

Spring MVC (часть 3)



ПРЕПОДАВАТЕЛЬ



Юрий Костяной

Java/Kotlin backend-разработчик

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

ПЛАН ЗАНЯТИЯ

1. Повторение
2. Основной блок
3. Вопросы по основному блоку
4. Домашняя работа

1

ОСНОВНОЙ БЛОК

Введение

- Проблемный html
- Перехватить и просеять
- Новый взгляд
- CRUDучись



Проблема

Актуальная версия стандарта, описывающего HTML – это HTML 5. Данный стандарт применяется во всех современных браузерах. Но стандарт HTML 5 поддерживает только [POST и GET](#) запросы, в то время как REST и его протокол http предусматривают применение множества других методов запросов.

Например, если мы передаём форму для редактирования профиля пользователя, то отправляемый формой запрос может быть только POST, а для нашего приложения такой запрос является PUT.

Как устранить данное несоответствие?

HTML в реальной жизни:



Проблемный html

Шаги решения

Шаг 1. В форму html-страницы, которую запрашивает клиент добавляется скрытое поле, хранящее значение http-метода, который должен указываться в запросе формы.

```
<h1>Edit user</h1>
<form th:object="${userForm}"
      th:action="@{/view/register}"
      method="post">
  <input type="hidden" name="_method" value="put">
  <div>
    <label for="firstName">First name</label>
    <input id="firstName" type="text" th:field="*{firstName}">
    <p th:if="${#fields.hasErrors('firstName')}"
      th:text="${#strings.listJoin(#fields.errors('firstName'), ', ')}"></p>
  </div>
  ...
  <button type="submit">Create userDto</button>
</form>
```



Проблемный html

Шаги решения

Thymeleaf делает этот трюк автоматически. Для этого достаточно вместо стандартного тего *method* нужно использовать *th:method=«НУЖНЫЙ_МЕТОД»*.

```
<h1>Edit user</h1>
<form th:object="${userForm}"
      th:action="@{/view/register}"
      th:method="PATCH"

  <div>
    <label for="firstName">First name</label>
    <input id="firstName" type="text" th:field="*{firstName}">
    <p th:if="${#fields.hasErrors('firstName')}"
      th:text="${#strings.listJoin(#fields.errors('firstName'), ', ')}"></p>
  </div>

  ...
  <button type="submit">Create userDto</button>
</form>
```



Проблемный html

Шаги решения

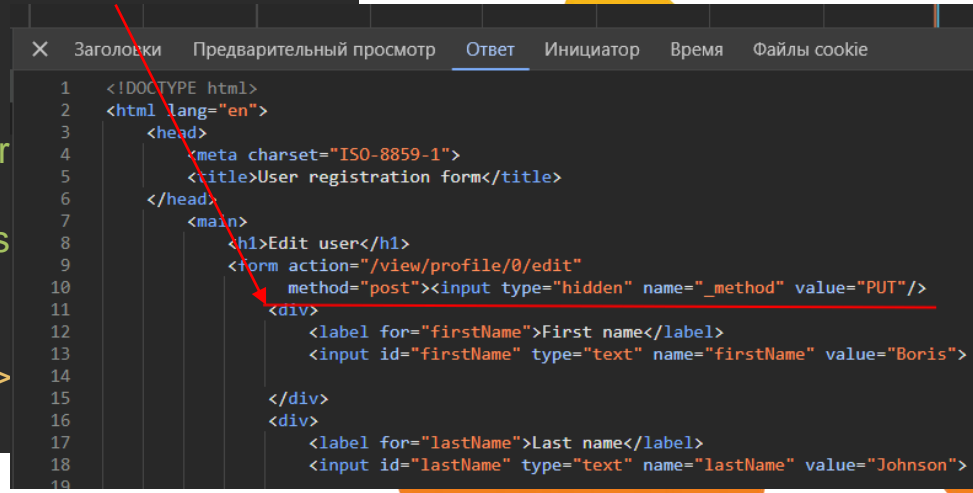
Thymeleaf делает этот трюк автоматически. Для этого достаточно вместо стандартного тего *method* нужно использовать *th:method=«НУЖНЫЙ_МЕТОД»*.

```
<h1>Edit user</h1>
<form th:object="${userForm}"
      th:action="@{/view/register}"
      th:method="PATCH">

  <div>
    <label for="firstName">First name</label>
    <input id="firstName" type="text" th:field="*{firstName}" />
    <p th:if="${#fields.hasErrors('firstName')}"
        th:text="${#strings.listJoin(#fields.errors('firstName'), ', ')}">
    </p>
  </div>

  ...

  <button type="submit">Create userDto</button>
</form>
```



Заголовки Предварительный просмотр Ответ Инициатор Время Файлы cookie

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="ISO-8859-1">
5     <title>User registration form</title>
6   </head>
7   <main>
8     <h1>Edit user</h1>
9     <form action="/view/profile/0/edit"
10        method="post"><input type="hidden" name="_method" value="PUT"/>
11       <div>
12         <label for="firstName">First name</label>
13         <input id="firstName" type="text" name="firstName" value="Boris">
14       </div>
15       <div>
16         <label for="lastName">Last name</label>
17         <input id="lastName" type="text" name="lastName" value="Johnson">
18       </div>
19     </main>
```

Проблемный html

Шаги решения

Шаг 2. Настроить фильтр на стороне web-приложения. Для этого в `MyWebAppInitializer` нужно переопределить метод `onStartup` и добавить в `ServletContext` фильтр. Класс фильтра уже написан в *Spring* – это класс `HiddenHttpMethodFilter`.



SpringMvcHtmlFormProblemExample.zip



Задание

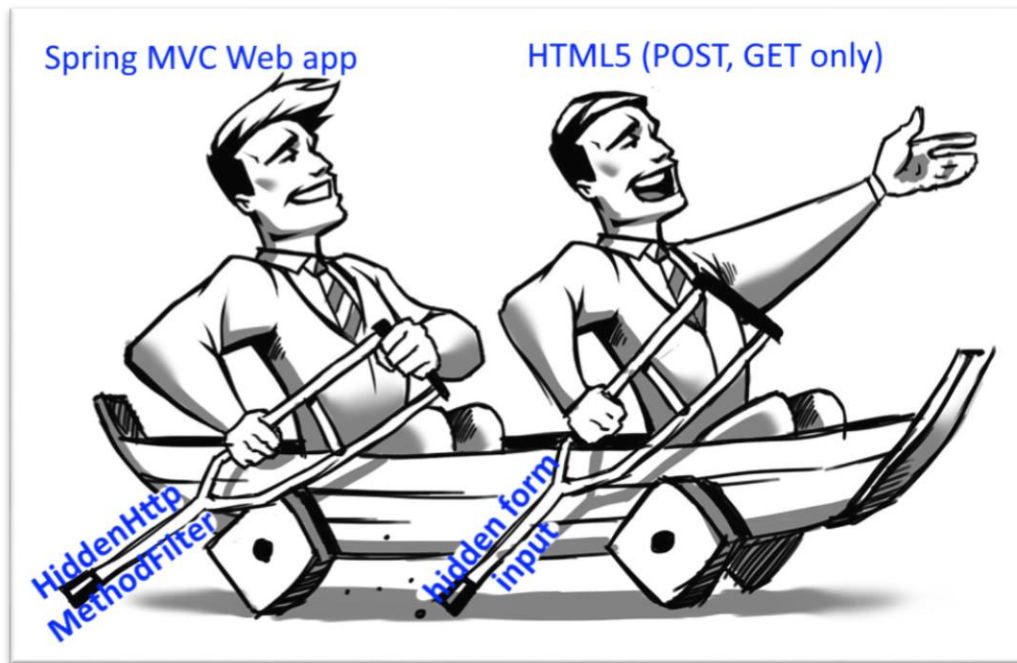
В проекте `SpringMvcHtmlFormProblemExample`, который был дополнен фильтром, реализуйте представление для пользователя, содержащее форму с кнопкой Delete и выполняющую http-метод DELETE. По нажатию на кнопку из хранилища данных должен удаляться пользователь с указанным в URL идентификатором.



Проблема

Проблема с HTML 5 – это проблема несовершенства технологий, которую приходится решать с помощью «костыля», использующего фильтр.

Но есть и штатные проблемы, которые нужно решать перехватом данных запроса. Например, если клиент делает запрос к серверу, но сперва запрос обрабатывается на некотором проху, а только потом обрабатывается бэкендом.

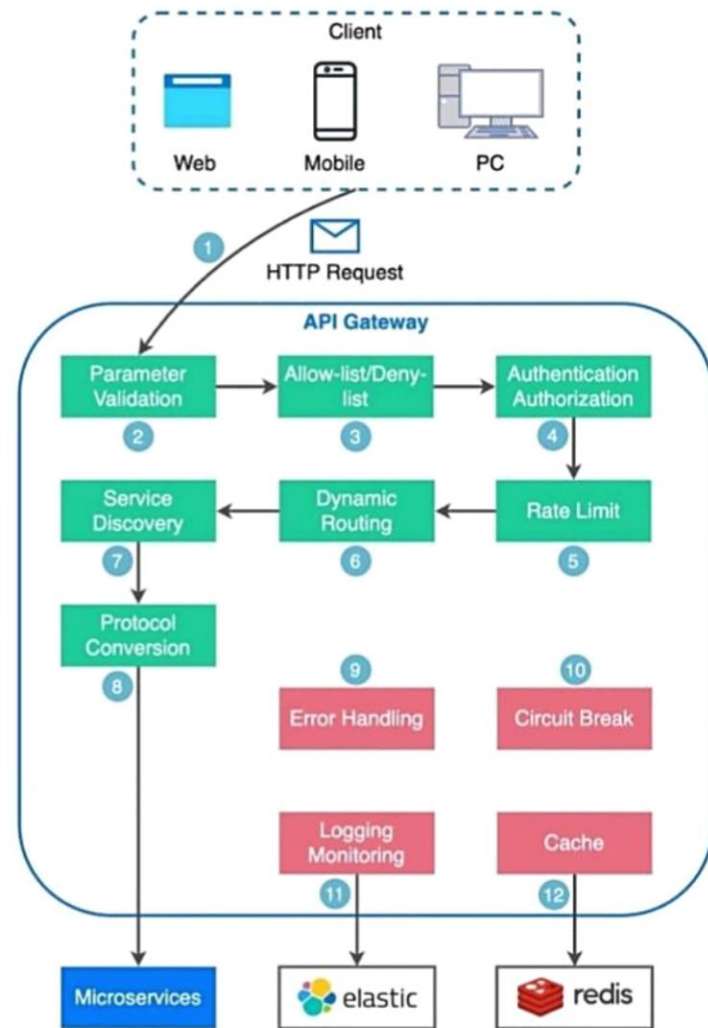


Проблема

На уровне микросервисной архитектуры такие проблемы решают с помощью дополнительного сервиса – API Gateway, который объединяет все микросервисы вашего приложения с помощью единого API (паттерн [Фасад](#)).

Но и на уровне отдельного сервера есть возможность выполнять предобработку запроса.

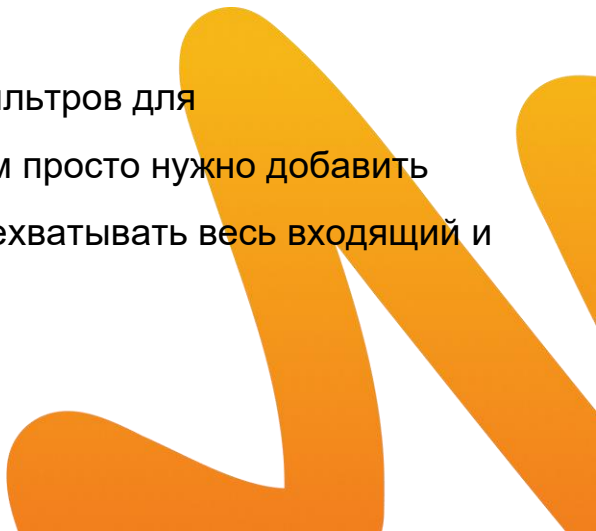
Каким образом можно создавать и настраивать фильтры и перехватчики в Spring MVC?



Перехватить и просеять Фильтры

Фильтры – это классы, обрабатывающие входящие запросы. Они являются частью веб-сервера, а не среды *Spring*. Для входящих запросов мы можем использовать фильтры, чтобы манипулировать запросами и даже блокировать запросы от достижения любого сервлета. Фильтры могут работать и с ответами сервера и, например, блокировать отправку ответа клиенту.

Spring Security (изучим позже) - отличный пример использования фильтров для аутентификации и авторизации. Чтобы настроить *Spring Security*, нам просто нужно добавить один фильтр, *DelegatingFilterProxy*. Затем *Spring Security* может перехватывать весь входящий и исходящий трафик.



Перехватить и просеять

Цепочка обязанностей

Чтобы понять, что такое фильтры, нужно изучить паттерн [Цепочка обязанностей](#) — поведенческий паттерн проектирования, который позволяет передавать запросы последовательно по цепочке обработчиков. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи.



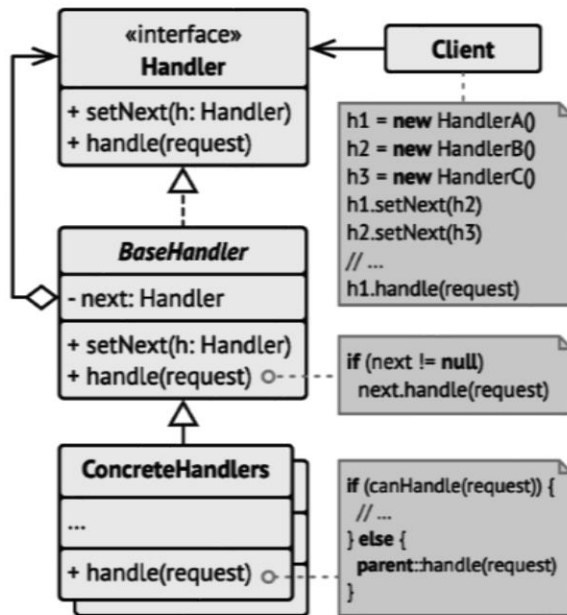
Перехватить и просеять

Цепочка обязанностей

1 **Обработчик** определяет общий для всех конкретных обработчиков интерфейс. Обычно достаточно описать единственный метод обработки запросов, но иногда здесь может быть объявлен и метод выставления следующего обработчика.

2 **Базовый обработчик** — опциональный класс, который позволяет избавиться от дублирования одного и того же кода во всех конкретных обработчиках.

Обычно этот класс имеет поле для хранения ссылки на следующий обработчик в цепочке. Клиент связывает обработчики в цепь, подавая ссылку на следующий обработчик через конструктор или сеттер поля. Также здесь можно реализовать базовый метод обработки, который бы просто перенаправлял запрос следующему обработчику, проверив его наличие.



4 **Клиент** может либо сформировать цепочку обработчиков единожды, либо перестраивать её динамически, в зависимости от логики программы. Клиент может отправлять запросы любому из объектов цепочки, не обязательно первому из них.

3 **Конкретные обработчики** содержат код обработки запросов. При получении запроса каждый обработчик решает, может ли он обработать запрос, а также стоит ли передать его следующему объекту.

В большинстве случаев обработчики могут работать сами по себе и быть неизменяемыми, получив все нужные детали через параметры конструктора.

Перехватить и просеять **Filter**

Шаги создания фильтра:

1. создаем класс, реализующий интерфейс **`javax.servlet.Filter`**.
2. переопределяем метод *`doFilter`*, с помощью которого мы можем получить доступ или управлять объектами *`ServletRequest`*, *`ServletResponse`* и *`FilterChain`*. Последний позволяет разрешать или блокировать запросы.
3. добавляем фильтр в контекст *`Spring`* в настройках *`MyWebAppInitializer`*. В *`Spring Boot`* достаточно поставить аннотацию *`@Component`* в классе фильтра и *`Spring`* самостоятельно внедрит бин фильтра в цепочку.

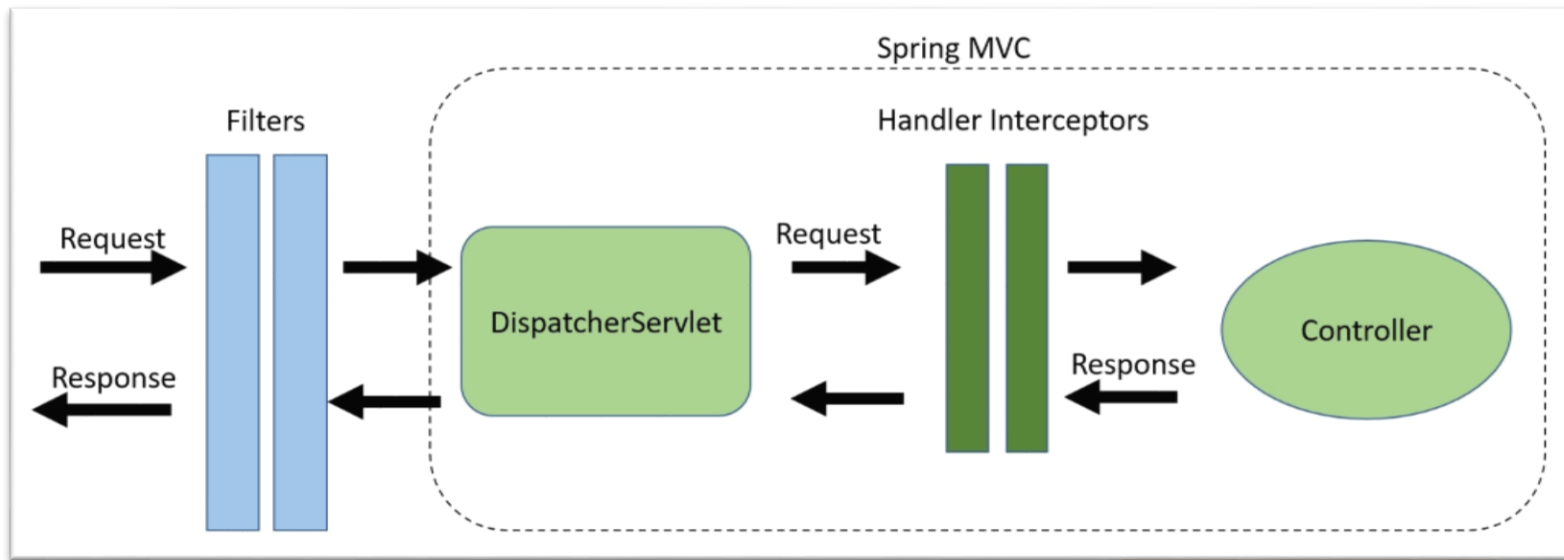


Перехватить и просеять

Filter vs HandlerInterceptors

Другой способ перехвата запроса – это interceptors (классы-перехватчики). Класс

HandlerInterceptors – это часть модуля Spring MVC, который контролирует запрос на этапе между *DispatcherServlet* и контроллерами, т.е. перехватывает запросы до их поступления в контроллер, до и после подготовки представления (*view*).



Перехватить и просеять

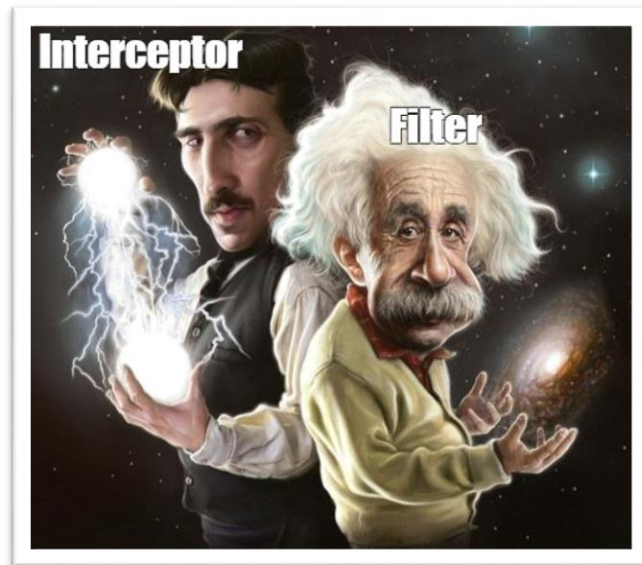
Filter vs HandlerInterceptor

Фильтры перехватывают запросы до того, как они достигают DispatcherServlet, что делает их идеальными для крупномасштабных задач:

- Аутентификация
- Ведение журнала и контроль
- Сжатие изображений и данных
- Любая функциональность, которую мы хотим отделить от Spring MVC

Перехватчики предоставляют доступ к объектам *Handler* и *ModelAndView*. Это уменьшает дублирование кода и обеспечивает:

- обработку сквозных задач (ведение журнала приложений)
- подробные проверки авторизации
- манипулирование контекстом *Spring* или моделью



Перехватить и просеять **HandlerInterceptors**

Шаги создания перехватчика:

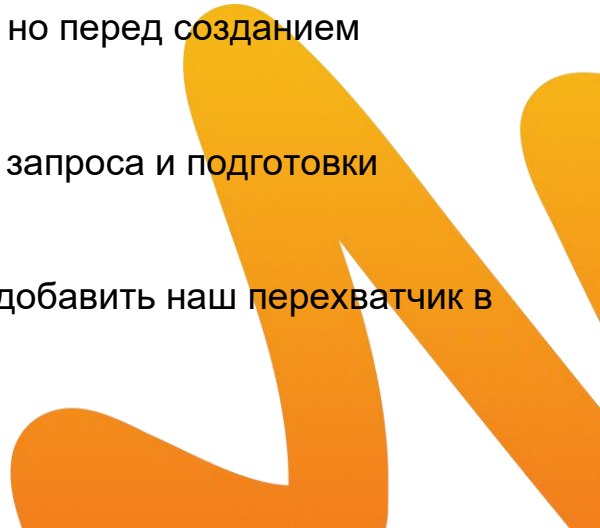
1 создать класс, который имплементирует интерфейс

org.springframework.web.servlet.HandlerInterceptor.

2 переопределить методы

- *preHandle()* – выполняется перед вызовом целевого обработчика
- *postHandle()* – выполняется после вызова целевого обработчика, но перед созданием представления для *DispatcherServlet*.
- *afterCompletion()* – обратный вызов после выполнения обработки запроса и подготовки представления.

3 В конфигурации переопределить метод *addInterceptors*, в котором добавить наш перехватчик в *registry*.



Перехватить и просеять

Пример реализации фильтра и перехватчика



SpringMvcFilterAndInterceptorExample.zip



Задание

В проекте `SpringMvcFilterInterceptorExample`. Добавьте фильтры, которые блокируют запрос и ответ по наличию заголовков *x-block-this-request* и *x-block-this-response* соответственно. Эти фильтры будут проверять наличие соответствующих заголовков и, если заголовок присутствует, блокировать обработку запроса или модификацию ответа.



Проблема

При изучении Spring Core мы говорили о том, что существуют области видимости бинов (scope): singleton и prototype. Но работа с REST предполагает отсутствие состояния у сервиса, значит, нам требуется пересоздавать некоторые бины для каждого запроса.

Кроме того, мы говорили о сессии взаимодействия клиента и сервера., т.е. бины нужны на время жизни сессии.

Следить за всеми бинами в контексте вручную очень сложно. Есть ли способ это автоматизировать?

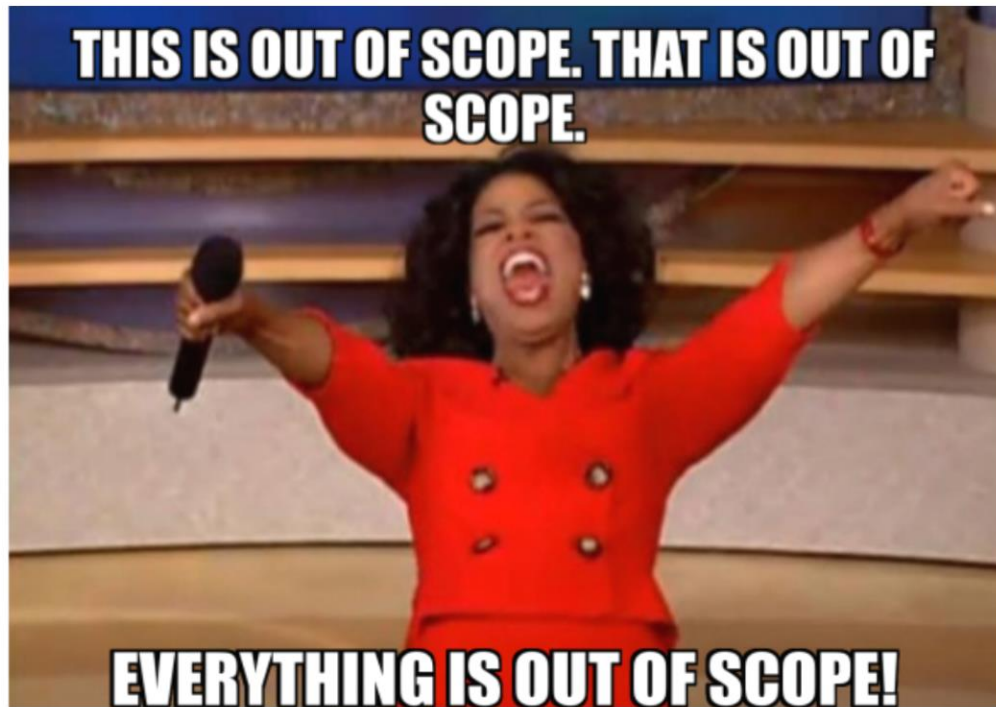


НОВЫЙ ВЗГЛЯД

Scope в Spring MVC

Spring MVC приложение, помимо ранее изученных *singleton* и *prototype*, поддерживает следующие веб-ориентированные области видимости бинов:

- *request*
- *session*
- *application*
- *websocket*



Новый взгляд Scope запроса

```
@Bean
@Scope(
    value = WebApplicationContext.SCOPE_REQUEST,
    proxyMode = ScopedProxyMode.TARGET_CLASS
)
public MessageGenerator helloMessage() {
    return new MessageGenerator();
}
```

Атрибут *proxyMode* нужен для того, чтобы в момент инициализации контекста приложения Spring создал прокси-объект (заглушку, см. паттерн [Заместитель](#)), который будет помещён в контекст вместо исходного бина. В момент появления запроса реальный бин будет создан и подставлен в заместителя, который делегирует все вызовы настоящему бину.

Вместо настройки аннотации `@Scope` можно воспользоваться готовой аннотацией **`@RequestScope`**.

Новый взгляд Scope сессии

Такой бин будет существовать в рамках http-сессии, т.е. хранить данные между запросами пользователя.

Для работы с атрибутами сессии существует аннотация *@SessionAttribute*.

```
@Bean
@Scope(
    value = WebApplicationContext.SCOPE_SESSION,
    proxyMode = ScopedProxyMode.TARGET_CLASS
)
// @SessionScope
public MessageGenerator sessionMessageGenerator() {
    return new MessageGenerator();
}
```

Scope приложения

Scope приложения создаёт бины для жизненного цикла *ServletContext*.

Он похож на *singleton*, но бин доступен многим приложениям на основе сервлетов в пределах сервера, а не одному приложению, как *singleton*.

```
@Bean
@Scope(
    value = WebApplicationContext.SCOPE_APPLICATION,
    proxyMode = ScopedProxyMode.TARGET_CLASS
)
// @ApplicationScope
public MessageGenerator appMessageGenerator() {
    return new MessageGenerator();
}
```

Новый взгляд Scope web-сокета

Современные приложения поддерживают взаимодействие через **WebSocket** – технологию, которая позволяет клиенту установить двухстороннюю («дуплексную») связь с сервером, что упрощает взаимодействие и снижает объём передаваемых данных.

Данная технология является альтернативой классическому REST и применяется в основном для работы с браузерами.

Мы можем настроить область видимости бина в рамках существования веб-сокета.

```
@Bean
@Scope(scopeName = "websocket", proxyMode = ScopedProxyMode.TARGET_CLASS)
public MessageGenerator websocketMessageGenerator() {
    return new MessageGenerator();
}
```

НОВЫЙ ВЗГЛЯД

Пример использования разных scope



SpringMvcScopesExample.zip



Задание

Создайте Spring MVC приложение с конфигурацией в Java-коде. В приложении создайте get-метод, который возвращает количество обращений к методу в рамках сессии. Когда счётчик обращений доходит до 5, начинается новая сессия.



2

Домашнее задание

Домашнее задание

1 Дополните проект, который мы использовали на лекции для работы с Users.

1.1 Добавьте в сущность User поле status. Создайте форму для установки статуса. Форма должна использовать метод PATCH. После обновления статуса пользователя происходит перенаправление на страницу его профиля, где должен отображаться также статус.

1.2 Напишите перехватчик, который логирует детали всех входящих HTTP запросов и ответов на них (URL, параметры, тело, заголовки и т.д.).

1.3 Используя Session scope бин, разработайте механизм, который автоматически определяет, новый пользователь это или возвращающийся, и выводит соответствующее приветствие на главной странице сайта – «Welcome, new user!» или «Welcome back!».

ЗАКЛЮЧЕНИЕ

