

## 6.5 セキュリティとセッション

- Spring Security
- Spring Session

# 目次

1 Spring Security

2 Spring Session

# 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、サーバー上の大事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

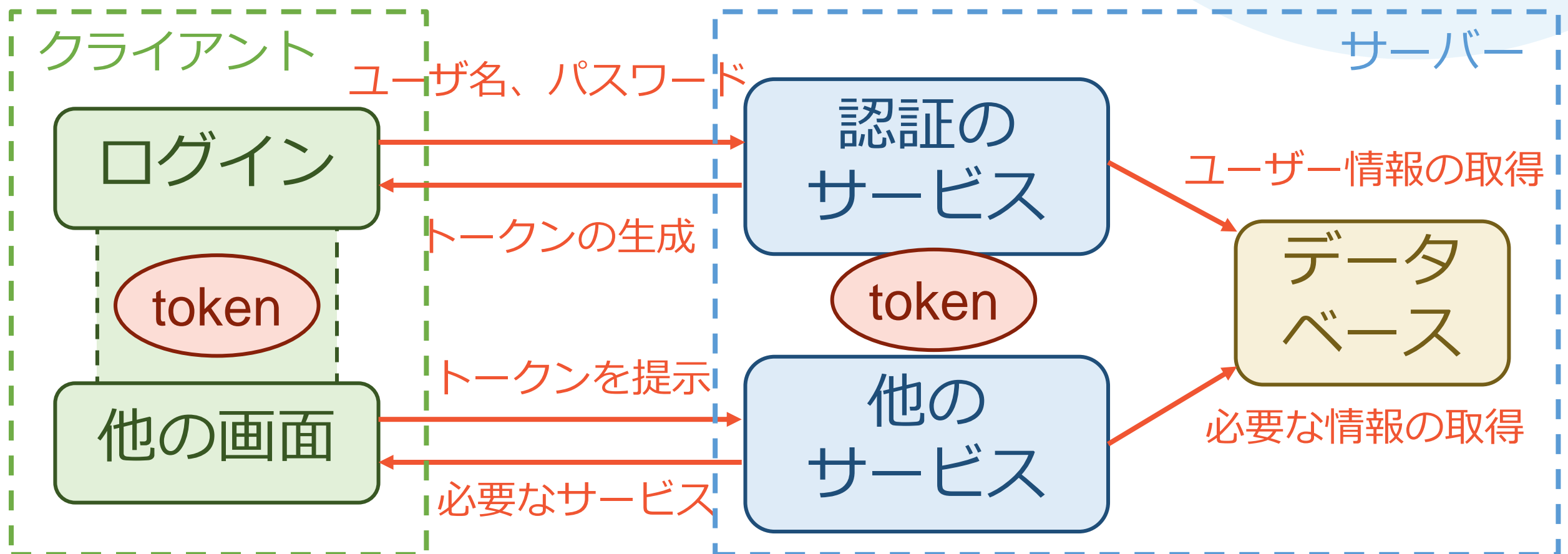
- 権限管理のプロセスは「3A」という 3 つのステップが必要：
  - **認証**[\[Authentication\]](#)：ユーザーを識別し、**誰が**サービスを利用しているのかをシステムが把握します。例えば、Alice がログインする際に正しいユーザー名とパスワードを使って、ウェブサイトは Alice 本人がサービスを利用していることを確認できるようにします。
  - **承認**[\[Authorization\]](#)：各ユーザーに適切な**権限**（どのサービスを利用できますか）を**割り当て**ます。例えば、Alice が一般ユーザーである場合、投稿と返信だけができます。Bob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

- 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。

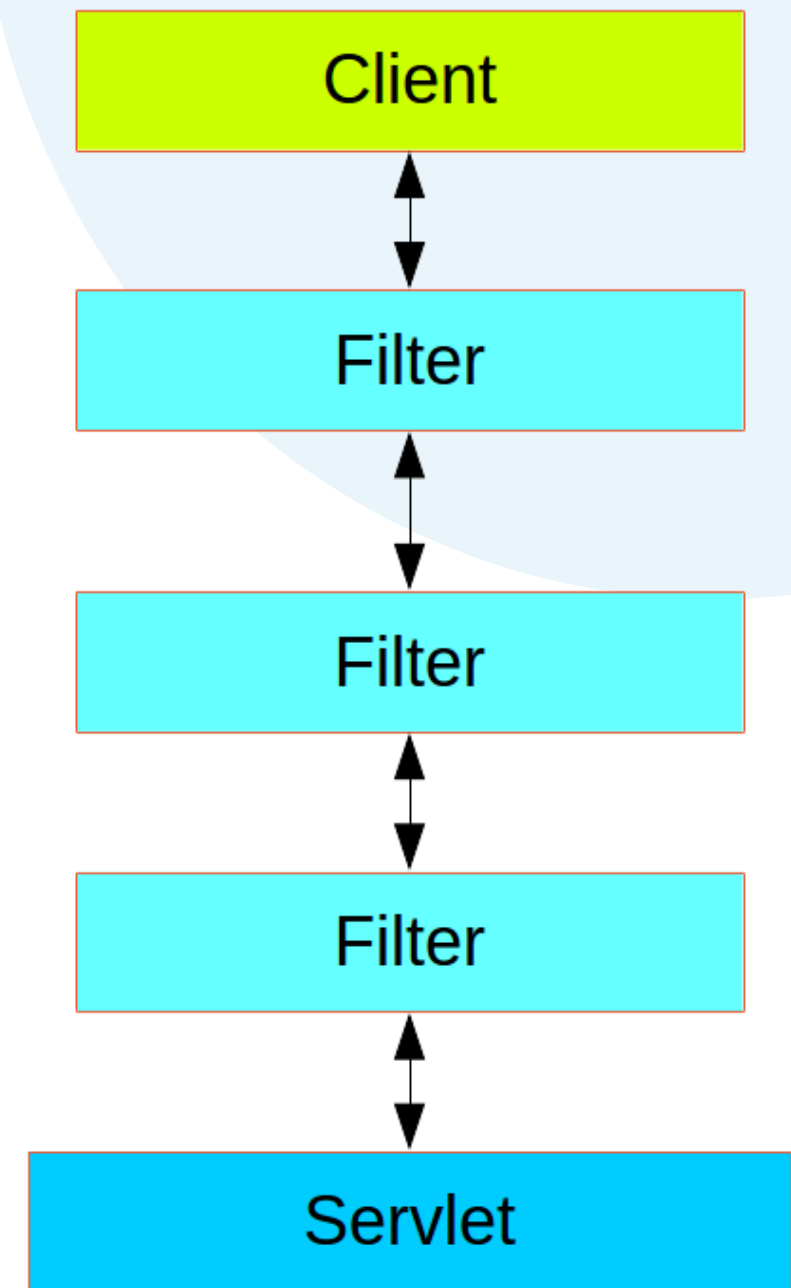
# Spring Securityの基礎原理

- 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**<sup>[Token]</sup> 認証について簡単に紹介します：



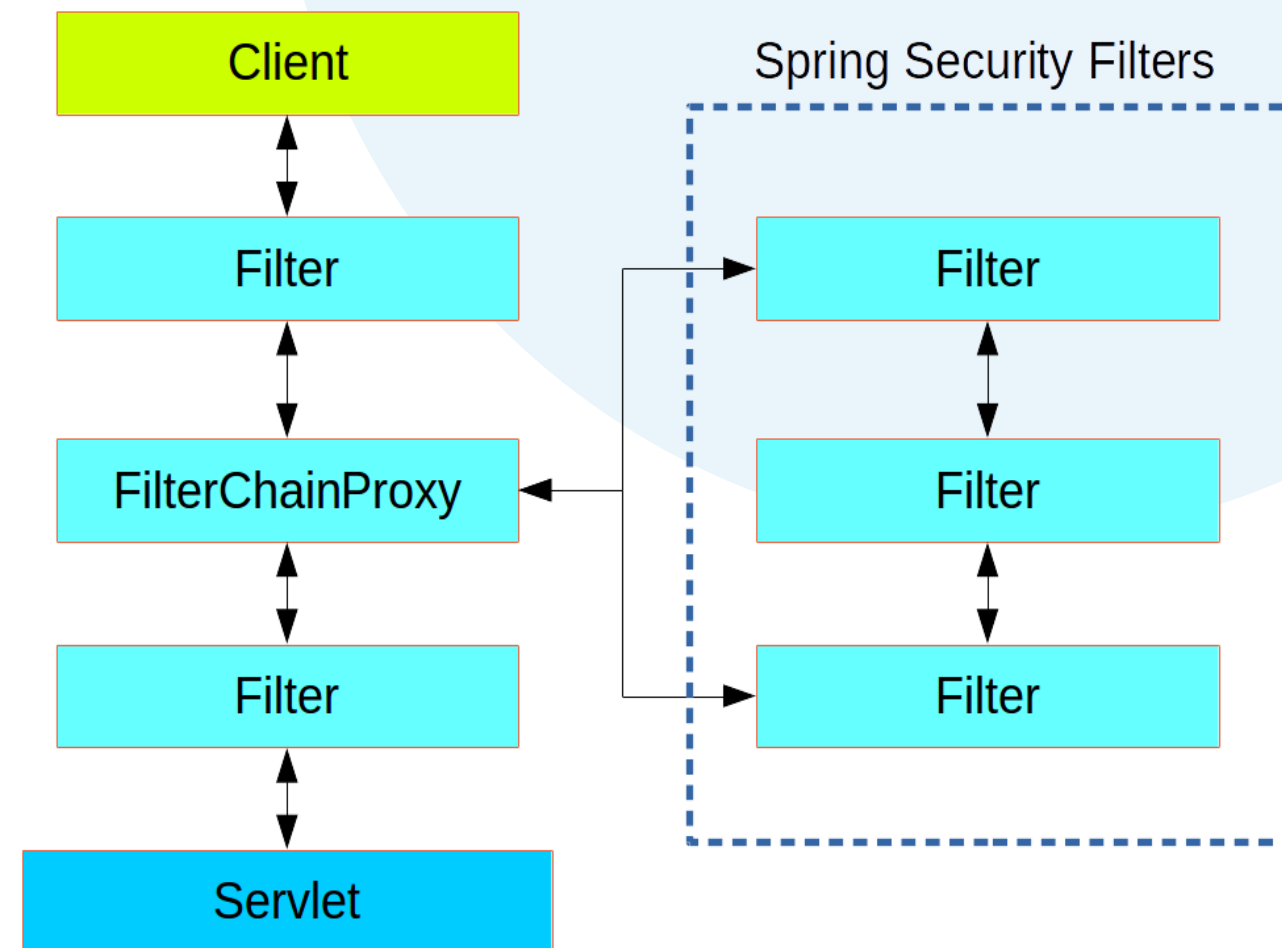
# フィルター

- 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。



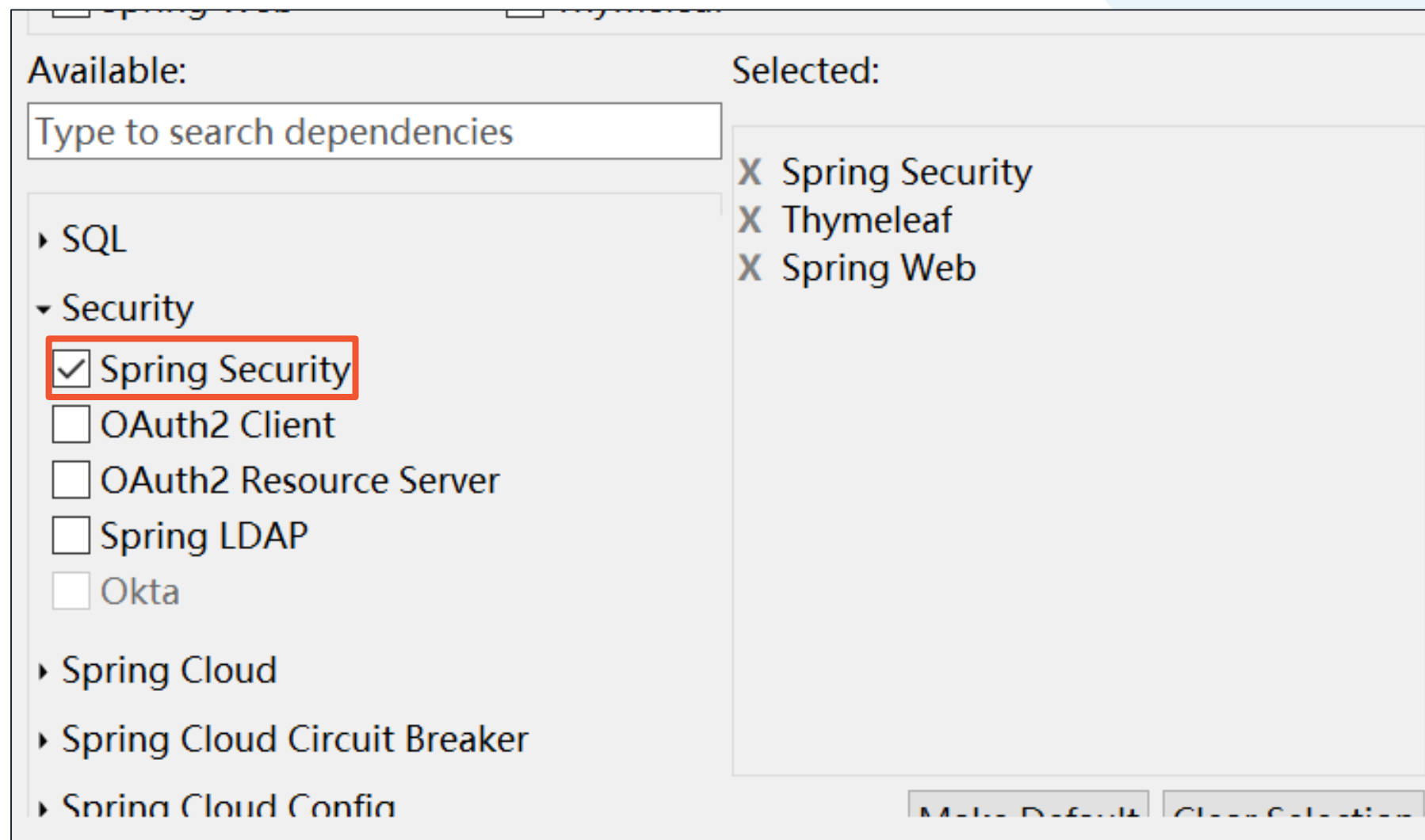


# 簡単な Security プロジェクト

- 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

- Spring Securityのフィルターチェーンを構成するためのメソッドを定義し、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 {
4     @Bean
5     public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
6         //ここに設定を書く
7     }

```

- これをパッケージ内の任意の場所、例えばconfigパッケージ内に置きます：

```

> SpringSecurityEx [boot] [LHP-backEnd ma
  > src/main/java
    > security.ex
      > config
        > WebSecurityConfig.java
      > controller
        > LoginController.java
        > SpringSecurityExApplication.java

```

# 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")
7             .password("123456")
8             .roles("USER")
9             .build();
10
11     return new InMemoryUserDetailsManager(user);
12 }

```

ユーザー情報を作成

ユーザー名 Alice

パスワード 123456

ユーザーのロールは「USER」（一般ユーザー）

このユーザーを含む認証システムを作成



# 認証カスタマイズ

- ログイン後の遷移先や、ログイン無しでアクセス可能かどうかを設定することが可能です。

```
1 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
2     http.formLogin(login -> login// フォーム認証の設定記述開始
3         .loginPage("/login")// ログイン画面のURL
4         .defaultSuccessUrl("/hello", true) // ログインが成功したときの遷移先
5         .usernameParameter("userName") // リクエストパラメータのname属性を明示
6         .passwordParameter("password")// リクエストパラメータのname属性を明示
7         .failureUrl("/login?error")// ログインに失敗したときの遷移先
8         .permitAll()// ログイン画面は未ログインでもアクセス可能
9     ).logout(logout -> logout// ログアウトの設定の記述を開始していきます。
10        .logoutSuccessUrl("/login")// ログアウト成功後の遷移先URL
11    ).authorizeHttpRequests(authz -> authz// URLごとの許可の設定の記述を開始します
12        .requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()
13        .requestMatchers("/register", "/css/**").permitAll()// ログイン無しでもアクセス可能
14        .anyRequest().authenticated()// 他のURLはログイン後のみアクセス可能
15    );
16    return http.build();
17 }
```



# 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：

```
.loginPage("/login")// ログイン画面のURL
.defaultSuccessUrl("/hello", true) //
```

```
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());
```

# Q&A

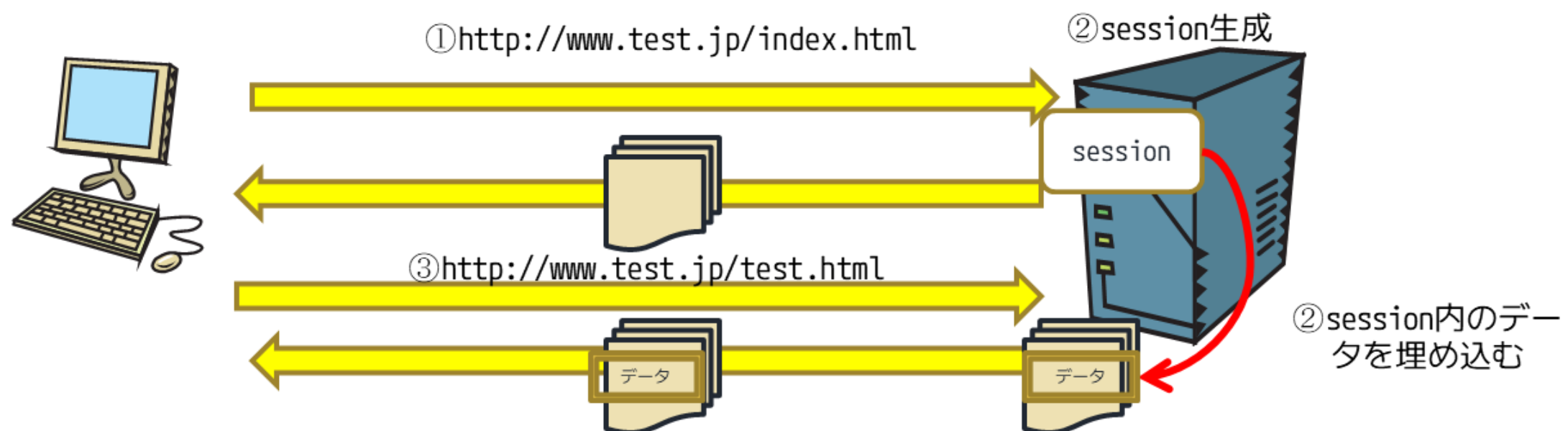
# 目次

1 Spring Security

2 Spring Session

# Sessionとは

- セッション：ユーザ情報を，一時的にサーバ側で保持される情報のこと．例えばショッピングカート。
- ページをまたいでも，データが保持される．**どのページでも共通のデータを使いたい時**に用いる機能。
- (名前, 値)という形式で保管される。





# Sessionの基本構文

- 【セッションオブジェクトの宣言】

@Autowired

HttpSession **session**;

- 【セッションに値を格納】

session.setAttribute("属性名 (key) ", "値 (value) ")

- 【セッションに格納されている値の取得】

session.getAttribute("属性名 (key) ")

- 【セッションの無効化】

session.invalidate()

# Sessionの使い方①

- 【セッションオブジェクトの宣言】

@Autowired

HttpSession session;

```
@Controller
@RequestMapping("/admin/")
public class AdminLoginController {
    @Autowired
    AdminService adminService;
    /**
     * ログイン済みユーザ情報を格納するための
     * セッションオブジェクト
     */
    @Autowired
    HttpSession session;
```

# Sessionの使い方②

- 【セッションに値を格納】

`session.setAttribute("属性名 (key) ", "値 (value) ")`

```
@PostMapping("/login")
public String adminAuth(@RequestParam String adminEmail, @RequestParam String password, Model model) {
    //adminServiceクラスのfindByAdminNameAndPasswordメソッドを使用して、該当するユーザー情報を取得する。
    AdminEntity adminEntity = adminService.findByAdminEmailAndPassword(adminEmail, password);
    //adminEntity == null
    if(adminEntity == null) {
        //admin_login.htmlが表示される。
        return "/admin/admin_login.html";
    }else {
        //adminEntityの内容をsessionに保存する
        session.setAttribute("admin", adminEntity);
    }
}
```

Entityの内容をセッションに  
セットしている

# Sessionの使い方③

- 【セッションに格納されている値の取得】  
`session.getAttribute("属性名 (key) ")`

```
//adminEntityの内容をsessionに保存する
session.setAttribute("admin",adminEntity);

//adminServiceクラスのselectFindAll()を使用して、一覧を取得する。
List<AdminEntity>adminList = adminService.selectAdminAll();
//adminListをキーにしてadminListをitem_list.htmlに渡す。
model.addAttribute("adminList",adminList);

//セッションから管理者の情報を取得する。
AdminEntity auth = (AdminEntity) session.getAttribute("admin");
//管理者情報(AdminEntity)から管理者名を取得する。
String loginAdminName = auth.getAdminName();

model.addAttribute("loginAdminName",loginAdminName);
return "/admin/admin_all_view.html";
```

↑ セッションに格納している値の取得をしている。

# Sessionの使い方④

- 【セッションの無効化】  
`session.invalidate()`

```
//ログアウト処理
@GetMapping("/logout")
public String adminLogout() {
    //セッションの情報を削除する。
    session.invalidate();
    //admin_login.htmlが表示される。
    return "/admin/admin_login.html";
}
```

Sessionの無効化  
ログアウト処理などに使用



# Q&A

# まとめ

Sum Up



## 1. セキュリティ：

- 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. セッション：

- ① セッションの概念。
- ② セッションの書き方。



*Light in Your Career.*  
**THANK YOU!**