertPostData, parameters);
            if (updateCount > 0)
            {
               return postId;
            }
         }

         return "";
      }

      @Override
      public List<PostDataModel> getAllPostData()
      {
         List<PostDataModel> retVal = new ArrayList<PostDataModel>();

         retVal = sqlDao.query(sql_queryAllPosts,
               (MapSqlParameterSource)null,
               (rs) -> {
                  List<PostDataModel> foundObjs = new ArrayList<PostDataModel>();
                  if (rs != null)
                  {
                     while (rs.next())
                     {
                        PostDataModel postToAdd = new PostDataModel();
                        postToAdd.setPostId(rs.getString("id"));
                        postToAdd.setTitle(rs.getString("subject"));
                        postToAdd.setContent(rs.getString("content"));

                        foundObjs.add(postToAdd);
                     }
                  }

                  return foundObjs;
               });

         return retVal;
      }

      private static String generateId()
      {
         UUID uuid = UUID.randomUUID();
         String retVal = uuid.toString().replaceAll("-", "");

         return retVal;
      }
   }
```

我展示这一点的原因是它使用 JDBC 模板对象进行数据库访问。就是这个JDBC模板对象可以访问的应用数据库。当用户登录并可以使用此服务时，会话数据库将有代表此会话的行。

最后，我想展示 Maven POM 文件，然后来测试这个示例应用程序。

Maven POM 文件

我想在 Maven POM 文件中指出的唯一一件事是依赖项。我添加了两个新的依赖项，一个是`spring-session-jdbc`，另一个是`spring-boot-starter-data-jpa`. `spring-session-jdbc`是使用数据库进行会话管理所需的依赖项。我认为我不需要其他依赖项，您可以尝试删除它，然后运行该应用程序并查看它是否有效。这是 POM 文件依赖项的整个部分：

XML 缩小▲ 复制代码

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

```
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.session</groupId>
      <artifactId>spring-session-jdbc</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</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-tomcat</artifactId>
      <scope>provided</scope>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
   </dependency>
   <dependency>
      <groupId>org.thymeleaf.extras</groupId>
      <artifactId>thymeleaf-extras-springsecurity5</artifactId>
      <version>3.0.4.RELEASE</version>
   </dependency>
   <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-dbcp2</artifactId>
      <version>2.5.0</version>
   </dependency>
</dependencies>
```

如何测试示例应用程序

将源文件压缩为 zip 文件后，请将所有*.*sj*文件重命名为*.*js*文件。

然后，在项目的基本目录（POM 文件所在的位置），运行以下命令：

复制代码

```
mvn clean install
```

成功构建项目后，您可以使用以下命令运行应用程序：

复制代码

```
java -jar target/hanbo-spring-session-mysql-sample-1.0.0.jar
```

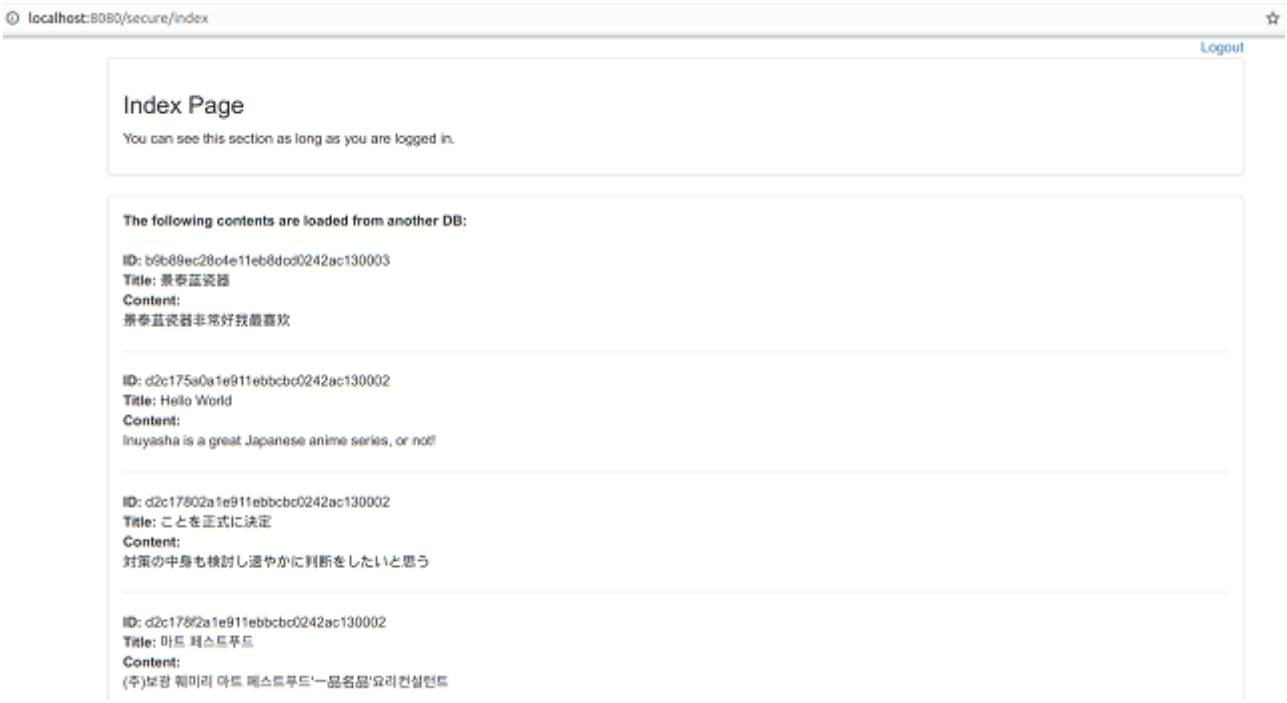
假设应用程序运行成功，您可以通过以下网址进入登录页面：

复制代码

```
http://localhost:8080
```

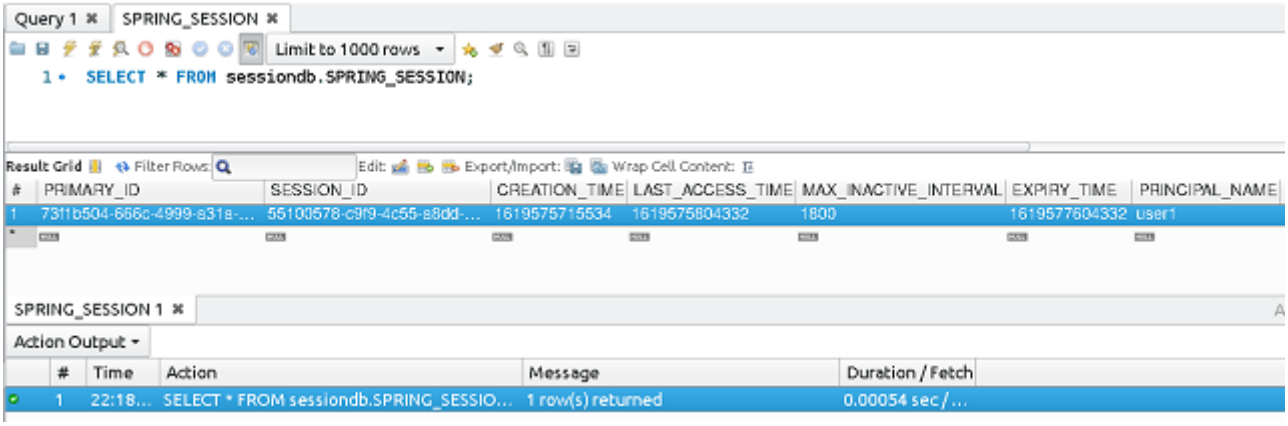
只有一个用户可以登录，用户名为“`user1`”，密码为“`user12345`”。如果您可以使用此用户登录，并看到索引页面显示，那么您就知道会话正在工作。如果应用程序配置有任何问题，您很可能会看到错误页面显示错误代码 500（内部服务器错误）。

如果您登录成功，并且已经为应用程序数据库执行了插入脚本，您将看到：

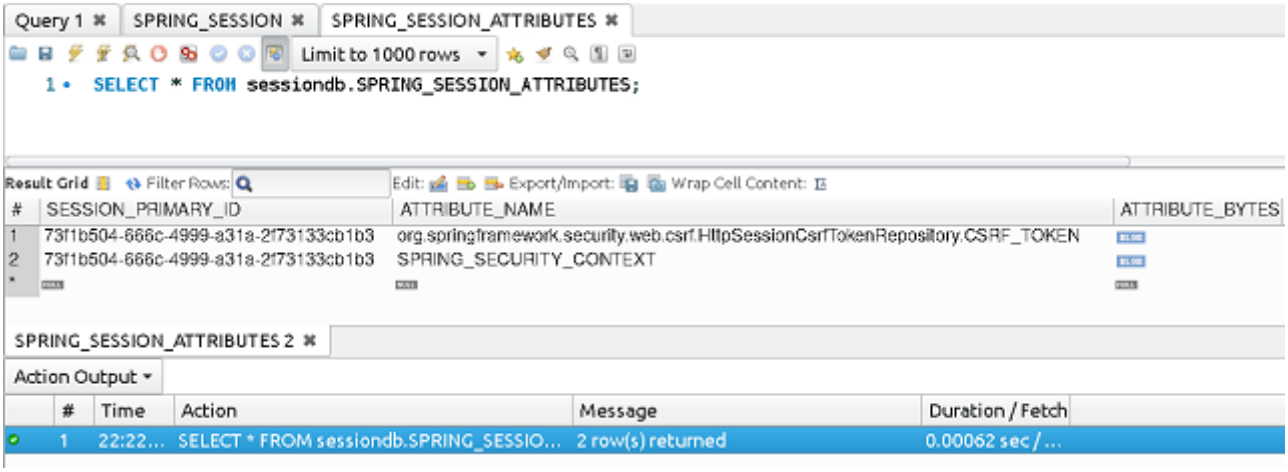


安全索引页面会显示应用数据库的帖子列表，所以请使用脚本“ *inserts.sql* ”（在项目基础文件夹下的子文件夹*DB*中）将这些数据行添加到应用数据库的唯一表中.

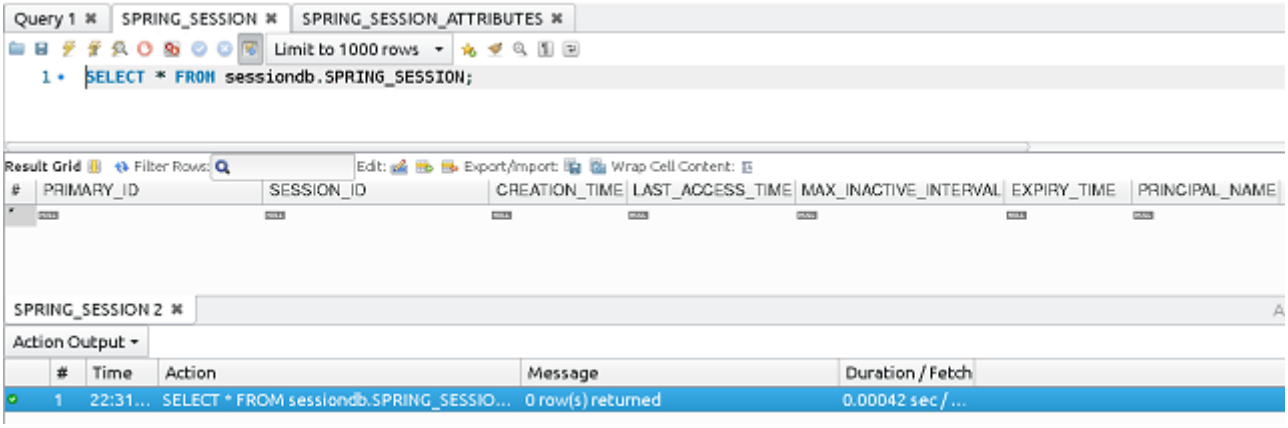
成功登录应用程序后，您可以检查会话数据库并查看会话数据：



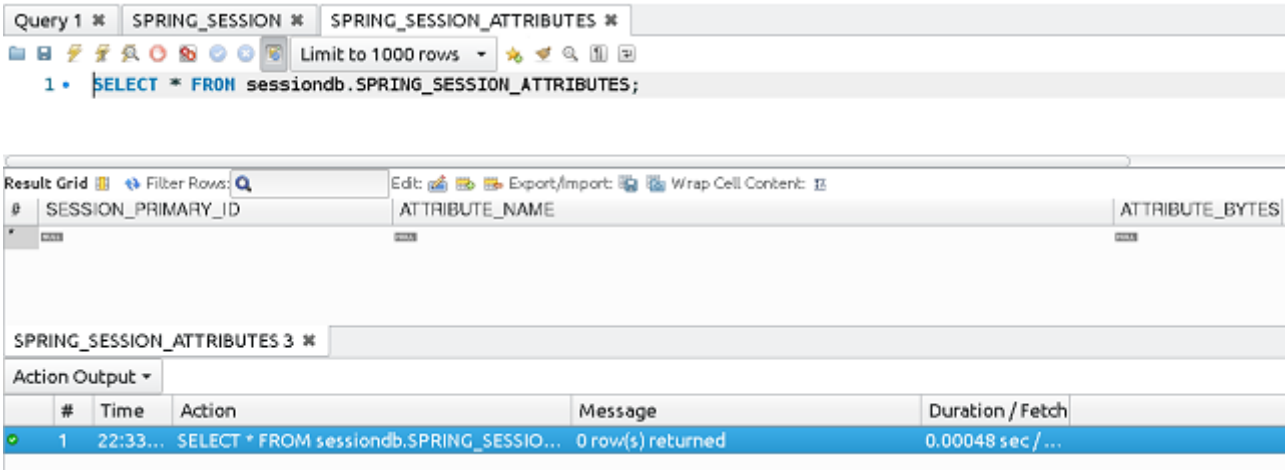
Spring Session 管理有两个数据表。上图显示了来自表“ SPRING_SESSION”的查询结果。这是来自第二个表“ SPRING_SESSION_ATTRIBUTES;”的查询结果：



对于这个示例应用程序，我提供了注销功能。当用户注销，查询会话数据表时，会看到：



和这个：



这是正确的！注销后，会话表将从关联的会话中清除。如果数据库有多个会话，则清除的会话将仅是用户注销的会话。

概括

这是一个相当复杂的教程。示例应用程序有很多活动部件。我必须使其工作如下：

1. 创建会话数据库和表。
2. 创建应用程序数据库。
3. 插入任何需要的应用程序数据。会话数据库最初应该是空的。
4. 添加 Spring Security 和 Spring JDBC 的配置（尤其是使用第二个数据源进行会话访问）。
5. 添加MVC控制器、服务类和数据访问类。在 MVC 控制器上，使用`@PreAuthorize`注释来强制用户登录。

关键是我可以先手动创建session数据库，然后使用两个数据源，一个专用于session访问。它所需要的只是一个注释`@SpringSessionDataSource`。其余的都是一样的。这又是一个看似棘手的技术问题，但总有一个简单的解决方案可用。

只想说最后一件事，我研究了使用 Google OpenID 授权进行身份验证，并决定不使用它。设置不是很困难，但是由于我必须使用基于 Google 的 API 并且不得不使用我自己的 Google 访问权限公开大量配置，因此安全风险太高。并且没有使用 Google OpenID 的功能需求。所以今年，我不会提供关于如何使用 Spring Security 配置 OpenID 的教程。

我希望你喜欢这个教程。我很高兴创造它。更多教程即将推出。敬请关注！

历史

- 2021 年 4 月27日 - 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 --