



SpringDoc枚举字段处理与SpringBoot接收枚举参数处理

叹雪飞花 2023-11-22 👁 459 ⌚ 阅读7分钟

关注

本期内容

1. 添加SpringDoc配置展示枚举字段，在文档页面中显示枚举值和对应的描述
2. 添加SpringMVC配置使项目可以接收枚举值，根据枚举值找到对应的枚举

默认内容

先不做任何处理看一下直接使用枚举当做入参是什么效果。

1. 定义一个枚举

▼ java 复制代码

```
1 package com.example.enums;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Getter;
5
6 /**
7  * 来源枚举
8  *
9  * @author vains
10  */
11 @Getter
12 @AllArgsConstructor
13 public enum SourceEnum {
14
15     /**
16      * 1-web网站
17      */
18     WEB(1, "web网站"),
```



```
22     */
23     APP(2, "APP应用");
24
25     /**
26     * 源代码
27     */
28     private final Integer value;
29
30     /**
31     * 来源名称
32     */
33     private final String source;
34
35 }
36
```

2. 定义一个入参类

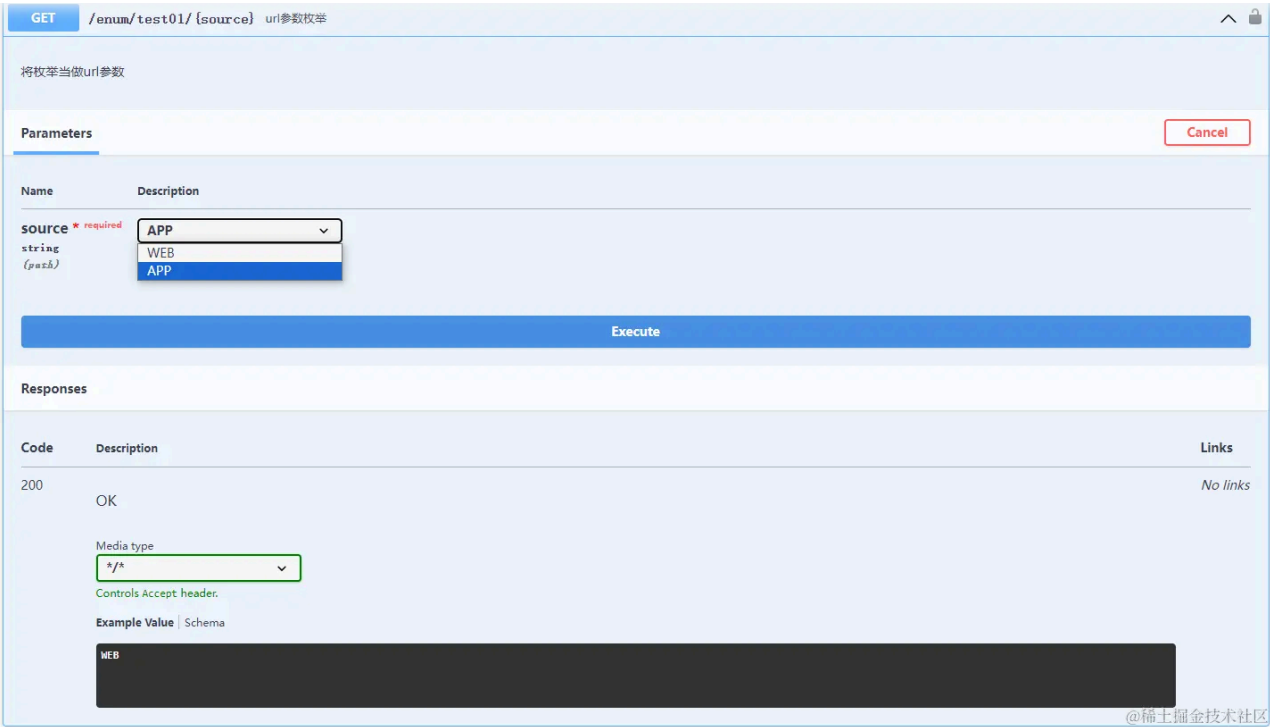


java 复制代码

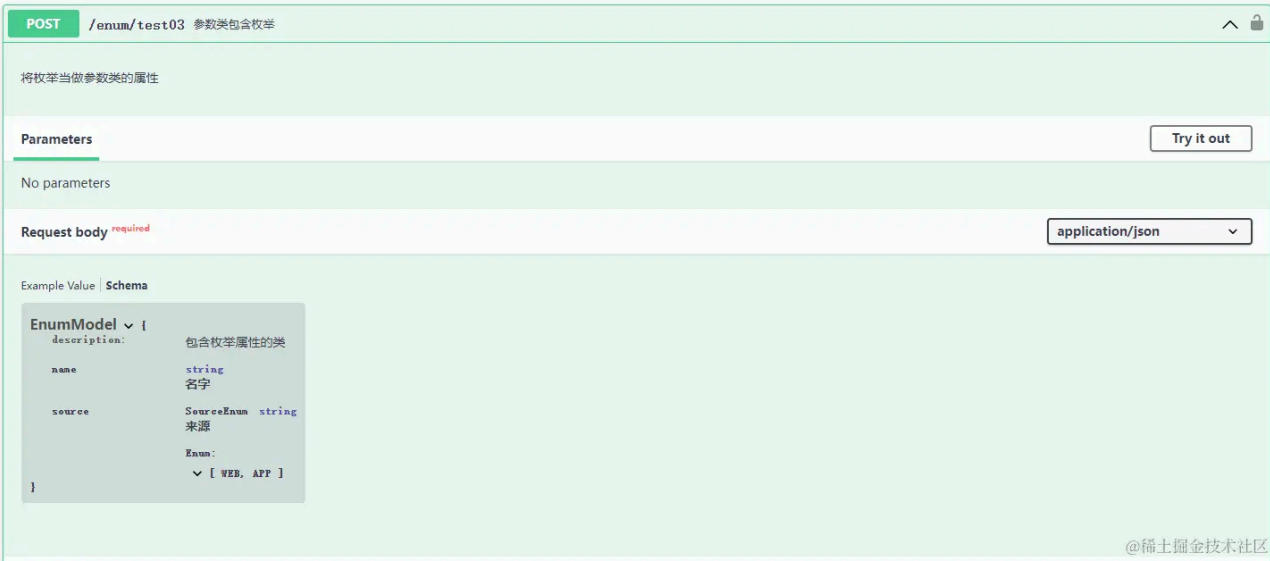
```
1 package com.example.model;
2
3 import com.example.enums.SourceEnum;
4 import io.swagger.v3.oas.annotations.media.Schema;
5 import lombok.Data;
6
7 /**
8  * 枚举属性类
9  *
10  * @author vains
11  */
12 @Data
13 @Schema(title = "包含枚举属性的类")
14 public class EnumModel {
15
16     @Schema(title = "名字")
17     private String name;
18
19     @Schema(title = "来源")
20     private SourceEnum source;
21
22 }
23
```

3. 定义一个接口，测试枚举入参的效果

```
2
3 import com.example.enums.SourceEnum;
4 import com.example.model.EnumModel;
5 import io.swagger.v3.oas.annotations.Operation;
6 import io.swagger.v3.oas.annotations.tags.Tag;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.PathVariable;
9 import org.springframework.web.bind.annotation.PostMapping;
10 import org.springframework.web.bind.annotation.RequestBody;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RestController;
13
14 /**
15  * 枚举接口
16  *
17  * @author vains
18  */
19 @RestController
20 @RequestMapping("/enum")
21 @Tag(name = "枚举入参接口", description = "提供以枚举作为入参的接口，展示SpringDoc自定义配置效果")
22 public class EnumController {
23
24     @GetMapping("/test01/{source}")
25     @Operation(summary = "url参数枚举", description = "将枚举当做url参数")
26     public SourceEnum test01(@PathVariable SourceEnum source) {
27         return source;
28     }
29
30     @GetMapping("/test02")
31     @Operation(summary = "查询参数枚举", description = "将枚举当做查询参数")
32     public SourceEnum test02(SourceEnum source) {
33         return source;
34     }
35
36     @PostMapping(value = "/test03")
37     @Operation(summary = "参数类包含枚举", description = "将枚举当做参数类的属性")
38     public EnumModel test03(@RequestBody EnumModel model) {
39         return model;
40     }
41
42 }
43
```



作为参数属性显示效果



文档中默认显示枚举可接收的值是定义的枚举名字(`APP` , `WEB`), 但是在实际开发中前端会传入枚举对应的值/代码(`1` , `2`), 根据代码映射到对应的枚举。

解决方案

单个处理方案

详细内容见[文档](#)

使用 `@Parameter` 注解(方法上/参数前)或者 `@Parameters` 注解来指定枚举参数可接受的值。如下所示

例1

▼

java 复制代码

```
1 @GetMapping("/test01/{source}")
2 @Parameter(name = "source", schema = @Schema(description = "来源枚举", type = "int32", allowable
3 @Operation(summary = "url参数枚举", description = "将枚举当做url参数")
4 public SourceEnum test01(@PathVariable SourceEnum source) {
5     return source;
6 }
```

◀ ▶

例2

▼

java 复制代码

```
1 @GetMapping("/test01/{source}")
2 @Operation(summary = "url参数枚举", description = "将枚举当做url参数")
3 public SourceEnum test01(@PathVariable
4     @Parameter(name = "source", schema =
5         @Schema(description = "来源枚举", type = "int32", allowableValues =
6     SourceEnum source) {
7     return source;
8 }
```

◀ ▶

单独枚举入参显示效果

Parameters

Try it out

Name	Description
source <small>required</small> int32 (path)	Available values: 1, 2 <div>1</div>

Responses

Code	Description	Links
200	OK <div>Media type */* Controls Accept header. Example Value Schema WEB</div>	No links

@稀土掘金技术社区

枚举作为参数类属性

单独处理没有好的办法，像上边添加 allowableValues 属性只会在原有列表上添加，如下

POST /enum/test03 参数类包含枚举

将枚举当做参数类的属性

Parameters

Try it out

No parameters

Request body required

application/json

Example Value | Schema

EnumModel {
description: 包含枚举属性的类
name > [...]
source SourceEnum string 来源
Enum: [WEB, APP, 1, 2]
}

@稀土掘金技术社区

全局统一处理方案

准备工作

1. 定义一个统一枚举接口

```

3  import com.fasterxml.jackson.annotation.JsonValue;
4
5  import java.io.Serializable;
6  import java.util.Arrays;
7  import java.util.Objects;
8
9  /**
10   * 通用枚举接口
11   *
12   * @param <V> 枚举值的类型
13   * @param <E> 子枚举类型
14   * @author vains
15   */
16  public interface BasicEnum<V extends Serializable, E extends Enum<E>> {
17
18      @JsonValue
19      V getValue();
20
21      /**
22       * 根据子枚举和子枚举对应的入参值找到对应的枚举类型
23       *
24       * @param value 子枚举中对应的值
25       * @param clazz 子枚举类型
26       * @param <B> {@link BasicEnum} 的子类型
27       * @param <V> 子枚举值的类型
28       * @param <E> 子枚举的类型
29       * @return 返回 {@link BasicEnum} 对应的子类实例
30       */
31      static <B extends BasicEnum<V, E>, V extends Serializable, E extends Enum<E>> B fromValue(V
32          return Arrays.stream(clazz.getEnumConstants())
33              .filter(e -> Objects.equals(e.getValue(), value))
34              .findFirst().orElse(null);
35      }
36
37  }
38

```

我这里为了通用性将枚举值的类型也设置为泛型类型了，如果不需要可以设置为具体的类型，比如 `String`、`Integer` 等，如果像我这样处理起来会稍微麻烦一些；另外我这里只提供了一个 `getValue` 的抽象方法，你也可以再提供一个 `getName`、`getDescription` 等获取枚举描述字段值的抽象方法。

2. 让项目中的枚举实现 `BasicEnum` 接口并重写 `getValue` 方法，如下



```
2
3 import lombok.AllArgsConstructor;
4 import lombok.Getter;
5
6 /**
7  * 来源枚举
8  *
9  * @author vains
10  */
11 @Getter
12 @AllArgsConstructor
13 public enum SourceEnum implements BasicEnum<Integer, SourceEnum> {
14
15     /**
16      * 1-web网站
17      */
18     WEB(1, "web网站"),
19
20     /**
21      * 2-APP应用
22      */
23     APP(2, "APP应用");
24
25     /**
26      * 来源代码
27      */
28     private final Integer value;
29
30     /**
31      * 来源名称
32      */
33     private final String source;
34
35 }
36
```

3. 定义一个基础自定义接口,提供一些对枚举的操作方法



java 复制代码

```
1 package com.example.config.basic;
2
3 import io.swagger.v3.core.util.PrimitiveType;
4 import io.swagger.v3.oas.models.media.ObjectSchema;
5 import io.swagger.v3.oas.models.media.Schema;
```



```
9  import java.lang.reflect.Field;
10 import java.lang.reflect.Method;
11 import java.lang.reflect.Modifier;
12 import java.lang.reflect.Type;
13 import java.util.Arrays;
14 import java.util.Comparator;
15 import java.util.List;
16 import java.util.Objects;
17 import java.util.stream.Collectors;
18
19 /**
20  * 基础自定义接口
21  *
22  * @author vains
23  */
24 public interface BasicEnumCustomizer {
25
26     /**
27      * 获取枚举的所有值
28      *
29      * @param enumClazz 枚举的class
30      * @return 枚举的所有值
31      */
32     default List<Object> getValues(Class<?> enumClazz) {
33         return Arrays.stream(enumClazz.getEnumConstants())
34             .filter(Objects::nonNull)
35             .map(item -> {
36                 // 收集values
37                 Method getValue = ReflectionUtils.findMethod(item.getClass(), "getValue");
38                 if (getValue != null) {
39                     ReflectionUtils.makeAccessible(getValue);
40                     return ReflectionUtils.invokeMethod(getValue, item);
41                 }
42                 return null;
43             }).filter(Objects::nonNull).toList();
44     }
45
46     /**
47      * 获取值和描述对应的描述信息，值和描述信息以“:”隔开
48      *
49      * @param enumClazz 枚举class
50      * @return 描述信息
51      */
52     default String getDescription(Class<?> enumClazz) {
53         List<Field> fieldList = Arrays.stream(enumClazz.getDeclaredFields())
54             .filter(f -> !Modifier.isStatic(f.getModifiers()))
```

```
58     fieldList.forEach(ReflectionUtils::makeAccessible);
59     return Arrays.stream(enumClazz.getEnumConstants())
60         .filter(Objects::nonNull)
61         .map(item -> fieldList.stream()
62             .map(field -> ReflectionUtils.getField(field, item))
63             .map(String::valueOf)
64             .collect(Collectors.joining(" : ")))
65         .collect(Collectors.joining("; "));
66 }
67
68 /**
69  * 根据枚举值的类型获取对应的 {@link Schema} 类
70  * 这么做是因为当SpringDoc获取不到属性的具体类型时会自动生成一个string类型的 {@link Schema} ,
71  * 所以需要根据枚举值的类型获取不同的实例, 例如 {@link io.swagger.v3.oas.models.media.IntegerSc
72  * {@link io.swagger.v3.oas.models.media.StringSchema}
73  *
74  * @param type          枚举值的类型
75  * @param sourceSchema 从属性中加载的 {@link Schema} 类
76  * @return 获取枚举值类型对应的 {@link Schema} 类
77  */
78 @SuppressWarnings({"unchecked"})
79 default Schema<Object> getSchemaByType(Type type, Schema<?> sourceSchema) {
80     Schema<Object> schema;
81     PrimitiveType item = PrimitiveType.fromType(type);
82
83     if (item == null) {
84         schema = new ObjectSchema();
85     } else {
86         schema = item.createProperty();
87     }
88
89     // 获取schema的type和format
90     String schemaType = schema.getType();
91     String format = schema.getFormat();
92     // 复制原schema的其它属性
93     BeanUtils.copyProperties(sourceSchema, schema);
94
95     // 使用根据枚举值类型获取到的schema
96     return schema.type(schemaType).format(format);
97 }
98
99 }
100
```

一个基础的 **Customizer** 接口。

实现枚举参数自定义

定义一个 **ApiEnumParameterCustomizer** 类并实现 **ParameterCustomizer** 接口，实现对枚举入参的自定义，同时实现 **BasicEnumCustomizer** 接口使用工具方法。

[java](#) [复制代码](#)

```
1 package com.example.config.customizer;
2
3 import com.example.config.basic.BasicEnumCustomizer;
4 import com.example.enums.BasicEnum;
5 import io.swagger.v3.oas.models.media.Schema;
6 import io.swagger.v3.oas.models.parameters.Parameter;
7 import org.springdoc.core.customizers.ParameterCustomizer;
8 import org.springframework.core.MethodParameter;
9 import org.springframework.stereotype.Component;
10
11 /**
12  * 枚举参数自定义配置
13  *
14  * @author vains
15  */
16 @Component
17 public class ApiEnumParameterCustomizer implements ParameterCustomizer, BasicEnumCustomizer {
18
19     @Override
20     public Parameter customize(Parameter parameterModel, MethodParameter methodParameter) {
21         Class<?> parameterType = methodParameter.getParameterType();
22
23         // 枚举处理
24         if (BasicEnum.class.isAssignableFrom(parameterType)) {
25
26             parameterModel.setDescription(getDescription(parameterType));
27
28             Schema<Object> schema = new Schema<>();
29             schema.setEnum(getValues(parameterType));
30             parameterModel.setSchema(schema);
31         }
32
33         return parameterModel;
34     }
```

实现枚举属性的自定义

定义一个 **ApiEnumPropertyCustomizer** 类并实现 **PropertyCustomizer** 接口，实现对枚举属性的自定义，同时实现 **BasicEnumCustomizer** 接口使用工具方法。

java 复制代码

```
1 package com.example.config.customizer;
2
3 import com.example.config.basic.BasicEnumCustomizer;
4 import com.example.enums.BasicEnum;
5 import com.fasterxml.jackson.databind.type.SimpleType;
6 import io.swagger.v3.core.converter.AnnotatedType;
7 import io.swagger.v3.oas.models.media.Schema;
8 import org.springdoc.core.customizers.PropertyCustomizer;
9 import org.springframework.stereotype.Component;
10 import org.springframework.util.ObjectUtils;
11
12 import java.lang.reflect.ParameterizedType;
13 import java.lang.reflect.Type;
14
15 /**
16  * 枚举属性自定义配置
17  *
18  * @author vains
19  */
20 @Component
21 public class ApiEnumPropertyCustomizer implements PropertyCustomizer, BasicEnumCustomizer {
22
23     @Override
24     public Schema<?> customize(Schema property, AnnotatedType type) {
25         // 检查实例并转换
26         if (type.getType() instanceof SimpleType fieldType) {
27             // 获取字段class
28             Class<?> fieldClazz = fieldType.getRawClass();
29             // 是否是枚举
30             if (BasicEnum.class.isAssignableFrom(fieldClazz)) {
31                 // 获取父接口
32                 if (fieldClazz.getGenericInterfaces()[0] instanceof ParameterizedType parameteri
33
34                 // 通过父接口获取泛型中枚举值的class类型
35                 Type actualTypeArgument = parameterizedType.getActualTypeArguments()[0];
36                 Schema<Object> schema = getSchemaByType(actualTypeArgument, property);
37
```

```
41         // 获取字段注释
42         String description = this.getDescription(fieldClazz);
43
44         // 重置字段注释和标题为从枚举中提取的
45         if (ObjectUtils.isEmpty(property.getTitle())) {
46             schema.setTitle(description);
47         } else {
48             schema.setTitle(property.getTitle() + " (" + description + ")");
49         }
50         if (ObjectUtils.isEmpty(property.getDescription())) {
51             schema.setDescription(description);
52         } else {
53             schema.setDescription(property.getDescription() + " (" + description + ")");
54         }
55         return schema;
56     }
57 }
58 }
59 return property;
60 }
61
62 }
63
```

如果读者不喜欢这样的效果可以自行修改枚举值、描述信息的显示效果

重启项目查看效果

 稀土掘金

首页 ▾

探索稀土掘金



将枚举当做url参数

Parameters

Try it out

Name	Description
source * required (path)	1 : web网站; 2 : APP应用 Available values: 1, 2 <div>1 ▾</div>

Responses

Code	Description	Links
200	OK <div>Media type */* ▾ Controls Accept header. Example Value Schema WEB</div>	No links

@稀土掘金技术社区

接口2

GET /enum/test02 查询参数枚举

将枚举当做查询参数

Parameters

Try it out

Name	Description
source * required (query)	1 : web网站; 2 : APP应用 Available values: 1, 2 <div>1 ▾</div>

Responses

Code	Description	Links
200	OK <div>Media type */* ▾ Controls Accept header. Example Value Schema WEB</div>	No links

@稀土掘金技术社区

将枚举当做参数类的属性

Parameters

Try it out

No parameters

Request body required

application/json ▾

Example Value | Schema

```
EnumModel ▾ {
  description: 包含枚举属性的类

  name 名字 > [...]
  source integer ($int32)
        title: 来源 (1 : web网站; 2 : APP应用)
        1 : web网站; 2 : APP应用

  Enum:
    ▾ [ 1, 2 ]
}
```

@稀土掘金技术社区

SpringBoot接收枚举入参处理

不知道大家有没有注意到 `BasicEnum` 接口中的抽象方法 `getValue` 上有一个 `@JsonValue` 注解，这个注解会在进行Json序列化时会将该方法返回的值当做当前枚举的值，例如：1/2，如果不加该注解则序列化时会直接变为枚举的名字，例如: APP/WEB。

如果Restful接口入参中有 `@RequestBody` 注解则在——统一枚举的 `getValue` 方法上有 `@JsonValue` 注解的基础上，无需做任何处理，对于Json入参可以这样处理，但是对于POST表单参数或GET查询参数需要添加单独的处理。

定义一个 `EnumConverterFactory`

根据枚举的class类型获取对应的 `converter`，并在 `converter` 中直接将枚举值转为对应的枚举，具体逻辑情况代码中的注释

▾

java 复制代码

```
1 package com.example.config.converter;
2
3 import com.example.enums.BasicEnum;
4 import com.fasterxml.jackson.databind.type.TypeFactory;
5 import lombok.NonNull;
6 import org.springframework.core.convert.converter.Converter;
7 import org.springframework.core.convert.converter.ConverterFactory;
8 import org.springframework.stereotype.Component;
9 import org.springframework.util.ReflectionUtils;
10
```

```
14 import java.lang.reflect.Type;
15 import java.util.function.Function;
16
17 /**
18  * 处理除 {@link org.springframework.web.bind.annotation.RequestBody} 注解标注之外是枚举的入参
19  *
20  * @param <V> 枚举值的类型
21  * @param <E> 枚举的类型
22  * @author vains
23  */
24 @Component
25 public class EnumConverterFactory<V extends Serializable, E extends Enum<E>> implements Converter<
26
27     @NonNull
28     @Override
29     @SuppressWarnings("unchecked")
30     public <T extends BasicEnum<V, E>> Converter<String, T> getConverter(Class<T> targetType) {
31         // 获取父接口
32         Type baseInterface = targetType.getGenericInterfaces()[0];
33         if (baseInterface instanceof ParameterizedType parameterizedType
34             && parameterizedType.getActualTypeArguments().length == 2) {
35             // 获取具体的枚举类型
36             Type targetActualTypeArgument = parameterizedType.getActualTypeArguments()[1];
37             Class<?> targetAawArgument = TypeFactory.defaultInstance()
38                 .constructType(targetActualTypeArgument).getRawClass();
39             // 判断是否实现自通用枚举
40             if (BasicEnum.class.isAssignableFrom(targetAawArgument)) {
41                 // 获取父接口的泛型类型
42                 Type valueArgument = parameterizedType.getActualTypeArguments()[0];
43                 // 获取值的class
44                 Class<V> valueRaw = (Class<V>) TypeFactory.defaultInstance()
45                     .constructType(valueArgument).getRawClass();
46
47                 String valueOfMethod = "valueOf";
48                 // 转换入参的类型
49                 Method valueOf = ReflectionUtils.findMethod(valueRaw, valueOfMethod, String.class);
50                 if (valueOf != null) {
51                     ReflectionUtils.makeAccessible(valueOf);
52                 }
53                 // 将String类型的值转为枚举值对应的类型
54                 Function<String, V> castValue =
55                     // 获取不到转换方法时直接返回null
56                     source -> {
57                         if (valueRaw.isInstance(source)) {
58                             // String类型直接强转
59                             return valueRaw.cast(source);
```



```
63         : (V) ReflectionUtils.invokeMethod(valueOf, valueRaw, source
64         );
65         return source -> BasicEnum.fromValue(castValue.apply(source), targetType);
66     }
67 }
68
69     return source -> null;
70 }
71
72 }
```

定义一个 `WebmvcConfig` 配置类，将 `EnumConverterFactory` 注册到添加到 mvc配置中



java 复制代码

```
1 package com.example.config;
2
3 import com.example.config.converter.EnumConverterFactory;
4 import lombok.AllArgsConstructor;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.format.FormatterRegistry;
7 import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
8
9 /**
10  * 添加自定义枚举转换配置
11  *
12  * @author vains
13  */
14 @AllArgsConstructor
15 @Configuration(proxyBeanMethods = false)
16 public class WebmvcConfig implements WebMvcConfigurer {
17
18     private final EnumConverterFactory<?, ?> enumConverterFactory;
19
20     @Override
21     public void addFormatters(FormatterRegistry registry) {
22         registry.addConverterFactory(enumConverterFactory);
23     }
24 }
```

ParametersCancel

Name	Description
source <small>* required</small> <small>(query)</small>	1: web网站; 2: APP应用

2

ExecuteClear

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8080/enum/test02?source=2' \
  -H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8080/enum/test02?source=2
```

Server response

Code	Details
200	<div><div>Response body</div><div>2</div><div>Download</div></div> <div>Response headers</div> <div><pre>connection: keep-alive content-type: application/json date: Wed, 22 Nov 2023 02:08:22 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre></div>

@稀土掘金技术社区

[Gitee地址](#)、[Github地址](#)

标签：

后端


Spring Boot

Java

话题：

每天一个知识点

本文收录于以下专栏



SpringDoc

专栏目录

适用于SpringBoot3.x版本的在线文档工具——SpringDoc框架。

14 订阅 · 4 篇文章

订阅

上一篇

SpringDoc基础配置和集成OA...

下一篇

Spring Cloud Gateway集成S...

评论 0





暂无评论数据

目录

收起 ^

本期内容

默认内容

解决方案

单个处理方案

枚举入参

枚举作为参数类属性

全局统一处理方案

准备工作

实现枚举参数自定义

实现枚举属性的自定义

重启项目查看效果

SpringBoot接收枚举入参处理

定义一个EnumConverterFactory

定义一个WebmvcConfig配置类，将EnumConverterFactory注册到添加到mvc配置中

重启项目并打开在线文档进行测试

相关推荐



SpringSecurity整合OAuth2授权的简单示例

1.5k阅读 · 3点赞

干货文：Spring Security 5.7+ 最新配置方案

5.4k阅读 · 6点赞

SpringBoot 集成 OAuth2 系列一（最简单配置篇）

2.9k阅读 · 1点赞

旧的Spring Security OAuth已停止维护，全面拥抱新解决方案Spring SAS

54阅读 · 0点赞

精选内容

刚刚，百度和苹果宣布联名

官水三叶的刷题... · 411阅读 · 1点赞

网络爬虫框架Scrapy的入门使用

CodeDevMaster · 212阅读 · 1点赞

Python is Easy. Go is Simple. Simple != Easy

Swindler · 200阅读 · 0点赞

深入了解Java 8 新特性：Optional类的实践应用（二）

凡夫贩夫 · 201阅读 · 1点赞

【一分钟快学】掌握 Go 语言：快速解析 map 的垃圾回收机制

路口IT大叔_KUMA · 245阅读 · 1点赞

为你推荐

Spring Authorization Server入门 (二) Spring Boot整合Spring Authorization Server

叹雪飞花 9月前 6.8k 31 86 Java

Spring Authorization Server入门 (十二) 实现授权码模式使用前后端分离的登录页面

叹雪飞花 8月前 4.8k 24 65 后端 Spring ... Spring

Spring Authorization Server入门 (十) 添加短信验证码方式登录

叹雪飞花 9月前 3.4k 20 9 Spring Spring ...

Spring Authorization Server入门 (十六) Spring Cloud Gateway对接认证服务

叹雪飞花 7月前  2.6k  17  44 Spring ... Spring ... 安全

Spring Authorization Server入门 (十三) 实现联合身份认证，集成Github与Gitee的OAuth2.0

叹雪飞花 8月前  2.3k  13  51 Spring Spring ... 安全

Spring Authorization Server入门 (十一) 自定义grant_type(短信认证登录)获取token

叹雪飞花 9月前  2.5k  15  39 Spring Spring ... 安全


Spring Authorization Server入门 (七) 登录添加图形验证码

叹雪飞花 9月前  2.9k  18  4 Spring ...

SpringBoot3.x最简集成SpringDoc-OpenApi

叹雪飞花 4月前  2.3k  17  评论 后端 Spring ... Java

Spring Authorization Server入门 (九) Spring Boot引入Resource Server对接认证服务

叹雪飞花 9月前  1.9k  13  8 Spring Spring ...


Spring Authorization Server优化篇：添加Redis缓存支持和统一响应类

叹雪飞花 8月前  1.7k  6  12 Spring Spring ... 安全

Spring Authorization Server入门 (十五) 分离授权确认与设备码校验页面

叹雪飞花 7月前  1.6k  14  8 Spring ... Spring Vue.js

Spring Authorization Server入门 (十九) 基于Redis的Token、客户端信息和授权确认信...

叹雪飞花 4月前  1.3k  8  19 Spring ... 后端 Redis

Spring Authorization Server入门 (二十) 实现二维码扫码登录

叹雪飞花 2月前  937  16  6 Spring ... Spring Java

Spring Authorization Server入门 (十七) Vue项目使用授权码模式对接认证服务

叹雪飞花 6月前  762  9  12 Vue.js 安全 Spring ...