<https://docs.spring.io/spring-security/site/docs/5.3.2.RELEASE/reference/html5/>

Spring Security requires **a Java 8** or higher Runtime Environment.

you need not configure a special **Java Authentication and Authorization Service (JAAS)** policy file or place Spring Security into common classpath locations.

Spring Security provides comprehensive support for [authentication](https://docs.spring.io/spring-security/site/docs/5.3.2.RELEASE/reference/html5/" \l "authentication), authorization, and protection against [common exploits](https://docs.spring.io/spring-security/site/docs/5.3.2.RELEASE/reference/html5/" \l "exploits). It also provides integration with other libraries to simplify its usage.

<https://github.com/spring-projects/spring-security/>

<https://github.com/spring-projects/spring-security/tree/5.3.2.RELEASE/samples>

#### Maven with SpringBoot

Spring Boot provides a spring-boot-starter-security starter that aggregates Spring Security-related dependencies together.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

**Since Spring Boot provides a Maven BOM to manage dependency versions, you do not need to specify a version.**

If you wish to **override** the Spring Security version, you may do so by providing a Maven property, as the following example shows:

<properties>

<spring-security.version>5.3.2.RELEASE</spring-security.version>

</dependencies>

#### Maven without SpringBoot

<dependencyManagement>

<dependencies>

<!-- ... other dependency elements ... -->

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-bom</artifactId>

<version>{spring-security-version}</version>

<type>pom</type>

<scope>import</scope>

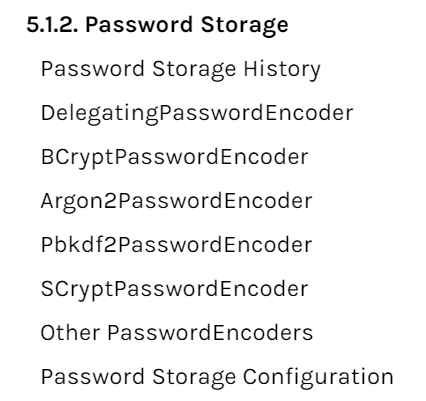
</dependency>

</dependencies>

</dependencyManagement>

#### 功能

#### Password Storage



Spring Security’s **PasswordEncoder**interface is used to perform a one way transformation of a password to allow the password to be stored securely **单向密码转换和安全保存**

**历史实现：比较新老密码的SHA256数值是否相同；数据库只需保留密码的单向hash结果**

Developers were then encouraged to store passwords after running them through a one way hash such as SHA-256. When a user tried to authenticate, the hashed password would be compared to the hash of the password that they typed.

 To defeat this new system malicious users decided to create lookup tables known as [Rainbow Tables](https://en.wikipedia.org/wiki/Rainbow_table).

Rather than doing the work of guessing each password every time, they computed the password once and stored it in a lookup table.

To mitigate the effectiveness of Rainbow Tables, developers were encouraged to use salted passwords.  **--- 应对方式：加盐值**

The salt and the user’s password would be ran through the hash function which produced a unique hash.

盐值和密码一起hash的得到唯一的hash结果

The salt would be stored alongside the user’s password in clear text.

Then when a user tried to authenticate, the hashed password would be compared to the hash of the stored salt and the password that they typed. The unique salt meant that Rainbow Tables were no longer effective because the hash was different for every salt and password combination.

In modern times we realize that cryptographic hashes (like SHA-256) are no longer secure. The reason is that with modern hardware we can perform billions of hash calculations a second

--- 每秒10亿级别的计算使得上述方式不再安全

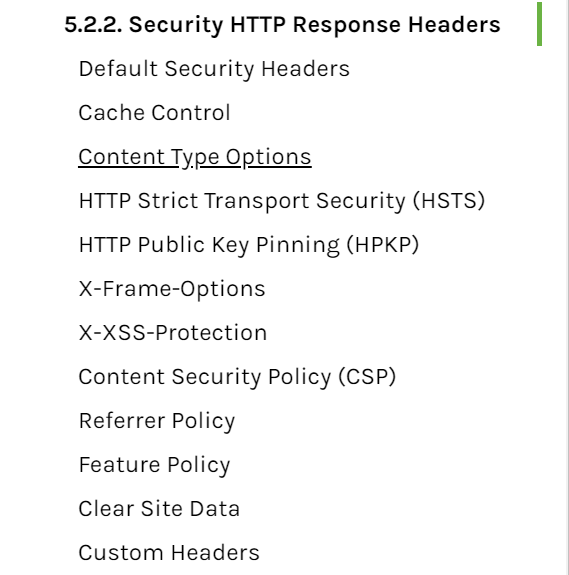
An adaptive one-way function allows configuring a "work factor" which can grow as hardware gets better. It is recommended that the "work factor" be tuned to take about 1 second to verify a password on your system. -- 当前推荐的方案

Because adaptive one-way functions are intentionally resource intensive, validating a username and password for every request will degrade performance of an application significantly.

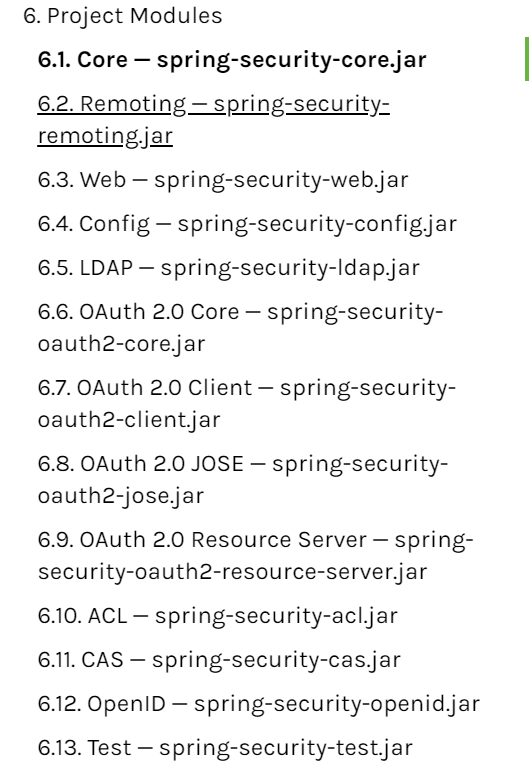
Examples of adaptive one-way functions that should be used include [bcrypt](https://docs.spring.io/spring-security/site/docs/5.3.2.RELEASE/reference/html5/" \l "authentication-password-storage-bcrypt), [PBKDF2](https://docs.spring.io/spring-security/site/docs/5.3.2.RELEASE/reference/html5/" \l "authentication-password-storage-pbkdf2), [scrypt](https://docs.spring.io/spring-security/site/docs/5.3.2.RELEASE/reference/html5/" \l "authentication-password-storage-scrypt), and [argon2](https://docs.spring.io/spring-security/site/docs/5.3.2.RELEASE/reference/html5/" \l "authentication-password-storage-argon2).

#### Cross Site Request Forgery (CSRF)

#### Security HTTP Response Headers



#### Project Modules



#### Start开始使用

**Spring Security integrates with the Servlet Container by using a standard Servlet Filter**. This means it works with any application that runs in a Servlet Container.

## Hello Spring Security -- 入门使用

[https://docs.spring.io/spring-security/site/docs/5.3.2.RELEASE/reference/html5/#samples](https://docs.spring.io/spring-security/site/docs/5.3.2.RELEASE/reference/html5/" \l "samples)

**Spring Boot Auto Configuration**

Spring Boot automatically:

1、Enables Spring Security’s default configuration, which creates a servlet Filter as a bean named **springSecurityFilterChain**. This bean is responsible for all the security (protecting the application URLs, validating submitted username and passwords, redirecting to the log in form, and so on) within your application.

2、Creates a UserDetailsService bean with a username of user and a randomly generated password that is logged to the console.

3、Registers the Filter with a bean named springSecurityFilterChain with the Servlet container for every request.

**Spring Boot is not configuring much, but it does a lot. A summary of the features follows:**

Require an authenticated user for any interaction with the application

Generate a default login form for you

Let the user with a username of user and a password that is logged to the console to authenticate with form-based authentication (in the preceding example, the password is 8e557245-73e2-4286-969a-ff57fe326336)

Protects the password storage with BCrypt

Lets the user log out

CSRF attack prevention

Session Fixation protection

**Security Header integration**

HTTP Strict Transport Security for secure requests

X-Content-Type-Options integration

Cache Control (can be overridden later by your application to allow caching of your static resources)

X-XSS-Protection integration

X-Frame-Options integration to help prevent Clickjacking

**Integrate with the following Servlet API methods:**

HttpServletRequest#getRemoteUser()

HttpServletRequest.html#getUserPrincipal()

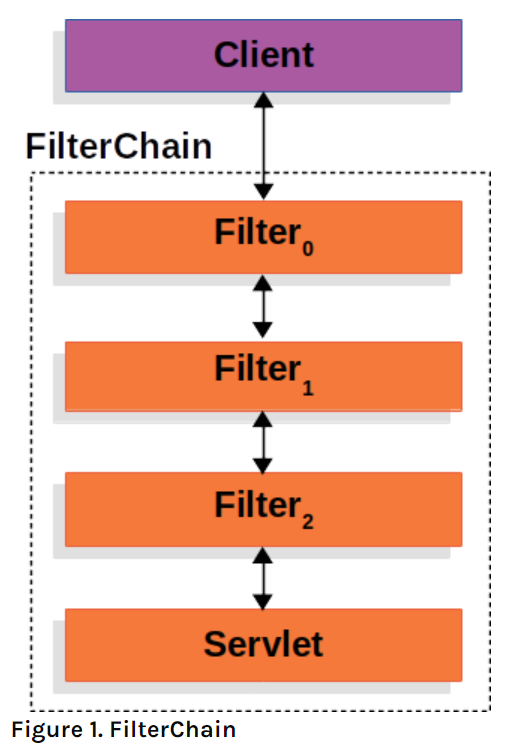
HttpServletRequest.html#isUserInRole(java.lang.String)

HttpServletRequest.html#login(java.lang.String, java.lang.String)

HttpServletRequest.html#logout()

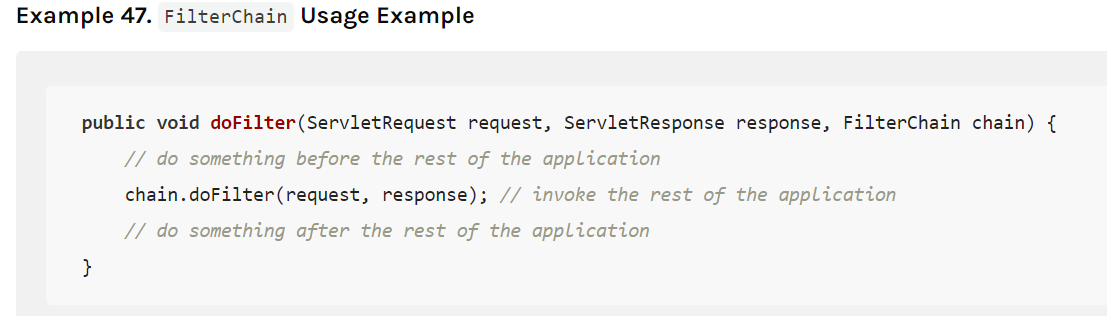
#### A Review of Filters

**Spring Security’s Servlet support is based on Servlet Filters**



The client sends a request to the application, and the container creates a **FilterChain**which contains the Filters and **Servlet**that should process the **HttpServletRequest**based on the path of the request URI.

 In a Spring MVC application the **Servlet is an instance of [DispatcherServlet](https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html" \l "mvc-servlet)**. At most one Servlet can handle a single HttpServletRequest and HttpServletResponse.



However, more than one Filter can be used to:

1. **Prevent downstream Filters or the Servlet from being invoked**. In this instance the Filter will typically write the HttpServletResponse.

**---- 拦截后续Filter执行，直接写Response字段**

1. Modify the HttpServletRequest or HttpServletResponse used by the downstream Filters and Servlet

---- 修改后续Filter处理的Request或响应

