

6.5 ウェブ開発補足

- Spring Security
- ログ出力
- Lombok
- プロパティファイル



- 1 Spring Security
- 2 ログ出力
- 3 Lombok
- 4 プロパティファ
イル

セキュリティの概要

- 実際のサーバー開発では、ユーザーに様々なサービスを提供するだけでなく、それらの**サービスの情報安全性**を確保する必要があります。サービスを安全に利用するために、各ユーザーに何らかの**権限**[Authority]を与えて、各ユーザーが利用を許可されたサービスのみを利用できるようにすること、即ち**アクセス制御**[Access Control]が一般的です。
- 例えば、Alice は自分のアカウントにログインしてメッセージを投稿することだけができ、Bob のアカウントにログインすることはできないはずです。別の例として、外部ユーザーの Carol が利用できるのは公開されたサービスのみで、サーバー上の大変なデータを変更するようなサービスは、ウェブサイトの管理者の Dave のみ利用できます。

権限管理の要素

- 権限管理のプロセスは「3A」という3つのステップが必要:
 - **認証**[Authentication]: ユーザーを識別し、**誰が**サービスを利用しているのかをシステムが把握します。例えば、Alice がログインする際に正しいユーザー名とパスワードを使って、ウェブサイトは Alice 本人がサービスを利用していることを確認できるようにします。
 - **承認**[Authorization]: 各ユーザーに適切な**権限**（どのサービスを利用できますか）を割り当てます。例えば、Alice が一般ユーザーである場合、投稿と返信だけができます。Bob は管理者で、人の投稿を削除したり、操作記録を閲覧することができます。
 - **記録**[Accounting]: 管理のために、ユーザーの操作をサーバに記録。

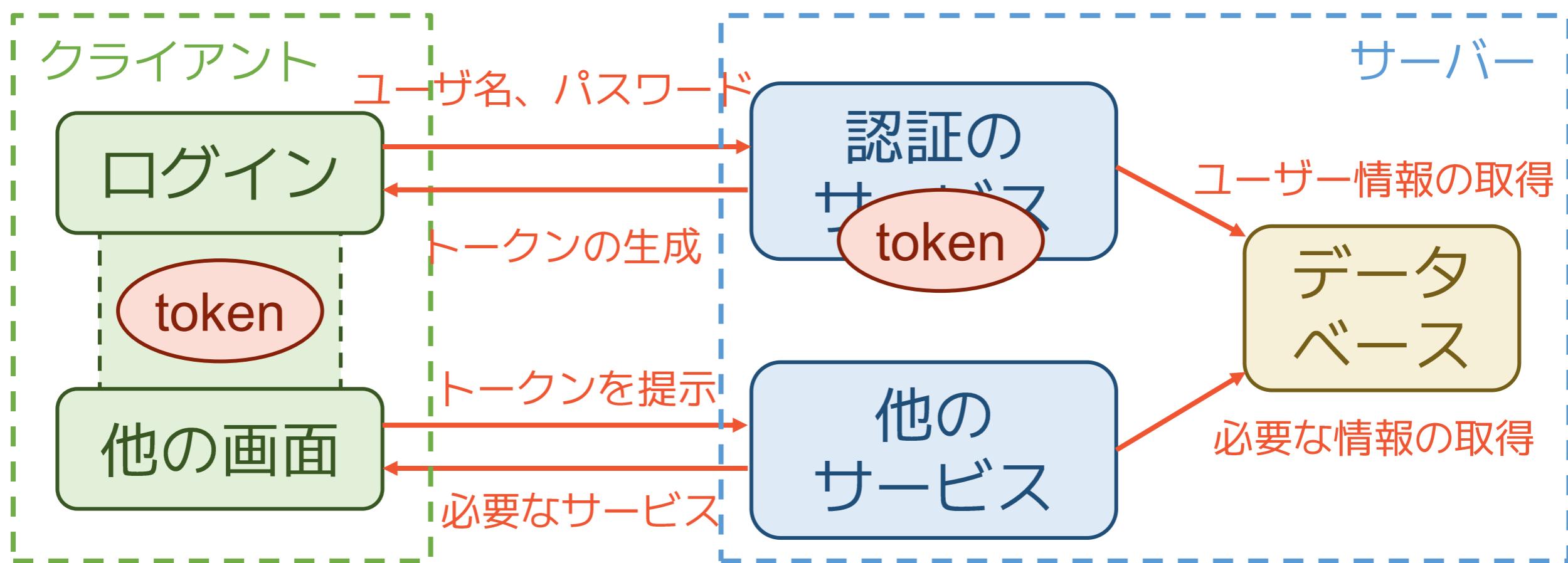
次へ 

 前へ

- 情報セキュリティに関するシステム設計は、2つの重要な点があります：一つは、上記のステップの実装を含むようにシステムを設計すること。もう一つは、上記のステップの秘匿性を確保するための**数学的なアルゴリズム**（例：暗号化アルゴリズム）。今回の授業では、前者、つまり Spring にどのような権限管理システムが実装されているかについて説明します。

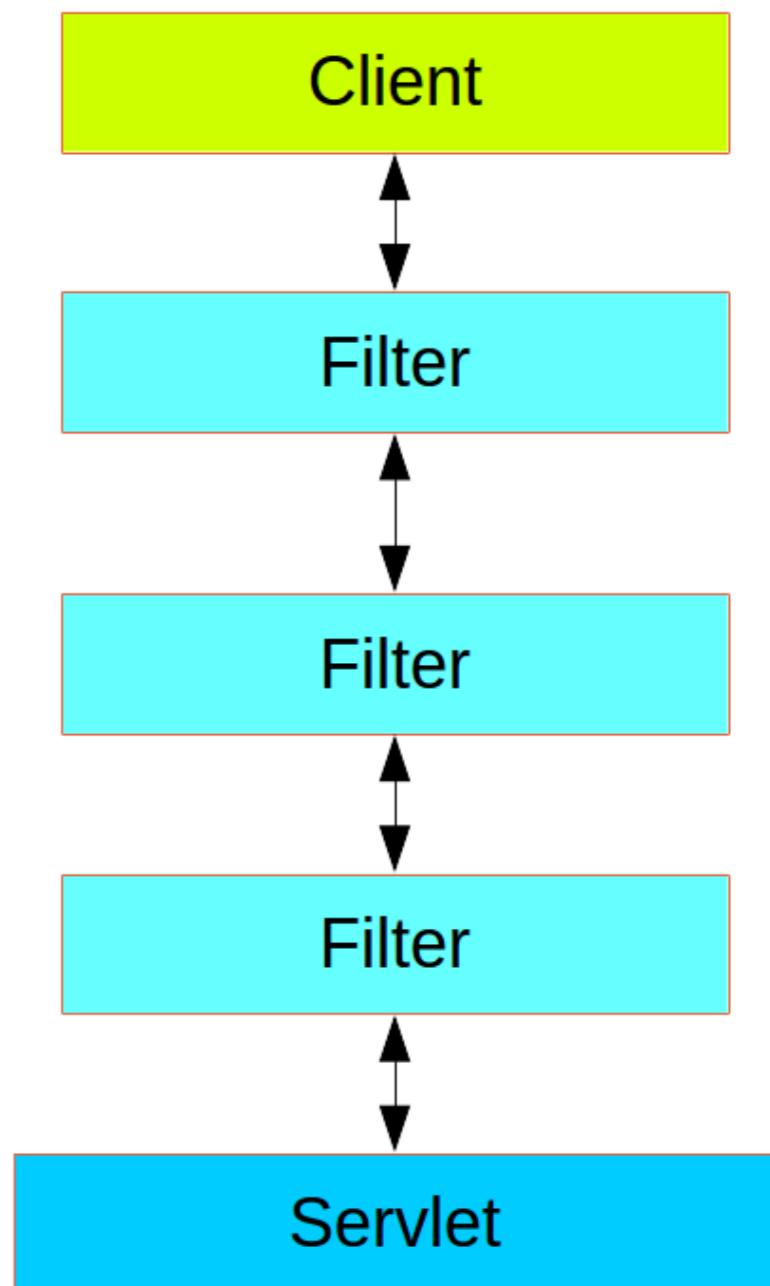
Spring Securityの基礎原理

- Spring は **Spring Security** というフレームワークを提供し、認証や承認に関する様々な操作を扱えます。認証の方法には様々なものがありますが、ここでは最も主流である**トークン認証**について簡単に紹介します：



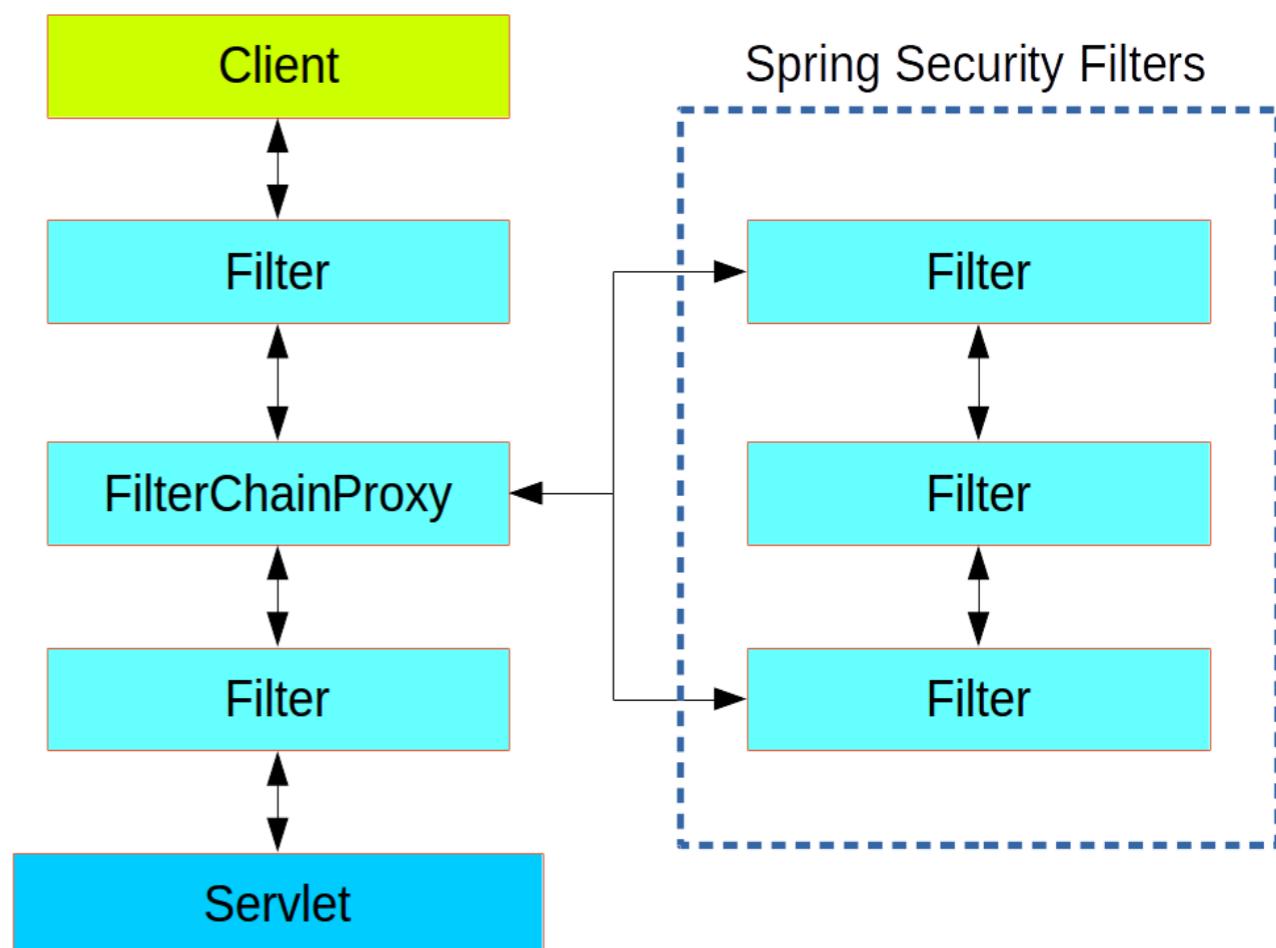
フィルター

- 承認する際、Spring Securityは **フィルターチェーン** [Filter Chain] という概念を利用します。
- フィルターとは、Servlet で HTTP リクエストとレスポンスを処理するオブジェクトです。クライアントから送信されたリクエストは、直接サーバに送信されずに、特定のフィルターを通過する必要があります。各フィルターは、次のレベルのフィルターに送る前に、リクエストに検証と修正を行えます。



Spring のフィルターチェーン

- Spring Security は Servlet にユーザーの認証と承認に関する操作を制御する `FilterChainProxy` という特殊なフィルタを加えます。
- このフィルタ自身は、Spring Security が提供するいくつかのフィルタを含んで、フィルタのチェーンを形成しています。それぞれのフィルタが独自の役割を担っていて、サーバ開発者は要求に応じて、適切なフィルターを選択してチェーンに入れます。

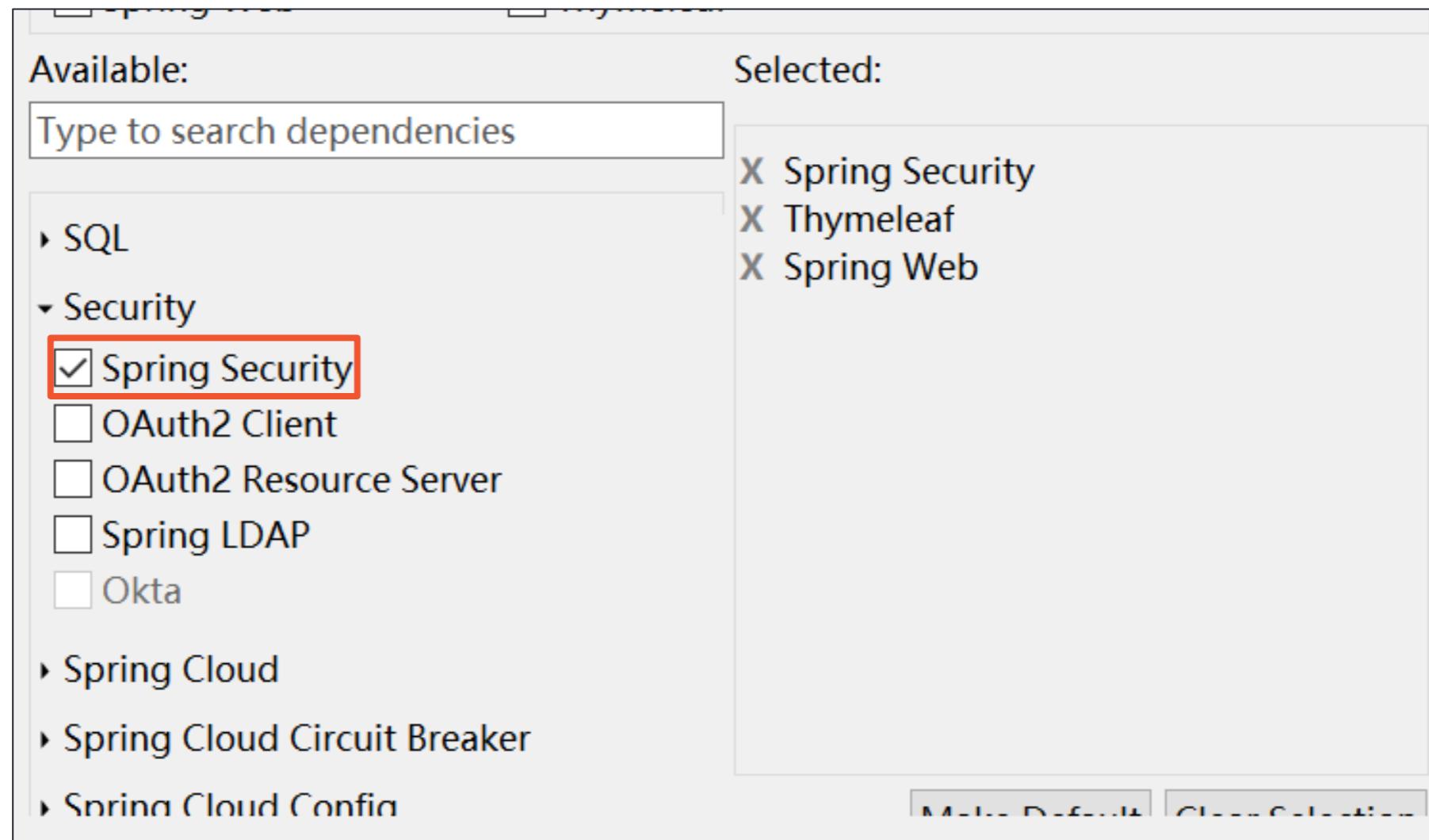


簡単な Security プロジェクト

- Spring Security は認証・承認関連のアルゴリズム、API、各種フィルタを多数提供し、学習が難しいので、Spring Security の公式チュートリアルへの参考をおすすめ：
 <https://spring.io/guides/topicals/spring-security-architecture/>
- しかし、最低限のログイン画面と登録画面に基づくログインシステムが含むウェブサイトを実現するだけであれば、Spring Boot は比較的簡単な API を提供しています。この方法では、安全性のリスクがあるため、**実際の製品に使用できません**が、コード例を通して Spring Security の基本パターンを体験することができます。
- さて、簡単なログインシステムを実装してみましょう。

プロジェクトの作成

- Spring Security の機能を利用するには、プロジェクト作成時に Spring Security の依存関係を追加する必要があります（もしくは作成後に Maven を利用）：

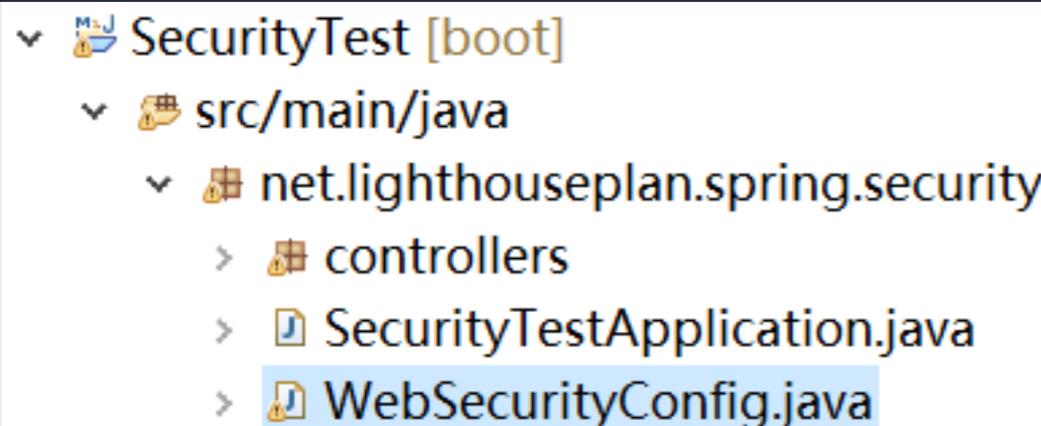


設定クラスの作成

- Security 機能は `WebSecurityConfigurerAdapter` を継承した Java クラスを作成し、関連するアノテーションを追加して設定する必要があります:

```
1 @Configuration  
2 @EnableWebSecurity  
3 public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  
4     //...  
5 }
```

- これをパッケージ内の任意の場所、例えば Application クラスの隣に置きます:



userDetailsService() をオーバーライド

- 設定クラスの `userDetailsService()` メソッドをオーバーライドすることで、**認証**の方法を設定することができます：

```
1 @Bean
2 @Override
3 public UserDetailsService userDetailsService() {
4     UserDetails user =
5         User.withDefaultPasswordEncoder()
6             .username("Alice")           ユーザー名 Alice
7             .password("123456")        パスワード 123456
8             .roles("USER")            ユーザーのロールは「USER」 (一般ユーザー)
9         .build();
10
11     return new InMemoryUserDetailsManager(user);
12 }
```

このユーザーを含む認証システムを作成

configure() をオーバーライド

- **configure()** メソッドをオーバーライドすることで、**承認の仕方**（アクセス制御）を設定することができます：

```
1 @Override  
2 protected void configure(HttpSecurity http) throws Exception {  
3     http  
4         .authorizeRequests()  
5             .antMatchers("/register").permitAll()  
6             .anyRequest().authenticated()  
7         .and()  
8         .formLogin()  
9             .loginPage("/login")  
10        .permitAll()  
11        .and()  
12        .logout()  
13        .permitAll();  
14 }
```

ログインしなくても登録ページに
アクセスできるように設定

他のすべてのリクエストは
ユーザーログインが必須

カスタムなログイン画面を設定

ログアウト関連の設定

ログイン画面の設定

- 設定を追加しない場合、Spring はデフォルトの login 画面を提供します。設定を追加した後、自分のコントローラと HTML テンプレートを実装し、ログイン画面が作成できます：

```
.and()  
.formLogin()  
    .loginPage("/login")  
    .permitAll()  
    .and()
```

```
1 @GetMapping("/login")  
2 public String getLoginPage() {  
3     return "login.html";  
4 }
```

- login.html では、name が username と password である <input> があって、「/login」に POST リクエストを送信する必要があります。コントローラは作らなくていいです。
- ログインに成功した後、ユーザーはログイン画面にアクセスする前にアクセスしようとした URL に転送されます。そうでない場合は、「/」にアクセスします。

ユーザー情報の取得

- Java では、いつでも以下の方法でユーザーに関する情報を取得することができます:

```
1 UserDetails user = (UserDetails) SecurityContextHolder  
2         .getContext().getAuthentication().getPrincipal();  
3  
4 mav.addObject("name", user.getUsername());
```

- ユーザー名を HTML で表示したいだけなら、Thymeleaf の便利な文法も利用できます:

```
<h1>Hello, [[${#httpServletRequest.remoteUser}]]!</h1>
```



Question and answer



- 1 Spring Security
- 2 ログ出力
- 3 Lombok
- 4 プロパティファ
イル

ログ

- これまででは、`System.out.println()`などのメソッドを使って、テストや Debug の情報を出力しています。実際の開発現場では、このようなテスト情報やシステムの動作記録を、**ログ**^[Log]と呼ばれる外部ファイルに保存しておきます。
- コンソールに出力するだけでなく、ログを使用することにはいくつかのメリットがあります：
 - **毎回の実行情報**が自動保存され、過去の記録はいつでも閲覧可能
 - エラーでプログラムが**異常終了**した場合でも記録が保存
 - 開発者は、ユーザーからもらえたログ情報をもとに**デブッグ**できる

ログのレベル

- 実際のログの出力は、出力情報の重要度をいくつかのレベルに分けて出力します。よく使われるレベル構造は右の表に示します：
- 開発か、テストか、リリースされた製品がユーザーに使用される時など、様々な状況でプログラムが実行された時に、どのレベル以上（どれくらい細かい）情報をコンソールや外部ログファイルに出力するかを制御することができます。

レベル	意味
TRACE	最も詳細な情報
DEBUG	デバッグに役立つくくらいの詳細情報
INFO	一般情報
WARN	問題が発生する可能性がある警告情報
ERROR	エラーや例外に関する重要な情報

SLF4J

- 出力ログに関する機能を提供する Java のライブラリには、Log4J、Logback、SLF4J などがあります。ここでは、簡単に使える **SLF4J** を簡単に紹介します。
- Spring Boot の依存関係には既に SLF4J が含まれているため、別のパッケージを Maven とか追加する必要はありません。関連するメソッドは org.slf4j パッケージにあります。

Logger の取得

- ログを出力するために、まず、**Logger** クラスのオブジェクトを取得する必要があります：

```
Logger logger = LoggerFactory.getLogger("Logger Name");
```

- **Logger Name** はこのロガーの名前です。一般的に、現在の**クラス名**をロガーの名前として使用します：

```
Logger logger = LoggerFactory.getLogger(MainController.class);
```

ログの出力

- ロガーを取得したら、その**同名のメソッド**を使用して、特定なレベルのログが出力できます。例えば、以下のコードではWARN レベルのログを出力します：

```
logger.warn("Caution!");
```

- `System.out.println()` と同様に、各レベルのログの出力をコンソールで確認できます：

```
2022-08-13 07:01:47.692 INFO 17308 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[:] : Initializing Spring Disp
2022-08-13 07:01:47.692 INFO 17308 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dis
2022-08-13 07:01:47.693 INFO 17308 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization
2022-08-13 07:01:47.720 INFO 17308 --- [nio-8080-exec-1] n.l.s.s.controllers.MainController : Hello!
2022-08-13 07:01:47.720 WARN 17308 --- [nio-8080-exec-1] n.l.s.s.controllers.MainController : Caution!
2022-08-13 07:01:47.720 ERROR 17308 --- [nio-8080-exec-1] n.l.s.s.controllers.MainController : Something went wrong...
```

- ログの出力日時、スレッド名なども記録されます。

ログ出力の設定

- 現在、INFO レベル以上のメッセージのみが出力されることがわかりました。設定情報を application.properties に追加することで、出力ログのレベル制限が変更できます：

```
logging.level.[ロガーのクラス名かパッケージ名]=[出力レベル]
```

- 例えば、この Security プロジェクトで使ったロガーの出力レベルを DEBUG 以上に設定するには、次のような設定を書くだけです：

```
logging.level.net.lighthouseplan.spring.security=DEBUG
```

- また、以下の設定でログを外部ファイルにも出力できます：

```
. logging.file.name=[ログファイルのパス]
```



Question and answer



- 1 Spring Security
- 2 ログ出力
- 3 Lombok
- 4 プロパティファイル

JavaBeans の仕様

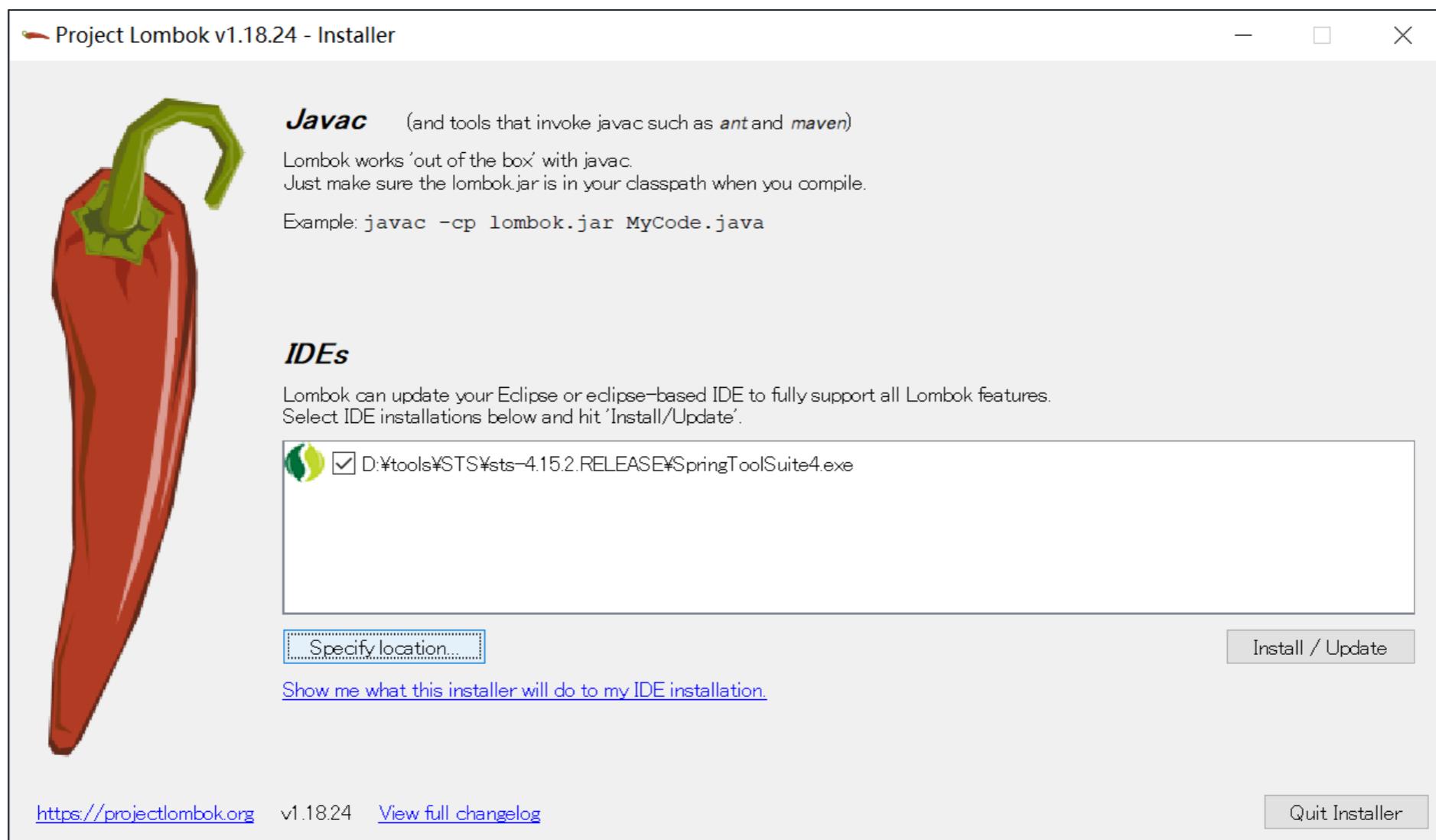
- クラスは以下の条件を満たす場合、JavaBean といいます：
 - すべての属性は `private` である
 - すべての属性は、それに対応するゲッターとセッターを持つ
 - `public` のパラメータなしコンストラクタがある
 - `Serializable` インタフェースを実装。
- 実際では、データを管理する DTO やパラメータクラスの標準として、JavaBean のような仕様がよく使われます。一部の仕様 (`Plain Old Java Object`、略して POJO など) は、コンストラクタが必ずしもパラメータなしではなく、`Serializable` を実装しないなど、若干の違いがあります。

Lombok

- 異なる JavaBean クラスは、ほとんど同様の方法で定義されています。更に、全て変数をパラメータとして持つコンストラクタや、`toString()` メソッドなどの、**毎回似たものを書かなければならぬ**コードがたくさんあります。
- **Lombok** は、このようなコードの生成を自動化し、開発を効率化するためのツールキットです。
- Lombok は追加インストールが必要です。以下のリンクからインストールパッケージをダウンロードしてください:
 <https://projectlombok.org/download>

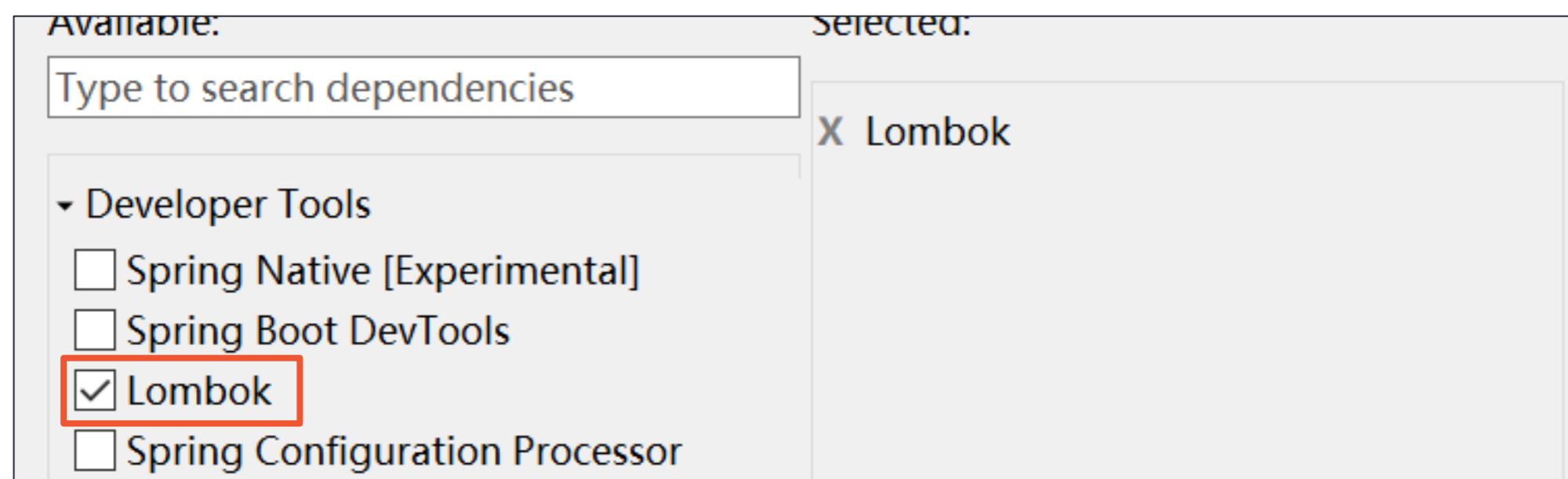
Lombok のインストール

- STS が表示されていない場合は、「Specify Location」をクリックして STS を探します。STS を選択し、「Install / Update」をクリックしてインストールします：



Lombok の依存関係の追加

- STS を再起動し、新規プロジェクトに Lombok の依存関係を追加します（もちろん、Maven 経由で後から追加することもできます）：



@Getter と @Setter

- Lombok の主な機能は、特定の宣言の前に付くアノテーションによって実現します。
- 例えば、**@Getter** または **@Setter** アノテーションを変数の宣言前に追加すると、その変数に対応するゲッターとセッターが自動的に生成されます：

```
1 public class Student {  
2     @Getter  
3     @Setter  
4     private String name;  
5 }
```

```
1 Student student = new Student();  
2 student.setName("Alice");  
3 System.out.println(student.getName()); // => Alice
```

よく使われるアノテーション

- その他によく使われるアノテーションを以下の表に：

アノテーション	機能
@NoArgsConstructor	パラメーターなしのコンストラクターの生成
@RequiredArgsConstructor	必須変数がパラメータなコンストラクタの生成
@AllArgsConstructor	全変数がパラメータなコンストラクタの生成
@ToString	toString() メソッドの生成
@EqualsAndHashCode	equals() および hashCode() の生成
@Data	クラスに @ToString、@EqualsAndHashCode、 @RequiredArgsConstructor を追加し、全て の変数に @Getter と @Setter を追加
@Log (@Slf4j)	log という (SLF4J) ロガーの生成

@Builder

- `@Builder` アノテーションは、オブジェクトを作成するための `builder()` メソッドを自動生成します。インスタンス化のコードをより読みやすく、拡張しやすくできます：

```
1 @Data
2 @Builder
3 public class Student {
4     private Long id;
5     private String name;
6     private int score;
7 }
```

```
1 Student student = Student.builder()
2         .id(1L)
3         .name("Alice")
4         .score(90)
5         .build();
6 System.out.println(student); // => Student(id=1, name=Alice, score=90)
```



Question and answer



- 1 Spring Security
- 2 ログ出力
- 3 Lombok
- 4 プロパティファイル

application.properties

- Spring Boot は、プロジェクト全体の**設定情報を格納する**ために、**application.properties** を使用します。
- src/main/resources ディレクトリ、またはプロジェクトパスの /config ディレクトリに配置する必要があります。
- この Properties ファイルを使って、ポート番号などのデフォルトの設定値が変更できます。ポート番号はデフォルトで 8080 に設定され、server.port で設定を変更できます：

```
server.port=8090
```

設定可能なパラメータ

- これまで使用したもの以外にも、application.propertiesで設定できる項目は多数あります：

パラメータ	機能
spring.application.name	アプリケーションの名前
server.address	サーバーのアドレス
spring.mail.host	サーバーのメールホストアドレス
logging.file.path	ログファイルのパス
spring.config.name	プロパティーのアドレス(デフォルトは application)

- 設定可能な全パラメータは、ここに記載されています：



<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>

Properties ファイルの問題

- 実際に使用されるプロパティのパラメータは、通常、一定の階層的な構造を満たします。しかし、従来の .properties ファイルはこの性質を利用していなかったため、パラメータが多くて名前が長い場合、非常に読みづらくなることがあります：

```
1 spring.datasource.url=jdbc:postgresql://localhost:5432/security
2 spring.datasource.username=postgres
3 spring.datasource.password=123456
4
5 logging.file.name=application.log
6 logging.level.root=INFO
7 logging.level.net.lighthouseplan.spring.security=DEBUG
```

YAML フォーマット

- Spring Boot は YAML 形式でのプロパティ設定もできます。情報を階層的に表現するため、プロパティの記述は簡単：

```
1 spring:  
2   datasource:  
3     url: jdbc:postgresql://localhost:5432/security  
4     username: postgres  
5     password: 123456  
6  
7   logging:  
8     file:  
9       name: application.log  
10    level:  
11      root: INFO  
12      net.lighthouseplan.spring.security: DEBUG
```

- YAML で記述されたプロパティは元の .properties ファイルを置き換えた application.yml ファイルに保存されます。



Question and answer

まとめ

Sum Up

1. Security:

① ウェブサイトの権限システムの一般的な概念：認証、承認、トークン。

② Spring Security の概念と簡単な使用方法。

2. ログの概念とロギングライブラリの使い方。

3. Lombok の使用方法。

4. プロパティファイルの書き方。

THANK YOU!