GeekBrains

Разработка интернет-магазина на Spring Framework

Полезные блоки

HTTP-сессии. Практика.

Оглавление

Работа с HTTP-сессией

Мы можем выбирать, каким образом **Spring Security** работает с HTTP-сессией. Для этого используется метод **HttpSecurity.sessionCreationPolicy()**:

- **always** — сессия всегда будет создаваться, если таковой еще нет;
- **isRequired** — сессия будет создана только по необходимости (этот режим включен по умолчанию);
- **never** — Spring никогда сам не создаст сессию, но будет использовать уже существующую;
- **stateless** — Spring Security не будет ни создавать, ни использовать сессию.

В настройках Spring Security это выглядит так:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
}
```

Только учтите, что эти настройки относятся только к Spring Security, а само веб-приложение может создать сессию и управлять ей при необходимости.

Перед выполнением аутентификации Spring Security запускает фильтр, ответственный за хранение **Security Context** между запросами, — **SecurityContextPersistanceFilter**. Context будет храниться в **HttpSessionSecurityContextRepository**, использующем HTTP-сессию как хранилище.

При указании stateless-режима такой вариант хранения будет заменен на **NullSecurityContextRepository**, и в таком случае сессия не будет создаваться и использоваться.

Можно настроить перенаправление пользователя при некорректной сессии или истечении срока ее действия.

```
http.sessionManagement()
  .expiredUrl("/sessionExpired")
  .invalidSessionUrl("/invalidSession");
```

Есть несколько дополнительных настроек безопасности для защиты **session cookie**:

- **httpOnly** — если **true**, скрипты браузера не будут иметь доступа к cookie;
- **secure** — если **true**, то cookie будут передаваться только по HTTPS-соединению.

Пример настройки:

```
public class WebAppInitializer implements WebApplicationInitializer {
    @Override
    public void onStartup(ServletContext servletContext) throws ServletException
{
        // ...
        servletContext.getSessionCookieConfig().setHttpOnly(true);
        servletContext.getSessionCookieConfig().setSecure(true);
    }
}
```

Или то же самое, только в **application.properties**:

```
server.servlet.session.cookie.http-only=true
server.servlet.session.cookie.secure=true
```

Бин может быть объявлен как сессионный — для этого достаточно в аннотации **@Scope** указать значение **session**:

```
@Component
@Scope("session")
public class AppBean {
   // ...
}
```

HTTP-сессию можно «заинжектить» в метод контроллера:

```
@GetMapping("/")
public void someMethod(HttpSession session) {
    Cart cart = new Cart();
    session.addAttribute("cart", cart);
}
```

Практическое задание

1. Реализовать сохранение покупок, которые пользователь добавил в корзину, в виде заказов, сохраняемых в БД.
2. * Подумать, возможно ли корзину реализовать через сессионный бин. Если возможно и целесообразно, то реализовать это в коде.

Дополнительные материалы

1. https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

Используемая литература

Для подготовки данного методического пособия литература не использовалась.