Поддержка архитектуры REST в Spring

Составил: Черниговский А.С.
Старший преподаватель кафедры “Информатика”
ИКИТ СФУ

Архитектура REST

- Прежде всего нацелена на данные
- Окружена массой ошибочных домыслов
- REST означает REpresentational State Transfer (Википедия: «передача состояния представления»). Это популярный архитектурный подход для создания API в современном мире.
- REST - набор архитектурных принципов построения сервис-ориентированных систем.
- RESTful - прилагательное, употребляющееся по отношению к сервисам, которые следуют принципам REST

Аббревиатура REST

- Representational State Transfer
- **Представление** – ресурсы REST могут быть представлены практически в любой форме, включая XML, формат представления объектов JavaScript (JavaScript Object Notation, JSON) или даже HTML, какая лучше подходит для потребителя ресурсов;
- **Состояние** – в архитектуре REST состояние ресурсов представляет больший интерес, чем операции, которые можно выполнить над ресурсами;
- **Передача** – архитектура предусматривает организацию передачи данных ресурса в некотором представлении из одного приложения в другое. Проще говоря, архитектура REST – это комплекс решений, связанных с передачей информации о состоянии ресурса в некоторой форме от сервера клиенту (или наоборот).

Поддержка REST в Spring

• Контроллеры , способные обрабатывать все типы HTTP-запросов, включая четыре основных: GET, PUT, DELETE и POST;

• Новая аннотация @PathVariable, позволяющая контроллерам обрабатывать запросы к параметризованным адресам URL (URL, имеющие переменную часть пути);

• JSP-тег <form:form> из библиотеки тегов JSP, входящей в состав Spring, а также новый компонент HiddenHttpMethodFilter, позволяющий отправлять запросы PUT и DELETE из HTML-форм даже в браузерах, не поддерживающих эти HTTP-запросы;

• Представления и арбитры представлений в Spring, включая новые реализации представлений для отображения моделей данных в форматах XML, JSON, Atom и RSS, позволяют представлять ресурсы в самых разных форматах;

Поддержка REST в Spring

- С помощью нового арбитра представлений ContentNegotiatingView Resolver можно выбрать формат отображения ресурса, наиболее подходящий для клиента;
- Механизм отображения данных с применением представлений можно полностью обойти с помощью новой аннотации @ResponseBody и различных реализаций интерфейса HttpMethodConverter;
- Аналогично, для преобразования данных HTTP-запроса в Java-объекты, передаваемые методам-обработчикам контроллера, можно воспользоваться новой аннотацией @RequestBody наряду с реализациями интерфейса HttpMethod Converter;
- Класс RestTemplate, упрощающий доступ к ресурсам REST на стороне клиента.

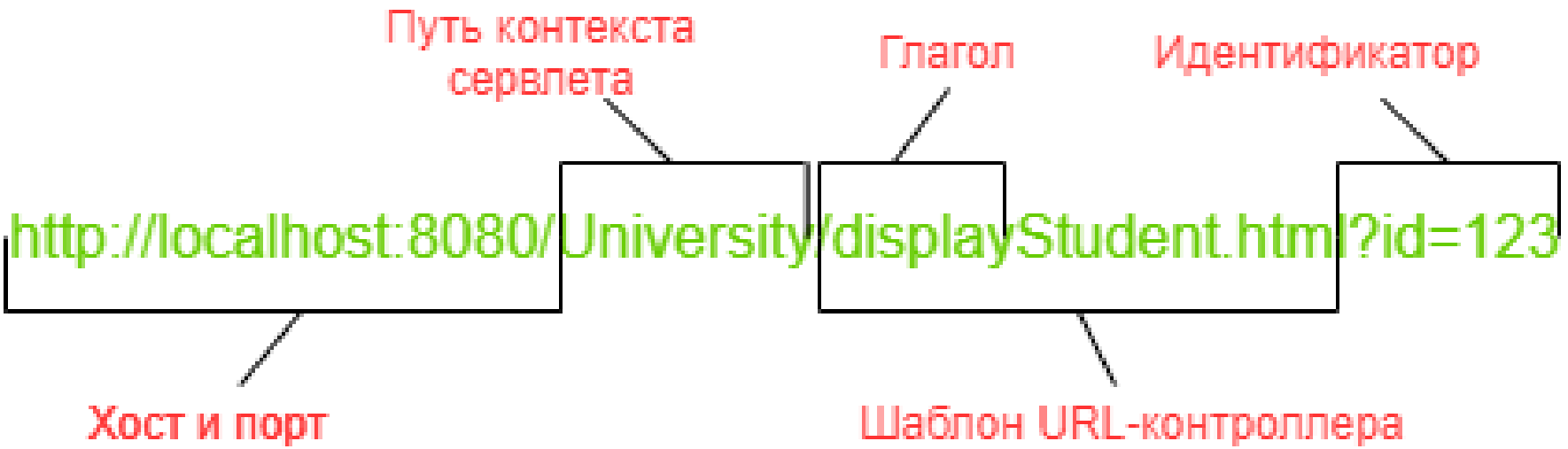
Устройство контроллера, противоречащего REST

```
@Controller
@RequestMapping("/displayStudent.html") // Отображение URL,
                                        // противоречащее REST
public class DisplayStudentController {
    private final StudentService studentService;

    @Autowired
    public DisplayStudentController(StudentService
                                                studentService) {
        this.studentService = srudentService;
    }

    @RequestMapping(method=RequestMethod.GET)
    public String showStudent(@RequestParam("id") long id,
                                                Model model)       {
        model.addAttribute(studentService.getStudentById(id));
        return "views/show";
    }
}
```

Обработка адресов URL в архитектуре RESTful

- URL – это аббревиатура от Uniform Resource Locator (унифицированный указатель ресурсов). Согласно этому названию, URL должен служить ссылкой на ресурс.
- Более того, все адреса URL являются также идентификаторами URI, или Uniform Resource Identifiers (унифицированные идентификаторы ресурсов).

URL, противоречащий архитектуре REST



Характеристики URL, соответствующих архитектуре REST

Иерархическая организация URL в стиле RESTful

- http://localhost:8080 – определяет доменное имя и порт. .
- http://localhost:8080/Univesity – определяет контекст сервлета приложения. Данный URL является более специализированным – он определяет приложение, выполняющееся на сервере.
- http://localhost:8080/University/students – определяет ресурс, представляющий список объектов Student внутри приложения University

- http://localhost:8080/University/students/123 – наиболее специализированный URL, определяющий конкретный ресурс Student. Самая интересная особенность адресов URL в стиле RESTful заключается в том, что пути в них являются параметризованными. В то время как URL, противоречащие архитектуре REST, несут входные данные в виде параметров запроса, адреса URL в стиле RESTful несут эти же данные внутри пути в URL.

Встраивание параметров в адреса URL

```
@Controller
@RequestMapping("/students")
public class StudentController {
	private StudentService studentService;
	@Inject
	public StudentController(StudentService studentService) {
		this.studentService = studentService;
	}
	@RequestMapping(value="/{id}", // Используется переменная-
					заполнитель
		method=RequestMethod.GET)
	public String getStudent(@PathVariable("id") long id, Model model) {
		model.addAttribute(studentService.getStudentById(id));
		return "students/view";
	}
}
```

@PathVariable без имени переменной заполнителя

```
@RequestMapping(value="/{id}", method=RequestMethod.GET)
public String getStudent(@PathVariable long id, Model model) {
      model.addAttribute(studentService.getStudentById(id));
       return "students/view";
}
```

Выполнение операций в стиле REST

Метод	Описание	Безопасный	Идемпотентный
GET	Извлекает ресурс с сервера. Ресурс идентифицируется адресом URL в запросе	Да	Да
POST	Посылает данные на сервер для обработки процессором, ожидающим поступления запросов по адресу URL в запросе	Нет	Нет
PUT	Помещает данные в ресурс на сервере, идентифицируемый адресом URL в запросе	Нет	Да
DELETE	Удаляет ресурс, идентифицируемый адресом URL в запросе	Нет	Да
OPTIONS	Запрашивает дополнительные параметры для взаимодействия с сервером	Да	Да
HEAD	Напоминает метод GET, за исключением того, что в ответ возвращаются только заголовки – содержимое не должно возвращаться в теле ответа	Да	Да
TRACE	В ответ на этот запрос сервер должен вернуть его тело обратно клиенту	Да	Да

Безопасность и идемпотентность

- Каждый из HTTP-методов характеризуется двумя чертами: безопасность и идемпотентность. Метод считается безопасным, если он не изменяет состояние ресурса. Идемпотентный метод может изменять или не изменять состояние ресурса, но повторные запросы, следующие за первым, не должны оказывать влияния на состояние ресурса. По определению все безопасные методы являются также идемпотентными, но не все идемпотентные методы являются безопасными.

Изменение ресурса с помощью PUT-запросов

```
@RequestMapping(value="/{id}",
method=RequestMethod.PUT)
@ResponseStatus(HttpStatus.NO_CONTENT)
public void putStudent(@PathVariable("id") long id,
                @Valid Student student) {
    studentService.saveStudent(student);
}
```

Обработка DELETE-запросов

```
@RequestMapping(value="/{id}",
method=RequestMethod.DELETE)
@ResponseStatus(HttpStatus.NO_CONTENT)
public void deleteStudent(@PathVariable("id") long id) {
    studentService.deleteStudent(id);
}
```

Создание ресурса с помощью POST-запроса

```
@RequestMapping(method=RequestMethod.POST)
                    // Обрабатывает POST-запросы
@ResponseStatus(HttpStatus.CREATED)
                    // Возвращает ответ HTTP 201
public @ResponseBody Student createStudent(@Valid Student
student, BindingResult result, HttpServletResponse response) throws
BindException {
    if(result.hasErrors()) {
            throw new BindException(result);
    }
    studentService.saveStudent(student);
    response.setHeader("Location", "/student/" + student.getId());
    return student; // Вернуть ресурс
}
```

Представление ресурсов

- Контроллеры обрабатывают ресурс в терминах Java-объектов. Но как только контроллер завершит свою работу, ресурс тут же будет трансформирован в формат, ожидаемый клиентом.
- Фреймворк Spring предоставляет два способа преобразования ресурса из представления на языке Java в представление, которое может быть отправлено клиенту:
    - отображение с помощью представлений на основе договоренностей;
    - преобразование HTTP-сообщений;

Договоренность о представлении ресурса

```
<bean id="contentNegotiationManager" class="org.springframework.web
                    .accept.ContentNegotiationManagerFactoryBean">
    <property name="favorPathExtension" value="true" />
    <property name="ignoreAcceptHeader" value="true"/>
    <property name="useJaf" value="false"/>
    <property name="defaultContentType" value="text/html" />

    <property name="mediaTypes">
        <map>
            <entry key="html" value="text/html"/>
            <entry key="json" value="application/json"/>
        </map>
    </property>
</bean>
```

Договоренность о представлении ресурса

```
<bean class="org.springframework.web.servlet.view.
                    ContentNegotiatingViewResolver">
    <property name="contentNegotiationManager"
        ref="contentNegotiationManager"/>
        <property name="viewResolvers">
    <list>
        <bean
class="org.springframework.web.servlet.view.freemarker.
                            FreeMarkerViewResolver"/>
        </list>
</property>
</bean>
```

Предоставление ресурса

- Чтобы понять принцип действия арбитра ContentNegotiatingView Resolver, необходимо знать, что договоренность о формате ресурса выполняется в два этапа:
- 1. Определяется тип возвращаемых данных.
- 2. Отыскивается представление для данного типа.

Определение запрошенного типа возвращаемых данных

- Клиент может явно указать требуемый формат в заголовке Accept запроса.
- Если клиент пользуется веб-браузером, нет никакой гарантии, что клиент желает получить именно то, что браузер отправляет в заголовке Accept.
- Арбитр ContentNegotiatingViewResolver учитывает содержимое заголовка Accept и выбирает то или иное представление в соответствии с ним, но только после того, как попытается определить расширение файла в URL.
- Если в конце URL присутствует расширение файла, будет выполнена попытка отыскать соответствующий ему элемент в свойстве mediaTypes. Свойство mediaTypes – это отображение, ключами в котором являются расширения имен файлов, а значениями – типы содержимого.

Изменение порядка определения типа содержимого

- Присвоив свойству favorPathExtension значение false, можно заставить ContentNegotiatingViewResolver игнорировать расширение файла в URL;
- Добавив фреймворк Java Activation Framework (JAF) в библиотеку классов (classpath), можно заставить ContentNegotiatingViewResolver обращаться к JAF за помощью в определении типа содержимого по расширению имени файла, если в свойстве mediaTypes не будет найден соответствующий элемент;
- Если присвоить свойству favorParameter значение true, тип содержимого будет определяться путем сопоставления параметра format в запросе (если присутствует) с элементами в свойстве mediaTypes;
- Присвоив свойству ignoreAcceptHeader значение true, можно исключить из рассмотрения заголовок Accept.

Поиск представления

- ContentNegotiatingView Resolver не определяет представление непосредственно, а предоставляет другим арбитрам представлений возможность выбрать наиболее подходящее представление, соответствующее требованиям клиента. Если не оговаривается иное, он будет использовать любые арбитры представлений, имеющиеся в контексте приложения. Но имеется возможность явно перечислить арбитры представлений, которые следует задействовать, перечислив их в свойстве viewResolvers.

Поиск представлений

- ContentNegotiatingViewResolver опросит все арбитры представлений, предложив им определить представление по логическому имени, и поместит полученные представления в список кандидатов. Кроме того, если определено свойство defaultView , представление, указанное в нем, также будет добавлено в конец списка.
- После составления списка кандидатов ContentNegotiatingViewResolver выполнит обход по всем запрашиваемым типам содержимого, пытаясь отыскать соответствующее представление из числа кандидатов. Поиск производится до первого совпадения.
- Если представление определить не удалось, ContentNegotiatingViewResolver вернет пустую ссылку (null). Или, если свойство useNotAcceptable StatusCode имеет значение true, будет возвращено представление с кодом состояния HTTP 406 (Not Acceptable).

Преобразование HTTP-сообщений

- Типичный метод контроллера Spring MVC завершается записью некоторых данных в модель и определением логического имени представления для отображения этих данных. Несмотря на большое разнообразие способов заполнения модели данными и идентификации представлений, все методы-обработчики контроллеров, встречавшиеся нам до сих пор, следовали этому основному шаблону.
- Но когда задача контроллера состоит в том, чтобы воспроизвести ресурс в некотором формате, существует более прямой путь к цели, минуя модели и представления. При таком подходе к реализации метода-обработчика возвращаемый им объект автоматически преобразуется в формат, запрошенный клиентом.
- Использование этого нового приема начинается с применения аннотации @ResponseBody к методу-обработчику контроллера.

Возврат ресурса в теле ответа

```
@RequestMapping(value = "/{username}", method =
RequestMethod.GET,
                headers = {"Accept=text/xml, application/json"})
 public @ResponseBody Student
getStudent(@PathVariable String username) {
    return studentService.getStudent(username);
}
```

Возврат ресурса в теле ответа

- Преобразование произвольных Java-объектов, возвращаемых методами обработчиками, в представление, доступное для клиента, выполняется одним из преобразователей HTTP-сообщений, входящих в состав Spring.
- Например, допустим, что в заголовке Accept запроса клиент сообщил, что он способен принимать данные в формате application/json. Допустим также, что в библиотеке классов приложения присутствует библиотека Jackson JSON. В этом случае объект, возвращаемый методом-обработчиком, можно передать преобразователю MappingJacksonHttpMessageConverter для преобразования его в формат JSON перед передачей клиенту.
- Обратите внимание, что преобразователи HTTP-сообщений регистрируются по умолчанию, поэтому их не требуется настраивать отдельно. Однако для их поддержки может понадобиться добавить дополнительные библиотеки в библиотеку классов приложения (classpath).

Прием ресурса в теле запроса

```
@RequestMapping(value = "/{username}",
method = RequestMethod.PUT, headers = "Content-Type=application/json")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void updateStudent(@PathVariable String username, @RequestBody Student student) {
    studentService.saveStudent(student);
}
```

Клиенты REST

```
public Student[] retrieveStudents(String name) {
try {
    HttpClient httpClient = new DefaultHttpClient();
    String studentsUrl = "http://localhost:8080/University/students/"
                        + name; // Подготовить URL
    HttpGet getRequest = new HttpGet(studentsUrl); // Создать запрос
    getRequest.setHeader( new BasicHeader("Accept", "application/json"));
        HttpResponse response = httpClient.execute(getRequest);
        HttpEntity entity = response.getEntity();
        ObjectMapper mapper = new ObjectMapper();
    return mapper.readValue(entity.getContent(), Student[].class);
} catch (IOException e) {
     throw new StudentClientException("Unable to retrieve Students", e);
}
}
```

Операции класса RestTemplate

- **exchange()** Выполняет HTTP-запрос требуемого типа к ресурсу с указанным URL и возвращает экземпляр ResponseEntity, содержащий объект, отображающий тело ответа
- **execute()** Выполняет HTTP-запрос требуемого типа к ресурсу с указанным URL, возвращает объект, отображающий тело ответа
- **getForEntity()** Выполняет запрос HTTP GET и возвращает экземпляр ResponseEntity, содержащий объект, отображающий тело ответа
- **getForObject()** Выполняет запрос HTTP GET и возвращает объект, отображающий тело ответа
- **headForHeaders()**
- **optionsForAllow()**
- **postForEntity()** Выполняет запрос HTTP POST и возвращает экземпляр ResponseEntity, содержащий объект, отображающий тело ответа
- **postForLocation()** Выполняет запрос HTTP POST и возвращает URL нового ресурса
- **postForObject()** Выполняет запрос HTTP POST и возвращает объект, отображающий тело ответа put() Выполняет запрос HTTP PUT, отправляя измененный ресурс с указанным URL

Чтение ресурсов

- <T> T getForObject(URI url, Class<T> responseType)

  throws RestClientException;
- <T> T getForObject(String url, Class<T> responseType, Object... uriVariables)

  throws RestClientException;
- <T> T getForObject(String url, Class<T> responseType, Map<String, ?> uriVariables)

  throws RestClientException;

Извлечение ресурсов

```
public Students[] retrieveStudents(String name) {
return new RestTemplate().getForObject(
        "http://localhost:8080/University/students/{name}",
         Student[].class, name);
}
```

Изменение ресурса

- public void updateStudent(Student student) throws StudentException {

  try {

  String url = "http://localhost:8080/University/students/"+ student.getId();

  new RestTemplate().put(new URI(url), student);

  } catch (URISyntaxException e) { throw new StudentUpdateException("Unable to update Student", e);
  }

  }

Удаление ресурсов

```
public void deleteStudent(long id) {
    try {
        restTemplate.delete(
            new
URI("http://localhost:8080/University/students/" + id));
    } catch (URISyntaxException wontHappen) { }
}
```

Прием объектов в ответах на POST-запросы

```
public Student postStudentForObject(Student student) {
RestTemplate rest = new RestTemplate();
return
rest.postForObject("http://localhost:8080/University/students",
    student, Student.class);
}
```

Заключение

Архитектура RESTful позволяет интегрировать приложения, основываясь на веб-стандартах, сохраняя взаимодействия простыми и естественными. Ресурсы в системе идентифицируются адресами URL, управляются с помощью методов HTTP и отображаются в форму, наиболее соответствующую требованиям клиента.

Заключение

В этой теме вы узнали, как писать контроллеры Spring MVC, обрабатывающие запросы управления ресурсами RESTful. Используя шаблоны параметризованных адресов URL и связывая методы контроллеров с конкретными методами HTTP, можно обеспечить обработку запросов GET, POST, PUT и DELETE на выполнение операций с ресурсами приложения.

- В ответ на запросы фреймворк Spring может обеспечить представление данных из ресурсов в формате, наиболее соответствующем требованиям клиента. При использовании представлений можно воспользоваться услугами арбитра представлений ContentNegotiatingViewResolver, который выберет представление из числа предлагаемых другими арбитрами представлений, лучше всего удовлетворяющее потребности клиента. Кроме того, метод-обработчик контроллера можно отметить аннотацией @ResponseBody, избавившись тем самым от этапа выбора представления, и поручить одному из нескольких преобразователей HTTP-сообщений превратить возвращаемое значение в ответ, который будет отправлен клиенту.

Спасибо за внимание!