

Spring Security (一) : 架构概述

2017/09/25 | 分类：基础技术 | 1 条评论 | 标签：SPRING SECURITY, 架构概述

分享到：         0

原文出处：[徐靖峰](#)

一直以来我都想写一写Spring Security系列的文章，但是整个Spring Security体系强大却又繁杂。陆陆续续从最开始的guides接触它，到项目中看了一些源码，到最近这个月为了写一写这个系列的文章，阅读了好几遍文档，最终打算尝试一下，写一个较为完整的系列文章。

较为简单或者体量较小的技术，完全可以参考着demo直接上手，但系统的学习一门技术则不然。以我的认知，一般的文档大致有两种风格：Architecture First和Code First。前者致力于让读者先了解整体的架构，方便我们对自己的认知有一个宏观的把控，而后者以特定的demo配合讲解，可以让读者在解决问题的过程中顺便掌握一门技术。关注过我博客或者公众号的朋友会发现，我之前介绍技术的文章，大多数是Code First，提出一个需求，介绍一个思路，解决一个问题，分析一下源码，大多如此。而学习一个体系的技术，我推荐Architecture First，正如本文标题所言，这篇文章是我Spring Security系列的第一篇，主要是根据Spring Security文档选择性翻译整理而成的一个架构概览，配合自己的一些注释方便大家理解。写作本系列文章时，参考版本为Spring Security 4.2.3.RELEASE。

1 核心组件

这一节主要介绍一些在Spring Security中常见且核心的Java类，它们之间的依赖，构建起了整个框

[java夜校](#) [Docker](#) [Nginx](#) [Redis](#)
[ActiveMQ](#) [Hessian](#) [FastDFS](#)
[MyCat](#) [Velocity](#) [SpringBoot](#)[更多前沿技术](#)[本周热门文章](#)[本月热门](#)[热门标签](#)

- 0 Spring 中获取 request 的几种方...
- 1 使用 Java 注解自动化处理对应关...
- 2 使用 losetup 帮你创建虚拟磁盘
- 3 Java 虚拟机16：Metaspace



架。想要理解整个架构，最起码得对这些类眼熟。

1.1 SecurityContextHolder

SecurityContextHolder用于存储安全上下文（security context）的信息。当前操作的用户是谁，该用户是否已经被认证，他拥有哪些角色权限...这些都被保存在SecurityContextHolder中。

SecurityContextHolder默认使用ThreadLocal 策略来存储认证信息。看到ThreadLocal 也就意味着，这是一种与线程绑定的策略。Spring Security在用户登录时自动绑定认证信息到当前线程，在用户退出时，自动清除当前线程的认证信息。但这一切的前提，是你在web场景下使用Spring Security，而如果是Swing界面，Spring也提供了支持，SecurityContextHolder的策略则需要被替换，鉴于我的初衷是基于web来介绍Spring Security，所以这里以及后续，非web的相关内容都一笔带过。

获取当前用户的信息

因为身份信息是与线程绑定的，所以可以在程序的任何地方使用静态方法获取用户信息。一个典型的获取当前登录用户的姓名的例子如下所示：

```
1 Object principal = SecurityContextHolder.getContext().getAuthentication().getPrinci
2 if (principal instanceof UserDetails) {
3     String username = ((UserDetails)principal).getUsername();
4 } else {
5     String username = principal.toString();
6 }
```

getAuthentication()返回了认证信息，再次getPrincipal()返回了身份信息，UserDetails便是Spring对身份信息封装的一个接口。Authentication和UserDetails的介绍在下面的小节具体讲解，本节重要的内容是介绍SecurityContextHolder这个容器。

1.2 Authentication

先看看这个接口的源码长什么样：

最新评论



Re: [Spring4 + Spring MVC + M...](#)

不建议这么整合，配置的东西太多了，可以用springboot，那样基本无缝整合，不需要啥配置的！
www.wuliaokankan.cn



Re: [深入剖析Java中的装箱和拆箱](#)

解释的很好，不过图挂了。

小木



Re: [推荐系统杂谈](#)

厉害厉害好文章！！赞一个

hznull



Re: [从 MVC 到前后端分离](#)

很不错！非常感谢！思路十分清晰，介绍很通俗易懂！
微笑感染嘴角



Re: [高并发Java（7）：并发设计模式](#)

看了受益颇多

个人



Re: [使用FastBootWeixin框架快速...](#)

好文章，不错！

www.wuliaokankan.cn



Re: [从 Spring Cloud 看一个微...](#)

用maven搭建springboot微服务框架杠杠的！以前搭个整合框架都麻烦的要命，现在几分钟搞定！
www.wuliaokankan.cn



Re: [Java代码优化](#)

对资源的close()建议分开操作 //释放资源不应该在finally里面吗
啊



```

1 package org.springframework.security.core; // <1>
2 public interface Authentication extends Principal, Serializable { // <1>
3     Collection<? extends GrantedAuthority> getAuthorities(); // <2>
4     Object getCredentials(); // <2>
5     Object getDetails(); // <2>
6     Object getPrincipal(); // <2>
7     boolean isAuthenticated(); // <2>
8     void setAuthenticated(boolean var1) throws IllegalArgumentException;
9 }

```

<1> Authentication是spring security包中的接口，直接继承自Principal类，而Principal是位于java.security包中的。可以见得，Authentication在spring security中是最高级别的身份/认证的抽象。

<2> 由这个顶级接口，我们可以得到用户拥有的权限信息列表，密码，用户细节信息，用户身份信息，认证信息。

还记得1.1节中，authentication.getPrincipal()返回了一个Object，我们将Principal强转成了Spring Security中最常用的UserDetails，这在Spring Security中非常常见，接口返回Object，使用instanceof判断类型，强转成对应的具体实现类。接口详细解读如下：

- getAuthorities(), 权限信息列表，默认是GrantedAuthority接口的一些实现类，通常是代表权限信息的一系列字符串。
- getCredentials(), 密码信息，用户输入的密码字符串，在认证过后通常会被移除，用于保障安全。
- getDetails(), 细节信息，web应用中的实现接口通常为WebAuthenticationDetails，它记录了访问者的ip地址和sessionId的值。
- getPrincipal(), 敲黑板！！！最重要的身份信息，大部分情况下返回的是UserDetails接口的实现类，也是框架中的常用接口之一。UserDetails接口将会在下的小节重点介绍。

Spring Security是如何完成身份认证的？

1 用户名和密码被过滤器获取到，封装成Authentication,通常情况下是UsernamePasswordAuthenticationToken这个实现类。

2 AuthenticationManager 身份管理器负责验证这个Authentication

3 认证成功后，AuthenticationManager身份管理器返回一个被填充满了信息的（包括上面提到的权限信息，身份信息，细节信息，但密码通常会被移除）Authentication实例。

4 SecurityContextHolder安全上下文容器将第3步填充了信息的Authentication，通过SecurityContextHolder.getContext().setAuthentication(...)方法，设置到其中。

这是一个抽象的认证流程，而整个过程中，如果不纠结于细节，其实只剩下一个 `AuthenticationManager` 是我们没有接触过的了，这个身份管理器我们在后面的小节介绍。将上述的流程转换成代码，便是如下的流程：

```
1 public class AuthenticationExample {
2     private static AuthenticationManager am = new SampleAuthenticationManager();
3     public static void main(String[] args) throws Exception {
4         BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
5         while(true) {
6             System.out.println("Please enter your username:");
7             String name = in.readLine();
8             System.out.println("Please enter your password:");
9             String password = in.readLine();
10            try {
11                Authentication request = new UsernamePasswordAuthenticationToken(name, pas
12                Authentication result = am.authenticate(request);
13                SecurityContextHolder.getContext().setAuthentication(result);
14                break;
15            } catch(AuthenticationException e) {
16                System.out.println("Authentication failed: " + e.getMessage());
17            }
18        }
19        System.out.println("Successfully authenticated. Security context contains: " +
20            SecurityContextHolder.getContext().getAuthentication());
21    }
22 }
23 class SampleAuthenticationManager implements AuthenticationManager {
24     static final List<GrantedAuthority> AUTHORITIES = new ArrayList<GrantedAuthority>()
25     static {
26         AUTHORITIES.add(new SimpleGrantedAuthority("ROLE_USER"));
27     }
28     public Authentication authenticate(Authentication auth) throws AuthenticationExcep
29         if (auth.getName().equals(auth.getCredentials())) {
30             return new UsernamePasswordAuthenticationToken(auth.getName(),
31                 auth.getCredentials(), AUTHORITIES);
32         }
33         throw new BadCredentialsException("Bad Credentials");
34     }
35 }
```

注意：上述这段代码只是为了让大家了解Spring Security的工作流程而写的，不是什么源码。在实际使用中，整个流程会变得更加的复杂，但是基本思想，和上述代码如出一辙。

1.3 AuthenticationManager

初次接触Spring Security的朋友相信会被AuthenticationManager，ProviderManager，AuthenticationProvider ...这么多相似的Spring认证类搞得晕头转向，但只要稍微梳理一下就可以理解清楚它们的联系和设计者的用意。AuthenticationManager（接口）是认证相关的核心接口，也

是发起认证的出发点，因为在实际需求中，我们可能会允许用户使用用户名+密码登录，同时允许用户使用邮箱+密码，手机号码+密码登录，甚至，可能允许用户使用指纹登录（还有这样的操作？没想到吧），所以说AuthenticationManager一般不直接认证，AuthenticationManager接口的常用实现类ProviderManager 内部会维护一个List<AuthenticationProvider>列表，存放多种认证方式，实际上这是委托者模式的应用（Delegate）。也就是说，核心的认证入口始终只有一个：

AuthenticationManager，不同的认证方式：用户名+密码

（UsernamePasswordAuthenticationToken），邮箱+密码，手机号码+密码登录则对应了三个AuthenticationProvider。这样一来四不四就好理解多了？熟悉shiro的朋友可以把AuthenticationProvider理解成Realm。在默认策略下，只需要通过一个AuthenticationProvider的认证，即可被认为是登录成功。

只保留了关键认证部分的ProviderManager源码：

```

1 public class ProviderManager implements AuthenticationManager, MessageSourceAware,
2     InitializingBean {
3     // 维护一个AuthenticationProvider列表
4     private List<AuthenticationProvider> providers = Collections.emptyList();
5
6     public Authentication authenticate(Authentication authentication)
7         throws AuthenticationException {
8         Class<? extends Authentication> toTest = authentication.getClass();
9         AuthenticationException lastException = null;
10        Authentication result = null;
11        // 依次认证
12        for (AuthenticationProvider provider : getProviders()) {
13            if (!provider.supports(toTest)) {
14                continue;
15            }
16            try {
17                result = provider.authenticate(authentication);
18                if (result != null) {
19                    copyDetails(authentication, result);
20                    break;
21                }
22            }
23            ...
24            catch (AuthenticationException e) {
25                lastException = e;
26            }
27        }
28        // 如果有Authentication信息, 则直接返回
29        if (result != null) {
30            if (eraseCredentialsAfterAuthentication
31                && (result instanceof CredentialsContainer)) {
32                // 移除密码
33                ((CredentialsContainer) result).eraseCredentials();
34            }
35            // 发布登录成功事件
36            eventPublisher.publishAuthenticationSuccess(result);
37            return result;
38        }
39        ...
40        // 执行到此, 说明没有认证成功, 包装异常信息
41        if (lastException == null) {
42            lastException = new ProviderNotFoundException(messages.getMessage(
43                "ProviderManager.providerNotFound",
44                new Object[] { toTest.getName() },
45                "No AuthenticationProvider found for {0}"));
46        }
47        prepareException(lastException, authentication);
48        throw lastException;
49    }
50 }

```

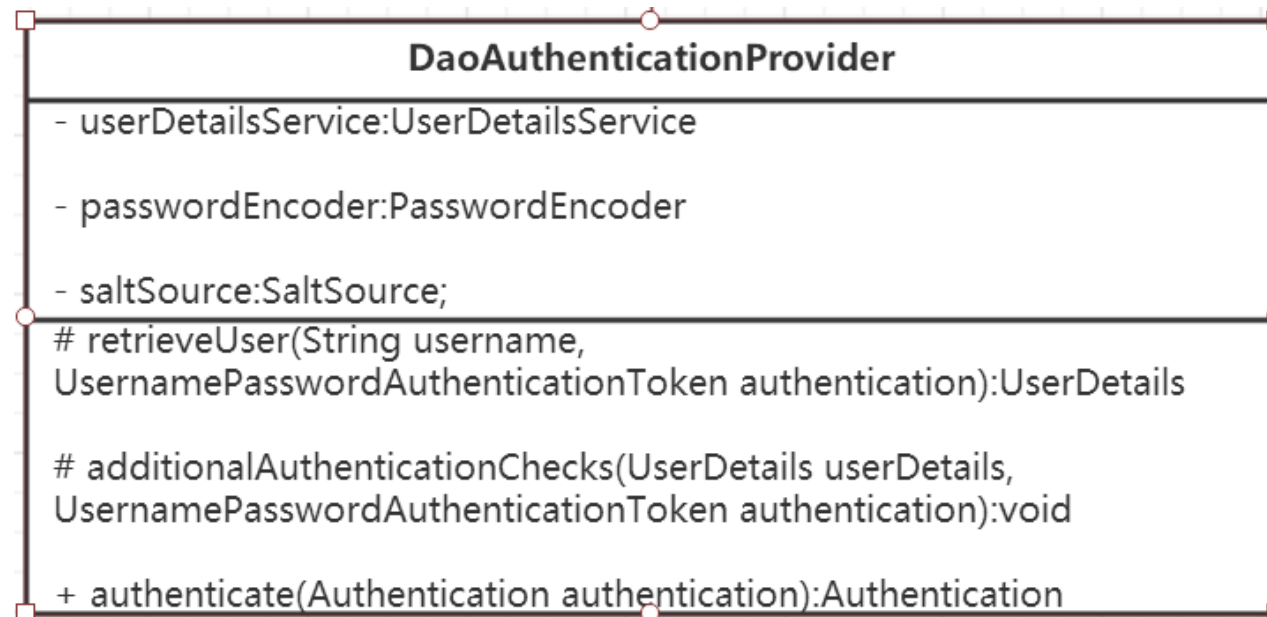
ProviderManager 中的List, 会依照次序去认证, 认证成功则立即返回, 若认证失败则返回null, 下一个AuthenticationProvider会继续尝试认证, 如果所有认证器都无法认证成功, 则

ProviderManager 会抛出一个ProviderNotFoundException异常。

到这里，如果不纠结于AuthenticationProvider的实现细节以及安全相关的过滤器，认证相关的核心类其实都已经介绍完毕了：身份信息的存放容器SecurityContextHolder，身份信息的抽象Authentication，身份认证器AuthenticationManager及其认证流程。姑且在这里做一个分隔线。下面来介绍下AuthenticationProvider接口的具体实现。

1.4 DaoAuthenticationProvider

AuthenticationProvider最最最常用的一个实现便是DaoAuthenticationProvider。顾名思义，Dao正是数据访问层的缩写，也暗示了这个身份认证器的实现思路。由于本文是一个Overview，姑且只给出其UML类图：



按照我们最直观的思路，怎么去认证一个用户呢？用户前台提交了用户名和密码，而数据库中保存了用户名和密码，认证便是负责比对同一个用户名，提交的密码和保存的密码是否相同便是了。在Spring Security中。提交的用户名和密码，被封装成了UsernamePasswordAuthenticationToken，而根据用户名加载用户的任务则是交给了UserDetailsService，在DaoAuthenticationProvider中，对应的方法便是retrieveUser，虽然有两个参数，但是retrieveUser只有第一个参数起主要作用，返回一个UserDetails。还需要完成UsernamePasswordAuthenticationToken和UserDetails密码的比对，这便是交给additionalAuthenticationChecks方法完成的，如果这个void方法没有抛异常，则认为比对成

功。比对密码的过程，用到了PasswordEncoder和SaltSource，密码加密和盐的概念相信不用我赘述了，它们为保障安全而设计，都是比较基础的概念。

如果你已经被这些概念搞得晕头转向了，不妨这么理解DaoAuthenticationProvider：它获取用户提交的用户名和密码，比对其正确性，如果正确，返回一个数据库中的用户信息（假设用户信息被保存在数据库中）。

1.5 UserDetails与UserDetailsService

上面不断提到了UserDetails这个接口，它代表了最详细的用户信息，这个接口涵盖了一些必要的用户信息字段，具体的实现类对它进行了扩展。

```
1 public interface UserDetails extends Serializable {
2     Collection<? extends GrantedAuthority> getAuthorities();
3     String getPassword();
4     String getUsername();
5     boolean isAccountNonExpired();
6     boolean isAccountNonLocked();
7     boolean isCredentialsNonExpired();
8     boolean isEnabled();
9 }
```

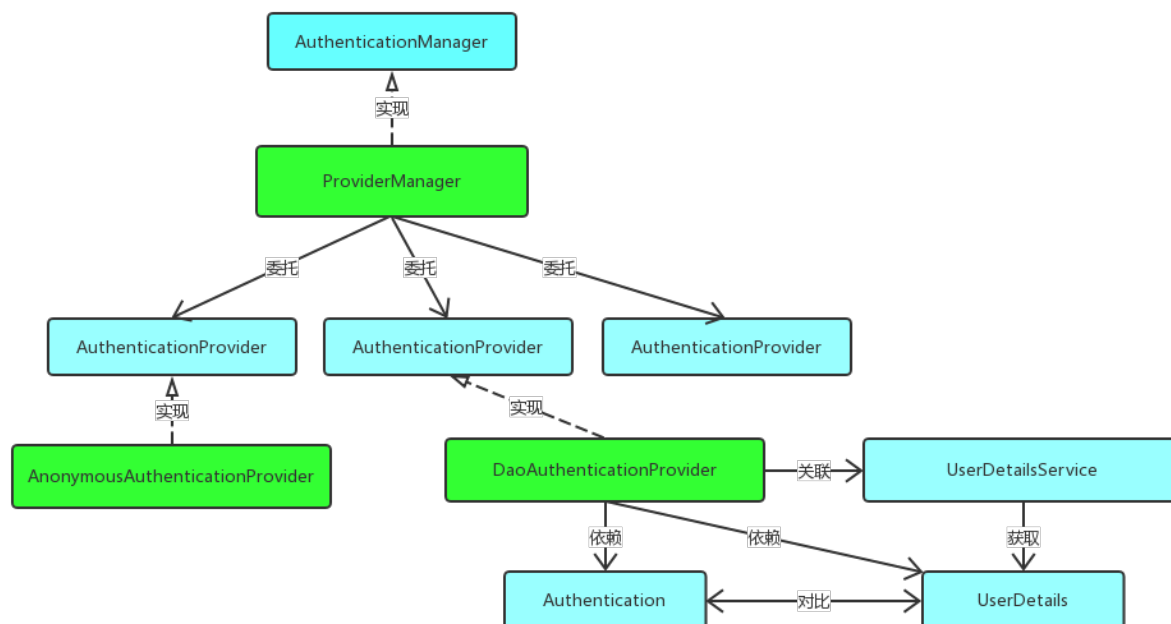
它和Authentication接口很类似，比如它们都拥有username，authorities，区分他们也是本文的重点内容之一。Authentication的getCredentials()与UserDetails中的getPassword()需要被区分对待，前者是用户提交的密码凭证，后者是用户正确的密码，认证器其实就是对这两者的比对。Authentication中的getAuthorities()实际是由UserDetails的getAuthorities()传递而形成的。还记得Authentication接口中的getUserDetails()方法吗？其中的UserDetails用户详细信息便是经过了AuthenticationProvider之后被填充的。

```
1 public interface UserDetailsService {
2     UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
3 }
```

UserDetailsService和AuthenticationProvider两者的职责常常被人们搞混，关于他们的问题在文档的FAQ和issues中屡见不鲜。记住一点即可，敲黑板！！！UserDetailsService只负责从特定的地方（通常是数据库）加载用户信息，仅此而已，记住这一点，可以避免走很多弯路。UserDetailsService常见的实现类有JdbcDaoImpl，InMemoryUserDetailsManager，前者从数据库加载用户，后者从内存中加载用户，也可以自己实现UserDetailsService，通常这更加灵活。

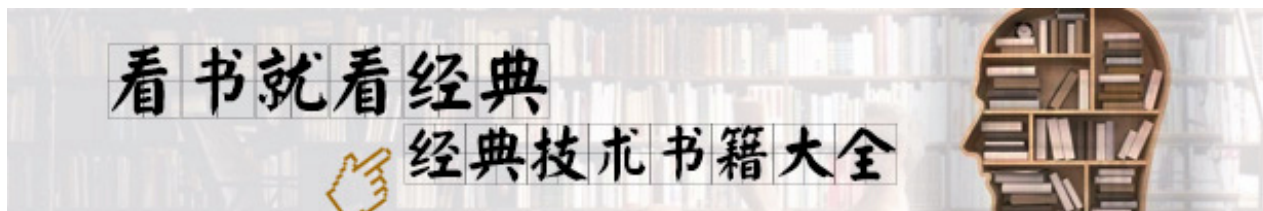
1.0 架构概览图

为了更加形象的理解上述我介绍的这些核心类，附上一张按照我的理解，所画出Spring Security的一张非典型的UML图



如果对Spring Security的这些概念感到理解不能，不用担心，因为这是Architecture First导致的必然结果，先过个眼熟。后续的文章会秉持Code First的理念，陆续详细地讲解这些实现类的使用场景，源码分析，以及最基本的：如何配置Spring Security，在后面的文章中可以不时翻看这篇文章，找到具体的类在整个架构中所处的位置，这也是本篇文章的定位。另外，一些Spring Security的过滤器还未囊括在架构概览中，如将表单信息包装成UsernamePasswordAuthenticationToken的过滤器，考虑到这些虽然也是架构的一部分，但是真正重写他们的可能性较小，所以打算放到后面的章节讲解。





相关文章

- [Spring Security \(三\) : 核心配置解读](#)
- [Spring Security \(二\) : 指南](#)
- [SPRING SECURITY JAVA配置 : 可读性](#)
- [SPRING SECURITY JAVA配置 : OAUTH](#)
- [SPRING SECURITY JAVA配置 : Method Security](#)
- [SPRING SECURITY JAVA配置 : Web Security](#)
- [SPRING SECURITY JAVA配置 : 简介](#)
- [深入解析Android关机](#)
- [SpringBoot\(十四\) : springboot整合shiro-登录认证和权限管理](#)
- [使用Java函数接口及lambda表达式隔离和模拟外部依赖方便单元测试](#)

发表评论

Name*

姓名

邮箱*

请填写邮箱

网站 (请以 http://开头)

请填写网站地址

请填写评论内容

(*) 表示必填项



提交评论

1 条评论

Xiaoyuan

2017/09/27 下午 1:45

赞

Well-loved. Like or Dislike:  5  0

回复

« [netty 实战 - netty client 连接池设计](#)

[Netty in action — 事件循环和线程模式](#) »

关于ImportNew

ImportNew 专注于 Java 技术分享。于2012年11月11日 11:11正式上线。是的，这是一个很特别的时刻：)

ImportNew 由两个 Java 关键字 import 和 new 组成，意指：Java 开发者学习新知识的网站。import 可认为是学习和吸收，new 则可认为是新知识、新技术圈子和新朋友.....



联系我们

Email : ImportNew.com@gmail.com

新浪微博 : [@ImportNew](#)

推荐微信号



反馈建议 : ImportNew.com@gmail.com

广告与商务合作QQ : 2302462408

推荐关注

小组 - 好的话题、有启发的回复、值得信赖的圈子

头条 - 写了文章？看干货？去头条！

相亲 - 为IT单身男女服务的征婚传播平台

资源 - 优秀的工具资源导航

翻译 - 活跃 & 专业的翻译小组

博客 - 国内外的精选博客文章

设计 - UI,网页，交互和用户体验

前端 - JavaScript, HTML5, CSS

安卓 - 专注Android技术分享

iOS - 专注iOS技术分享

Java - 专注Java技术分享

Python - 专注Python技术分享