**Flask\_request上下文详解：**

**请求上下文：**

请求上下文也是存放到一个`LocalStack`的栈中。

和请求相关的操作就必须用到请求上下文，比如使用`url\_for`反转视图函数。

**注意1：**

在视图函数中，不用担心请求上下文的问题。

因为视图函数要执行，那么肯定是通过访问url的方式执行的，

那么这种情况下，Flask底层就已经自动的帮我们把应用上下文和请求上下文都推入到了相应的栈中。

**注意2：**

如果想要在视图函数外面执行相关的操作，

比如反转url，那么就必须要手动推入请求上下文：

**底层代码执行说明：**

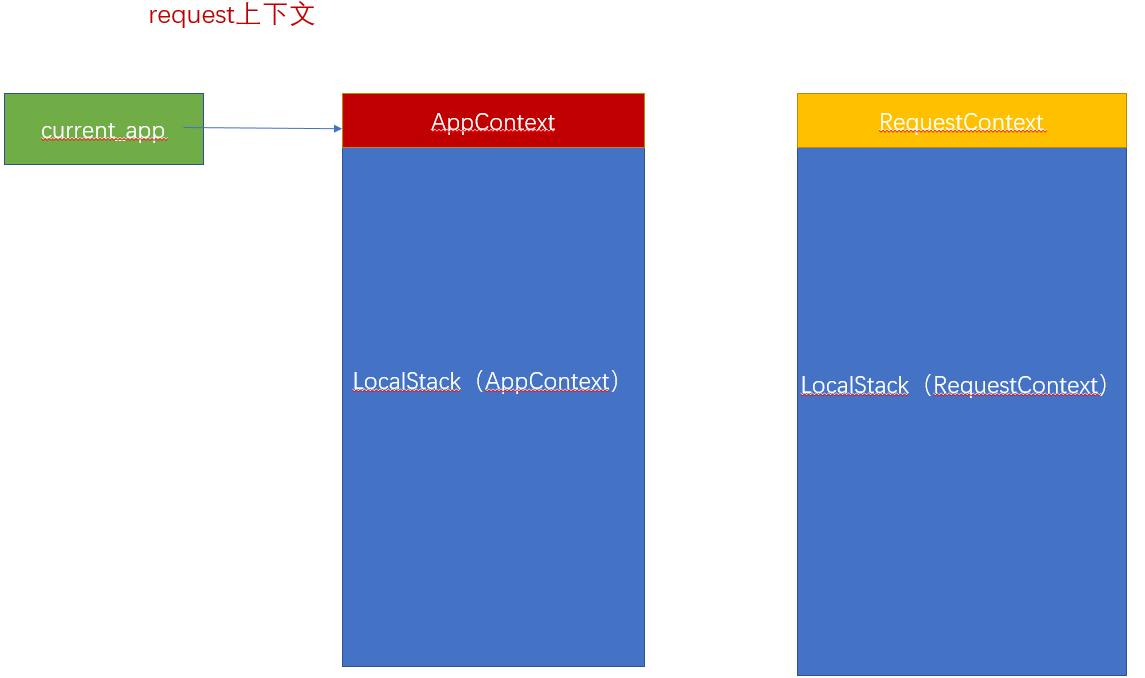
\* 推入请求上下文到栈中，会首先判断有没有应用上下文，

\* 如果没有那么就会先推入应用上下文到栈中，

\* 然后再推入请求上下文到栈中

|  |
| --- |
| **from** flask **import** Flask,url\_for  app = Flask(\_\_name\_\_)  *#request上下文*  @app.route(**'/hi'**)  **def** hi():  print(url\_for(**'my\_list'**)) *#获取构建得到的url*  **return 'Hello World!'**  @app.route(**'/list/'**)  **def** my\_list():  **return '返回列表'**  *# print(url\_for('my\_list')) #获取构建得到的url*  *# with app.app\_context():*  *# print(url\_for('my\_list'))*  *#查看报错源码*  **with** app.test\_request\_context():  *#手动推入一个请求上下文到请求上下文栈中*  *#如果当前app应用上下文栈中没有app应用上下文*  *#那么会首先推入一个app应用上下文到栈中*  print(url\_for(**'my\_list'**))  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  app.run(debug=**True**) |

**图解：**



**总结：**

为什么上下文需要放在栈中？

**1.**应用上下文：Flask底层是基于werkzeug，werkzeug是可以包含多个app的，所以这时候用一个栈来保存。

如果你在使用app1，那么app1应该是要在栈的顶部，如果用完了app1，那么app1应该从栈中删除。方便其他代码使用下面的app。

**2.**如果在写测试代码，或者离线脚本的时候，我们有时候可能需要创建多个请求上下文，这时候就需要存放到一个栈中了。使用哪个请求上下文的时候，就把对应的请求上下文放到栈的顶部，用完了就要把这个请求上下文从栈中移除掉。