Spring Security：安全上下文持有者SecurityContextHolder介绍与Debug分析

原创 ITKaven 已于 2022-01-25 18:47:49 修改 阅读量5.1k 收藏 11 点赞数 1

分类专栏： Spring Security 文章标签： spring 安全 java

SecurityContextHolder

将给定的 SecurityContext 与当前执行线程相关联，此类提供了一系列委托给 SecurityContextHolderStrategy 实例的静态方法，此类的目的是提供一法来指定应该用于给定 JVM 的策略（持有 SecurityContext 的策略），这是 JVM 范围的设置，因为此类中的所有内容都是 static 修饰，方便调用。

要指定使用哪种策略，必须提供模式设置，模式设置是定义为 static final 字段的三个有效设置之一，或者是提供公共 无参构造方法 的 SecurityContextHolderStrategy 具体实现的完全限定类名（会通过反射进行创建）。有两种方法可以指定所需的 策略模式 ，第一种是通过键入 SYSTEM_PROPERTY 的系统属性来指定它，第二种是在使用类之前调用 setStrategyName(String) 进行设置。如果这两种方法都没有使用，则该类将默认 MODE_THREADLOCAL ，它向后兼容，具有较少的 JVM 不兼容性并且适用于服务器（而 MODE_GLOBAL 不适合服务器使用）。

SecurityContextHolder 类源码：

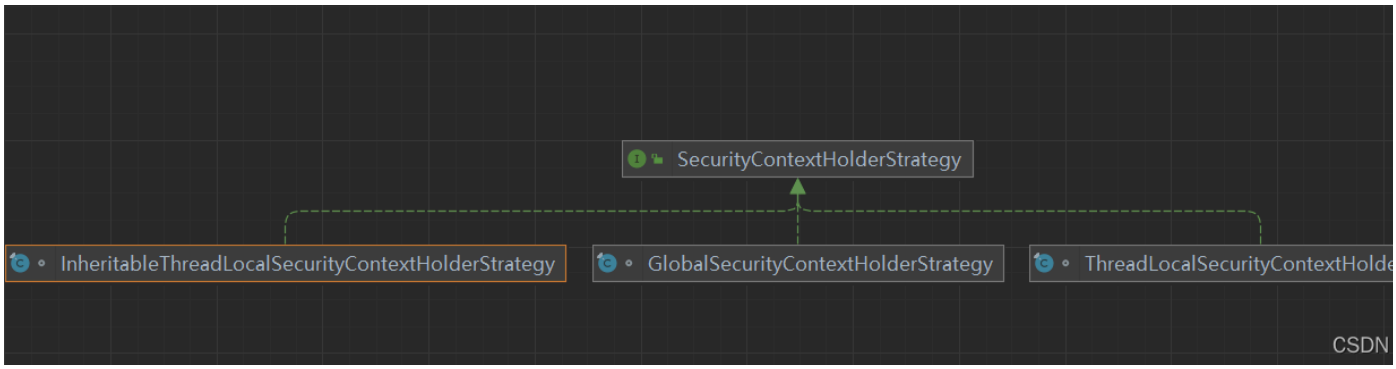
```
public class SecurityContextHolder {
    // 三种模式设置，代表三种策略
    public static final String MODE_THREADLOCAL = "MODE_THREADLOCAL";
    public static final String MODE_INHERITABLETHREADLOCAL = "MODE_INHERITABLETHREADLOCAL";
    public static final String MODE_GLOBAL = "MODE_GLOBAL";
    // 系统属性
    public static final String SYSTEM_PROPERTY = "spring.security.strategy";
    // 获取系统属性spring.security.strategy的值
    private static String strategyName = System.getProperty(SYSTEM_PROPERTY);
    // 安全上下文持有策略
    private static SecurityContextHolderStrategy strategy;
```

SecurityContextHolderStrategy

针对线程存储安全上下文信息的策略，首选策略由 SecurityContextHolder 加载。

```
public interface SecurityContextHolderStrategy {

    /**
     * 清除当前上下文
     */
    void clearContext();

    /**
     * 获取当前上下文
     */
    SecurityContext getContext();
```

SecurityContextHolderStrategy 接口有三个实现类，如下图所示，分别对应 SecurityContextHolder 类中的三种模式设置。



ThreadLocalSecurityContextHolderStrategy

基于 ThreadLocal 的 SecurityContextHolderStrategy 实现。

```
final class ThreadLocalSecurityContextHolderStrategy implements
        SecurityContextHolderStrategy {

    // 使用ThreadLocal持有SecurityContext实例
    private static final ThreadLocal<SecurityContext> contextHolder = new ThreadLocal<>();

    // 删除ThreadLocal持有的SecurityContext实例
    public void clearContext() {
        contextHolder.remove();
    }
    // 获取ThreadLocal持有的SecurityContext实例
```

InheritableThreadLocalSecurityContextHolderStrategy

基于 InheritableThreadLocal 的 SecurityContextHolderStrategy 实现。

InheritableThreadLocal 类扩展了 ThreadLocal 类，以提供从父线程到子线程的值继承：当创建子线程时，子线程接收父线程具有值的所有可继承线程的初始值。

```
final class InheritableThreadLocalSecurityContextHolderStrategy implements
        SecurityContextHolderStrategy {
    // 使用ThreadLocal持有SecurityContext实例，但该ThreadLocal是一个InheritableThreadLocal实例
    private static final ThreadLocal<SecurityContext> contextHolder = new InheritableThreadLocal<>();

    // 删除ThreadLocal持有的SecurityContext实例
    public void clearContext() {
        contextHolder.remove();
    }
    // 获取ThreadLocal持有的SecurityContext实例
    public SecurityContext getContext() {
```

GlobalSecurityContextHolderStrategy

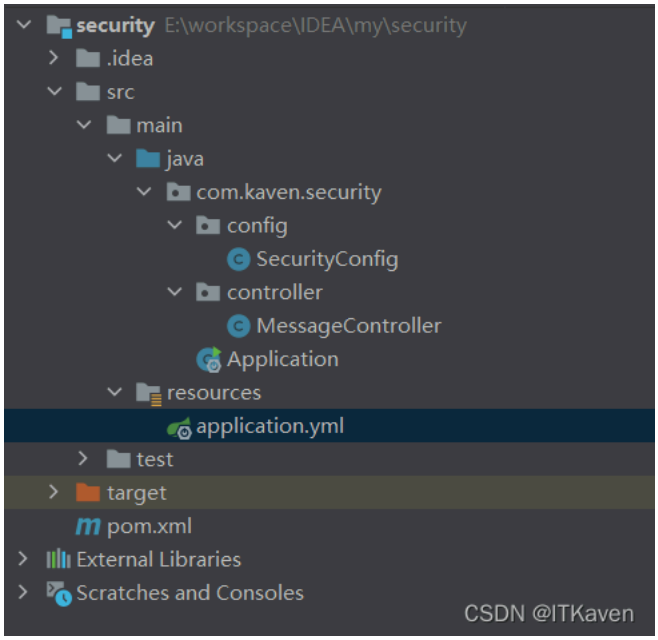
基于 static 字段的 SecurityContextHolderStrategy 实现，这意味着 JVM 中的所有实例共享相同的 SecurityContext 。

```
final class GlobalSecurityContextHolderStrategy implements SecurityContextHolderStrategy {
    // 将SecurityContext实例定义成一个静态属性
    private static SecurityContext contextHolder;

    // 清除上下文，即设置contextHolder属性为null
    public void clearContext() {
        contextHolder = null;
    }
    // 获取上下文
```

```
    public SecurityContext getContext() {
```

Debug分析

项目结构图：



pom.xml：

```xml
<?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>com.kaven</groupId>
    <artifactId>security</artifactId>
    <version>1.0-SNAPSHOT</version>

    <parent>
```

application.yml：

```yaml
spring:
  security:
    user:
      name: kaven
      password: itkaven
      roles: USER
logging:
  level:
    org:
      springframework:
        security: DEBUG
```

MessageController （定义接口）：

```
package com.kaven.security.controller;

import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MessageController {
    @GetMapping("/message")
    public String getMessage() {
        return "hello spring security";
    }
}
```

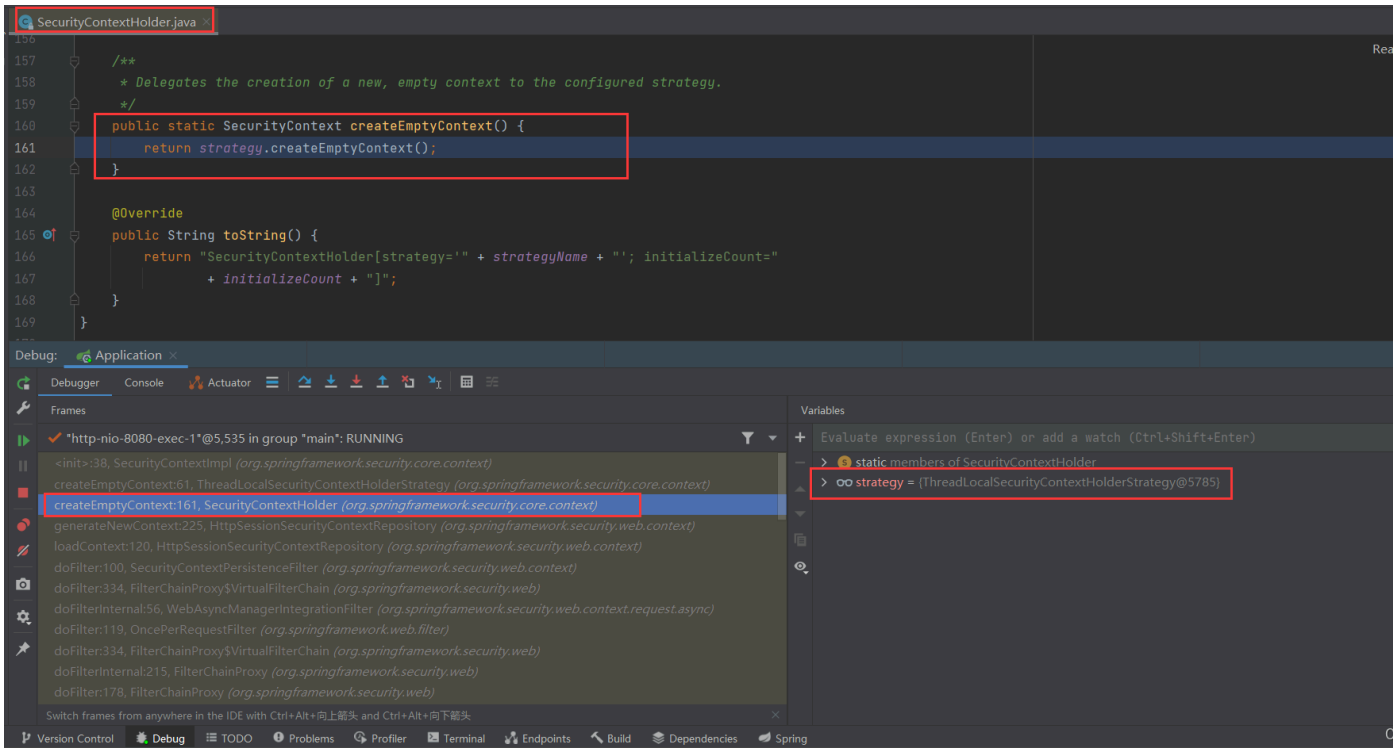
启动类：

```
package com.kaven.security;

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

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

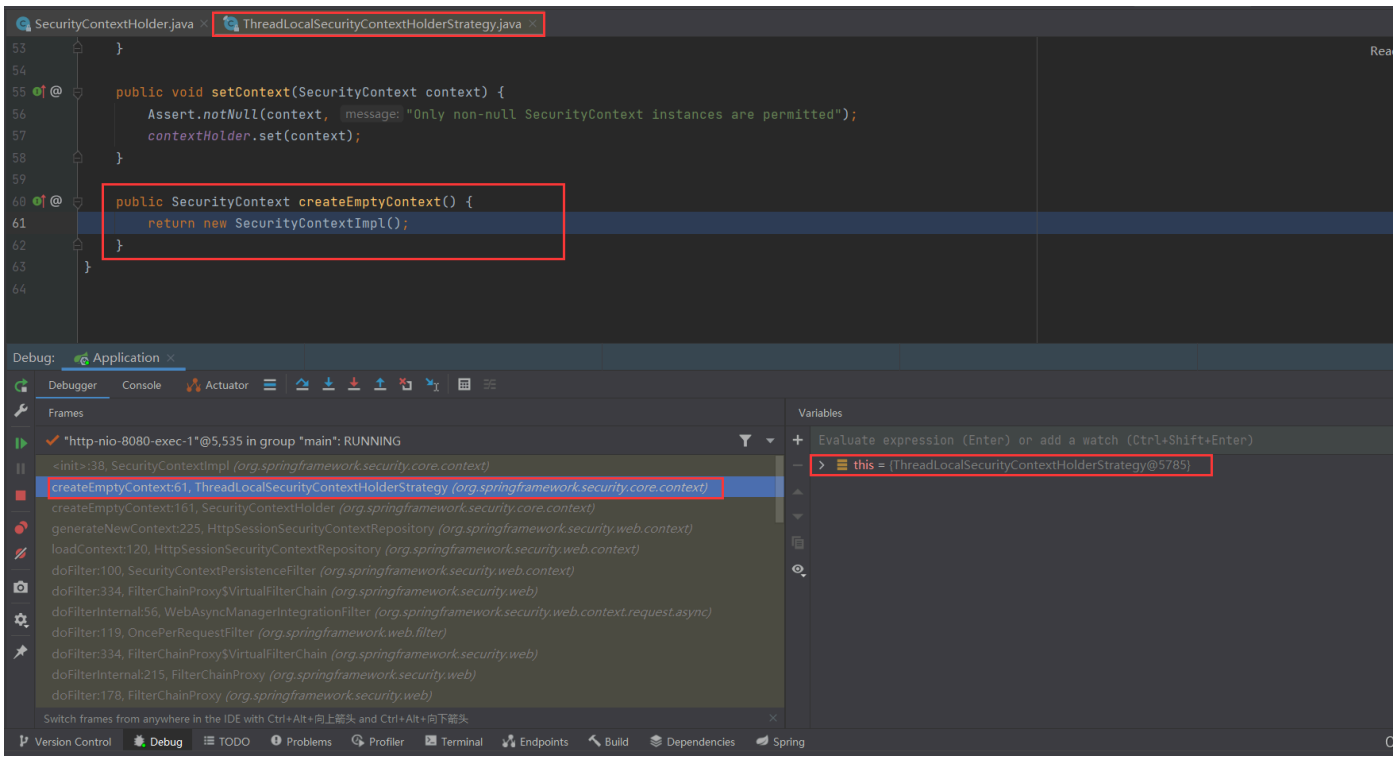
`SecurityConfig` （`Spring Security` 的配置类，不是必须的，因为有默认的配置）：

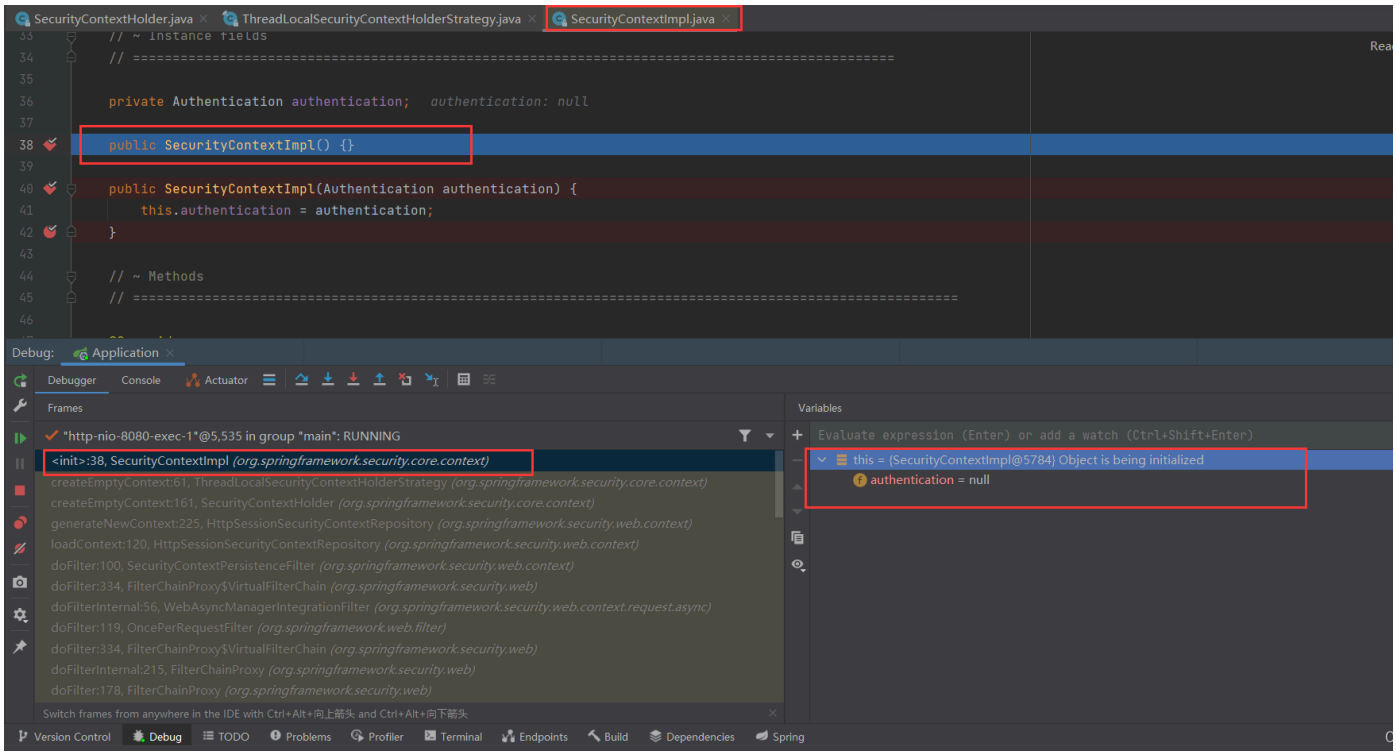
```
package com.kaven.security.config;

import org.springframework.security.config.Customizer;
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;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
```

`Debug` 方式启动应用，访问 `http://localhost:8080/message` ， `Spring Security` 会通过 `SecurityContextHolder` 创建空 `SecurityContextImpl` 实例（实 `authentication` 属性为空）。

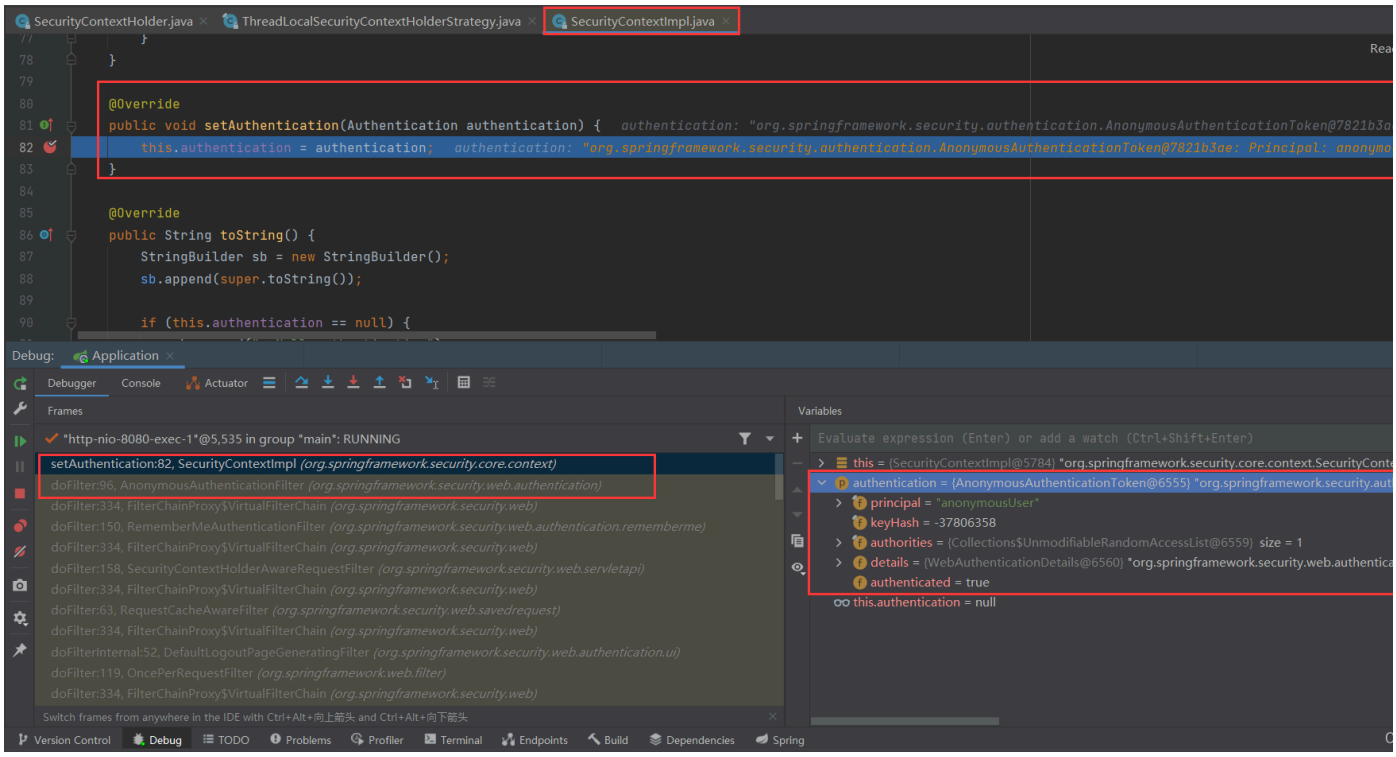


SecurityContextHolder 使用 ThreadLocalSecurityContextHolderStrategy 实例（默认策略）创建空 SecurityContextImpl 实例（实例的 authenticati
空）。

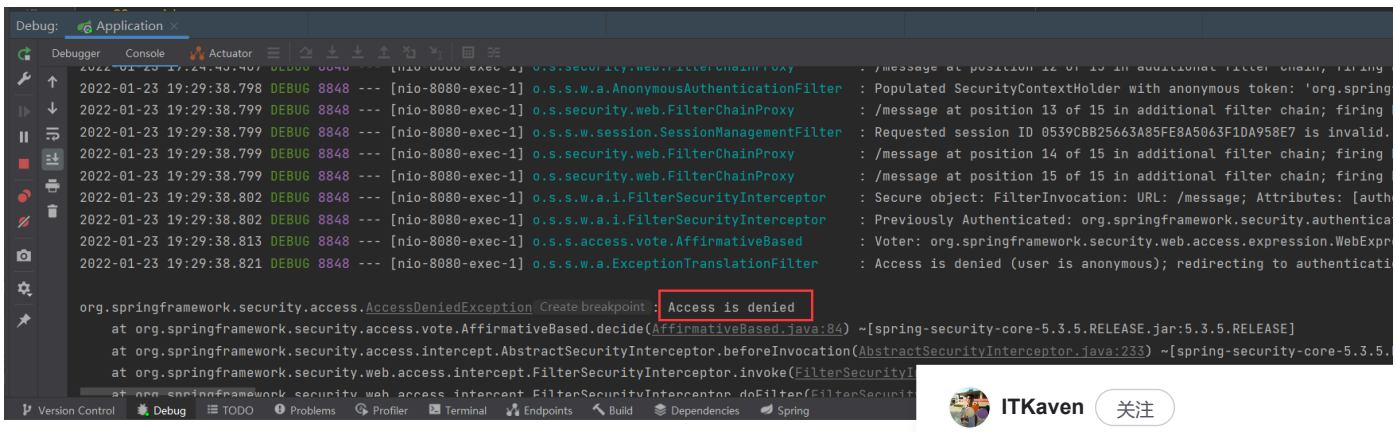




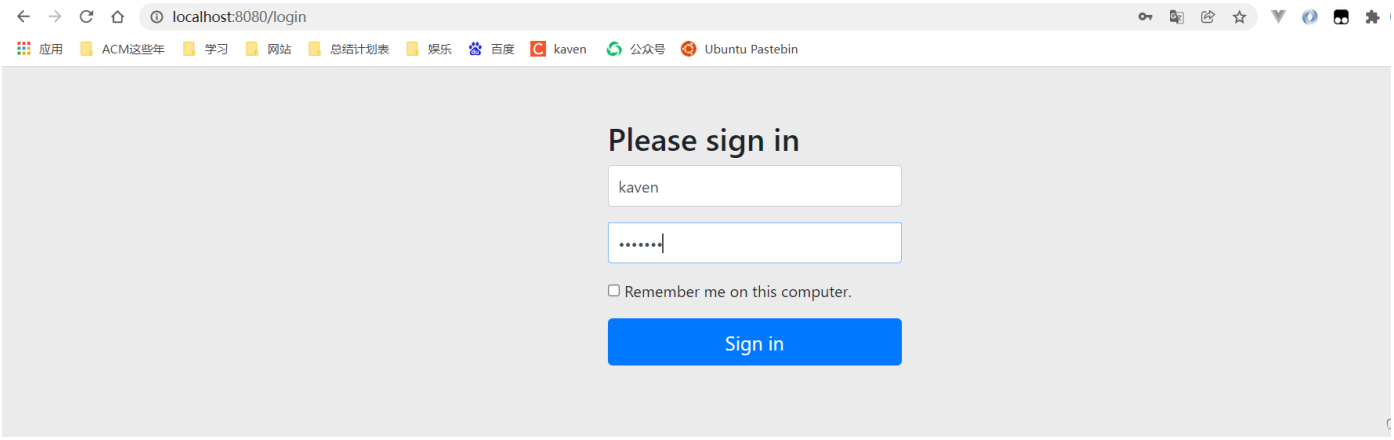
然后 AnonymousAuthenticationFilter 会处理请求（当前是匿名访问资源），该过滤器会设置该空 SecurityContextImpl 实例的 authentication 属性为 AnonymousAuthenticationToken 实例。



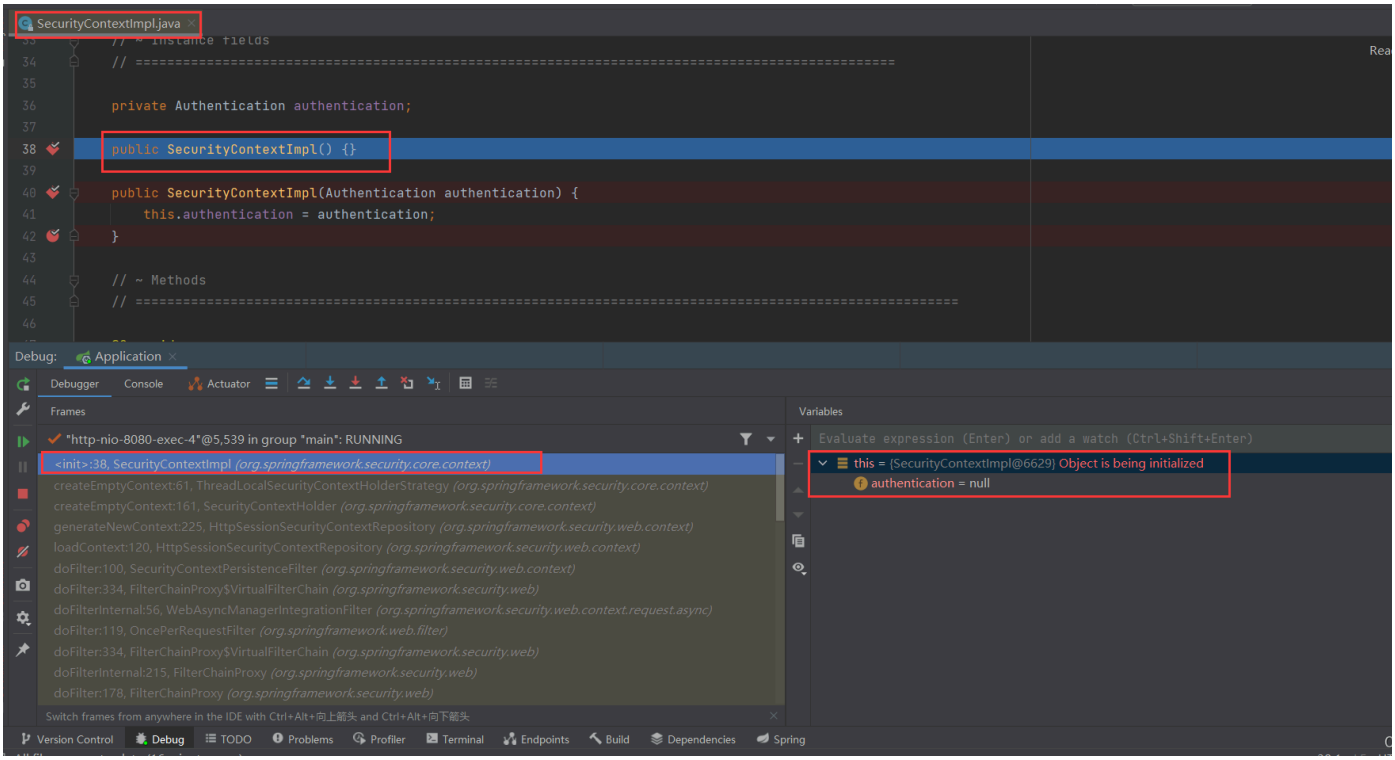
而通过身份验证（ authenticated 属性为 true ）的 AnonymousAuthenticationToken 实例（身份验证请求令牌），也是没有权限访问 /message 接口的，拒绝访问了。



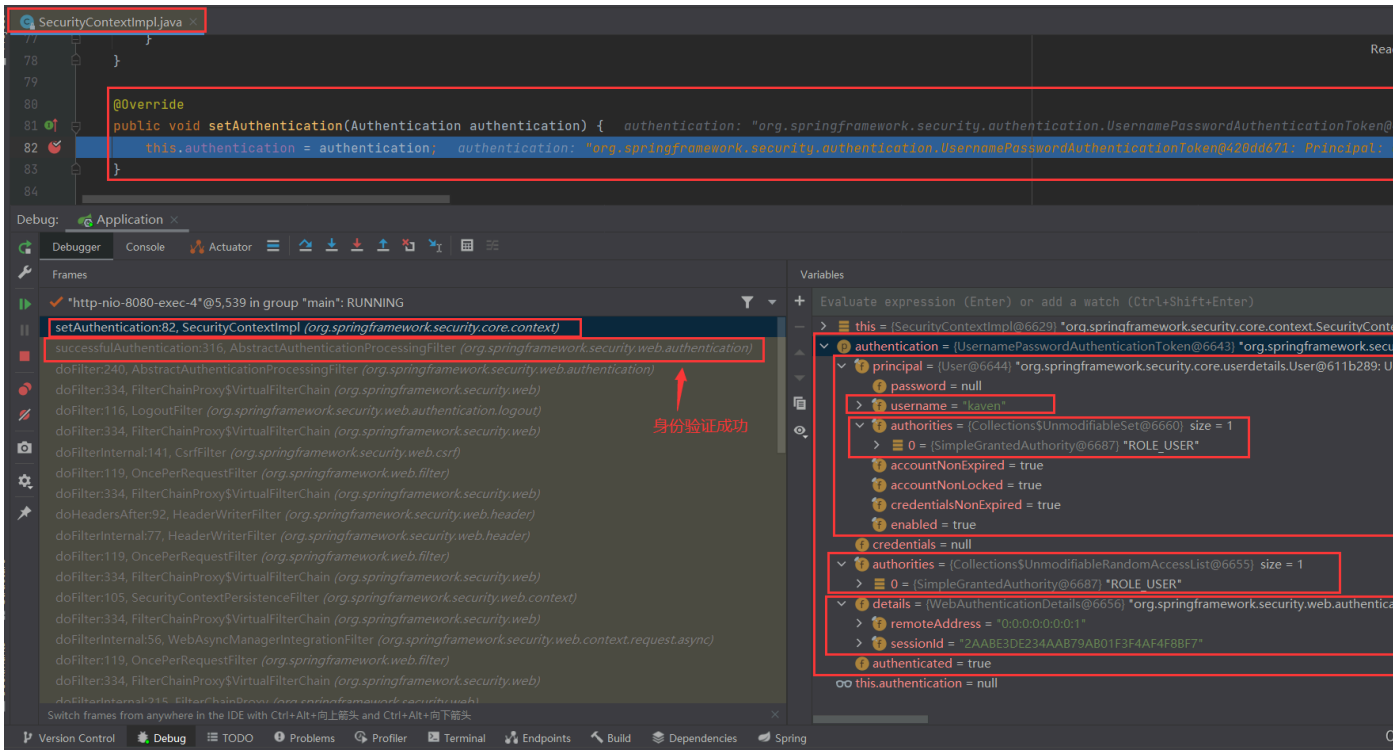
之后请求会被重定向到表单登陆页面，需要填写用户名和密码进行身份验证。



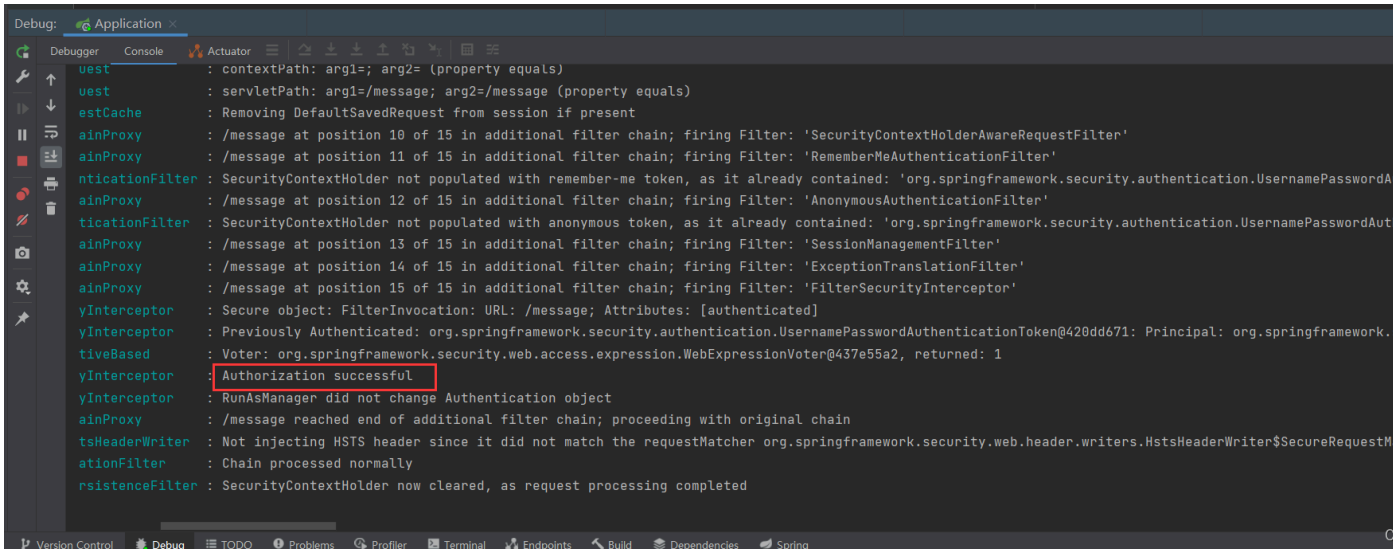
点击登陆后，又会创建一个空 `SecurityContextImpl` 实例（实例的 `authentication` 属性为空）。



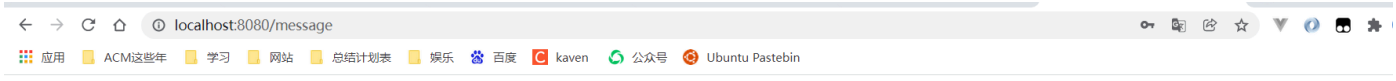
之后会设置该空 `SecurityContextImpl` 实例的 `authentication` 属性为 `UsernamePasswordAuthenticationToken` 实例，并且该 `UsernamePasswordAuthenticationToken` 实例的 `authenticated` 属性为 `true` （通过身份验证）。



控制台的日志也能说明身份验证成功了。



请求也成功访问到了 `/message` 接口。



安全上下文持有者 `SecurityContextHolder` 介绍与 `Debug` 分析就到这里，如果博主有说错的地方或者大家有不同的见解，欢迎大家评论补充。

以后每当有请求到来时,Spring Security 就会先从 Session 中取出用户登录数据,保存到SecurityContextHolder 中,方便在该请求的后续处理过程中使用,同时在请求结束时将

...登录用户数据的获取,超详细的源码分析_securitycontextholder...

SecurityContextHolder 依赖 SecurityContext,SecurityContext 封装了 Authentication,而Authentication 即是我们所指的认证后的用户数据信息。(从这关系以及上面的分析,

SpringSecurity （八）用户数据获取之SpringSecurityContextHolder深度剖析（下） 陈

在上一篇中我们大致的说明了从Security中获取登录数据的逻辑以及SecurityContextHolder保存数据的策略，最后也遗留下了一个问题。—SpringBoot中不同的请求都是由

核心组件之SecurityContextHolder dieqiuxie4160的

作用：保留系统当前的安全上下文细节，其中就包括当前使用系统的用户的信息。 上下文细节怎么表示？ 用SecurityContext对象来表示 每个用户都会有它的上下文，那

...身份验证的基本类/架构_securitycontextholder

// 定义一个Authentication -> 放入SecurityContext -> 放入 SecurityContextHolder 在SecurityContextHolder上设置完SecurityContext。Spring Security会使用此信息进行授

【SpringCloud】Spring Security核心组件_springsecurity组件详解-CS...

2.2 Spring Security是如何完成身份认证的 一、核心组件 1.1SecurityContextHolder类 1.1.1 概念 SecurityContextHolder看名知义,他是一个holder,用于持有的是安全上下文

spring-security SecurityContextHolder m13012606980的

翻译版本【spring-security 6.2.1】SecurityContextHolder

Spring Security(学习笔记)-SecurityContextHolder！ **最新发布** TONGZHOUGONGDU的

匿名访问，多线程获取用户信息。

SpringSecurity之SecurityContextHolder详解

简单来说,SecurityContextHolder是用来存储认证信息的,以方便保存用户的状态,供线程内所有方法使用SecurityContext,也就是用户信息。 SecurityContextHolder工作模式

Spring Security详解_springsecurity securitycontextholder-CSDN...

SecurityContextcontext=SecurityContextHolder.getContext(); Authenticationauthentication=context.getAuthentication(); Stringname=authentication.getName(); return"hel

SpringSecurity---SecurityContextHolder详解 池

巴拉巴拉： 之前在使用SpringSecurity的过程中并没有把很多东西理解透彻，找了很多学习资料也都只是是很浅显了告诉你怎么使用这个东西。现在只能自己回过头来研究

spring-security:弹簧安全

- Spring Security允许开发者根据需求自定义认证提供者、权限决策器、用户详情服务等。 9. **多层架构支持**： - 支持Web应用、RESTful API、WebSocket、JMS等不同

安全认证之SecurityContextHolder

1) SecurityContextHolder是SpringSecurity最基本的组件了,是用来存放SecurityContext的对象,默认是使用ThreadLocal实现的,这样就保证了本线程内所有的方法都可以获

...securitycontextholder to empty securitycontext

整体来说,SecurityContextPersistenceFilter主要做了两年事情。 (1)、当一个请求到来时,从HttpSession中获取SecurityContext并存入SecurityContextHolder中,这样在同一

Spring Security详细介绍及使用含完整代码（值得珍藏）

- **WebAsyncManagerIntegrationFilter**：集成Security上下文与Spring Web中处理异步请求映射的WebAsyncManager。 - **SecurityContextPersistenceFilter**：加载与

spring-security:这是演示弹簧安全

- `SecurityContextHolder`：存储当前的认证上下文，提供当前安全主体的信息。 - `AccessDecisionManager`：处理授权决策，根据投票器的结果决定是否允许访问。 5. `

Spring Security in Action

SecurityContext 是 Spring Security 中的一个重要概念，表示当前用户的安全上下文。SecurityContext 中包含用户的身份信息、权限信息等。SecurityContext 是通过 Sec

SecurityContextHolder weixin_57128596的

SecurityContextHolder是Spring Security的一个组件，其实它是一个工具类，只提供一些静态方法。这个工具类的目的是用来保存应用程序中当前使用人的安全上下文。(

springSecurity系列之 SecurityContextHolder与SecurityContext weixin_39802680的

SecurityContextHolder SecurityContextHolder 是spring security 的基础工具类。这是一个工具类，只提供一些静态方法。这个工具类的目的是用来保存应用程序中当前使

Spring Security 的基本组件 SecurityContextHolder **热门推荐** Details Inside Sp

Spring Security 中最基本的组件应该是SecurityContextHolder了。这是一个工具类，只提供一些静态方法。这个工具类的目的是用来保存应用程序中当前使用人的安全上下

[SpringSecurity5.6.2源码分析九]：SecurityContextHolder qq_41662584的

【代码】[SpringSecurity5.6.2源码分析九]：SecurityContextHolder。

SecurityContextHolder, SecurityContext and Authentication Objects Stainky的

SecurityContextHolder, SecurityContext and Authentication Objects 最重要并且最根本的object是SecurityContextHolder，我们将当前应用security上下文的所有数据保存在

springsecurity6.1存放上下文

Spring Security 6.1中的上下文存储在SecurityContextHolder类中。SecurityContextHolder是一个用于存储和访问当前用户的安全上下文的工具类。它提供了静态方法来获

ITKaven
码龄7年 暂无认证

542	2万+	105万+	227万+	
原创	周排名	总排名	访问	等级

2万+	1368	1992	937	3844
积分	粉丝	获赞	评论	收藏

私信 关注

搜博主文章

热门文章

分类专栏

最新评论

ElasticJob-Lite：作业监听器
一只烤鸭朝北走啊: 好久没看了，有点忘记了，我那会儿做的是定时任务在分布式环 ...

ElasticJob-Lite：作业监听器
AKKO111: 那感觉没必要用这个Distribute的listener了，直接用ElasticJobServiceLoa ...

Nginx：Nginx添加SSL实现HTTPS访问
风度翩翩609: 我这边配完 http+域名可以，但是https+域名就不行了

使用Vue和Spring Boot实现文件下载
一入程序无退路: 下载按钮点第二次就不太好使怎么回事

Docker容器中使用PING命令报错：bash:…
桂哥317: 干脆利落、实测有效

最新文章

MySQL: 备份 & 导入备份

RabbitMQ：Docker Compose部署RabbitMQ集群

VirtualBox：系统安装重启后又要重新安装

2022年 80篇	2021年 55篇
2020年 206篇	2019年 10篇
2018年 220篇	2017年 31篇

目录