**10.1异常处理·业务异常与全局异常处理**

异常处理也是非常重要的，我们将异常分为以下两种：Flask内置异常和自定义异常。当然，不管是什么异常，我们都可以使用Flask的@errorhandler进行全局处理。这里重点讲一下自定义异常。

# 关于业务错误码

在编写业务代码的时候，时常需要定义不同的状态码来表示不同的执行结果。而一般主流的做法主要有以下两种：一、业务方法中定义统一的返回体，外层统一的返回体做全局处理；二、业务方法中抛出自定义异常， 外层做统一的异常处理。这里不对这两者进行对比，感兴趣的同学，可以去了解一下。这里主要讲的是第二种方式，也是本快速开发框架使用的方式。

# 错误码定义与规范

## 错误码枚举

仅一个错误码字段可能还不足以描述一个错误码，所以一般会单独定义枚举类来描述错误码。这里如下定义：

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 说明 |
| code | int | 错误码值 |
| msg | String | 错误码名称或描述 |

## 错误码规范

不同的业务，错误码就不一样，所以一般每做一个业务，需要提前申请或者定义一下错误码范围。在这里使用8位数字组成的错误码。如：80000001、99990401等。一般来说，建议一个业务类，对应一个错误码枚举类。不过如果需要判断的业务不多，可以一个模块中的多个业务类共用一个错误码枚举类。

## 返回结果样例

**成功**

http status: 200

{ "code": 0, "msg": "成功" }

**失败**

http status: 200

{ "code": 80009001, "msg": "用户不存在" }

# 创建目录mexception

mkdir mexception

# 新增mexception/\_\_init\_\_.py

主要有三个类：

* ExceptionConfig异常配置类
* ErrorEnum 错误码枚举基类
* GlobalErrorEnum 全局错误码，定义一些常用的错误码
* BizException 自定义异常，方便全局捕获
* AssetTool 异常断言工具类，用于抛出自定义异常的

from enum import Enum  
  
from werkzeug.exceptions import HTTPException  
  
from controllers import R  
  
  
class ExceptionConfig:  
 """  
 异常处理配置  
 """  
  
 def \_\_init\_\_(self, app=None):  
 self.app = app  
 if app:  
 self.init\_app(app)  
  
 def init\_app(self, app):  
 self.app = app  
  
 @app.errorhandler(HTTPException)  
 def error\_http(e):  
 """  
 HTTP异常处理(401,403,404,500等)  
 :param e: 异常  
 :return:  
 """  
 app.logger.error(e)  
 return R.fail(e.deｓｃｒｉｐｔion, code=e.code)  
  
 @app.errorhandler(BizException)  
 def error\_biz(e):  
 """  
 业务异常处理  
 :param e: 异常  
 :return:  
 """  
 app.logger.error(e)  
 return R.fail(e.msg, e.code)  
  
 @app.errorhandler(Exception)  
 def error(e):  
 """  
 其他异常处理  
 :param e: 异常  
 :return:  
 """  
 app.logger.error(e)  
 return R.fail(GlobalErrorEnum.GL99999999.msg, code=GlobalErrorEnum.GL99999999.code)  
  
  
class ErrorEnum(Enum):  
 """  
 错误码基类  
 """  
  
 def \_\_new\_\_(cls, code, msg):  
 obj = ｏｂｊｅｃｔ.\_\_new\_\_(cls)  
 obj.code = code  
 obj.msg = msg  
 return obj  
  
  
class GlobalErrorEnum(ErrorEnum):  
 """  
 全局错误码  
 """  
 GL99999999 = (99999999, "服务器异常")  
 GL99990100 = (99990100, "参数异常")  
 GL99990401 = (99990401, "未授权")  
 GL99990500 = (99990500, "未知异常")  
 GL99990403 = (99990403, "token过期或不存在")  
 GL99990404 = (99990404, "找不到指定资源")  
 GL99990406 = (99990406, "无访问权限")  
 GL99990001 = (99990001, "注解使用错误")  
 GL99990002 = (99990002, "微服务不在线,或者网络超时")  
 GL99990003 = (99990003, "没有数据")  
 GL99990004 = (99990004, "演示账号，无写权限")  
 GL99990005 = (99990005, "数据库插入异常")  
 GL99990006 = (99990006, "文件后辍不允许")  
 GL99990007 = (99990007, "文件上传异常")  
 GL99990008 = (99990008, "文件上传配置不存在")  
 GL99990009 = (99990009, "文件超过上传最大值")  
 GL99990010 = (99990010, "审核不通过原因不能为空")  
 GL99990011 = (99990011, "图片验证码生成异常")  
 GL99990012 = (99990012, "图片验证码错误或不存在")  
 GL99990013 = (99990013, "演示站无访问权限")  
  
  
class BizException(Exception):  
 """  
 业务异常  
 """  
  
 def \_\_init\_\_(self, error: ErrorEnum):  
 self.code = error.code  
 self.msg = error.msg  
  
  
class AssertTool:  
 """  
 断言工具  
 """  
  
 @staticmethod  
 def raise\_biz(error\_enum: ErrorEnum):  
 raise BizException(error\_enum)

# 修改mtoken/\_\_init.py

将abort(403)->AssetTool.raise\_biz(GlobalErrorEnum.GL99990403)

from functools import wraps  
  
from mauth.mtoken import TokenStrategyFactory  
from mexception import AssetTool, GlobalErrorEnum  
  
  
class HasPerm(ｏｂｊｅｃｔ):  
 """  
 权限装饰器类  
 """  
  
 def \_\_init\_\_(self, access=None, name=None):  
 self.access = access  
 self.name = name  
  
 def \_\_call\_\_(self, func):  
 @wraps(func)  
 def wrapped\_function(\*args, \*\*kwargs):  
 # 这里处理权限拦截的逻辑  
 if not TokenStrategyFactory.check\_token(self.access):  
 AssertTool.raise\_biz(GlobalErrorEnum.GL99990403)  
 else:  
 return func(\*args, \*\*kwargs)  
  
 return wrapped\_function

# 修改app.py

初始化日志配置

代码片段：

from mexception import ExceptionConfig  
  
# 配置异常处理  
ExceptionConfig(app)

# 运行Flask服务

默认为控制台输出

flask run

# 使用Postman接口测试工具访问

# 不携带token访问/user/get

