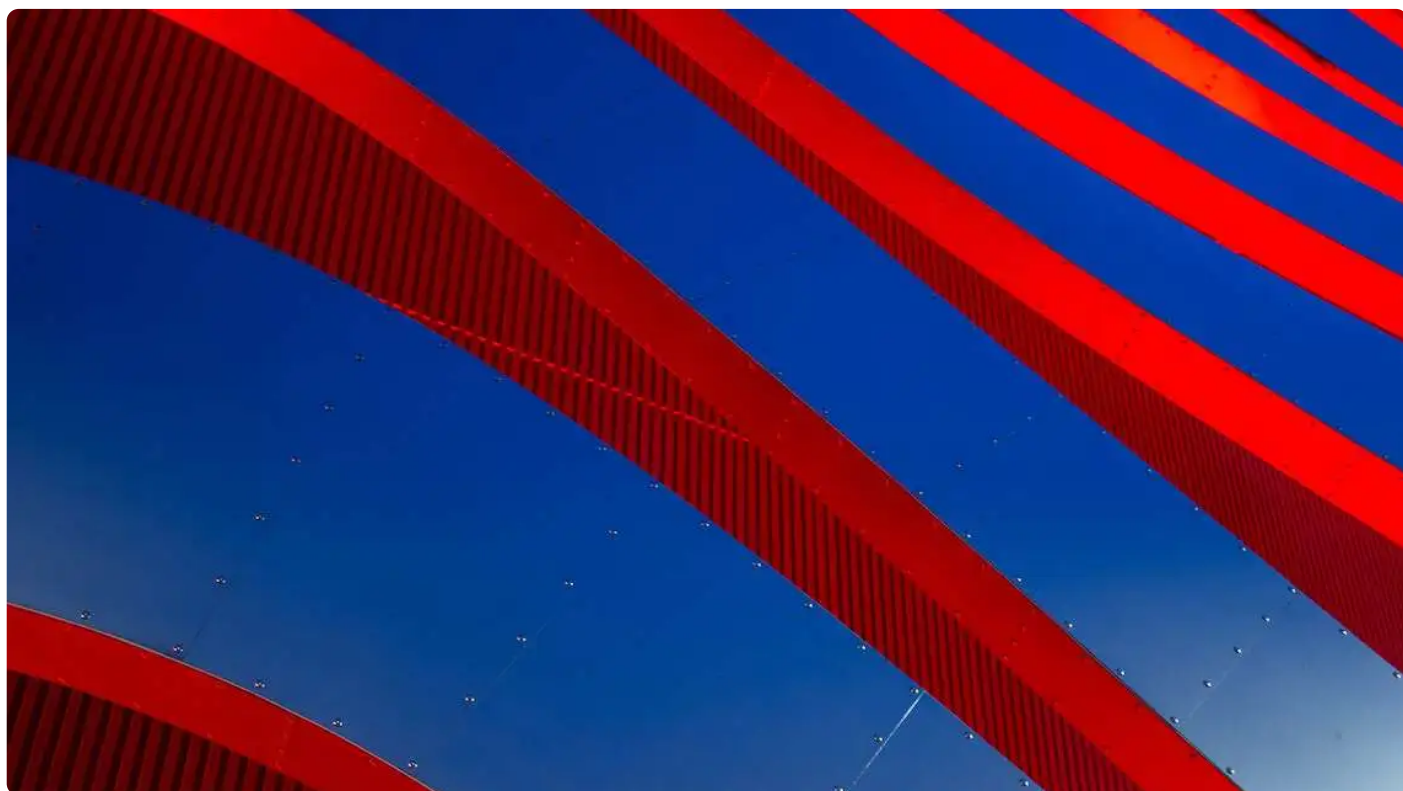


25 | Blueprint与Redprint: 如何让程序进行模块化处理?

2023-06-19 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。

通过前面课程的数据库实战，相信你已经能熟练应用数据库了。接下来，我们就来学习功能接口模块化开发。

为什么需要模块化开发呢？随着 Flask 项目的程序越来越复杂，我们在项目开发和迭代管理上都会成倍消耗精力。为了提升效率，就需要对项目里的请求方法进行封装管理，并且把项目划分成多个单独的功能模块，让每个模块负责不同的处理功能，再通过路由分配把各模块连接成一个完整的 Flask 项目。


那么在 Flask 框架中，我们如何实现模块化呢？这就要用到今天要学的内容——蓝图和红图了。

什么是蓝图？

我们先通过一个案例，来了解一下蓝图能为我们解决什么问题。

案例解析

在我们的视频项目中，包含了首页、分区列表、视频详情等模块，我们先看看代码实现。

 复制代码

```
1 源程序app.py文件：
2 from flask import Flask
3
4 app=Flask(__name__)
5
6 @app.route('/')
7 def VideoIndex():
8     return 'VideoIndex'
9
10 @app.route('/VideoList')
11 def VideoList():
12     return 'VideoList'
13
14 @app.route('/VideoDetail')
15 def VideoDetail():
16     return 'VideoDetail'
17
18 if __name__=='__main__':
19     app.run()
```

按照常规的编写逻辑，这时候我们需要在 app.py 文件中再定义添加、修改、发布的实现方法，代码如下。

shikey.com转载分享

 复制代码

```
1 源程序app.py文件：
2 from flask import Flask
3
4 app=Flask(__name__)
5
6 @app.route('/')
7 def VideoIndex():
8     return 'VideoIndex'
9
10 @app.route('/VideoList')
11 def VideoList():
```

shikey.com转载分享

```

12     return 'VideoList'
13
14 @app.route('/VideoDetail')
15 def VideoDetail():
16     @app.route('/')
17     def admin_home():
18         return 'admin_home'
19
20 @app.route('/add')
21 def new():
22     return 'add'
23
24 @app.route('/edit')
25 def edit():
26     return 'edit'
27
28 @app.route('/publish')
29 def publish():
30     return 'publish'
31
32 if __name__ == '__main__':
33     app.run()

```

像上面这种情况，我们需要在文件内写大量的路由，内容繁杂不说，整个功能模块管理起来也极其不友好。

这时就要引入**模块化的思维**，引入 Flask 内置模块化处理的类 Blueprint，也就是蓝图。

我们来看看优化之后的代码实现是什么样，后面是 app.py 文件的具体代码。

shikey.com转载分享

 复制代码

```

1 源程序app.py文件：
2 from flask import Flask
3
4 app=Flask(__name__)
5
6 @app.route('/')
7 def VideoIndex():
8     return 'VideoIndex'
9
10 @app.route('/VideoList')
11 def VideoList():
12     return 'VideoList'

```

shikey.com转载分享

```

13
14 @app.route('/VideoDetail')
15 def VideoDetail():
16     return 'VideoDetail'
17
18 if __name__ == '__main__':
19     app.run()
20
21 // admin.py文件, 进行模块化分别管理
22 @app.route('/')
23 def admin_home():
24     return 'admin_home'
25
26 @app.route('/add')
27 def new():
28     return 'add'
29
30 @app.route('/edit')
31 def edit():
32     return 'edit'
33
34 @app.route('/publish')
35 def publish():
36     return 'publish'
37

```

像上面这样，我们可以用蓝图把每个项目的每个模块当作一个独立的 app，这里的 app 是一个 Flask 应用对象，用于管理应用程序的各个方面，如路由、模板、中间件等。然后把功能模块拆分开。这样不管在编写过程里，还是后期维护时模块划分都更加清晰，也更有利于项目模块的管理。

到了这里，你应该对蓝图的思想有了初步的概念：**通过蓝图，开发者能够把项目中的应用拆分成不同组件，并完成对各个应用的控制和实现。**

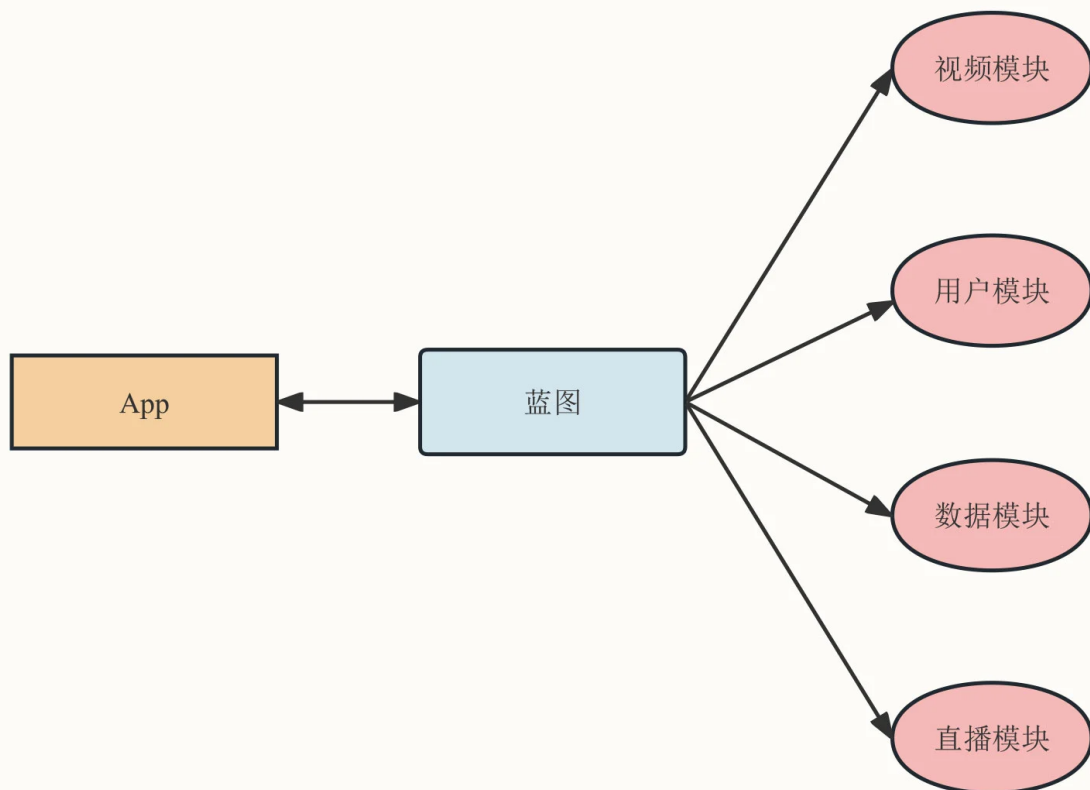
蓝图概念 shikey.com转载分享

了解了蓝图可以解决的问题之后，现在我们一起揭开蓝图的面纱，看看它的定义和属性。

我们可以把蓝图理解成一个操作方法的集合容器，各个模块的路由都会绑定在该模块的蓝图上。 Flask 可以通过蓝图来组织 URL 和处理请求，这就是蓝图的概念。

例如，视频模块的路由只放在视频模块的蓝图上，用户模块的路由放在用户模块的蓝图上。

你可以结合后面的图解加深印象。



极客时间

了解了概念后，我们还必须全面掌握蓝图的属性，方便以后去编写和设计蓝图。

首先，我们要知道，一个应用中不是只能定义一个蓝图。你可以根据你的模块需求定义多个蓝图；还可以将一个 Blueprint 注册到任何一个未使用的 URL 下，例如 “/”、或者子域名。

其次，在一个应用中，你的每一个模块都可以注册多次。这样的设计模式可以根据需求做多次拆分，让应用管理更细致。


最后，蓝图在实现的过程中，可以单独具有自己的模板、静态文件或者其它的通用操作方法，它并不是必须要实现应用的视图和函数。当然这时你要记住，在一个应用初始化时，就应该要注册需要使用的 Blueprint。

此外，我们还需要注意，单独的 Blueprint 并不代表一个完整的 app，换句话说，蓝图不可以独立运行，不可以独立应用，必须注册在某个应用之下。

初识蓝图

理解了蓝图的概念和属性，想要进一步认识它，还是要结合例子。接下来我用伪代码写一个单文件，带你看一下蓝图究竟长什么样子，要怎么用。蓝图的使用和 Flask 对象差不多，最大的区别是蓝图对象没有办法独立运行，必须将它注册到一个应用对象上才能生效。

我们先来初始化一个蓝图，导入 Blueprint。其中的 video 就是我们指定的模块名称，__name__ 代表模块的地址。

 复制代码

```
1 from flask import Blueprint //导入蓝图
2 video=Blueprint('video',__name__)
```

然后我们可以像往常写项目一样，定义各个功能模块的路由。有了这个蓝图对象，我们就可以像 app 加路由一样把它 “.route” 一下。这时我们调用了蓝图的对象，让它只注册路由指定静态文件夹，然后注册模块过滤器。后续我们可以在里面写上对应的视图函数，让它处理相应的业务逻辑。

 复制代码

```
1 //注册与业务逻辑
2 @admin.route('/')
3 def video_index():
4     return 'video_index'
5
6 //注册蓝图，并且指定前缀
7 app.register_blueprint(video,url_prefix='/admin')
```

最后，别忘了在我们的 app 里面注册蓝图，让你定义的蓝图的路径，也就是接口能够挂载到我们的 app 下。注册的时候也可以指定访问路由的前缀。

重要规则

看完伪代码你是不是想直接动手改造代码了？别着急，想要更好运用蓝图，还得多多了解蓝图的“游戏规则”，即实现蓝图的规则、设计方式、命名方式。掌握了这些前置规则，后面使用蓝图的时候才能少踩坑。

运行机制

蓝图运行机制分为以下四点。

第一，蓝图保存了一组可以在应用对象上执行的操作，注册路由就是其中的一种。它的意思是，我们注册蓝图之后，就可以根据蓝图来写相应的接口了。接下来通过蓝图中定义的 URL，就可以调用应用下模块的功能接口，直接访问该模块的视图函数并处理相应的请求。

第二，在应用对象上调用 route 装饰器完成注册后，这个操作将修改对象的 url_map，也就是我们存储的路由。url_map 就是路由的映射关系表，有了它才能根据访问请求里的访问接口找到对应的视图函数。

第三，蓝图对象本来没有路由表，当我们在蓝图对象上调用 route 装饰器注册路由时，它只是在内部的延迟操作记录列表 deferred_functions 中添加了一个项，在蓝图的底层代码中会有一些相应的方法。然后，**执行注册路由时 deferred_functions 会存储在表中对应的位置**，以便后续处理相应业务逻辑时，deferred_functions 能找到对应的视图函数。


第四：执行注册蓝图的注册方法，就是从**这个蓝图对象的 deferred_functions 列表中取出每一项，把它作为参数，然后执行该匿名函数（也就是这个注册方法）**。执行函数的过程实际上就是调用蓝图这个方法实现了对 add url rule 的加入。然后我们再修改应用对象，这样就把最后的路由表修改成了总的路由表。最后根据这种映射关系来即可找到对应的路由，让项目能够跑起来。

蓝图的 URL 前缀

我们在应用对象上注册一个蓝图时，还要着重关注一下蓝图的前缀，这样我们在注册和命名的过程中才能更好地封装路由。

具体做法就是在应用最终的路由表 `url_map` 中，指定一个 `url_prefix` 关键字参数（这个参数默认是 `/`）。这个前缀将会自动添加到之后注册的 URL 上，这样即使多个蓝图定义了相同 URL，也不会发生冲突。

我们这就结合代码示例看一下。

 复制代码

```
1 // url_for
2
3 url_for('admin.index') # /admin/
```

通过这种方式来获取我们的接口，然后访问蓝图的视图函数，你就能找到对应的一个接口，可以看到你的路由是什么。当然也可以通过 `url_for` 倒推查找视图函数，直接找到你对应的 URL。


说到 URL，蓝图中必不可少的还有静态路由，接下来我们一起看看如何注册静态路由。

注册静态路由

和应用对象不同，蓝图在对象创建时不会默认注册静态目录的路由，而是需要我们在创建时指定 `static_admin` 目录，将其设置为静态目录。

我们看看下面的示例，这段代码将蓝图所在目录下的 `static_admin` 目录设置成了静态目录。


shikey.com转载分享

 复制代码

```
1
2
3 admin = Blueprint("admin", __name__, static_folder='static_admin')
4 app.register_blueprint(admin, url_prefix='/admin/')
5
```

有了前面的设置，现在我们就可以使用 `/admin/static_admin/` 来访问 `static_admin` 目录下的静态文件了。定制静态目录 URL 规则就是，在创建蓝图对象时使用 `static_url_path` 改变静态目录的路由。

就像后面的例子这样，把 `static_admin` 文件夹的路由设置为 `/lib` 即可指定 `admin` 的静态文件。

 复制代码

```
1
2 admin = Blueprint("admin",__name__,static_folder='static_admin',static_url_path='
3 app.register_blueprint(admin,url_prefix='/admin')
```


静态文件 `static_folder` 可以指定蓝图静态文件。对应到视图文件里就是：如果你注册时设置了 URL 前缀，之后访问这个蓝图下面的所有接口时，都需要在前面单独再加一个前缀，这样所有的接口都会有统一的前缀。

在静态文件中还可以看到指定的静态路由，如果没手动添加过，它可能就是 “`static_folder`” 前面的 `static`。如果你写了前缀，就可能会改变静态路由的接口写法，通过这种方式来修改。

项目操练环节

接下来我们来尝试实操一下怎么搭建蓝图，帮助你更全面地掌握蓝图。话不多说，我们按步骤分解一下。

第一步，在项目的 `app` 文件之下创建一个 `index package`。`_init_.py` 文件之下的代码如下所示。

 复制代码

```
1 app的_init_.py文件
2 from flask import Blueprint //导入蓝图
3
4 index_blu = Blueprint('index', __name__) //实例化Blueprint对象 同时指定模块名index, 同
```


第二步，还是在 `index package` 下创建一个视图文件 `views.py`。为了满足文件查找需求，需要在我们的 `init` 文件中导入视图文件，代码如下。

 复制代码

```
1 app/index/__init__.py文件
2
3
4 from flask import Blueprint //导入蓝图
5 from . import views // 导入视图文件
6
7 index_blu = Blueprint('index', __name__) //实例化Blueprint对象 同时指定模块名index, 同
```


第三步就是定义视图函数。由于视图函数对应蓝图模块，所以这里面肯定需要蓝图对象。你需要把你的蓝图对象 `index_blu` 导入进来，视图函数会通过装饰器路由 `route` 去找到下一个路由的地址。

我们直接指定一个 `'/'`，`def` 是一个 `index`，然后我们 `return` 一个 `index`，这个 `index` 是已经创建好了的路由。下面是具体的代码实现。

 复制代码

```
1 views.py文件
2
3 from app.index import index_blu //导入Blueprint
4
5 @index_blu.route('/') //使用装饰器进行定义route
6 def index():
7     return 'index'
```

定义好了视图函数后，第四步就是在 `app` 的 `__init__.py` 下注册蓝图。

 复制代码

```
1 app/__init__.py
2
3 from app.index import index_blu //导入Blueprint
4 app.register_blueprint(index_blu)
```

第五步，在 `manager.py` 直接右键选择 `run manager`，这时我们就可以在控制台上看到启动的地址，如下图所示。

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
Note: NumExpr detected 12 cores but "NUMEXPR_MAX_THREADS" not set, so enforcing safe limit of 8.
NumExpr defaulting to 8 threads.
* Debugger is active!
* Debugger PIN: 131-701-694
```

极客时间

然后我们直接通过浏览器访问 “127.0.0.1:5000” 页面的 index，效果图如下。



极客时间

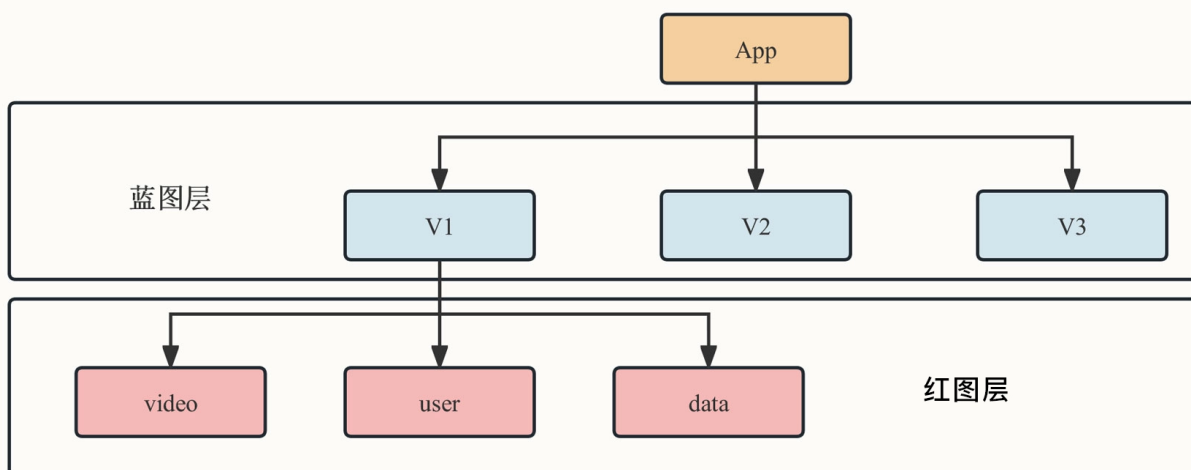
到这里就代表我们成功完成了蓝图的搭建，在之后的项目中你就可以应用起来了。当然，我们后边做功能实践时，它也是必要的模块。

什么是红图？

通过前面的学习我们知道蓝图是模块级别的拆分，但是它的设计不是用来拆分视图函数的。那如果我们要实现比模块级别更细分的视图函数的拆分，又该怎么办呢？这时就要用到红图了。

当然使用红图也是基于蓝图的。红图就是在蓝图的基础上又分了一层，负责实现比模块级别更具体的视图拆分，这就是红图的价值所在。

我给你画了一个简单的层级关系图，在红图层之下就是视图函数，这一点非常好理解。



我们这就来看看红图的实现，全面了解它的运行机制。

红图的实现

我们首先需要了解蓝图的 route 方法，这是红图对象创建的前置知识。我们直接来看源码。

复制代码


```
1
2 def route(self, rule, **options):
3     def decorator(f):
4         endpoint = options.pop("endpoint", f.__name__)
5         self.add_url_rule(rule, endpoint, f, **options)
6         return f
7     return decorator
8
```

shikey.com转载分享

我来解读一下这段代码。蓝图的 route 实现就是内部调用了 add_url_rule 方法。它的参数 rule 就是我们装饰器中定义的 URL，这也就是装饰器所作用的方法；endpoint 我们直接复用即可，**options 其实就是一系列关键字参数。

我们来梳理一下流程，装饰器接收了一组参数，并且调用了 add_url_rule 方法完成视图函数向蓝图的注册。我们要让红图的 route 代替蓝图的 route，就需要在我们的 Redprint 里把视

图函数注册到蓝图里去，因为要传参 f 嘛，实现形式我们通过模仿蓝图得到。过程你可以参考后面的代码。


 复制代码

```
1 class Redprint:
2     def __init__(self, name):
3         self.name = name
4         self.mound = [] # 把参数记录下来
5         pass
6
7     def route(self, endpoint, f, **options):
8         def decorator(f):
9             self.mound.append((f, rule, options)) # self是蓝图对象，以元组形式添加到
10            return f
11        return decorator
12
13    def register(self)
14        pass
```

到这里，我们对红图已经有一个较为全面的了解了，接下来我们进入实操练习环节。

代码实操环节


第一步，我们需要在 app/api/lib 目录下创建一个红图类的包文件，将其命名为 redprint。
init.py 文件内容放的就是红图的类，因为我们之后要调用它，代码如下。

 复制代码

```
1 redprint/__init__.py文件:
2
3 class Redprint:
4     def __init__(self, name):
5         self.name = name
6         self.mound = [] # 把参数记录下来
7         pass
8
9     def route(self, endpoint, f, **options):
10        def decorator(f):
11            self.mound.append((f, rule, options)) # self是蓝图对象，以元组形式添加到
12            return f
13        return decorator
14
```


```
15     def register(self)
16         pass
```

第二步，因为红图是依附于蓝图之下的，所以我们先模拟创建好蓝图 V1 和 V2，这一步你直接在 api 文件下新建即可。重点在创建蓝图这部分，具体代码与之前的创建相似。

 复制代码

```
1 v1/__init__.py
2
3 from flask import Blueprint
4 //创建V1蓝图
5 def creat_blueprint_v1():
6     bp_v1 = Blueprint('v1',__name__)
7     return bp_v1
```


有一点相当重要，我提醒你一下，一定要记得**在 app 下注册蓝图**，这样之后你才能访问到对应的 URL。

 复制代码

```
1 api/__init__.py
2
3 from flask import Flask
4 from app.index import index_blu //导入Blueprint
5
6 app = Flask(__name__)
7
8 //注册蓝图
9 //先导入
10 from app.v1 import creat_blueprint_v1
11 //注册蓝图,并且制定前缀"v1"
12 app.register_blueprint(creat_blueprint_v1(), url_prefix='/v1')
```

shikey.com转载分享

第三步，我们在 v1 蓝图下新建两个红图模块，将其命名为 user.py 和 admin.py。。这一步我们相当于对蓝图 V1 进行了分层。

 复制代码


```
1 user.py文件
```

```

2 // 因为红图类是我们自己创建的，所以我们要使用创建的红图
3 // 我们先来导入红图类包
4 from app.lib.redprint import Redprint
5
6 // 定义红图
7 api = Redprint('user')
8
9 //定义路由
10 @api.route('/')
11 def user_index():
12     return 'user_index'
13 -----
14
15 admin.py文件
16 from app.lib.redprint import Redprint
17
18 // 定义红图
19 api = Redprint('admin')
20
21 //定义路由
22 @api.route('/')
23 def admin_index():
24     return 'admin_index'
25

```

第四步， 将我们的红图对象注册在蓝图上。

 复制代码

```

1 v1/__init__.py
2
3 from flask import Blueprint
4 导入user、 admin 红图对象
5 from app.v1 import user, admin
6 //创建v1蓝图
7 def creat_blueprint_v1():
8     bp_v1 = Blueprint('v1', __name__)
9     //将红图对象使用register方法注册到蓝图上
10    user.api.register(bp_v1, user_prefix='/user') //使用url_prefix指定前缀
11    admin.api.register(bp_v1, user_prefix='/admin')
12    return bp_v1

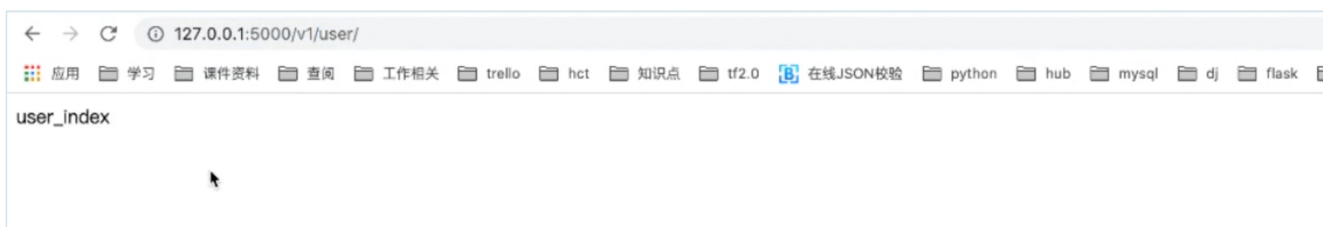
```

第五步， 直接启动命令在浏览器中查看效果。


```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 546-492-686
```

极客时间

这里有个容易踩坑的地方，那就是访问路径。我们已经看到启动的地址为 **127.0.0.1:5000**，但是直接访问你会发现浏览器是 “Not found”，这时候你应该想到，我们在注册蓝图时已经指定了访问前缀，所以应该访问的是 “**127.0.0.1:5000/v1/user**” 这个域名，访问后你会在浏览器中看到页面的效果。



极客时间

进行到这一步，说明操作就成功了，现在你就可以在项目里使用红图了。

总结

好，我们来做个总结回顾。


除了要明白红蓝图的属性以及它们解决了哪些问题，更重要的是如何把红蓝图熟练用起来。所以最后我重点带你梳理总结一下蓝图创建的过程，关键动作有三步。

首先，创建蓝图对象。

```
1 from flask import Blueprint
2 # 创建蓝图对象,设置访问前缀,所有的访问该蓝图的请求都需要加上/admin
3 admin_blu = Blueprint("admin", __name__, url_prefix="/admin")
4 from . import views
```

复制代码

其次，注册定义视图文件。

 复制代码

```
1 from app.index import index_blu //导入Blueprint
2
3 @index_blu.route('/') //使用装饰器进行定义route
4 def index():
5     return 'index'
```

最后，在程序实例中注册该蓝图。

 复制代码

```
1 # 注册admin蓝图对象
2 from api.modules.admin import admin_blu
3 app.register_blueprint(admin_blu)
```

当然，只记住这关键的内容还不够，我建议你自己尝试按课程讲解的内容，进行全链路的实践，这样可以掌握得更扎实。

红图可以实现比模块级别更细分的视图函数的拆分。整体实现的逻辑也是模仿了蓝图 route 设计原理，进一步对模块层级进行分层，把业务对象的函数拆分开，让项目模块化的管理更精细。你掌握蓝图之后，红图应用起来就没什么难度了，重点同样是实现过程。

下节课我们继续学习 “Restful API 与 Flask-Restful”，敬请期待。

shikey.com转载分享

思考题

如果在 user.py 红图模块下再创建一个 route 命名为 message，你觉得这时浏览器访问的地址应该是怎样的呢？**如果我们就以 “127.0.0.1:5000” 为案例，那这个路由地址最后是什么样子呢？**

欢迎你在留言区和我交流互动。如果这节课对你有启发，也欢迎你转发给朋友、同事，说不定就能帮他解决疑问。

精选留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

shikey.com转载分享

shikey.com转载分享