

REST



Григорий
Скобелев



Григорий Скобелев
Java Developer, ЮMoney



План занятия

1. [REST](#)
2. [Работы с ошибками в Spring](#)
3. [Валидация запросов](#)
4. [WebMvcConfigurer](#)
5. [Итоги](#)
6. [Домашнее задание](#)



REST



REST

REST (Representational State Transfer) - это архитектурный стиль взаимодействия между разными компонентами системы, которые могут находиться в разных местах.

RESTful приложение - это приложение, которое построено с учетом ограничений, наложенных архитектурой REST.

REST

Главные принципы REST:

- Архитектура клиент-сервер
- Отсутствие состояния
- Кэширование
- Единообразный интерфейс
- Слои
- Код по запросу

Повторим

Основные аннотации для работы с REST в Spring:

- **@RestController.** Указывает на класс-обработчик пользовательских запросов
- **@RequestMapping.** Формирует соответствие путь<->обработчик
- **@RequestBody.** Указывает, что надо десериализовать объект из тела запроса
- **@RequestParam.** Указывает, что надо десериализовать объект из query запроса
- **@PathVariable.** Указывает, что надо десериализовать объект из пути запроса

Повторим

```
@RestController
@RequestMapping("/test")
public class ExampleController {
    @RequestMapping(value = "/get", method = RequestMethod.GET)
    public String get() {
        return "Request with GET method";
    }

    @PostMapping(value = "/post")
    public String post() {
        return "Request with POST method";
    }
}
```

Маркер класса-контроллера

Указание начала пути для всех обработчиков этого класса

Тело ответа

Объявление POST обработчика для пути /test/post

Тело ответа



Работы с ошибками в Spring



Работа с ошибками в Spring

Для работы с ошибками и удобным управлением потоком выполнения программы на основе этих ошибок, Spring предоставляет расширяемые механизмы отлова этих ошибок как локально, так и глобально.

HandlerExceptionHandlerResolver

В Spring реализации интерфейса **HandlerExceptionHandlerResolver** обрабатывают неожиданные исключения, возникающие во время выполнения методов контроллера.

```
public interface HandlerExceptionHandlerResolver {  
    /** Try to resolve the given exception that got thrown during handler execution, ...*/  
    @Nullable  
    ModelAndView resolveException(  
        HttpServletRequest request, HttpServletResponse response, @Nullable Object handler, Exception ex);  
}
```

HandlerExceptionResolver

Spring предоставляет несколько базовых реализацию интерфейса `HandlerExceptionResolver`:

- **`SimpleMappingExceptionResolver`**. Позволяют декларативно маппить исключения в определенные представления вместе с некоторой дополнительной логикой
- **`DefaultHandlerExceptionResolver`**. Переводит стандартные исключения Spring в REST статусы для ответа.
- **`ExceptionHandlerExceptionResolver`**. Позволяет обрабатывать ошибки в контроллерах на основе типа ошибки.



HandlerExceptionResolver

Первые два класса по умолчанию включаются DispatcherServlet'ом.

Третий же класс включается, если в вашем контроллере есть методы, умеющие работать с exception'ами.

Про них и другие способы обрабатывать ошибки мы и поговорим.

Работа с exception'ами

Для работы с ошибками Spring предлагает несколько решений:

- **ResponseStatusException.** Это RuntimeException, который инкапсулирует в себе логику работы с REST статусами.
- **@ExceptionHandler.** Аннотация, которая помещается над методом, умеющим обрабатывать ошибки контроллера.
- **@ControllerAdvice.** Аннотация, которой помечается класс со сквозной логикой для нескольких контроллеров.

ResponseStatusException

```
@GetMapping(value =("/{id}")  
public Person findById(@PathVariable("id") Long id) {  
    try {  
        Person resourceById = service.findById(id);  
        return resourceById;  
    }  
    catch (PersonNotFoundException exc) {  
        throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Person Not Found", exc);  
    }  
}
```

@ExceptionHandler

@ExceptionHandler помечают методы, в которые Spring может передать следующие переменные:

- **ServletRequest, ServletResponse.** Объекты, которые содержат данные о переданном запросе, или о отдаваемом ответе.
- **@ResponseBody.** Объекты для REST ответа в виде json/xml.
- **ResponseEntity.** Класс инкапсулирующий в себе работы со статусами, телом и заголовками ответа.
- **void.** В случае, если ошибку нет необходимости в отправке ошибки на клиент.
- **И другие.**

@ExceptionHandler

Указываем какую ошибку
обрабатываем

```
@ExceptionHandler(IllegalArgumentException.class)
ResponseBody<String> handleIAE(IllegalArgumentException e) {
    return new ResponseEntity<>() {
        body: "Got an exception: " + e.getMessage(), HttpStatus.BAD_REQUEST);
    }
}
```

Можем указать какой
возвращаем REST статус

```
@ResponseStatus(HttpStatus.NOT_FOUND)
@ExceptionHandler(PersonNotFoundException.class)
String handlePersonNotFound(PersonNotFoundException e) {
    return "Can't find person: " + e.getMessage();
}

@ExceptionHandler(RuntimeException.class)
ResponseBody<String> handleRuntime(PersonNotFoundException e) {
    return new ResponseEntity<>() {
        body: "Something went wrong: " + e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

@ControllerAdvice

@ControllerAdvice помечают классы, в которых будет находиться сквозная логика для нескольких контроллеров. В отличие от `ExceptionHandler` применяет логику работы с ошибками к нескольким контроллерам из указанного пакета.

@ControllerAdvice

```
public @interface ControllerAdvice {  
  
    /** Alias for the {@link #basePackages} attribute. ...*/  
    @AliasFor("basePackages")  
    String[] value() default {};  
  
    /** Array of base packages. ...*/  
    @AliasFor("value")  
    String[] basePackages() default {};  
  
    /** Type-safe alternative to {@link #value()} for specifying the packages ...*/  
    Class<?>[] basePackageClasses() default {};  
  
    /** Array of classes. ...*/  
    Class<?>[] assignableTypes() default {};  
  
    /** Array of annotations. ...*/  
    Class<? extends Annotation>[] annotations() default {};  
  
}
```

Указываем пакет, в котором
будет вестись поиск
контроллеров(оба способа
равноценны)

Указываем, что контроллеры
с этой аннотации будут
обрабатываться этим
ControllerAdvice классом



Валидация запросов

Валидация запросов

Для валидации объектов запросов в контроллерах Spring использует predetermined бины Java Validation API. Для этого либо контроллер, либо объект, который валидируем, должны быть помечены аннотацией `@Validated`.

Пример валидации полей объекта

```
@Validated
public class Input {

    @Min(1)
    @Max(10)
    private int numberBetweenOneAndTen;

    @Pattern(regexp = "[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}$")
    private String ipAddress;

    public int getNumberBetweenOneAndTen() {
        return numberBetweenOneAndTen;
    }

    public void setNumberBetweenOneAndTen(int numberBetweenOneAndTen) {
        this.numberBetweenOneAndTen = numberBetweenOneAndTen;
    }

    public String getIpAddress() {
        return ipAddress;
    }

    public void setIpAddress(String ipAddress) {
        this.ipAddress = ipAddress;
    }
}
```

Говорим Spring, что у объекта
есть аннотации для
ограничений

Конкретные ограничения

Пример валидации полей объекта

```
@RestController
class ValidateRequestBodyController {
    @PostMapping("/validateBody")
    ResponseEntity<String> validateBody(@Valid @RequestBody Input input) { return ResponseEntity.ok("valid"); }
}
```

Пример валидации объектов метода

```
@RestController
@RequestMapping("validate")
@Validated
class ValidateParametersController {
    @GetMapping("/pathVariable/{id}")
    ResponseEntity<String> validatePathVariable(@PathVariable("id") @Min(5) int id) { return ResponseEntity.ok("valid"); }

    @GetMapping("/requestParameter")
    ResponseEntity<String> validateRequestParameter(@RequestParam("param") @Min(5) int param) {
        return ResponseEntity.ok("valid");
    }
}
```

Теперь на уровне класса

Ограничение на длину для объекта метода



Валидация запросов

Для валидации объектов запросов в контроллерах Spring использует predetermined бины Java Validation API.



WebMvcConfigurer

REST в Spring boot

При помощи автоконфигураций создается множество сконфигурированных бинов:

- **RequestMappingHandlerAdapter.** Создает базовый бин этого класса, с заполненными конвертерами входящих сообщений в теле запроса. Умеет сериализовывать/десериализовывать json/xml и тд.
- **DispatcherServlet.** Создает бин с настроенными маппингами и настройками из вашего application.properties.
- **И многое другое.** Типовые валидаторы, конвертеры и еще много упрощающих жизнь бинов и настроек.



REST в Spring boot

Но также в Spring boot есть возможность изменять/дополнять ту функциональность, с которой вы работаете в своих контроллерах, в одном месте.

WebMvcConfigurer

WebMvcConfigurer - это интерфейс, с помощью которого можно конфигурировать то, как будет ваше приложение работать с REST. Он может:

- Регистрировать Interceptor классы, которые могут добавлять дополнительную логику перед и после выполнения методов контроллера
- Регистрировать контроллер для возвращения статичных ресурсов в ответ на запрос
- Кастомные конвертеры объектов для пользовательских типов данных, которые не могут быть десериализованы стандартным образом

Пользовательский конвертер

Для работы пользовательских конвертеров, необходимо реализовать интерфейс `Converter` и зарегистрировать его.

`@Configuration`

```
public class RulesWebMvcConfig implements WebMvcConfigurer {  
    private static final Map<String, InstrumentType> types = Stream.of(Bond.values(), Equity.values())  
        .flatMap(Stream::of).collect(Collectors.toMap(InstrumentType::name, Function.identity()));
```

`@Override`

```
public void addFormatters(formatterRegistry registry) {  
    registry.addConverter(new SpringRateFeatureConverter());  
}
```

```
private static class SpringRateFeatureConverter implements Converter<String, InstrumentType> {  
    @Override  
    public InstrumentType convert(String source) {  
        return types.getOrDefault(source, Unknown.UNKNOWN);  
    }  
}
```

Пользовательский конвертер

Для удобной работы со статичным ресурсами необходимо поставить в соответствие относительный путь, по которому надо вернуть ресурс, и название и путь ресурса, который Spring искать в папке resources/<путь к ресурсу и его название>

```
@Configuration
public class StaticResourceMapping implements WebMvcConfigurer {
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        String index = "index";
        Consumer<String> registerViewController = path -> registry.addViewController( urlPath: "/" + path).setViewName(index);
        Stream.of("/", "ui", index).forEach(registerViewController);
    }
}
```

Пользовательские аннотации

Этот способ похож на работу с пользовательским конвертером тем, что мы можем обрабатывать определенные объекты способом, который в Spring не предусмотрен. Но здесь дается большая свобода относительно входных данных

```
@Configuration
public class FeatureGroupWebMvcConfig implements WebMvcConfigurer {
    private static final Map<String, InstrumentType> types = Stream.of(Bond.values(), Equity.values())
        .flatMap(Stream::of).collect(Collectors.toMap(InstrumentType::name, Function.identity()));

    @Override
    public void addArgumentResolvers(List<HandlerMethodArgumentResolver> resolvers) {
        resolvers.add(new CustomParamResolver());
    }

    private class CustomParamResolver implements HandlerMethodArgumentResolver {
        @Override
        public boolean supportsParameter(MethodParameter parameter) {
            return parameter.hasParameterAnnotation(Instrument.class);
        }

        @Override
        public Object resolveArgument(MethodParameter parameter, ModelAndViewContainer mavContainer,
            NativeWebRequest webRequest, WebDataBinderFactory binderFactory) {
            String feature = webRequest.getParameter(paramName: "feature");

            return types.getOrDefault(feature, Unknown.UNKNOWN);
        }
    }
}
```




Итоги



Итоги

Spring дает нам много возможностей для работы с REST окружением нашего приложения.

Он дает нам удобно управлять потоком выполнения программы с помощью перехватчиков exception'ов как локально, так и глобально.

Также, он позволяет нам удобно валидировать и конфигурировать наши пользовательские типы данных.

Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и
пишите отзыв о лекции!**

Григорий Скобелев