

Chapter-9:

Using the Spring web scopes

Upcode Software
Engineer Team



CONTENT

1. Using the request scope in a Spring web app
 - 1.1. Key aspects of request-scoped beans
2. Using the session scope
3. Using the application scope
4. Conclusion
5. References



USING REQUEST SCOPE IN A SPRING WEB APP

Области веб-видимости бинов Spring

-Область видимости в рамках запроса — Spring создает отдельный экземпляр класса бина для каждого HTTP-запроса. Конкретный экземпляр существует только для конкретного HTTP-запроса;



USING REQUEST SCOPE IN A SPRING WEB APP

Мы рассмотрим пример, где создадим функционал для аутентификации, где пользователю предоставляется возможность зарегистрироваться и войти в учетную запись.

Этап 1

Построение логики аутентификации

Если пользователь предоставляет правильные учетные данные, то приложение распознает его и подтверждает успешную аутентификацию.

Нам нужно, чтобы Spring сохранял учетные данные пользователя в памяти приложения не дольше, чем необходимо для выполнения запроса на аутентификацию, поэтому для реализации этого функционала мы будем использовать бин с областью видимости в рамках запроса





USING REQUEST SCOPE IN A SPRING WEB APP

Этап 2



Хранение данных об аутентифицированном пользователе

После того как пользователь аутентифицировался в системе, введя правильные учетные данные, нужно в течение какого-то времени хранить статус входа выполненным.

Чтобы сохранить данные о пользователе и в течение достаточно длительного времени помнить, что данный пользователь аутентифицирован в приложении, мы будем использовать HTTP-сессию и бин с областью видимости в рамках этой сессии



Этап 3

Подсчет запросов на аутентификацию

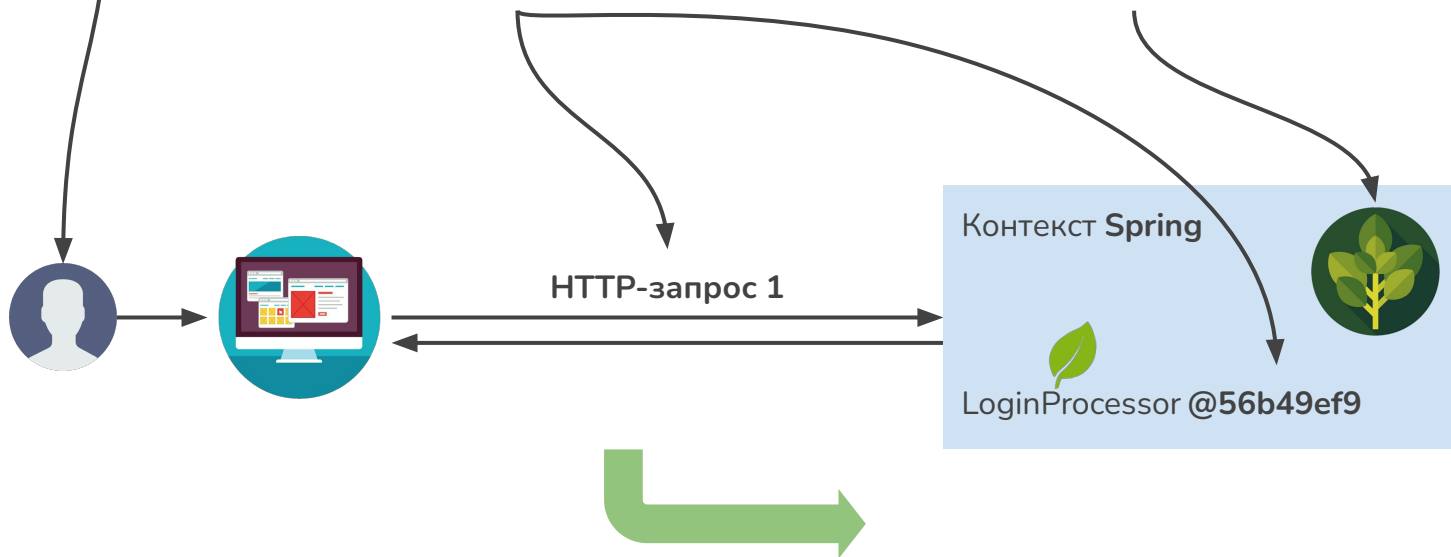
Наконец, мы хотим, чтобы приложение учитывало все запросы пользователей на аутентификацию. Чтобы реализовать этот функционал, нужно воспользоваться бином с областью видимости в рамках приложения.

USING REQUEST SCOPE IN A SPRING WEB APP

Пользователь

Для первого HTTP-запроса Spring создает экземпляр бина LoginProcessor с областью видимости в рамках запроса

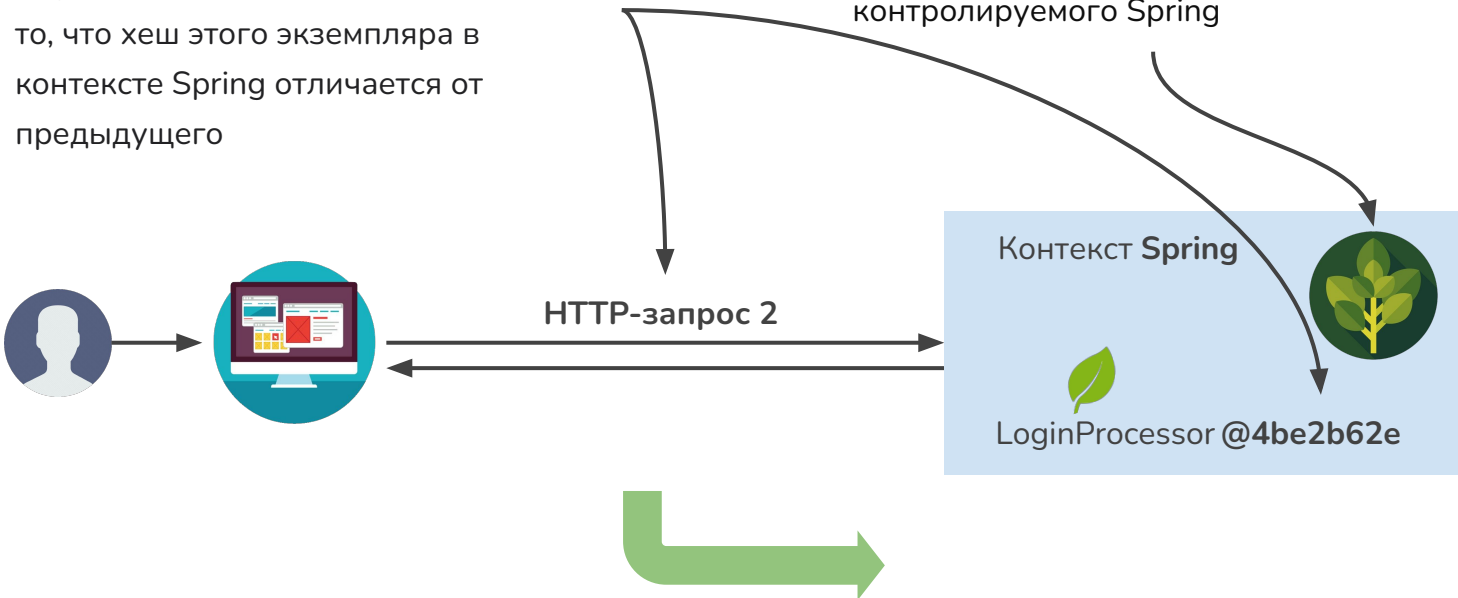
Spring управляет типом бина и создает новый экземпляр для каждого HTTP-запроса. На этом рисунке дерево изображает тип бина, контролируемого Spring



USING REQUEST SCOPE IN A SPRING WEB APP

Для второго HTTP-запроса Spring создает другой экземпляр бина LoginProcessor. Обратите внимание на то, что хеш этого экземпляра в контексте Spring отличается от предыдущего

Spring управляет типом бина и создает новый экземпляр для каждого HTTP-запроса. На этом рисунке дерево изображает тип бина, контролируемого Spring





USING REQUEST SCOPE IN A SPRING WEB APP



Для каждого HTTP-запроса Spring создает новый экземпляр бина с областью видимости в рамках запроса. Используя такие бины, вы можете быть уверены, что данные, сохраняемые в нем, будут доступны только для того HTTP-запроса, для которого бин был создан. Spring управляет типом бина (кофейное дерево) и использует его, чтобы получать экземпляры (кофейные зерна) для каждого следующего запроса.



USING REQUEST SCOPE IN A SPRING WEB APP

Главные особенности бинов с областью видимости в рамках запроса
(Key aspects of request-scoped beans)(1/2)

Факты	Следствия	Что учесть	Чего избегать
Spring создает новый экземпляр бина для каждого HTTP-запроса от каждого клиента	В процессе выполнения приложения Spring создает в его памяти множество экземпляров этого бина	Как правило, количество экземпляров не является большой проблемой, так как время их жизни невелико. Они нужны приложению не дольше, чем выполняется HTTP-запрос. После завершения HTTP-запроса приложение уничтожает эти экземпляры и их подбирает сборщик мусора	Главное — проследить, чтобы в таких запросах не выполнялась затратная по времени логика, обычно необходимая Spring для создания экземпляров (такая как получение данных из базы данных или вызов функции по сети). Старайтесь не писать логику в конструкторах таких бинов или методах с аннотацией <code>@PostConstruct</code>



USING REQUEST SCOPE IN A SPRING WEB APP

Главные особенности бинов с областью видимости в рамках запроса
(Key aspects of request-scoped beans) (2/2)

Факты	Следствия	Что учесть	Чего избегать
Экземпляр бина с областью видимости в рамках запроса доступен только одному запросу	Экземпляры бинов с областью видимости в рамках запроса не годятся для многопоточных задач, так как доступны только для одного потока (того, к которому принадлежит запрос)	В атрибутах такого экземпляра можно сохранять данные, используемые в запросе	Не используйте методы синхронизации для атрибутов таких бинов. Эти методы не сработают и лишь снизят производительность приложения



USING REQUEST SCOPE IN A SPRING WEB APP

Реализация(1/5)

Создадим проект Spring Boot и внедрим необходимые зависимости:

Maven

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```



USING REQUEST SCOPE IN A SPRING WEB APP

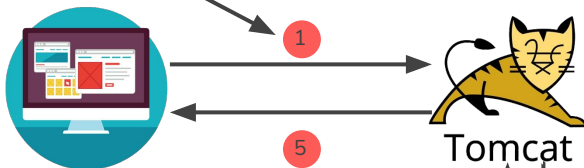
Реализация(2/5)

Нам нужно создать контроллер и представление. В контроллере мы сформулируем действие, которое будет определять, являются ли учетные данные, переданные в запросе аутентификации, достоверными. Контроллер передает сообщение в представление, а представление выводит это сообщение на экран

USING REQUEST SCOPE IN A SPRING WEB APP

Реализация(3/5)

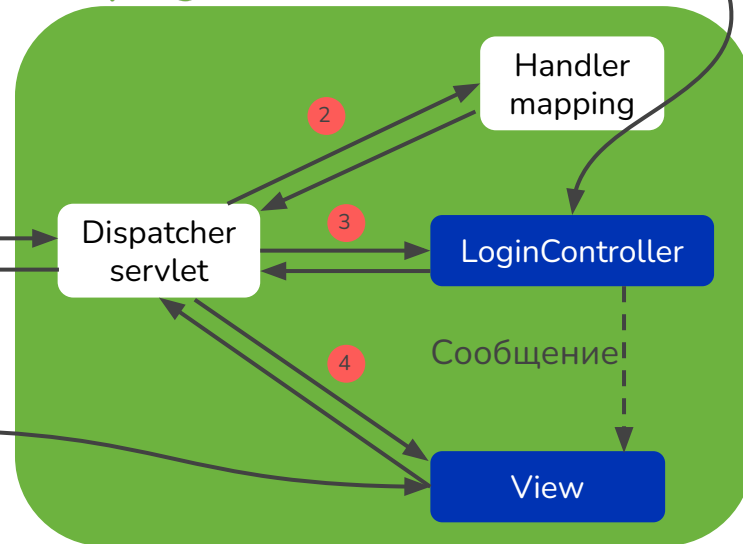
HTTP-запрос POST
/?username=natalie&password=password



Контроллер определяет, являются ли учетные данные достоверными, и по результатам выполнения этой логики передает соответствующее сообщение в представление

Клиент передает HTTP-запрос с учетными данными пользователя

В представлении выводится сообщение, полученное от контроллера



USING REQUEST SCOPE IN A SPRING WEB APP

Реализация(4/5)

Код страницы аутентификации login.html

Определяем префикс th, чтобы
использовать возможности
шаблонизатора Thymeleaf

```
<!DOCTYPE html>
```

```
<html lang="en" xmlns:th="http://www.thymeleaf.org" >
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <title>Login</title>
```

```
</head>
```

```
<body>
```

```
<form action="/" method="post">
```

```
  Username: <input type="text" name="username" /><br />
```

```
  Password: <input type="password" name="password" /><br />
```

```
</form>
```

```
  <button type="submit">Log in</button>
```

```
</body>
```

```
<p th:text="${message}" ></p>
```

```
</html>
```

Под HTML-формой выводим сообщение с результатом
запроса аутентификации

Создаем HTML-форму для отправки учетных данных
пользователя на сервер

Поля ввода учетных данных
— имени пользователя и
пароля

Когда пользователь щелкнет на
кнопке Submit, клиент создаст
HTTP-запрос типа POST с
учетными данными этого
пользователя



USING REQUEST SCOPE IN A SPRING WEB APP

Реализация(5/5)

Действие контроллера, связанное с корневым путем приложения



USING REQUEST SCOPE IN A SPRING WEB APP

Результат:

Приложение выводит форму, в которую пользователь вносит свои учетные данные для аутентификации

Форма аутентификации:

Username:

Password:

Если введенные пользователем учетные данные неверны, то приложение сообщает о неудачной попытке аутентификации

ДА

Учетные данные верны?

НЕТ

Username:

Password:

You are now logged in

Если введенные пользователем учетные данные верны, то приложение сообщает пользователю, что он аутентифицирован

Username:

Password:

Login failed!



USING REQUEST SCOPE IN A SPRING WEB APP

Действие контроллера, выполняющее аутентификацию(1/6)

```
@Controller
public class LoginController {
    @GetMapping("/")
    public String loginGet () {"login.html";}

    @PostMapping("/")
    public String loginPost (RequestParam String username,
                             RequestParam String password, Model model) {
        boolean loggedIn = false;
        if (loggedIn) {
            model.addAttribute("message", "You are now logged in.");
        } else {
            model.addAttribute("message", "Login failed!");
        }
        return "login.html";
    }
}
```

Связываем действие контроллера с
HTTP-запросом типа POST,
отправляемым со страницы
аутентификации



USING REQUEST SCOPE IN A SPRING WEB APP

Действие контроллера, выполняющее аутентификацию(2/6)

Извлекаем учетные данные из параметров HTTP-запроса

```
@Controller
public class LoginController {
    @GetMapping("/")
    public String loginGet () {"login.html";}

    @PostMapping ("/")
    public String loginPost (RequestParam String username,
                             RequestParam String password, Model model) {
        boolean loggedIn = false;
        if (loggedIn) {
            model.addAttribute("message", "You are now logged in.");
        } else {
            model.addAttribute("message", "Login failed!");
        }
        return "login.html";
    }
}
```

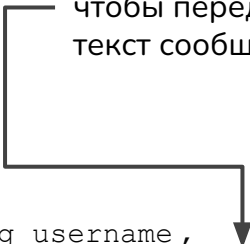
USING REQUEST SCOPE IN A SPRING WEB APP

Действие контроллера, выполняющее аутентификацию(3/6)

```
@Controller
public class LoginController {
    @GetMapping("/")
    public String loginGet () {"login.html";}

    @PostMapping ("/")
    public String loginPost (RequestParam String username,
                             RequestParam String password, Model model) {
        boolean loggedIn = false;
        if (loggedIn) {
            model.addAttribute("message", "You are now logged in.");
        } else {
            model.addAttribute("message", "Login failed!");
        }
        return "login.html";
    }
}
```

Объявляем параметр типа Model,
чтобы передавать в представление
текст сообщения





USING REQUEST SCOPE IN A SPRING WEB APP

Действие контроллера, выполняющее аутентификацию(4/6)

```
@Controller
public class LoginController {
    @GetMapping("/")
    public String loginGet () {"login.html";}

    @PostMapping ("/")
    public String loginPost (RequestParam String username,
                             RequestParam String password, Model model) {
        boolean loggedIn = false;
        if (loggedIn) {
            model.addAttribute("message", "You are now logged in.");
        } else {
            model.addAttribute("message", "Login failed!");
        }
        return "login.html";
    }
}
```

Позже, когда мы напишем логику аутентификации, в этой переменной будет храниться результат обработки запроса на вход

USING REQUEST SCOPE IN A SPRING WEB APP

Действие контроллера, выполняющее аутентификацию(5/6)

В зависимости от результата аутентификации, отправляем в представление то или иное сообщение

```
@Controller
public class LoginController {
    @GetMapping("/")
    public String loginGet () {"login.html";}

    @PostMapping("/")
    public String loginPost (RequestParam String username,
                             RequestParam String password, Model model) {
        boolean loggedIn = false;
        if (loggedIn) {
            model.addAttribute("message", "You are now logged in.");
        } else {
            model.addAttribute("message", "Login failed!");
        }
        return "login.html";
    }
}
```





USING REQUEST SCOPE IN A SPRING WEB APP

Действие контроллера, выполняющее аутентификацию(6/6)

```
@Controller
public class LoginController {
    @GetMapping("/")
    public String loginGet () {"login.html";}

    @PostMapping("/")
    public String loginPost (RequestParam String username,
                             RequestParam String password, Model model) {
        boolean loggedIn = false;
        if (loggedIn) {
            model.addAttribute("message", "You are now logged in.");
        } else {
            model.addAttribute("message", "Login failed!");
        }
        return "login.html";
    }
}
```

Возвращаем имя представления. Это по-прежнему login.html, так что мы остаемся на той же странице

USING REQUEST SCOPE IN A SPRING WEB APP

```
@Controller
public class LoginController {
    @GetMapping("/")
    public String loginGet() {"login.html";}
    @PostMapping("/")
    public String loginPost(RequestParam String username, RequestParam String password,
        Model model) {
        boolean loggedIn = false;
        if (loggedIn) {model.addAttribute("message", "You are now logged in.");}
        else {model.addAttribute("message", "Login failed!");}
        return "login.html";
    }
}
```

Это действие контроллера вызывается при отправке формы

Действие контроллера принимает учетные данные из параметров запроса

```
<body>
<form action="/" method="post">
  Username: <input type="text" name="username" />
  <br />
  Password: <input type="password" name="password" />
  <br />
  <button type="submit">Log in</button>
</form>
<p th:text="${message}"></p>
</body>
```

Действие контроллера передает в представление сообщение, содержание которого зависит от результатов обработки запроса на аутентификацию. Представление выводит сообщение под формой аутентификации



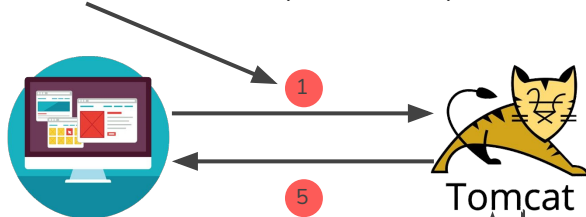
USING REQUEST SCOPE IN A SPRING WEB APP

Давайте дополним приложение, добавив бин `LoginProcessor` с областью видимости в рамках запроса. Этот бин извлекает учетные данные из запроса и проверяет их.

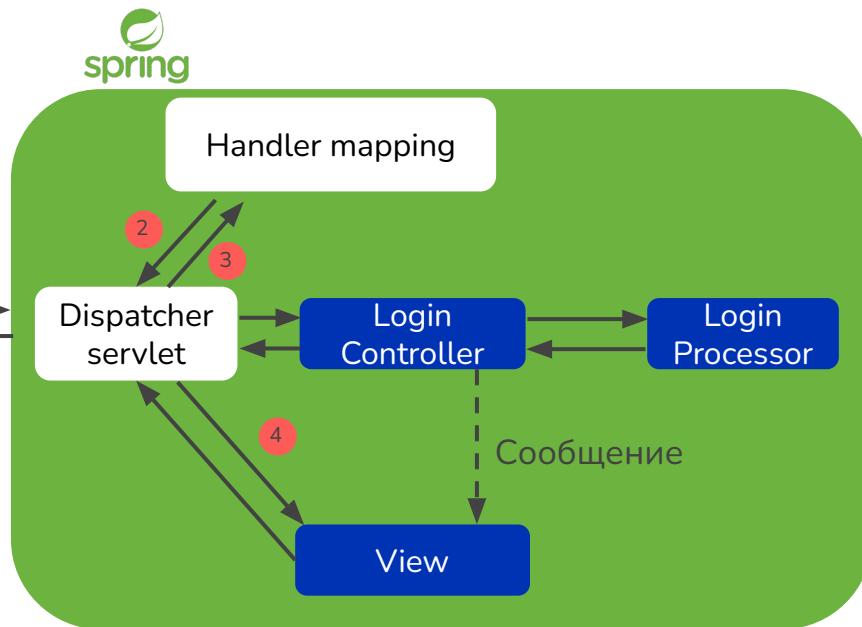
USING REQUEST SCOPE IN A SPRING WEB APP

Клиент отправляет HTTP-запрос, в котором содержатся учетные данные для аутентификации(1/4)

HTTP-запрос POST
/?username=natalie&password=password



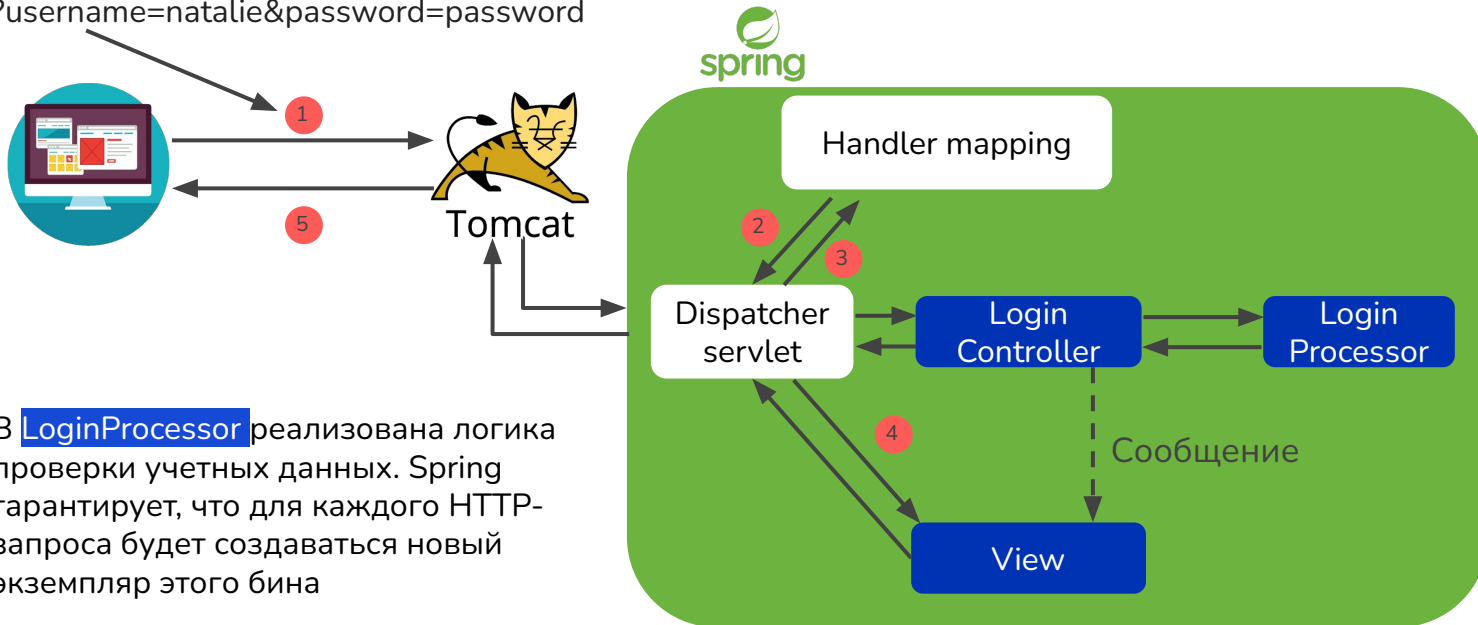
Контроллер с помощью `LoginProcessor` определяет, являются ли эти учетные данные правильными, и передает в представление сообщение о результатах аутентификации



USING REQUEST SCOPE IN A SPRING WEB APP

Клиент отправляет HTTP-запрос, в котором содержатся учетные данные для аутентификации(2/4)

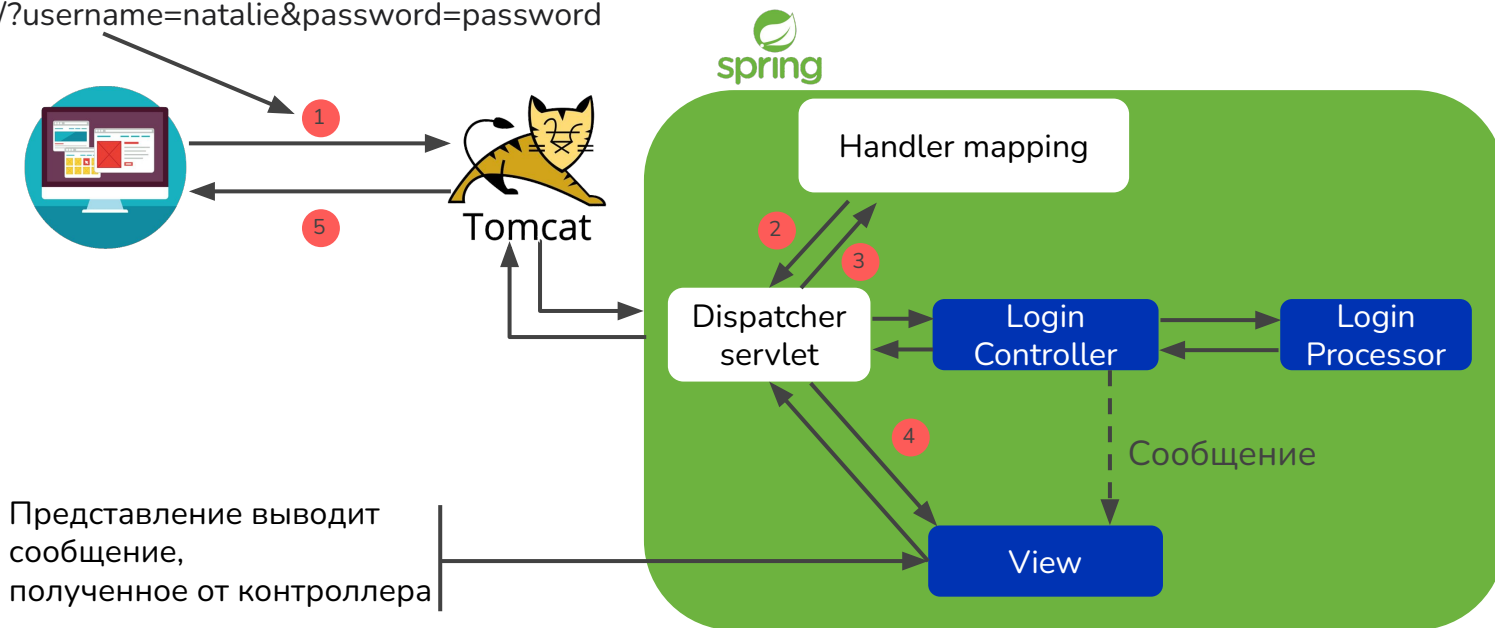
HTTP-запрос POST
/?username=natalie&password=password



USING REQUEST SCOPE IN A SPRING WEB APP

Клиент отправляет HTTP-запрос, в котором содержатся учетные данные для аутентификации(3/4)

HTTP-запрос POST
/?username=natalie&password=password





USING REQUEST SCOPE IN A SPRING WEB APP

Клиент отправляет HTTP-запрос, в котором содержатся учетные данные для аутентификации(4/4)

Бин `LoginProcessor` имеет область видимости в рамках запроса. Spring гарантирует, что для каждого HTTP-запроса будет создаваться новый экземпляр этого бина. В `LoginProcessor` реализована логика аутентификации. Контроллер вызывает метод, выполняющий эту логику. Метод возвращает `true`, если учетные данные верны, и `false`, если нет. Сообщение, которое `LoginController` передает в представление, определяется тем, какое значение возвращает `LoginProcessor`

USING REQUEST SCOPE IN A SPRING WEB APP

Бин `LoginProcessor` с областью видимости в рамках запроса, реализующий логику аутентификации

С помощью стереотипной аннотации сообщаем Spring, что это бин

`@Component`
`@RequestScope`

```
public class LoginProcessor {
```

```
    private String username;
```

```
    private String password;
```

```
    public boolean login() {
```

```
        String username = this.getUsername();
```

```
        String password = this.getPassword();
```

```
        if ("natalie".equals(username) && "password".equals(password)) {
```

```
            return true;
```

```
        } else {
```

```
            return false;
```

```
        }
```

```
    }
```

```
    // геттеры и
```

```
}
```

С помощью аннотации

`@RequestScope` ограничиваем область видимости бина текущим запросом.

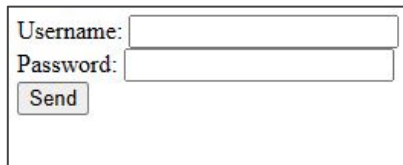
Теперь Spring будет создавать новый экземпляр класса для каждого HTTP-запроса

Учетные данные хранятся в бине в качестве атрибутов

В бине определен метод, в котором реализована логика аутентификации

USING REQUEST SCOPE IN A SPRING WEB APP


Форма аутентификации



A diagram of an empty authentication form. It contains two input fields: 'Username:' and 'Password:'. Below the 'Password:' field is a 'Send' button.

Открыв веб-страницу, вы сначала увидите пустую форму для аутентификации


Учетные данные верны



A diagram of an authentication form showing a successful login. It contains two input fields: 'Username:' and 'Password:'. Below the 'Password:' field is a 'Send' button. Below the 'Send' button, the text 'You are now logged in!' is displayed.

Если ввести правильные учетные данные - имя пользователя natalie и пароль password - и нажать Log in, то приложение выведет сообщение об успешной аутентификации

Учетные данные не верны



A diagram of an authentication form showing a failed login. It contains two input fields: 'Username:' and 'Password:'. Below the 'Password:' field is a 'Send' button. Below the 'Send' button, the text 'Login failed!' is displayed.

При использовании неверных учетных данных приложение выводит сообщение Login failed! (Аутентификация не удалась!)



USING REQUEST SCOPE IN A SPRING WEB APP

Описание:

Когда страница открывается в браузере, приложение выводит форму аутентификации. Если ввести правильные учетные данные, приложение выдаст сообщение об успешной аутентификации. Если учетные данные будут неверными, появится текст Login failed! (Аутентификация не удалась!)



USING THE SESSION SCOPE

ИСПОЛЬЗОВАНИЕ ОБЛАСТИ ВИДИМОСТИ В РАМКАХ СЕССИИ В ВЕБ-ПРИЛОЖЕНИЯХ SPRING.

-Область видимости в рамках сессии — Spring создает экземпляр и хранит его в памяти сервера в течение всей HTTP-сессии. Фреймворк связывает этот экземпляр в контексте сессии данного клиента.



USING THE SESSION SCOPE

Бины с областью видимости в рамках сессии позволяют реализовать, в частности, такие функции, как:

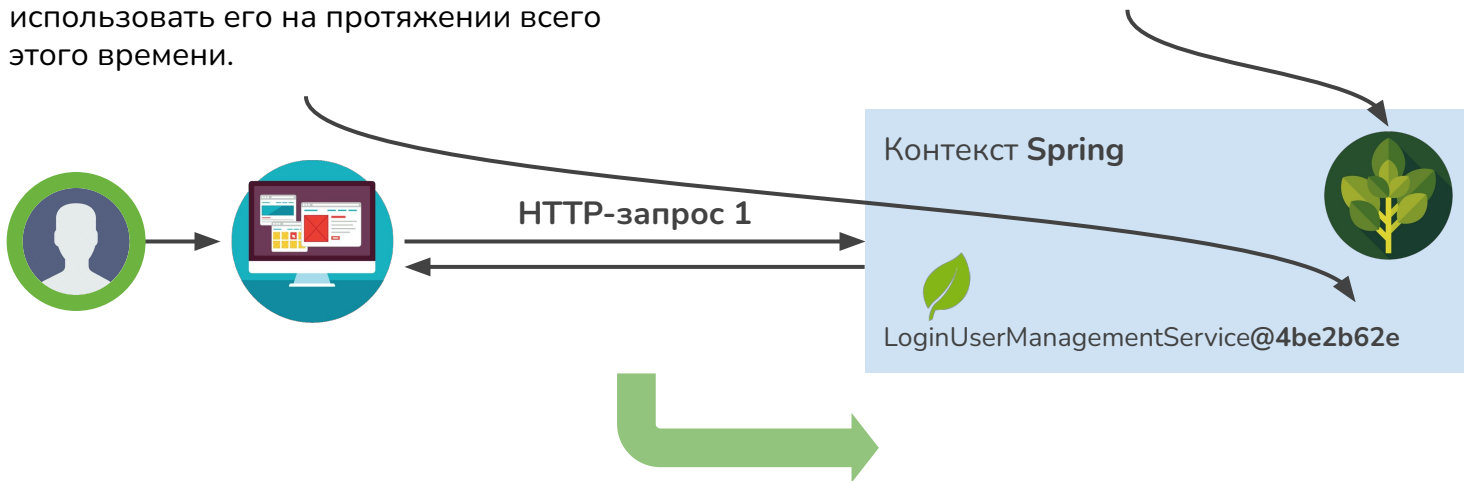
- Аутентификация. В бине сохраняются данные об аутентифицированном пользователе в течение всего времени, пока он посещает различные страницы приложения и отправляет запросы.
- Корзина интернет-магазина. Пользователь посещает разные страницы приложения в поисках товаров, которые он хочет добавить в корзину. Корзина запоминает все наименования, которые в нее поместил клиент.

USING THE SESSION SCOPE

Пример(1/3)

Когда Ричард отправляет первый запрос, начинается новая HTTP-сессия и Spring создает новый экземпляр бина с областью видимости в рамках сессии. Приложение будет использовать его на протяжении всего этого времени.

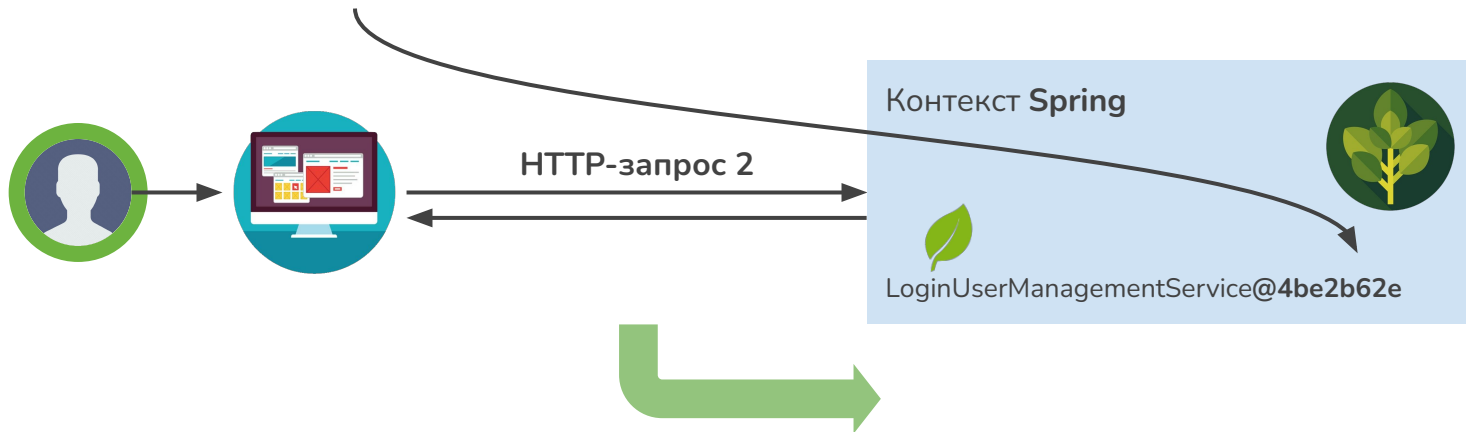
Бин с областью видимости в рамках сессии отмечен значком дерева. Spring отслеживает тип объекта, но создает и обслуживает несколько экземпляров этого объекта. Spring создает по одному экземпляру для каждой HTTP-сессии.



USING THE SESSION SCOPE

Пример(2/3)

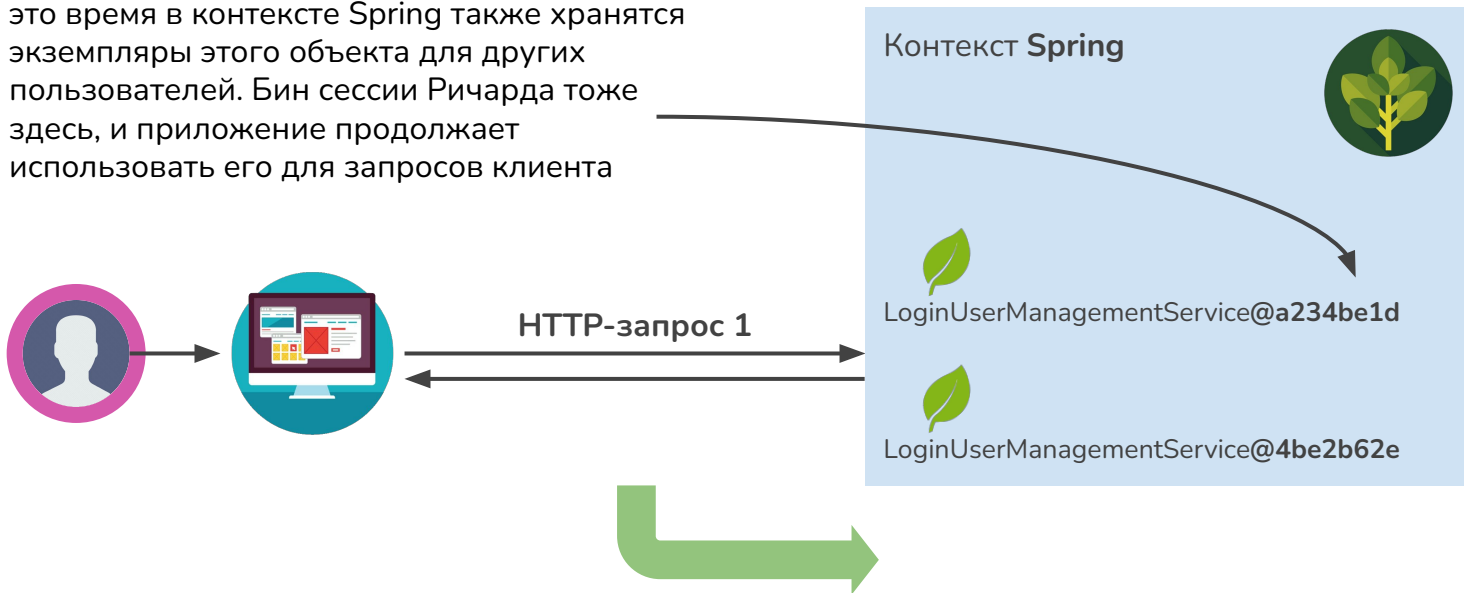
Затем Ричард отправляет запрос на использование уже созданного экземпляра бина с областью видимости в рамках сессии. Приложение хранит этот экземпляр в памяти и многократно его использует для всех запросов HTTP-сессии



USING THE SESSION SCOPE

Пример(3/3)

Когда Даниэла начинает свою HTTP-сессию, Spring создает экземпляр бина и для нее. В это время в контексте Spring также хранятся экземпляры этого объекта для других пользователей. Бин сессии Ричарда тоже здесь, и приложение продолжает использовать его для запросов клиента





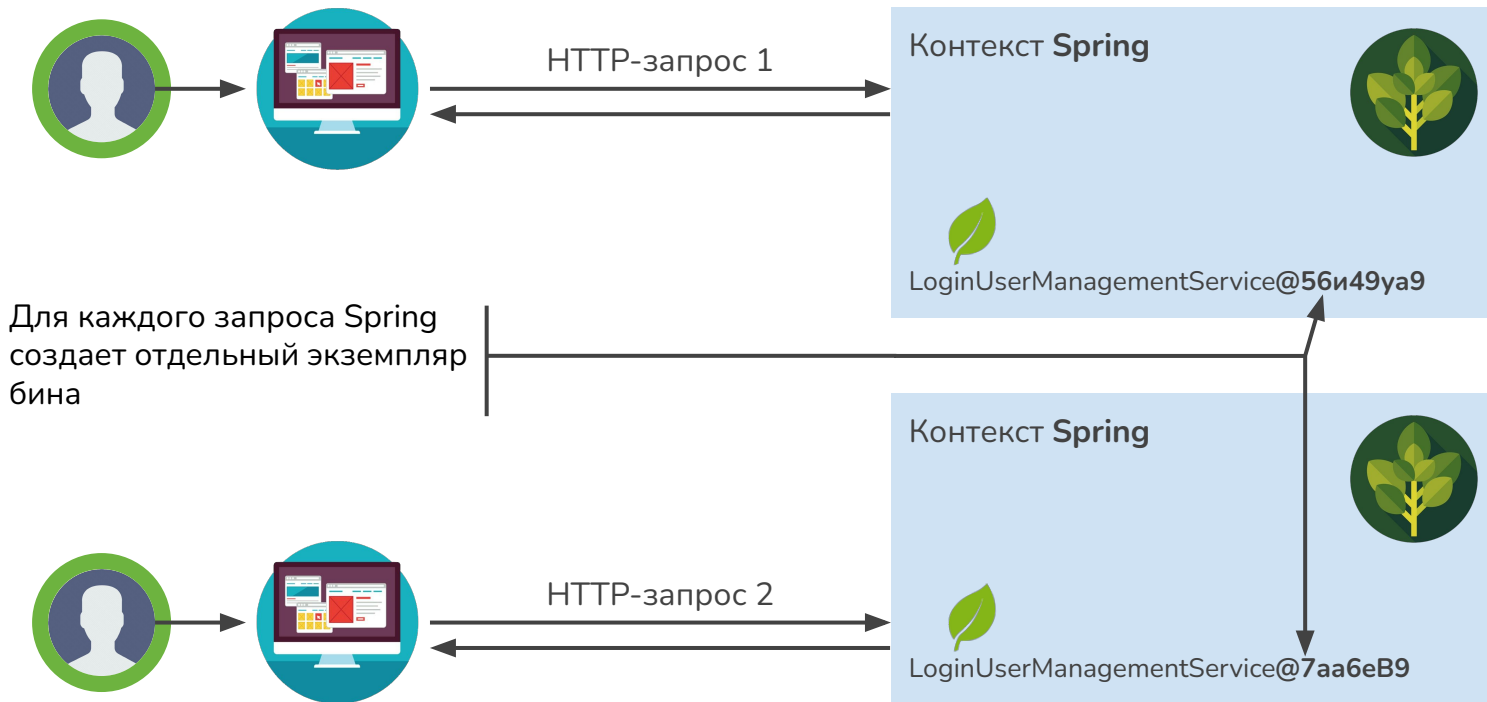
USING THE SESSION SCOPE

Анализ:

Бин с областью видимости в рамках сессии можно хранить в контексте в течение всей HTTP-сессии клиента. Spring создает экземпляр такого бина для каждой сессии, открываемой клиентом.

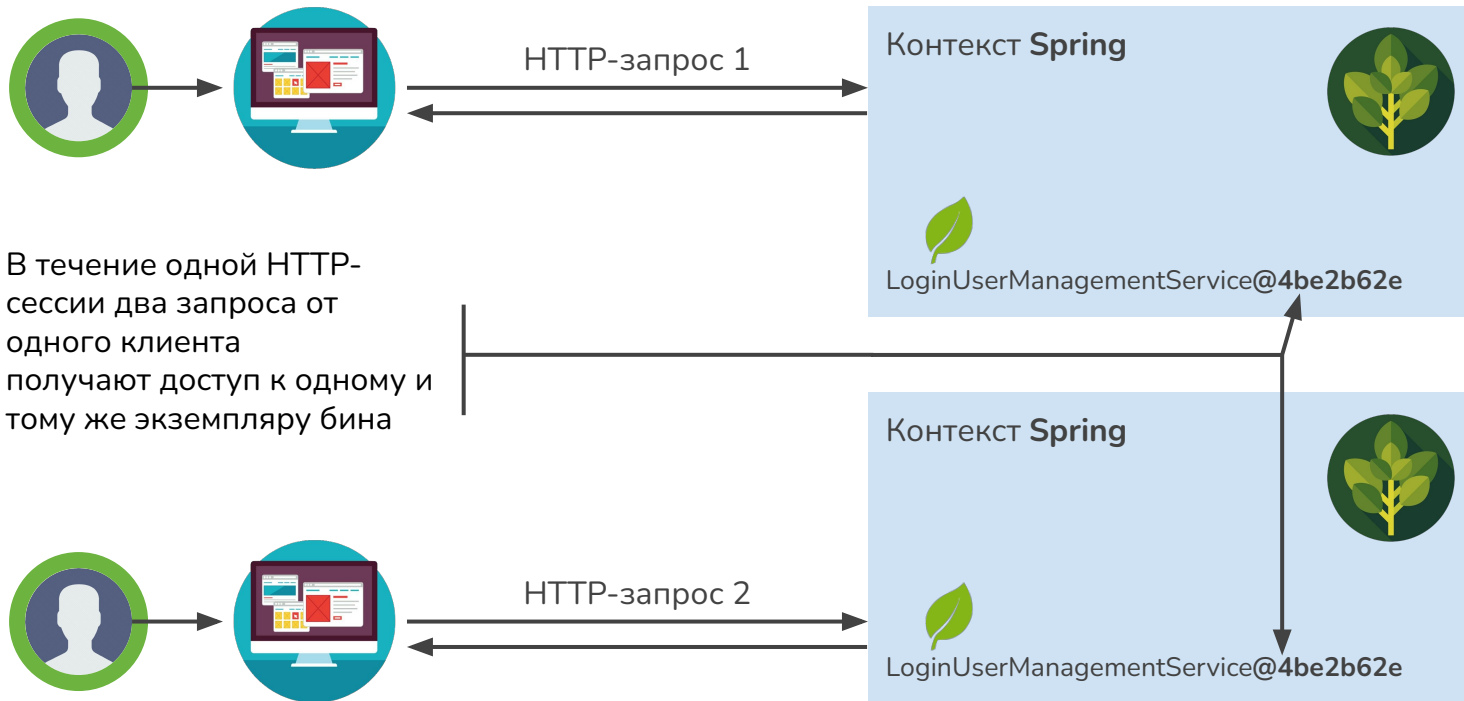
USING THE SESSION SCOPE

Сравнение request и session(1/2): Бины с областью видимости в рамках запроса.



USING THE SESSION SCOPE

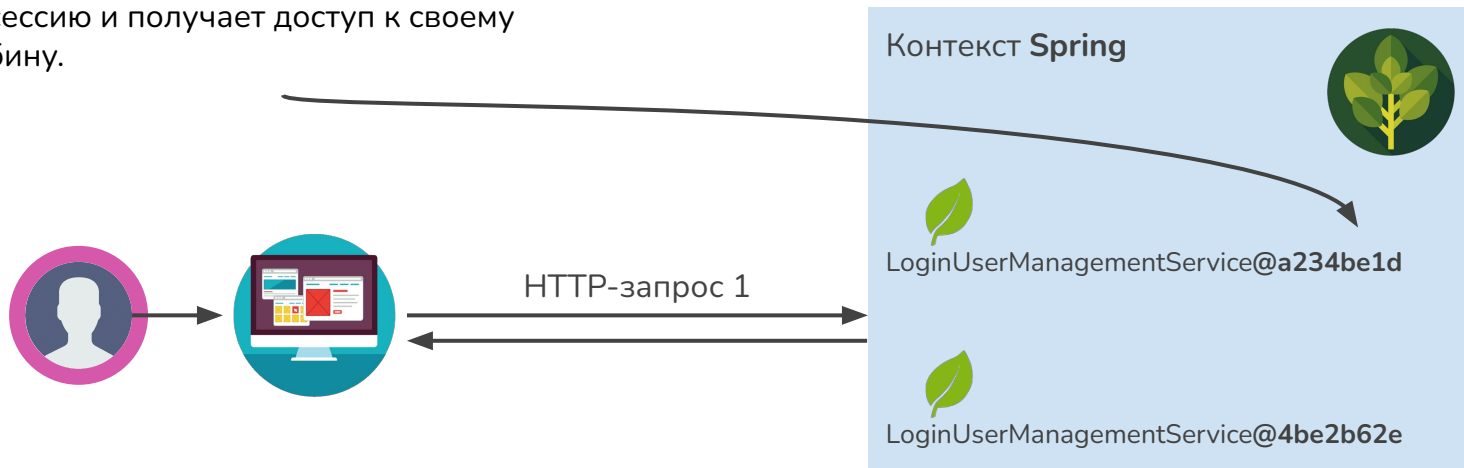
Сравнение request и session(2/2): Бины с областью видимости в рамках сессии.



USING THE SESSION SCOPE

Сравнение request и session(2/2): Бины с областью видимости в рамках сессии.

Каждый клиент создает свою HTTP-сессию и получает доступ к своему бину.





USING THE SESSION SCOPE

Анализ:

Сравнение бинов с областью видимости в рамках запроса и в рамках сессии поможет вам наглядно представить различия между этими двумя областями видимости. Первые бины применяют, когда нужно создать новый экземпляр бина для каждого запроса. Вторые используют в тех случаях, когда бин (и все данные, которые в нем хранятся) должен быть доступен в течение всей HTTP-сессии данного клиента



USING THE SESSION SCOPE

Основные свойства бинов с областью видимости в рамках сессии(1/3)

Факты	Следствия	Что учесть	Чего избегать
Экземпляры бинов с областью видимости в рамках сессии сохраняются в течение всей HTTP-сессии	Время жизни таких бинов дольше, чем у бинов с областью видимости в рамках запроса, и они не так часто попадают в сборку мусора	Данные, сохраненные в бинах с областью видимости в рамках сессии, приложение помнит дольше	Не стоит хранить в сессии слишком много данных — это может привести к проблемам с производительностью. И тем более не следует помещать в атрибуты бинов с областью видимости в рамках сессии конфиденциальную информацию, такую как пароли, частные ключи и др.



USING THE SESSION SCOPE

Основные свойства бинов с областью видимости в рамках сессии(2/3)

Факты	Следствия	Что учесть	Чего избегать
Один экземпляр бина с областью видимости в рамках сессии может быть доступен нескольким запросам	Если один и тот же клиент сделает несколько конкурентных запросов, изменяющих данные в таком экземпляре, возможны проблемы многопоточности, например состояние гонки	Возможно, чтобы избежать конкурентности, стоит воспользоваться механизмами синхронизации. Но я обычно рекомендую подумать, можно ли не допускать появления такой проблемы и оставить синхронизацию на самый крайний случай	



USING THE SESSION SCOPE

Основные свойства бинов с областью видимости в рамках сессии(3/3)

Факты	Следствия	Что учесть	Чего избегать
Бины с областью видимости в рамках сессии — это способ сделать данные доступными для нескольких запросов, сохраняя эти данные на стороне сервера	Реализуемая вами логика может потребовать запросов, зависящих друг от друга	Когда данные о состоянии хранятся в памяти приложения, клиенты становятся зависимыми от этого конкретного объекта приложения. Принимая решение о реализации какого-либо функционала посредством бина с областью видимости в рамках сессии, рассмотрите другие варианты хранения данных, которые вы хотите сделать общедоступными, например, не в сессии, а в базе данных, чтобы HTTP-запросы остались независимыми друг от друга	



USING THE SESSION SCOPE

Процесс реализация(1/4)

Внесем изменения в код приложения, добавим в него страницу, которая открывается только для аутентифицированных пользователей. После того как пользователь аутентифицируется, приложение перенаправляет его на эту страницу.

Создадим новый класс **LoggedInUserManagementService** с областью видимости в рамках сессии.



USING THE SESSION SCOPE

Процесс реализации(2/4)

Определение бина с областью видимости в рамках сессии для хранения данных об аутентифицированном пользователе.

Добавляем стереотипную аннотацию **@Service**, чтобы Spring создал бин этого класса и добавил его в контекст

С помощью аннотации **@SessionScope** меняем область видимости бина на видимость в рамках сессии

```
@Service
@SessionScope
public class LoggedUserManagementService {
    private String username;
    // геттеры и сеттеры
}
```



USING THE SESSION SCOPE

Процесс реализации(3/4)

Создать бин с областью видимости в рамках сессии, чтобы хранить в нем данные об аутентифицированном пользователе

```
@Service
@SessionScope
public class
    LoggedUserManagementService {
        private String username;
    }
```

Шаг-1



Создать страницу, на которую пользователь может попасть только после аутентификации
resources/templates/main. html

```
<!DOCTYPE html>
<html lang="en"
    xmlns:th=" http://www.thymeleaf.org " >
<head>
    <meta charset="UTF-8">
    <title>Welcome</title>
</head>
<body>
<h1>Welcome</h1>
</body>
</html>
```

Шаг-2



USING THE SESSION SCOPE

Процесс реализации(4/4)

После успешной аутентификации перенаправить пользователя со страницы аутентификации на главную страницу

```
if (loggedIn) {  
    return "redirect:/main";  
}else {  
    model.addAttribute("message",  
        "Login failed!");  
}
```

Шаг-4

Проследить, чтобы пользователь не мог открыть страницу, созданную в пункте 2, предварительно не аутентифицировался

```
String username =  
    loggedInUserManagementService.getUsername();  
if (username == null) {  
    return "redirect:/";  
}
```

Шаг-3



USING THE SESSION SCOPE

Анализ:

С помощью бина, имеющего область видимости в рамках сессии, мы создадим раздел приложения, доступный только аутентифицированным пользователям. Приложение будет перенаправлять пользователя на эту страницу только после успешной аутентификации. Если он попытается открыть страницу не зарегистрировавшись, приложение вернет его в форму аутентификации



USING THE SESSION SCOPE

Применение бина **LoggedUserManagementService** в логике аутентификации.

```
@Component
@RequestScope
public class LoginProcessor {
    private final LoggedUserManagementService loggedUserManagementService;
    private String username;
    private String password;
    public LoginProcessor(
        LoggedUserManagementService loggedUserManagementService) {
        this.loggedUserManagementService = loggedUserManagementService;
    }
    public boolean login() {
        String username = this.getUsername();
        String password = this.getPassword();
        boolean loginResult = false;
        if ("natalie".equals(username) && "password".equals(password)) {
            loginResult = true;
            loggedUserManagementService.setUsername(username);
        }
        return loginResult;
    }
    // геттеры и сеттеры
}
```

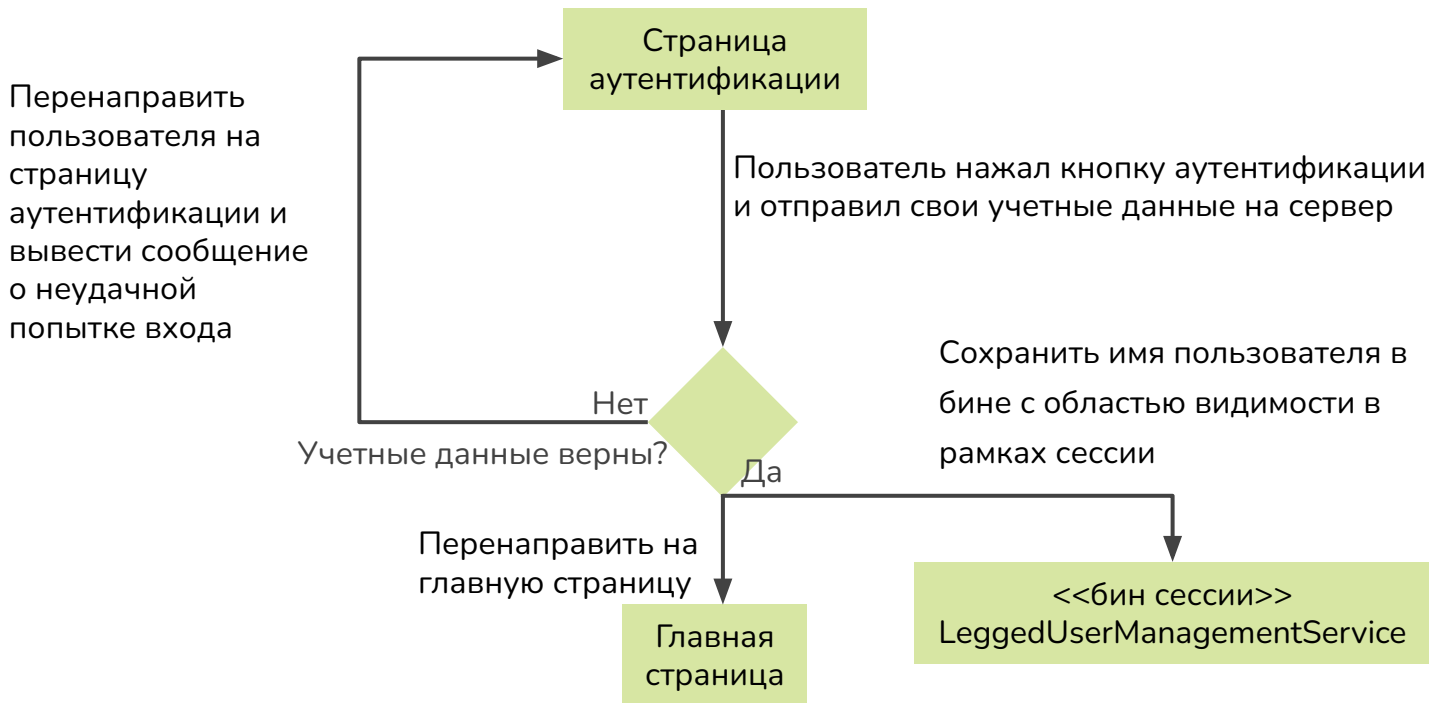
Автомонтируем бин LoggedUserManagementService

Сохраняем имя пользователя в бине LoggedUserManagementService



USING THE SESSION SCOPE

Процедура аутентификации, реализованная в данном примере.





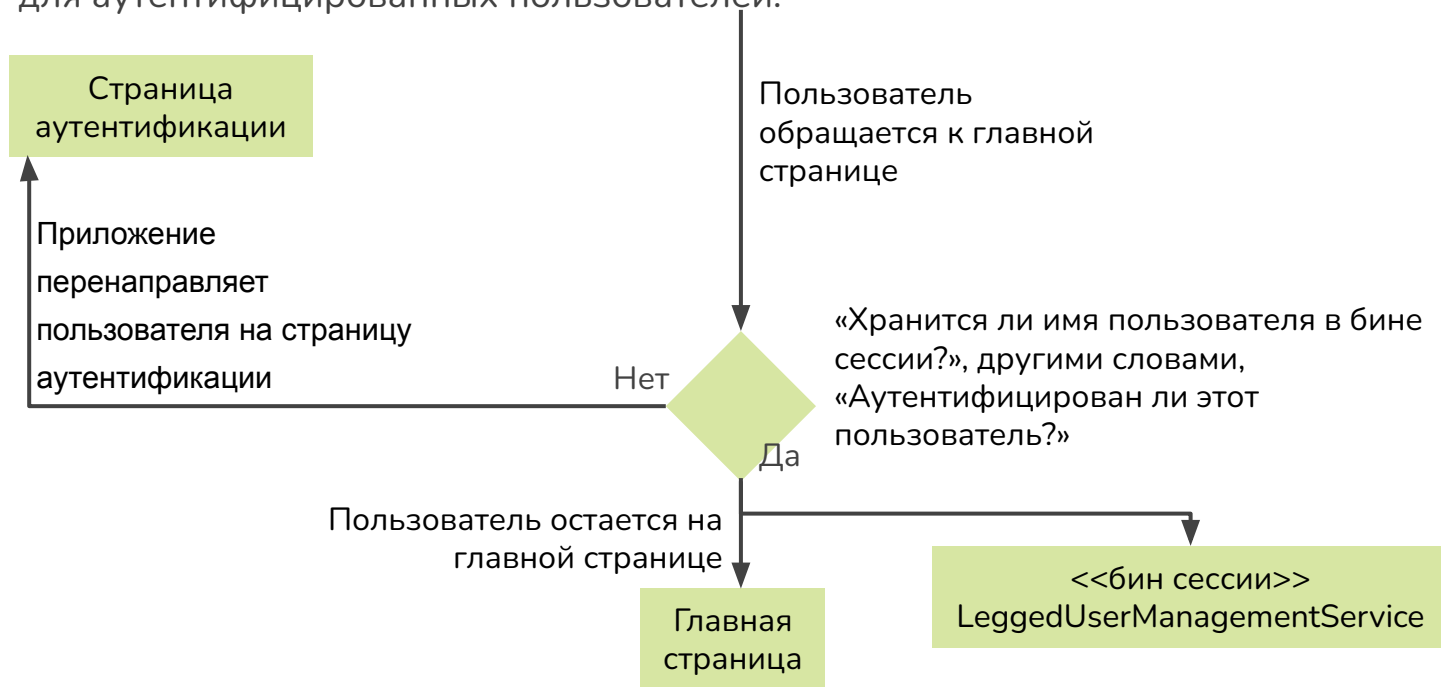
USING THE SESSION SCOPE

Анализ:

После того как пользователь передаст на сервер свои учетные данные, начинается процесс аутентификации. Если данные верны, имя пользователя сохраняется в бине с областью видимости в рамках сессии, и приложение перенаправляет пользователя на главную страницу. Если же данные неверны, приложение возвращает пользователя на страницу аутентификации и выводит сообщение о неудачной попытке входа

USING THE SESSION SCOPE

Теперь создадим новую страницу и проследим, чтобы она была доступна только для аутентифицированных пользователей.





USING THE SESSION SCOPE

Анализ:

Пользователь может получить доступ к главной странице только после аутентификации. Когда приложение аутентифицирует пользователя, оно сохраняет его имя в бине сессии. Таким образом, приложение знает, что данный пользователь аутентифицирован. Когда кто-нибудь пытается получить доступ к главной странице, но в бине его сессии имени пользователя не будет (ведь он не аутентифицирован), приложение перенаправляет его на страницу с формой входа

USING THE SESSION SCOPE

Класс MainController

```
@Controller
public class MainController {
    private final LoggedUserManagementService loggedUserManagementService ;

    public MainController (
        LoggedUserManagementService loggedUserManagementService ) {
        this.loggedUserManagementService = loggedUserManagementService ;
    }
    @GetMapping ("/main")
    public String home () {
        String username =
            loggedUserManagementService .getUsername () ;
        if (username == null) {
            return "redirect:/" ;
        }
        return "main.html" ;
    }
}
```

Автомонтируем(Injecting) бин
LoggedUserManagementService, чтобы
можно было узнать, аутентифицирован ли
пользователь

Получаем значение username — если
пользователь аутентифицирован, то
оно не должно быть равно null

Если пользователь не аутентифицирован,
перенаправляем его на страницу
аутентификации

Если пользователь аутентифицирован, возвращаем представление главной страницы



USING THE SESSION SCOPE

Теперь нужно создать в папке **resources/templates** Spring Boot файл **main.html**, для главной страницы.

```
<!DOCTYPE html>
<html lang="en"
xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Welcome</title>
</head>
<body>
    <h1>Welcome</h1>
</body>
</html>
```




USING THE SESSION SCOPE

Добавление ссылки для выхода из приложения на страницу **main.html**

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Login</title>
</head>
<body>
  <h1>Welcome, <span th:text="${username}"></span></h1>
  <a href="/main?Logout">Log out</a>
</body>
</html>
```

Получаем от контроллера имя
пользователя и выводим его на
странице в приветствии

Добавляем на страницу ссылку,
которая отправляет HTTP-запрос с
параметром **Logout**. Получив этот
параметр, контроллер удалит из
сессии значение атрибута username

USING THE SESSION SCOPE

Выход пользователя из приложения с помощью параметра запроса Logout.

```
@Controller
public class MainController {
    // Остальной код
    @GetMapping("/main")
    public String home(
        @RequestParam(required = false) String Logout,
        Model model
    ) {
        if (Logout != null) {
            loggedUserManagementService.setUsername( null);
        }
        String username = loggedUserManagementService.getUsername();
        if (username == null) {
            return "redirect:/";
        }
        model.addAttribute("username", username);
        return "main.html";
    }
}
```

Извлекаем из запроса параметр
Logout, если он там есть

Добавляем параметр
Model, чтобы передать имя
пользователя в
представление

Если в запросе есть параметр
Logout, удаляем из бина
LoggedUserManagementServic
е имя пользователя

Передаем имя пользователя в представление



USING THE SESSION SCOPE

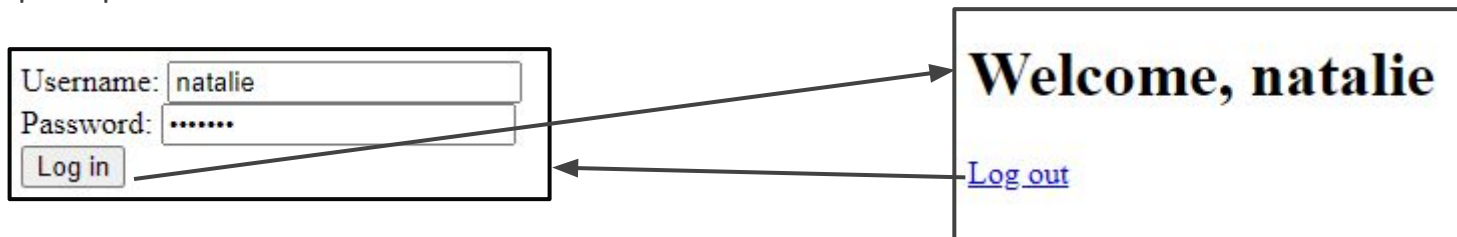
Перенаправление пользователя на главную страницу после аутентификации

```
@Controller
public class LoginController {
    // Остальной код
    @PostMapping("/")
    public String loginPost(
        @RequestParam String username,
        @RequestParam String password,
        Model modelModel model
    ) {
        loginProcessor.setUsername( username);
        loginProcessor.setPassword( password);
        boolean loggedIn = loginProcessor.login();
        if (loggedIn) {
            return "redirect:/main";
        }
        model.addAttribute( "message", "Login failed!");
        return "login.html";
    }
}
```

После успешной аутентификации приложение перенаправляет пользователя на главную страницу

USING THE SESSION SCOPE

Проверка



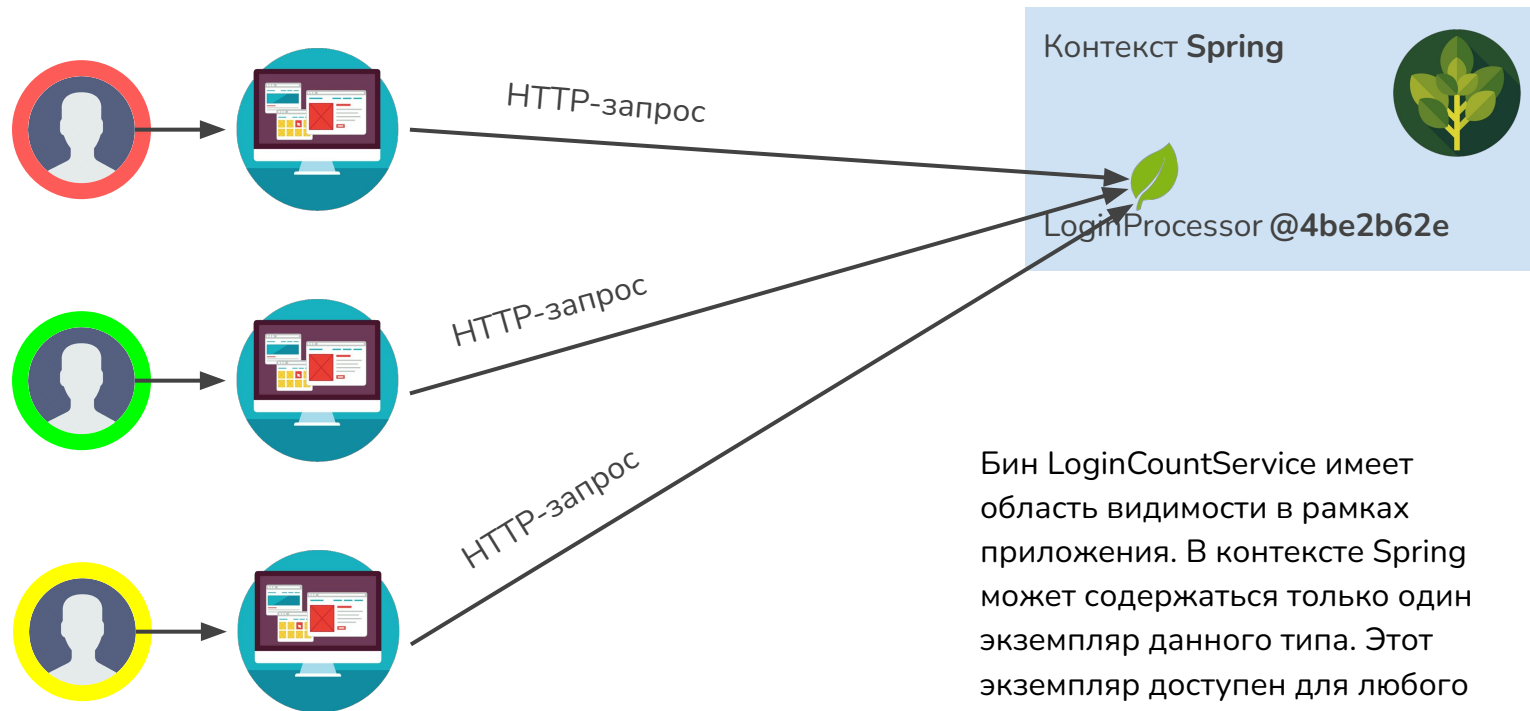
Перенаправление пользователя между двумя страницами: после того как пользователь аутентифицировался, приложение отправляет его на главную страницу. Пользователь может щелкнуть на ссылке выхода из приложения, и тогда он будет перенаправлен обратно к форме аутентификации



USING THE APPLICATION SCOPE

- ❖ Область видимости в рамках приложения — экземпляр является уникальным в контексте приложения и доступен все время работы приложения.
- ❖ Бин с областью видимости в рамках приложения доступен для всех запросов от всех клиентов. Он похож на одиночный бин. Различие состоит в том, что в данном случае нельзя создать в контексте несколько экземпляров.
- ❖ Если сделать атрибуты неизменяемыми, вместо бина с областью видимости в рамках приложения можно просто использовать одиночный бин.

USING THE APPLICATION SCOPE



Бин `LoginCountService` имеет область видимости в рамках приложения. В контексте Spring может содержаться только один экземпляр данного типа. Этот экземпляр доступен для любого запроса от любого клиента



USING THE APPLICATION SCOPE

Анализ:

Так выглядит бин с областью видимости в рамках всего веб-приложения Spring. Экземпляр этого бина доступен для всех HTTP-запросов от всех клиентов. В контексте Spring может существовать только один экземпляр бина данного типа, и его могут использовать все, кому он нужен



USING THE APPLICATION SCOPE

Поскольку нам нужно подсчитать все попытки аутентификации от всех пользователей, мы будем хранить счетчик в бине с областью видимости в рамках приложения. Создадим такой бин **LoginCountService** и разместим счетчик в его атрибуте.



USING THE APPLICATION SCOPE

Класс LoginCountService для подсчета попыток аутентификации

```
@Service
@ApplicationScope
public class LoginCountService {
    private int count;
    public void increment () {
        count++;
    }
    public int getCount () {
        return count;
    }
}
```

← Аннотация @ApplicationScope
распространяет область видимости
бина на все приложение

Этот бин может автомонтироваться в бин LoginProcessor, который может вызывать метод increment() при каждой попытке аутентификации, как показано в листинге

USING THE APPLICATION SCOPE

Увеличение счетчика попыток аутентификации при каждом запросе на аутентификацию

```
@Component
@RequestScope
```

```
public class LoginProcessor {
    private final LoggedUserManagementService loggedUserManagementService;
    private final LoginCountService loginCountService;
    private String username;
    private String password;
    public LoginProcessor(
        LoggedUserManagementService loggedUserManagementService,
        LoginCountService loginCountService) {
        this.loggedUserManagementService = loggedUserManagementService;
        this.loginCountService = loginCountService;
    }
    public boolean login() {
        loginCountService.increment();
        String username = this.getUsername();
        String password = this.getPassword();
        boolean loginResult = false;
        if ("natalie".equals(username) && "password".equals(password)) {
            loginResult = true;
            loggedUserManagementService.setUsername(username);
        }
        return loginResult;
    }
}
```

Внедряем бин
LoginCountService через
параметры конструктора

Увеличиваем счетчик
при каждой попытке
аутентификации

USING THE APPLICATION SCOPE

Передача значения счетчика из контроллера
для отображения на главной странице

```
@Controller
public class MainController {
    // Остальной код
    @GetMapping("/main")
    public String home(
        @RequestParam(required = false) String Logout,
        Model model
    ) {
        if (Logout != null) {
            loggedUserManagementService.setUsername(null);
        }
        String username = loggedUserManagementService.getUsername();
        int count = loginCountService.getCount();
        if (username == null) {
            return "redirect:/";
        }
        model.addAttribute("username", username);
        model.addAttribute("loginCount", count);
        return "main.html";
    }
}
```

Получаем значение счетчика
из бина с областью
видимости в рамках
приложения

Передаем значение счетчика
в представление



USING THE APPLICATION SCOPE

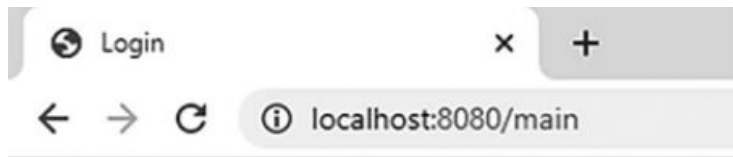
Вывод значения счетчика на главной странице

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Login</title>
</head>
<body>
  <h1>Welcome, <span th:text="${username}"></span>!</h1>
  <h2>
    Your login number is
    <span th:text="${loginCount}"></span>
  </h2>
  <a href="/main?Logout">Log out</a>
</body>
</html>
```

Выводим значение счетчика на
странице

USING THE APPLICATION SCOPE

Результат работы приложения — веб-страница, на которой отображается общее количество попыток аутентификации для всех пользователей. Это число выводится на главной странице



Welcome, natalie!

Your login number is 5

[Log out](#)



CONCLUSION(1/4)

- Кроме одиночной и прототипной (рассмотренных в главах 2–5), в веб-приложениях Spring доступны еще три области видимости бина. Они применимы только в веб-приложениях, и поэтому их называют областями веб-видимости:
 - область видимости в рамках запроса — Spring создает новый экземпляр бина для каждого HTTP-запроса;
 - область видимости в рамках сессии — Spring создает новый экземпляр бина для каждой HTTP-сессии конкретного клиента. Этот экземпляр доступен для всех запросов данного клиента в рамках данной сессии;
 - область видимости в рамках приложения - во всем приложении может существовать только один экземпляр такого бина. Этот экземпляр доступен для всех запросов от всех клиентов.



CONCLUSION(2/4)

- Spring гарантирует, что экземпляр бина с областью видимости в рамках запроса доступен только для данного HTTP-запроса. Поэтому можно смело использовать его атрибуты, не беспокоясь о проблемах конкурентности. Можно также не думать о том, что такие экземпляры займут всю память приложения: их время жизни очень коротко и они попадают в сборку мусора сразу же после завершения выполнения HTTP-запроса.
- Spring создает экземпляры бинов с областью видимости в рамках запроса для каждого HTTP-запроса, что происходит весьма часто. Поэтому не рекомендуется усложнять процесс за счет использования логики в конструкторе или в методе `@PostConstruct`.
- Экземпляры бинов с областью видимости в рамках сессии Spring связывает с конкретной HTTP-сессией клиента. Таким образом, их можно использовать для совместного доступа к данным для нескольких HTTP-запросов от одного и того же клиента.



CONCLUSION(3/4)

- Но даже один и тот же клиент может отправлять конкурентные HTTP-запросы. Если такие запросы изменяют данные, хранящиеся в экземпляре бина с областью видимости в рамках сессии, это может привести к состоянию гонки. Необходимо учитывать возможность подобных ситуаций и либо избегать их, либо синхронизировать код, чтобы поддерживать конкурентность.
- Рекомендую по возможности не использовать бины с областью видимости в рамках приложения. Такие бины доступны для всех запросов веб-приложения, поэтому любая операция записи, скорее всего, будет нуждаться в синхронизации. Это приведет к образованию узких мест и значительно снизит производительность приложения. Более того, такие бины хранятся в памяти приложения в течение всего времени жизни приложения и не попадают в сборку мусора. Как вы узнаете в главе 11, лучше хранить данные непосредственно в базе данных.



CONCLUSION(4/4)

- При использовании бинов с областями видимости в рамках сессии и в рамках приложения запросы становятся менее независимыми. В подобных случаях говорят, что приложение управляет состоянием, необходимым для запросов (или что это приложение с сохранением состояния). Приложениям с сохранением состояния свойственны различные проблемы архитектуры, которых лучше избегать. Описание этих проблем выходит за рамки нашей книги, но вам стоит заранее знать о них, чтобы сразу поискать альтернативы.



Reference

- 1: [Spring Start Here](#)
- 2: [Spring MVC](#)
- 3: [Spring MVC tutorial](#)
- 4: [Spring Annotations tutorial](#)



Thank you!

Presented by

Moxirbek Maxkamov

(mokhirbek.makhkam@gmail.com)