в предыдущих сериях

ранее использовали от спринга
mvc
spring core (отвечал за жизненный цикл бинов?) -- мы лишь расставляли аннотации, и после не использовали слово new чтобы создавать объекты а считали что спринг сам за нас их создаст когда надо будет (синглтоны) и пропихнет нужные зависимости -- это был IoC
частный случай IoC это DI
в основном мы его (ди) осуществляли через параметры конструкторов
это упрощает в частности вещи связанные с тестированием
иногда можем и через properties аннотацией autowired ???
использовали validation , jpa

теперь

будет больше абстракций, взаимосвязи между объектами предметной области

виды взаимосвязей

грубо говоря есть 4 способа как сущности могут быть взаимосвязаны:
`<many/one>-to-<many/one>`
для `one-to-one` `one-to-man` у чтобы хранить информацию достаточно поставить какую-нибудь колоночку (или две)
чтобы хранить `many-to-many` энивей придется завести отдельную таблицу

примеры

пользователи (как авторы) -- посты = `one-to-many`
посты -- пользователи = `many-to-one`
юзер -- социальная информация = `one-to-one`
юзеры -- группы юзеров = `many-to-many`

практика

```
@ManyToOne
@JoinColumn(name = "user_id", nullable = false)
private User user;
```

что означает:
у объекта `Post` есть мэппинг many-to-one с `User`
и осуществлять этот мэппинг нужно через дополнительную колонку с именем `user_id`

```
@OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
private List<Post> posts;
```

что означает:
отображение осуществляется обратно через объект `user`

```
@ManyToMany
@JoinTable(name = "user_role",
        joinColumns = @JoinColumn(name = "user_id", nullable = false),
        inverseJoinColumns = @JoinColumn(name = "role_id", nullable = false))
private Set<Role> roles;
```

дополнительная таблица `user_role` , имена колонок

делаем аннотацию guest

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Guest {
}
```

"не нужно удалят эту аннотацию после компиляции, она нужна будет в рантайме"
"аннотация на методы"

```
@Guest
@PostMapping("/enter")
public String register(@Val
                       Bind
                       Http
    if (bindingResult.hasEr
        return "EnterPage";
    }

    setUser(httpSession, us
    setMessage(httpSession,

    return "redirect:";
}
```

ставим ее вот в таких местах

interceptor

interceptor -- типа фильтра будет перехватывать доступ к методам там и т п

```
@Component
public class SecurityInterceptor implements HandlerInterceptor {
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Obj
        return HandlerInterceptor.super.preHandle(request, response, handler);
    }
}
```

мы заставим этот класс всегда когда дергаем какой-то метод из Page (нашего контроллера) вызываться

```
ɔlic class SecurityInterceptor implements HandlerInterceptor {
  @Override
  public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
                           Object handler) throws Exception {
      if (handler instanceof HandlerMethod) {
          Method method = ((HandlerMethod) handler).getMethod();
          if (Page.class.isAssignableFrom(method.getDeclaringClass())) {
              if (method.getAnnotation(Guest.class) == null) {
                  ??
              }
          }
      }

      return true;
```

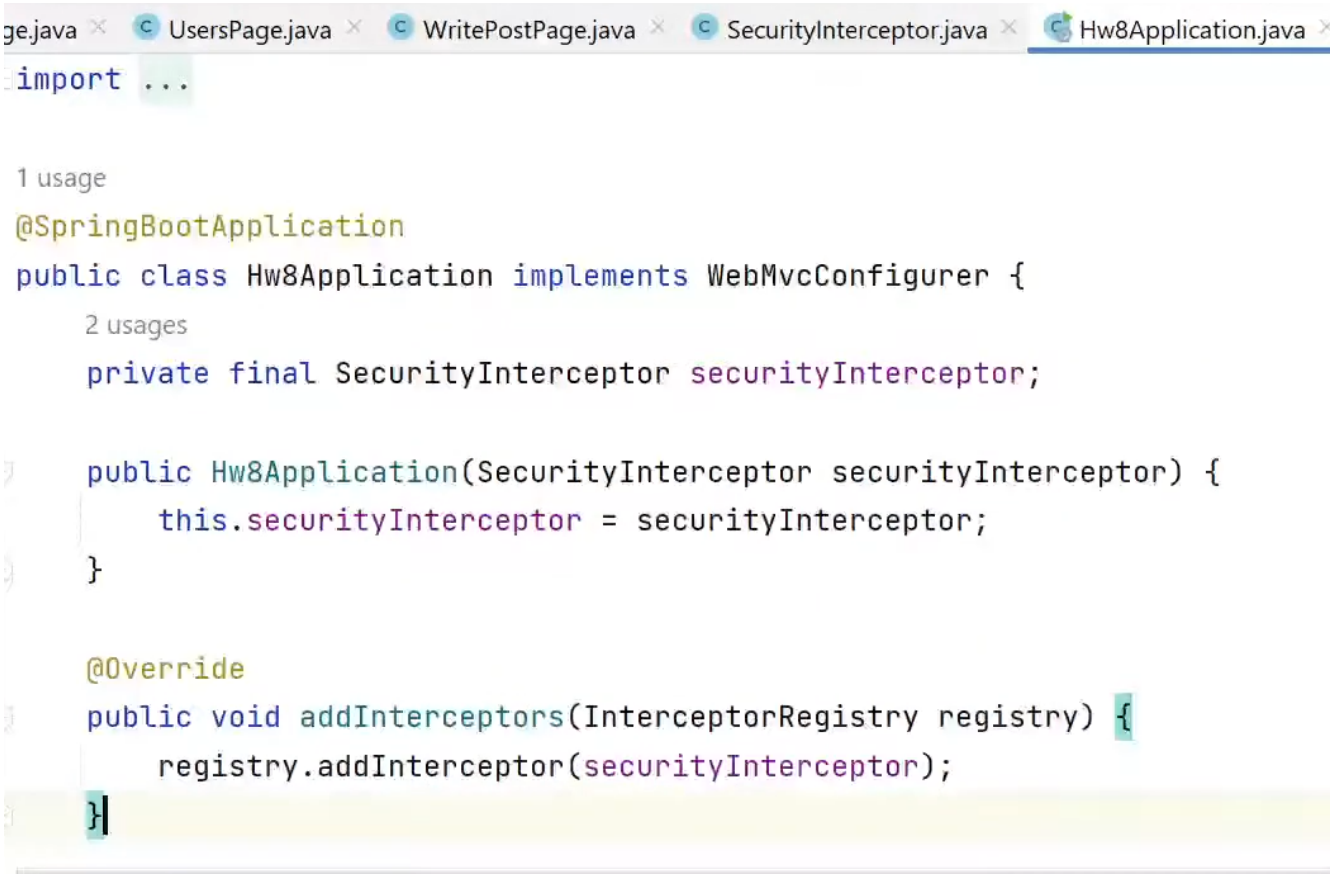
```
    if (handler instanceof HandlerMethod) {
        Method method = ((HandlerMethod) handler).getMethod();
        if (Page.class.isAssignableFrom(method.getDeclaringClass())) {
            if (method.getAnnotation(Guest.class) == null) {
                User user = indexPage.getUser(request.getSession());

                if (user == null) {
                    indexPage.setMessage(request.getSession(), message: "Enter into th
                    response.sendRedirect( location: "/enter");
                    return false;
                }
            }
        }
    }
```

если мы поймали метод
-> если метод определяется из `Page`
--> если нет аннотации `@Guest`
---> залетаем в `indexPage` *(но в проде так делать не надо)*, используем его функцию `getUser(request)` и смотрим залогинен ли юзер, возвращаем `true` или `false`
#Note сам `request`, `request.getSession()` у нас есть итак, мы обращаемся к `indexPage` только чтобы она по сути распарсила данные и посмотрела для нас в них Юзер сессию.

`true` если все ок, можно передавать процесс обработки дальше в цепочку
`false` -- вызов штуки не будет осуществляться
ну короче прям литералли фильтр просто в других цепочках обработки

донастройка interceptor-a



вот она
вроде типа добавляем интерцептор в регистр всего

делаем роли

через аннотации и бд
есть доменный объект `Role` со связями `many-to-many` с юзерами

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface AnyRole {
    Role.Name[] value() default {};
}
```

аннотация

```
@AnyRole(Role.Name.WRITER)
@GetMapping("/writePost")
public String writePostGet(
```

расставляем аннотацию

и прокачиваем интерцептор

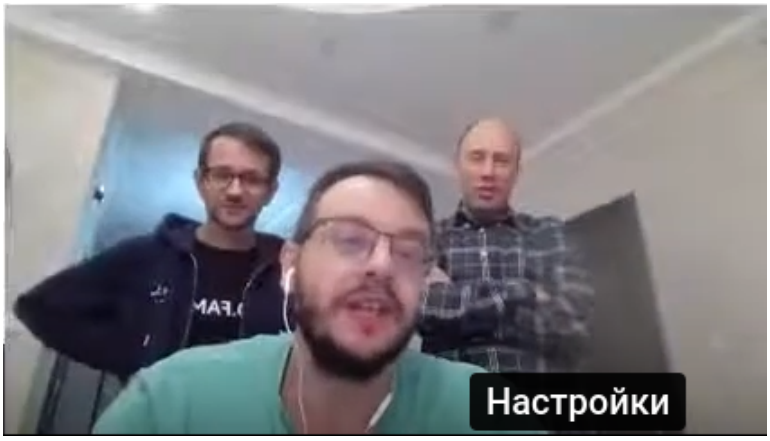
еще там какую то штуку в юзер сервисе пишем чтобы ьона проверяла что все роли есть в бд?

про ленивость

если есть мэппинг `чтото-to-many` то используются коллекции которые не хочется доставать на каждый чих поэтому достаются они всегда лениво, по требованию (on demand)
это приводит к тому что в пользователе которого мы достали из бд и который отображается как экземпляр класса нет готового списка постов есть только что-то типа динамического прокси который при попытке обращения к соответствующему геттеру подгрузит посты
такова стратегия по умолвчанию

SecurityInterceptor живет в не совсем обычный момент жизни приложения, и в нем подключение к бд находится в несколько другом состоянии нежели в момент жизни контроллеров и поэтому нельзя порезолвить ленивость
как фиксится:
указываем стратегию чтобы сет ролей всегда сразу доставался

```
@ManyToMany(fetch = FetchType.EAGER)
@JoinTable(name = "user_role",
        joinColumns = @JoinColumn(nam
        inverseJoinColumns = @JoinCol
private Set<Role> roles;
```



правило трех сигм

note про DI

видимо классический DI:

```
private final IndexPage indexPage;

public SecurityInterceptor(IndexPage indexPage) {
    this.indexPage = indexPage;
}
```

он здесь примерно это сказал

ИТОГ

по сути в этом занятии глубже познакомились с JPA
вспомнили отношения между парами объектов и как мы можем их декларативно размечать