Разработка серверного ПО. Лекция 4. Создание веб-приложений с использованием Spring Boot и Spring MVC

Процесс обработки запросов Spring MVC выглядит следующим образом.

1. Клиент отправляет на сервер HTTP-запрос.
2. Диспетчер сервлетов с помощью карты обработчиков находит нужный контроллер и действие, которое необходимо выполнить.
3. Диспетчер вызывает это действие.
4. После того как действие будет выполнено, контроллер возвращает имя представления, которое диспетчер сервлетов должен преобразовать в HTTP-ответ.
5. HTTP-ответ возвращается клиенту.

Чтобы сформировать динамическое представление, контроллер должен передать для него данные. Эти данные могут различаться в зависимости от запроса. Например, для корзины интернет-магазина контроллер вначале предоставляет список, состоящий из одного товара. После того как пользователь добавляет в корзину еще несколько продуктов, контроллер включает в перечень и их. Таким образом, одно и то же представление дает разную информацию в ответ на эти запросы.

Здесь нужно внести изменения в пункт 4. Нужно, чтобы контроллер не только возвращал имя представления, но и каким-то образом передавал в представление данные для их вставки в HTTP-ответ. Таким образом, если сервер предоставит список с одним товаром, на странице, отображающей данный перечень, будет выведена только эта позиция. Но если потом контроллер передаст для этого же представления два товара, данные на экране изменятся — количество продуктов в корзине удвоится.

Пример в `com.sibsutis.MVCWithDynamicView.controllers.simpletemplate;`

Получение данных из HTTP-запроса

В большинстве случаев данные передаются посредством HTTP-запроса одним из следующих способов:

- в виде параметра HTTP-запроса — это простой способ передачи значений от клиента к серверу в формате пары «ключ — значение». Параметры добавляются в URI как выражения запроса. Поэтому их также называют параметрами запроса. Этот метод следует использовать только для передачи небольших объемов данных;

- в виде параметров заголовка HTTP-запроса. Как и параметры запроса, параметры заголовка передаются в заголовке HTTP-запроса. Главное различие между ними — параметры заголовка не попадают в URI. Этот способ также не подходит для больших объемов данных; переменная пути передает данные через сам путь запроса. Как и в случаях выше, переменные пути используются для небольших объемов данных. Но данный вариант передачи следует применять, если данные являются обязательными;

- в теле HTTP-запроса. Этот метод обычно применяется в случаях, когда нужно передать много данных (строки, но иногда и двоичные данные, такие как файлы).

Передача данных от клиента серверу посредством параметров запроса

Параметры запроса применяют в следующих случаях:

- объем данных не очень велик. Параметры запроса задаются посредством переменных запроса (как будет показано далее). Этот метод позволяет отправить максимум около 2000 символов;
- нужно передать необязательные данные. Параметры запроса — простой способ выслать значения, которые клиент не обязан передавать. Сервер будет готов не получить значения определенных параметров запроса.

Типичный случай использования параметров запроса — определение критериев поиска и фильтрации.

Чтобы присвоить значения параметрам HTTP-запроса, нужно поставить в конце пути вопросительный знак (?), после которого перечислить параметры пар в виде «ключ — значение», разделенных символом &. Например:

```
http://localhost:8080/home?color=red&name=Anton
```

По умолчанию параметр запроса является обязательным. Если клиент не передает значение для него, сервер возвращает HTTP-статус 400 Bad Request. Чтобы значение перестало быть обязательным, нужно явно прописать это в аннотации с помощью дополнительного атрибута: `@RequestParam(optional=true)`

Пример в `com.sibsutis.MVCWithDynamicView.controllers.withparams`.

Передача данных от клиента серверу с помощью переменных пути

Параметры запроса:

```
http://localhost:8080/home?color=blue
```

Переменные пути:

```
http://localhost:8080/home/blue
```

Теперь не приходится указывать ключ для значения. Достаточно вставить значение в определенное место пути — и на стороне сервера оно будет извлечено из этой позиции. В пути может быть несколько значений как переменных пути, но обычно рекомендуется использовать одно-два.

Параметры запроса	**Переменные пути**
Могут использоваться для необязательных значений	Не могут использоваться для необязательных значений

Параметры запроса	Переменные пути
Не рекомендуется использовать много таких параметров. Если их больше трех, лучше поместить их в тело запроса — хотя бы для удобства чтения/Не стоит передавать более трех переменных пути. Лучше даже ограничиться максимум двумя	
По мнению некоторых разработчиков, выражения запроса труднее читать, чем выражения пути	Переменные пути легче читать, чем переменные запроса. В случае общедоступных веб-сайтов страницы с переменными пути легче индексируются поисковыми системами (такими как Google), благодаря чему веб-сайт проще найти через поисковик

Если создаваемая вами страница зависит всего лишь от одного или двух значений, определяющих ее итоговый вид, лучше вставить их непосредственно в путь, чтобы запрос было проще читать. Кроме того, подобный URL будет проще найти, если сохранить его в браузере в виде закладки и он будет легче индексироваться поисковыми системами (если это важно для вашего приложения).

Пример в `com.sibsutis.MVCWithDynamicView.controllers.pathvariables`

Использование HTTP-методов GET и POST

HTTP-метод определяется ключевым словом, соответствующим намерению клиента. Если запрос клиента требует только получения данных, мы создаем конечную точку, используя HTTP-метод GET. Если же при этом данные на сервере как-то изменяются, для точного описания действий клиента нужно использовать другое ключевое слово.

HTTP-метод	Описание
GET	Клиентский запрос лишь получает данные
POST	Клиентский запрос передает новые данные, которые должны быть сохранены на сервере
PUT	Клиентский запрос изменяет данные, хранящиеся на сервере
PATCH	Клиентский запрос требует частичного изменения данных, хранящихся на сервере
DELETE	Клиентский запрос удаляет данные на сервере

Несмотря на то что хорошим тоном программирования считается различать полное (PUT) и частичное (PATCH) изменение данных, при разработке реальных приложений различия между этими HTTP-методами часто не делают.

Пример в `com.sibsutis.MVCWithDynamicView.controllers.getpostmethods`

Это упрощенная структура, позволяющая сосредоточиться на HTTP-методах. По умолчанию бины Spring имеют одиночную область определения, а веб-приложение создает несколько потоков (по одному для каждого запроса). В реальном приложении, когда сразу несколько клиентов будут добавлять новые товары, изменение списка, определенного как атрибут бина, приведет к состоянию гонки. Но пока что используем этот вариант: далее мы будем изменять список в базе данных, так что проблем не возникнет. Тем не менее следует помнить, что это порочная практика и не следует использовать подобный подход в реальных приложениях. При наличии потоков одиночные бины небезопасны!

Области веб-видимости бинов в Spring

- одиночный бин — область видимости бинов в Spring по умолчанию. Фреймворк однозначно идентифицирует каждый такой экземпляр в контексте по его имени;
- прототип — область видимости бина в Spring, в случае которой фреймворк управляет только типом объектов и создает новый экземпляр класса для каждого запроса (сразу из контекста либо посредством монтажа или автомонтажа).
- область видимости в рамках запроса — Spring создает отдельный экземпляр класса бина для каждого HTTP-запроса. Конкретный экземпляр существует только для конкретного HTTP-запроса;
- 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 с сессией данного клиента;
- область видимости в рамках приложения — экземпляр является уникальным в контексте приложения и доступен все время работы приложения.

Область видимости в рамках запроса

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 @PostConstruct

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

Область видимости в рамках сессии

Бин с областью видимости в рамках сессии — это управляемый Spring объект, для которого фреймворк создает экземпляр, привязанный к текущей HTTP-сессии. Когда клиент посылает запрос на сервер, сервер выделяет в памяти место для этого запроса на все время сессии, к которой данный запрос относится. Spring создает экземпляр бина с областью видимости в рамках сессии в начале HTTP-сессии для данного клиента. Этот экземпляр может многократно использоваться одним и тем же клиентом, пока HTTP-сессия остается активной. Данные в атрибутах бина с областью видимости в рамках сессии доступны для всех запросов клиента в рамках сессии. Такой способ хранения данных позволяет не потерять информацию о том, что делает пользователь, пока переходит по страницам приложения.

При всех запросах, передаваемых в рамках одной HTTP-сессии, клиент получает доступ к одному и тому же экземпляру бина. У каждого пользователя есть своя сессия, поэтому они получают доступ к собственным бинам, имеющим область видимости в рамках сессии.

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.

Факты	Следствия	Что учесть	Чего избегать

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.
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	

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	

Чтобы создать бин с областью видимости в рамках сессии, достаточно добавить к классу бина аннотацию @SessionScope.

Область видимости в рамках 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. Кроме того, когда мы говорим о жизненном цикле бинов с областью видимости в веб-приложениях (включая область видимости в рамках всего веб-приложения), отправной точкой всегда являются HTTP-запросы. В случае бинов с областью видимости в рамках приложения возникают те же проблемы конкурентности, которые были описаны для одиночных бинов. Желательно, чтобы атрибуты одиночных бинов были неизменяемыми. Тот же совет касается и бинов с областью видимости в рамках приложения. Но если сделать атрибуты неизменяемыми, вместо бина с областью видимости в рамках приложения можно просто использовать одиночный бин.

Пример в `com.sibsutis.MVCWithDynamicView.controllers.beanscope`

Рекомендуется 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.

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.

Обмен данными посредсвом REST-сервисов

Эндпоинты REST — это всего лишь способ организовать коммуникацию между двумя приложениями. Они представляют собой обычное действие контроллера, связанное с HTTP-методом и путем. Приложение вызывает действие контроллера через HTTP. Поскольку таким образом приложение делает сервис доступным через веб-протокол, конечные точки можно назвать веб-сервисами.

При использовании REST эндпоинтов могут возникнуть некоторые проблемы:

- если действие контроллера выполняется долго, то HTTP-вызов конечной точки может завершиться и соединение разорвется;
- при попытке передать большое количество данных за один вызов (один HTTP-запрос) времени вызова может не хватить и соединение разорвется. Передача за один REST-вызов больше пары мегабайт обычно не лучшее решение;
- слишком много конкурентных обращений к одной конечной точке бэкенда может привести к чрезмерной нагрузке на приложение и вызвать его сбой;
- HTTP-вызовы выполняются за счет сети, но сеть никогда не бывает абсолютно надежной. Всегда есть вероятность того, что вызов конечной точки REST завершится неудачно из-за сетевого сбоя.

Создание REST эндпоинта

Аннотация `@ResponseBody`. Она сообщает диспетчеру сервлетов, что действие контроллера не возвращает имя представления, а передает данные непосредственно в HTTP-запрос.

Дублирования кода лучше избегать. Желательно как-нибудь избавиться от повторения `@ResponseBody` для каждого метода. Чтобы решить эту проблему, в Spring есть аннотация @RestController, которая заменяет сочетание `@Controller` и `@ResponseBody`. С помощью `@RestController` мы сообщаем Spring, что все действия контроллера являются эндпоинтами REST, поэтому в многократном использовании аннотации `@ResponseBody` нет необходимости.

Пример в `com.sibsutis.MVCWithDynamicView.controllers.rest.simple`

Для проверки можно использовать специальные инструменты, например:

- Postman — имеет приятный GUI и удобен в использовании;
- cURL — инструмент командной строки, пригодный в тех случаях, когда нельзя использовать GUI (например, при соединении с виртуальной машиной по SSH или если вы пишете сценарий пакетной обработки).

Postman предоставляет удобный интерфейс для настройки парамеров и передачи HTTP-запросов. Чтобы отправить HTTP-запрос, нужно выбрать HTTP-метод, указать URI запроса и нажать кнопку Send. При необходимости можно задать и другие элементы конфигурации — параметры запроса, заголовки, тело запроса. При нажатии на кнопку Send, Postman отправит HTTP-запрос. Когда он будет выполнен, Postman выведет содержимое HTTP-ответа.

Примеры работы cURL:

Запрос:

```
```

```
curl http://localhost:8080/hello
```

Ответ:

```
Hello!
```

Если в запросе используется HTTP-метод GET, его можно не указывать явно. Если же это какой-либо другой метод или если вы все же хотите четко прописать GET, нужно воспользоваться флагом -X:

Запрос:

```
curl -X GET http://localhost:8080/hello
```

Ответ:

```
Hello!
```

Чтобы получить другую информацию о HTTP-ответе, добавьте в команду флаг -v:

Запрос:

```
curl -v http://localhost:8080/hello
```

Ответ:

```
Trying ::1:8080...
* Connected to localhost (::1) port 8080 (#0)
> GET /hello HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.73.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: text/plain;charset=UTF-8
< Content-Length: 6
< Date: Fri, 25 Dec 2020 23:11:02 GMT
<
{ [6 bytes data]
100 6 100 6 0 0 857 0 --:--:-- --:--:-- --:--:--
1000
```

```
Hello!
* Connection #0 to host localhost left intact
```

Передача объектов в теле HTTP-ответа

Чтобы отправить объект клиенту в HTTP-ответе, нужно всего лишь сделать так, чтобы действие контроллера возвращало этот объект. По умолчанию Spring создает строку, представляющую объект, и дает ее в формате JSON. JavaScript Object Notation (JSON). В теле ответа можно также передавать экземпляры, представляющие собой коллекции объектов.

JSON — наиболее распространенный способ представления объектов при работе с конечными точками REST. При желании можно использовать в Spring другие форматы для тела запроса (такие как XML или YAML), подключив к объектам специальный преобразователь.

Пример в `com.sibsutis.MVCWithDynamicView.controllers.rest.responsebody`

Создание HTTP-ответа со статусом и заголовками

Статус ответа — это важный признак HTTP-ответа, который сообщает о результате запроса. По умолчанию Spring присваивает ответам следующие основные HTTP-статусы:

- 200 — OK — при обработке запроса на стороне сервера не возникло никаких исключений;
- 404 — Not Found — запрошенный ресурс не существует;
- 400 — Bad Request — часть запроса не соответствует тем данным, которые ожидает сервер;
- 500 — Error on server — при обработке запроса на стороне сервера по какой-то причине возникло исключение. Как правило, с подобным исключением клиент ничего не может сделать. Предполагается, что проблема будет решена на стороне бэкенда.

Самый простой и наиболее распространенный способ изменить HTTP-ответ — применить класс `ResponseEntity`. Этот класс Spring позволяет модифицировать тело, статус и заголовки HTTP-ответа.

Пример в `com.sibsutis.MVCWithDynamicView.controllers.rest.responsestatus`

Управление исключениями на уровне эндпоинта

Очень важно продумать, что случится, если действие контроллера выбросит исключение. Один из способов управления исключениями состоит в том, чтобы перехватывать их в действии контроллера и, используя класс ResponseEntity передавать другую конфигурацию ответа в случае возникновения исключения.

Пример в `com.sibsutis.MVCWithDynamicView.controllers.rest.exception`

Это хорошая методика, и многие разработчики ею пользуются для обработки исключений. Но в более сложных приложениях удобнее отделить управление исключениями от других обязанностей. Во-первых, иногда одно и то же исключение должно обрабатываться для нескольких конечных точек и, разумеется, мы не хотим, чтобы это привело к дублированию кода. Во-вторых, когда вам придется разбираться в работе тех или иных исключений, гораздо удобнее знать, что вся логика их обработки размещается в одном месте. Исходя из этих соображений, предпочтительно использовать совет REST-

контроллера — аспект, который перехватывает исключения, выдаваемые действиями контроллера, и применяет к ним написанную вами логику в зависимости от конкретного случая.

Пример в `com.sibsutis.MVCWithDynamicView.controllers.rest.aopexception`

На практике иногда приходится передавать из действия контроллера в совет дополнительную информацию о возникшем исключении. В таком случае нужно добавить параметр к методу-обработчику исключения в классе совета. Spring догадается, что нужно передать из контроллера в метод-обработчик ссылку на исключение. Затем можно будет использовать в логике совета любую информацию, извлеченную из экземпляра исключения.

Извлечение данных из тела запроса

Поскольку в основе конечных точек REST лежит все тот же механизм Spring MVC, синтаксис передачи данных через параметры запроса и переменные пути в этом случае ничем не отличается от уже известных. Эндпоинты REST создаются точно так же, как и действия контроллера для веб-страниц; в обоих случаях используются одни и те же аннотации.

У HTTP-запроса есть тело и его можно использовать для передачи данных от клиента серверу. Тело HTTP-запроса часто используется при отправке информации в конечные точки REST. Если нужно передать большой объем данных (все, что занимает больше 50–100 символов), следует использовать тело запроса.

Чтобы передать данные в теле запроса, достаточно снабдить параметр действия контроллера аннотацией @RequestBody. По умолчанию Spring предполагает, что данные в параметре, сопровождаемом аннотацией, представлены в формате JSON, и пытается преобразовать строку JSON в объект, соответствующий типу параметра. Если фреймворку это не удается, приложение возвращает HTTP-ответ со статусом `400 Bad Request`.

Пример в `com.sibsutis.MVCWithDynamicView.controllers.rest.requestbody`.

Тестирование контроллеров