

20 | 匹配路由&异常捕获：如何用转化器实现匹配规则？

2023-06-07 Barry 来自北京

《Python实战·从0到1搭建直播视频平台》



你好，我是 Barry。

这节课我们继续学习路由。静态路由和动态路由的作用都是处理用户请求的 URL，根据 URL 匹配相应的路由规则，从而将请求分发给相应的视图函数处理。


在上节课，我们学习了静态路由，但动态路由部分我们并没有展开讲解。这节课，我们一起来学习一下动态路由以及正则匹配路由，这也是在 Flask 框架中非常核心的技术应用。

动态路由 shikey.com转载分享

我们先来认识一下动态路由。动态路由可以根据界面的不同，生成不一样的 URL。不同于静态路由，动态路由往往在后边会有一个变量作为动态参数，我们通常将其命名为。

动态路由的形式通常是 `@app.route('/index/')`，这个参数可以传入到我们的视图函数当中。

接下来，我们通过一个简单的案例来测试一下。

 复制代码

```
1 from flask import Flask
2 app = Flask(__name__)
3 @app.route('/index/<name>')
4 # http://127.0.0.1:5000/index/flask
5 def index(name):
6     return f'当前URL中的动态参数为: {name}'
7 # 当前请求的动态参数为: flask
```

在上面的例子当中，`index()` 函数可以返回 URL 中输入的动态参数 `name`。当我们在 URL 中输入 “**127.0.0.1:5000/index/flask**” 时，页面也同样会打印我们传入的动态参数 `flask`。


还有另一种方式就是在输入路由的时候指定传入参数。

 复制代码

```
1 from flask import Flask
2 app = Flask(__name__)
3 @app.route('/index/n=<name>')
4 # http://127.0.0.1:5000/index/n=flask
5 def index(name):
6     return f'当前URL中的动态参数为: {name}'
7 # 当前请求的动态参数为: flask
```

shikey.com转载分享

其实我们使用赋值的方式，同样可以传入这个参数，在 URL 中输入以下域名地址就能实现和上一段代码相同的效果。

 复制代码

```
1 http://127.0.0.1:5000/index/n=flask。
```

不过，动态路由在处理参数时通常只能根据一定规则进行匹配，无法直接实现参数类型、规则和格式的转换。如果路由参数类型或者路由参数格式需要转化，这时候我们就要用到一个叫转化器的东西。


转换器

在动态路由中使用转化器，可以方便地对路由参数进行类型、规则和格式的转换，从而提高路由的灵活性和可靠性。转换器分为两种类型，一种是系统自带的，另一种是自定义转化器。我们这就来看看这两种转化器要怎么使用。

系统自带转换器

在 Flask 应用中，动态路由参数默认是字符串类型。如果需要使用其他类型，可以使用系统自带转换器来实现，如 `int`、`float`、`bool` 等。这些转换器可以自动将路由参数转换为指定类型，方便之后视图函数的处理。

下面我会通过一个 `Int` 转化器的案例，带你了解一下系统自带转化器的应用方法。

 复制代码

```
1 from flask import Flask
2 app = Flask(__name__)
3 @app.route("/index/<int:id>",)
4 def index(id):
5     if id == 1:
6         return 'first'
7     elif id == 2:
8         return 'second'
9     elif id == 3:
10        return 'thrid'
11    else:
12        return 'hello world!'
13 if __name__ == '__main__':
14    app.run()
```

上一段代码中使用了 `path` 转化器。在 `@app.route("/index/<int:id>")` 这行代码中，使用了 `<int:id>` 来定义一个整数类型的参数 `id`，并将其绑定到 URL 路径中的变量 `id`。

这里的 `<int:id>` 就是一个转化器，它将传入的 URL 路径中的 `id` 参数从字符串类型转换成整数类型。当然，我们选取静态路由也可以实现这个效果，只是它需要定义四个路由，没有转化器的方法灵活，我们还是追求更优的方式。

自定义转化器

除了系统自带的转化器，我们还需要用到自定义转化器。这是因为动态路由匹配机制无法直接使用正则表达式。

如果我们需要实现更灵活的 URL 匹配方式，就需要用到自定义转换器，它可以更加灵活地处理 URL 和参数。

这样我们就可以通过自定义转换器来扩展 URL 规则。自定义转换器可以将 URL 中的字符串转换成指定类型的变量，并将其传递给视图函数。为了让你加深理解，我们通过一个案例来实践一下。

 复制代码

```
1 from werkzeug.routing import BaseConverter #导入转换器的基类，用于继承方法
2 # 自定义转换器类
3 class Flask_DIY_Converter(BaseConverter):
4     def to_python(self, value):
5         # 将字符串转换成指定类型的变量
6         return int(value) # 这里指定转换成int类型
7     def to_url(self, value):
8         # 将变量转换成字符串
9         return str(value)
```

shikey.com转载分享

对照代码可以看到，我们定义了一个名为 `Flask_DIY_Converter` 的自定义转换器，继承自 `BaseConverter` 类。

shikey.com转载分享

这里的 `BaseConverter` 是一个基础转换器，主要用于处理 URL 中的字符串。


在 `Flask_DIY_Converter` 类中，我们定义了两个方法，接下来我们来说一下它们分别的作用。

1.to_python(value) 方法是将 URL 中的字符串转换成 int 类型的变量。

2.to_url(value) 方法将变量转换成字符串，用于生成 URL。

我再给你梳理一下前面代码的完整执行过程：我们先通过 to_url() 函数，将动态参数 name 转换成字符串，从而生成 URL。然后，我们再把 URL 中的字符串传入函数 to_python 中。最后输出成 int 整数类型。这样就实现了系统自带转换器 int 的功能。

当然，我们在 Flask 中使用自定义转换器之前，还需要将其注册到应用中。具体的代码是后面这样。

 复制代码

```
1 from flask import Flask
2 app = Flask(__name__)
3 app.url_map.converters['my_cv'] = Flask_DIY_Converter
4 #将自定义的转换器my_cv,
5 #添加到Flask类下url_map属性（一个Map类的实例）包含的转换器字典属性中
6 # converters就是这个转换器字典
7 # 作用是将URL路径与视图函数进行匹配，并将请求分发到对应的视图函数中处理
```


我们在 Flask_DIY_Converter 里注册一个名为 my_cv 的转换器。这里的 app.url_map 和 [上节课](#)的 app.add_url_to 用法类似。

其中 url_map 是 Flask 中的 URL 映射表，它的作用同样是将 URL 路径与视图函数进行匹配，并将请求分发到对应的视图函数中处理。

其中 converters 是一个字典，它的键是自定义参数转换器的名称，值是实现参数转换逻辑的函数或类。

在前面的代码中，自定义的参数转换器是 my_cv，对应的实现逻辑是 Flask_DIY_Converter 类。my_cv 被用在 URL 映射规则中，这意味着在匹配 URL 路径时，Flask 会使用 my_cv 对应的参数转换逻辑来转换 URL 路径中的参数，从而实现参数的类型转化。


在 app 实例对象当中添加好转化器字典以后，就可以在 URL 规则中使用自定义转换器了。使用方法和系统自带转换器大体一样，只需要把函数换成我们自己定义的 my_cv 转化器即可。具体代码我给你写在了后面，供你参考。

 复制代码

```
1 @app.route('/user/<my_cv:user_id>')
2 def show_(user_id):
3     # user_id 是一个整数类型的变量
4     return 'User %d' % user_id
```

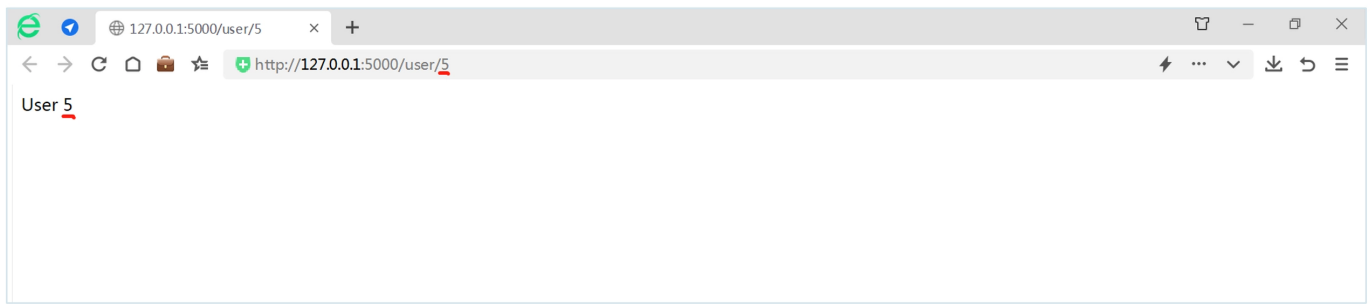
接下来，我们使用 my_cv 转换器来将 URL 中的字符串转换成整数类型的变量。这样一来，在视图函数中就可以直接使用整数类型的变量了。

我把具体的代码都整合在一起了，你可以对照文稿来理解。

 复制代码

```
1 from werkzeug.routing import BaseConverter
2 from flask import Flask
3 app = Flask(__name__)
4 class Flask_DIY_Converter(BaseConverter):
5     def to_python(self, value):
6         # 将字符串转换成指定类型的变量
7         return int(value)
8     def to_url(self, value):
9         # 将变量转换成字符串
10        return str(value)
11 app.url_map.converters['my_cv'] = Flask_DIY_Converter
12 @app.route('/user/<my_cv:user_id>')
13 # 实现和 @app.route('/user/<int:user_id>') 一样的功能
14 def show_(user_id):
15     # user_id 是一个整数类型的变量
16     return 'User %d' % user_id
17 if __name__ == '__main__':
18     app.run()
```

运行前面这段代码，页面就会呈现后面的截图。运行到这里，我们已经成功通过自定义的路由参数转化器，将字符串类型的用户 ID 参数转换成了整数类型。



极客时间

正则匹配路由

说到这里，你或许会有疑问：为了修改自定义转换器的功能，我们每次都需要在自定义类 Flask_DIY_Converter 中，修改其中的 to_url() 和 to_python() 函数的返回值太不方便了，有没有更简便的方式呢？

这个当然是有的，我们可以使用 regex。那什么是 regex？

regex 就是正则表达式。它是一种文本模式，用来匹配处理我们文本字符串。常规的正则表达式是由元字符和普通字符构成的。

元字符有特殊含义，比如 “.” 表示任意字符、“\d” 表示数字、“\s” 表示空白字符等。

普通字符是正则表达式中除了元字符以外的所有字符，例如字母、数字、标点符号等。

我们在 Flask 当中用正则表达式去匹配路由，能够很好地利用正则的匹配字符串的模式、数据校验字符串筛选等等。

shikey.com转载分享

跟住我的思路，我带你结合例子了解一下什么是正则表达式，还有如何用它来实现我们想要的自定义匹配。

shikey.com转载分享

复制代码

```
1 r'^1[3-9]\d{9}$'
```

下面我也写了对应的验证条件，你可以看一下，这个就是我们常用到的手机号格式验证。

- 1 ^1: 开头必须为1
- 2 [3-9]: 第二位是3到9之间的数字
- 3 \d: 匹配一个数字
- 4 {9}: 匹配9个数字
- 5 \$: 字符串的结尾

那我们如何在动态路由当中使用正则表达式呢？我们让每个 URL 都可以包含动态参数以及参数化的一个元素。通过这样的方式，可以更好地支持我们后面去应用 restful-API，以及 Web 应用程序的动态开发。

了解了正则表达式以后，我们使用 regex 来实践一下。

```

1 from werkzeug.routing import BaseConverter
2 from flask import Flask
3 app = Flask(__name__)
4 # 自定义转换器类
5 class RegexConverter(BaseConverter):
6     def __init__(self, url_map, regex):
7         # 定义这个类中的属性，有url_map和regex
8         # 重写父类定义方法
9         super(RegexConverter, self).__init__(url_map)
10        # RegexConverter类继承了BaseConverter类的所有属性和方法
11        self.regex = regex
12        # 一个名为regex的正则表达式赋值给当前对象的属性self.regex。
13        # 这样，就可以在类中使用self.regex来匹配字符串或执行其他与正则表达式相关的操作
14    def to_python(self, value):
15        return value # 由于使用了regex，这里无需再手动定义返回什么类型
16 app.url_map.converters['my_cv'] = RegexConverter
17 # 此处my_cv后括号内的匹配语句，被自动传给我们定义的转换器中的regex属性
18 # user_id值会与该语句匹配，匹配成功则传达给url映射的视图函数
19 @app.route("/user/<my_cv('1\d{10}')>:user_id")
20 def show_(user_id):
21     return 'User %d' % user_id
22 if __name__ == '__main__':
23     app.run()

```

在这段代码中我们先要定义一个类，同时需要定义这个类的属性。我们先使用 super 继承 RegexConverter 类中的 url_map 属性，再自定义一个 regex 属性，这样就可以直接设定规

则并且使用，无需在类的操作 `to_python` 中再去手动定义返回类型了。

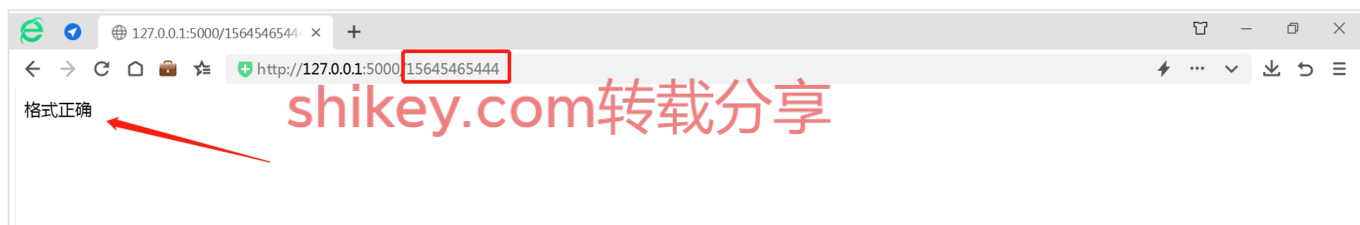
另外这么做还有一个好处：因为 `to_url` 操作一般用于将字符串转换为 URL 格式，而 `regex` 已经实现了字符串的替换和格式化，所以再定义一个 `to_url` 函数就略显多余，所以 `to_url` 操作也就不需要了。

接下来，我们通过模拟实现登录界面来练习一下，我们可以在路由装饰器中自定义规则，例如设置 URL 输入的格式必须为手机号格式。具体的代码是后面这样。

复制代码

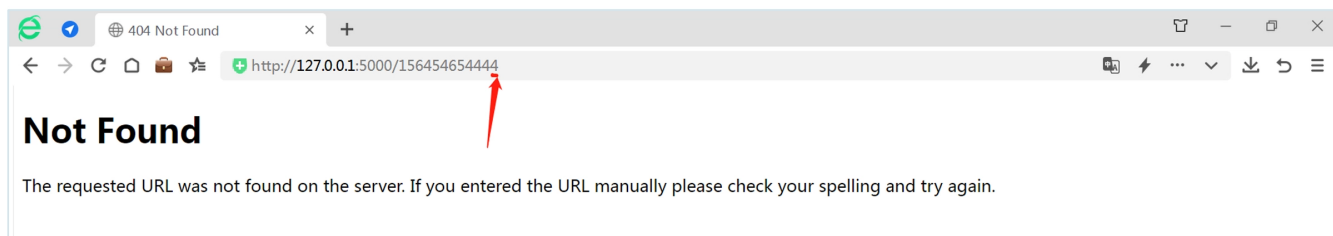
```
1 from flask import Flask, abort
2 from werkzeug.routing import BaseConverter
3 app = Flask(__name__)
4 class RegexConverter(BaseConverter):
5     def __init__(self, url_map, regex):
6         super(RegexConverter, self).__init__(url_map)
7         self.regex = regex
8     def to_python(self, value):
9         return value
10 app.url_map.converters['my_cv'] = RegexConverter
11 @app.route("/<my_cv(r'1[3456789]\d{9}')>:user_id")
12 def index(user_id):
13     return '格式正确'
14 if __name__ == '__main__':
15     app.run()
16 return
```

后面截图展示了输入格式正确情况下的运行结果。



极客时间

如果不按照规则，比如多输入一位数字 4，则会显示界面找不到。



但是像上面的这种情况，如果用户遇到错误之后，我们程序中没有任何异常处理的话，体感非常不好。所以我们要尽可能全面地去捕获异常错误，优化程序运行过程的同时，也要去提升用户体验，我们趁热打铁，继续学习如何进行异常捕获。

异常捕获

异常捕获通常指的是接口异常捕获。当程序出现异常的时候，刚好用户请求了服务器资源，而这个资源又不存在，那么用户就得不到有效的反馈，这种情况就需要我们用到接口异常捕获。我们先捕捉到错误，然后再进行优化处理，这样就能提高用户体验和代码流畅度。

异常捕获可以通过三种方式来处理，我们分别来看看。

abort 函数

我们先来认识一下 abort 函数，abort 函数是一个用于在视图函数中终止请求并返回 HTTP 错误响应的函数。

当你在编写 Flask 应用时，需要中止请求并返回一个 HTTP 错误响应，而不是继续执行视图函数，这时 abort 函数就能派上用场。

我们还是以正则路由匹配手机号格式的代码为例，结合代码来看看。

shickey.com转载分享

复制代码

```
1 from flask import Flask, abort
2 import re
3 from werkzeug.routing import BaseConverter
4 app = Flask(__name__)
5 class RegexConverter(BaseConverter):
6     def __init__(self, url_map, regex):
7         super(RegexConverter, self).__init__(url_map)
```

```

8     self.regex = regex
9     def to_python(self, value):
10         return value
11 app.url_map.converters['my_cv'] = RegexConverter
12 @app.route("/<my_cv(r'1[3456789]\d{9}')>:user_id")
13 def index(user_id):
14     if not re.match('1[3456789]\d{9}', user_id):
15         # abort的用法类似于python中的raise, 在网页中主动抛出错误
16         abort(404)
17         return None
18     else:
19         return '格式正确'
20 if __name__ == '__main__':
21     app.run()

```

对照代码可以看到，我们同样需要在路由中自定义规则。其中，re 是 Python 中的正则表达式模块。re 模块中包含了许多函数和方法，re.match 就是用来检测 user_id 和我们自定义的规则是否吻合。如果不符合，我们就可以直接使用 abort 函数，对应括号当中 404 会主动抛出这样一个错误；如果符合，则返回“格式正确”的字样。

errorhandler 装饰器


利用 abort 函数抛出的 404 错误，有些用户可能知道原因，但是还有改善的空间。遇到异常时，怎么做才可以给用户看得懂的提示呢？

此时，我们可以使用 errorhandler 这个注释装饰器来修改。

```

1 @app.errorhandler(404)
2 def page_not_found(error):
3     return "404错误, 您访问的页面不存在, 原因是您所输入的URL格式有误!"
4 # 比如说404, 当abort抛出这个404异常之后, 就会访问以下的这个代码。

```

 复制代码

这里我们定义了一个名叫 page_not_found 的函数，它会接收一个参数 error，表示发生的错误。函数返回一个包含错误消息的响应，例如提示用户访问的页面不存在。

同样地，要让这段代码起作用，需要将其注册到 Flask 应用程序中。我们可以使用 `@app.errorhandler` 装饰器把它注册为一个全局的错误处理程序，当 Flask 应用程序发生 404 错误时，就会自动调用该函数。

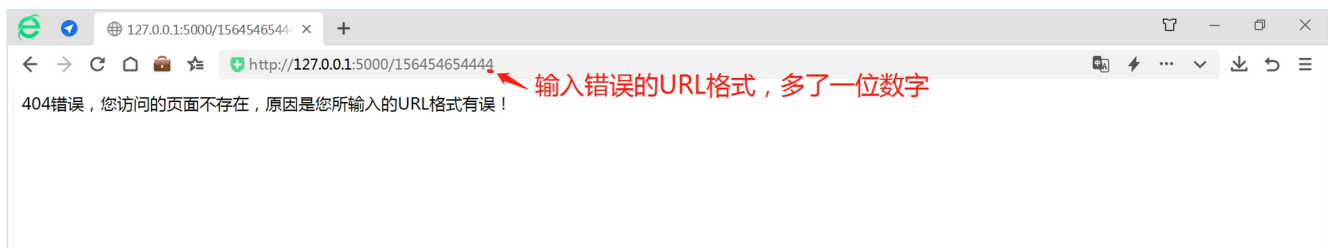
当然，这里接收的 `error=404`，也可以进行修改。比如 `error=405`，HTTP 请求方法不允许，相应地 `abort` 抛出的错误也可以修改。

梳理完了思路，我们再来看看完整代码。

 复制代码

```
1 from flask import Flask, abort
2 import re
3 from werkzeug.routing import BaseConverter
4 app = Flask(__name__)
5 class RegexConverter(BaseConverter):
6     def __init__(self, url_map, regex):
7         super(RegexConverter, self).__init__(url_map)
8         self.regex = regex
9     def to_python(self, value):
10         return value
11 app.url_map.converters['my_cv'] = RegexConverter
12 @app.errorhandler(404)
13 def page_not_found(error):
14     return "404错误，您访问的页面不存在，原因是您所输入的URL格式有误！"
15 # 比如说404，当abort抛出这个404异常之后，就会访问以下的这个代码。
16 @app.route("/<my_cv(r'1[3456789]\\d{9}')>:user_id", methods=['GET', 'POST'])
17 def index(user_id):
18     if not re.match('1[3456789]\\d{9}', user_id):
19         # abort的用法类似于python中的raise，在网页中主动抛出错误
20         abort(404)
21     return None
22 else:
23     return '格式正确'
24 if __name__ == '__main__':
25     app.run()
```

此时异常捕获后，就会给用户展示后面这样的界面。



极客时间

try-except 语法结构

前面两块应用都是对异常的处理以及对异常信息做友好的提示，方便用户知道系统出现问题的原因，提升产品体验。

但如果出现程序崩溃的情况我们又该如何处理呢？这时候我们就需要用到 try-except 语法结构。

使用 try-except 语法结构主要有两点好处。首先就是避免程序崩溃。异常处理是一种保护机制，可以避免程序因未处理的异常而崩溃。要知道如果程序出现异常错误并崩溃，用户体验将受到很大影响，使用 try-except 可以避免这种情况的发生。

另外，使用 try-except 还支持错误信息处理，我们在 except 块中可以获取并处理错误信息。

我们来看一个出现异常的例子。在 Practice 下新建 lb_03.py，注意，此时的 templates 中只有两个模板，不存在 index2.html。



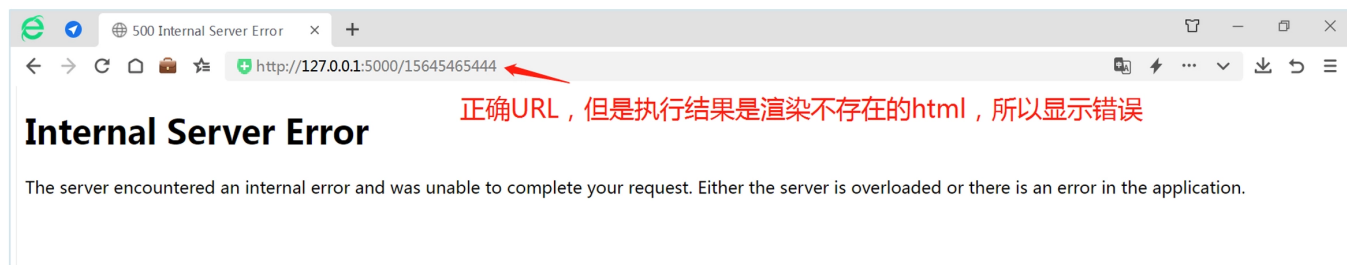
极客时间

lb_03.py 代码如下。

复制代码

```
1 from flask import Flask,render_template,abort
2 import re
3 from werkzeug.routing import BaseConverter
4 app = Flask(__name__)
5 class RegexConverter(BaseConverter):
6     def __init__(self,url_map,regex):
7         super(RegexConverter,self).__init__(url_map)
8         self.regex = regex
9     def to_python(self, value):
10         return value
11 app.url_map.converters['my_cv'] = RegexConverter
12 @app.route("/<my_cv(r'[0-9]{1,9}'):user_id>")
13 def index(user_id):
14     if re.match('[0-9]{1,9}', user_id):
15         return render_template('index2.html')
16 if __name__ == '__main__':
17     app.run()
```

这段代码中，我们先不进行异常处理。在 user_id 正确匹配格式时，我们渲染一个并不存在的模板 index2.html，来测试一下会发生什么异常。



极客时间

虽然我们的 url 输入了正确的格式，执行语句 render_template('index2.html')，但却因为我们渲染了一个不存在的 index2.html，就会导致程序崩溃。

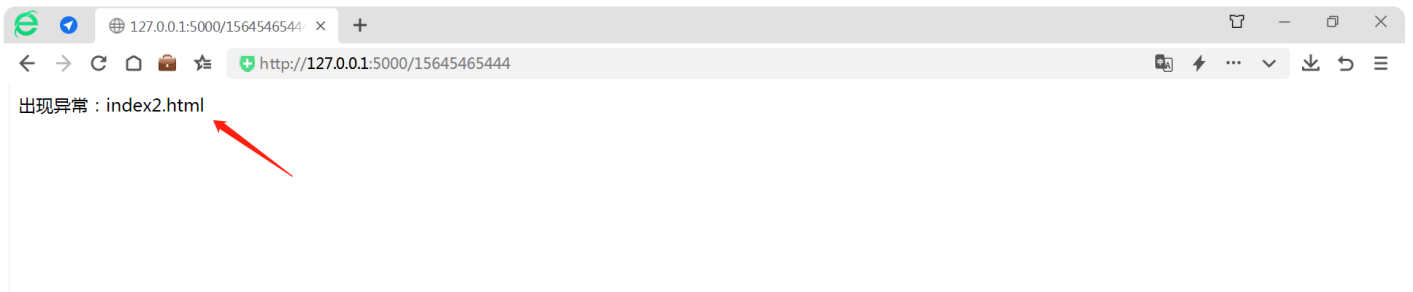
这里我们可以使用 try-except Exception as e，不让程序崩溃，并且处理这个异常，然后返回给用户。

复制代码

```
1 from flask import Flask,render_template,abort
2 import re
3 from werkzeug.routing import BaseConverter
4 app = Flask(__name__)
5 class RegexConverter(BaseConverter):
```

```
6 def __init__(self,url_map,regex):
7     super(RegexConverter,self).__init__(url_map)
8     self.regex = regex
9     def to_python(self, value):
10         return value
11 app.url_map.converters['my_cv'] = RegexConverter
12 @app.route("/<my_cv(r'1[3456789]\\d{9}')>:user_id>")
13 def index(user_id):
14     try:
15         if re.match('1[3456789]\\d{9}', user_id):
16             return render_template('index2.html')
17     except Exception as e:
18         return "出现异常: " + str(e)
19 if __name__ == '__main__':
20     app.run()
```

运行结果是后面这样。



你可以看一下效果图。我们不仅捕获了异常，还避免了程序发生崩溃。在此基础上又对程序进行了简单处理，提示用户 index2.html 这个模板出现异常，帮助用户精准定位问题。

在整个在线视频平台中 try-except 这种异常捕获的方法用得比较多，因为这种异常捕获方式允许我们对异常进行操作处理。使用 try-except 机制来处理可能出现的异常，可以保证程序的稳定性和可靠性。

总结

今天的课程告一段落，我们一起来回顾总结一下。

这节课的第一个学习重点是如何在路由中添加动态参数，这是为了根据我们自己的需要，灵活匹配生成不同 URL。

第二就是转换器（系统自带的或者我们自定义的转换器）和正则表达式的用法。学会了这些，就能定义我们自己想要的规则，比如定义输入的动态参数必须为手机号格式。

另外，在遇到异常发生时，为了避免用户体验不佳，我们可以使用异常捕获的方式，先处理捕获到的异常，再将处理后的结果返回给用户。

异常捕获有多种方法。我们既可以使用 `abort` 函数抛出异常，也可以用 `errorhandler` 装饰器把异常处理成我们想要的结果，再返回给用户。不过我更推荐的还是 `try-except` 方式，思路就是先尝试执行 `try` 代码块中的语句，如果出现异常，则执行 `except` 代码块中的语句。这样，就可以在出现异常的情况下继续执行程序，而不会导致程序崩溃。

异常的捕获和处理是在开发中必不可少的，我们一定要建立好异常处理的规则机制，这样功能开发才更完整。希望你课后结合配套代码加强练习，这样学习效果会更好。

思考题

正则表达式能解决的实际场景有很多，你可以举几个例子吗？

欢迎你在留言区和我交流讨论，如果这节课对你有启发，也推荐你把它发给身边更多朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (2)



Forest

2023-06-09 来自四川

正则的其他使用场景：数据格式校验、数据检索和提取

作者回复：回答的非常精准，感谢分享，一起加油。



1



peter

2023-06-07 来自北京

Q1: “VSCode中编辑vue文件，而vue文件是node解析的，这个关系应该是在VSCode中指定的吧。请问，是在VSCode的什么地方指定的？”，我在第17课中问过这个问题，不过老师误解我的意思了。我知道VSCode不是node.js。我的问题是：那么，vue怎么就自动找到node解析了？谁告诉vue去找node解析？就是说vue和node之间存在一种关系，这种关系是在哪里指定的？（比如，一种可能是在VSCode中配置的）

Q2: try .. except出异常后会自动释放资源吗？

Q3: 正则表达式字符串最前面的r表示什么？

作者回复: 1、Vue是通过webpack来打包编译的，而webpack又是通过配置文件来进行配置的，其中就包括了使用哪种loader来解析.vue文件，而vue-loader就是其中之一。而vue-loader又依赖于node.js来进行解析和编译，所以在使用Vue开发时需要先安装node.js，并配置好环境变量，这样才能让Vue正常工作。在VSCode中并没有直接指定这种关系的地方，而是通过配置webpack来实现的。

2、在 Flask 中使用 try-except 语句捕获异常后，Flask 会自动释放资源。这是因为 Flask 框架使用 Werkzeug WSGI 服务器，该服务器会自动处理资源释放。

3、r表示“raw”，即原始字符串，它告诉Python解释器不要对字符串中的反斜杠进行转义。这是它的含义和用途。

继续加油！！



shikey.com转载分享

shikey.com转载分享