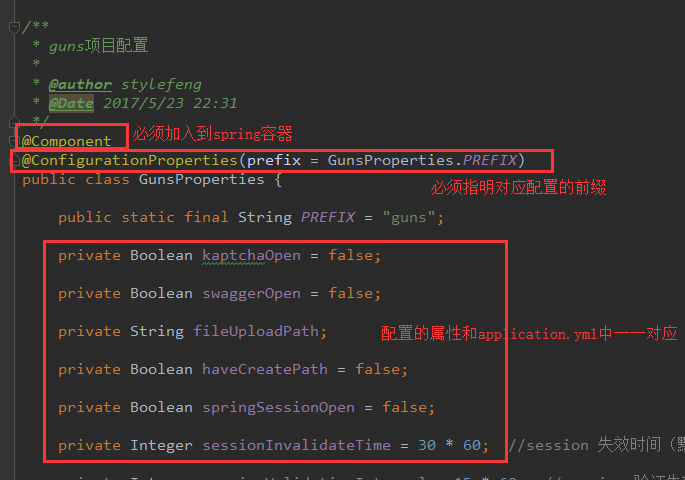
[4.1 分包](https://www.stylefeng.cn/doc/guns/#/core/?id=_41-%e5%88%86%e5%8c%85)

在日常开发中，业务模块的包结构划分一般划分为三个config、core、modular或者四个common、config、core、modular

其中common为模块内通用的注解、常量、枚举、异常和持久化的实体等，若common不单独划分一个包，则可以把common包放到core包下面

config包存放整个模块的配置类，因为项目基于spring boot开发，大部分的spring配置都换成了java bean方式的配置，所以单独分一个包来存放配置，config包中除了存放配置类，还有一些以Properties结尾的类，这些类的作用是启动应用的时候把application.yml中的配置映射到类的属性上，使用时需要注意以下几点 

modular存放按业务划分的业务代码，若本模块中包含多个模块业务，则在modular中建立多个业务包，在具体的业务包下再建立controller、dao、service、transfer、warpper这几个包，其中transfer为前后端传输数据所用的属性封装，warpper为对返回结果的包装器(下面会介绍到)，如果当前模块中只存在一类业务，那么没有必要在modular包下再建立多个业务模块，可直接在modular模块建立controller、dao、service、transfer、warpper

core包存放当前模块所运行的一些核心机制，例如全局的异常拦截器，日志AOP，权限的AOP，项目初始化后的监听器，工具类等，还可以存放一些对某些框架的扩展，例如对beetl模板的扩展配置和工具类，对flowable的扩展类，Shiro的一些拓展类等等

这样拆分的好处在于把业务，配置和运行机制清晰的拆分开，提高项目的可维护性，加快项目的开发效率!

[4.2 统一异常拦截](https://www.stylefeng.cn/doc/guns/#/core/?id=_42-%e7%bb%9f%e4%b8%80%e5%bc%82%e5%b8%b8%e6%8b%a6%e6%88%aa)

[4.2.1 介绍](https://www.stylefeng.cn/doc/guns/#/core/?id=_421-%e4%bb%8b%e7%bb%8d)

统一异常拦截指对程序抛出的异常利用@ControllerAdvice在统一的一个类中做catch处理，在Guns中，我们在GlobalExceptionHandler类中做统一异常拦截处理，GlobalExceptionHandler类中可以拦截所有控制器执行过程中抛出的异常，若需要拦截其他包下的异常可以参考SessionInterceptor这个类中AOP的写法，来拦截其他特定包的异常。统一异常拦截的写法注意一下几点 

[4.2.2 优点](https://www.stylefeng.cn/doc/guns/#/core/?id=_422-%e4%bc%98%e7%82%b9)

对异常进行统一处理，不需要再在业务代码中进行try catch操作，尽情写业务，有异常也会被自动拦截到，并且自动处理返回给前端提示

[4.2.3 关于性能](https://www.stylefeng.cn/doc/guns/#/core/?id=_423-%e5%85%b3%e4%ba%8e%e6%80%a7%e8%83%bd)

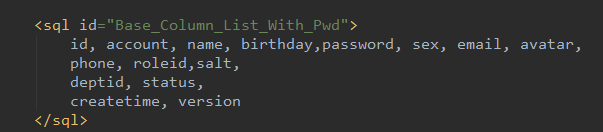
有人可能会认为利用异常拦截这种机制，把业务逻辑的错误都用业务异常抛出进入aop的执行器，对性能会有所影响，经过笔者的调研和测试，频繁的抛出异常和try catch不会有性能损耗，主要的性能损耗在catch方法内部，并且在catch内，记录日志比较占用大部分的时间

所以，如果是系统特别注重性能等问题，可以把业务异常分为两类，一类是较为频繁抛出的业务异常，一类是较少出现次数的业务异常，第一类异常可以再@ExceptionHandler中不做日志记录，只进行简单的返回操作，第二类可以着重做异常处理，并做结果返回

[4.3 结果包装器](https://www.stylefeng.cn/doc/guns/#/core/?id=_43-%e7%bb%93%e6%9e%9c%e5%8c%85%e8%a3%85%e5%99%a8)

我们在进行列表查询或详情查询的过程中，查到的结果中，有些值可能在数据库中存的是一些列数字(一般为状态值等)，但是我们要返回给前端，业务人员看的时候不希望直接返回给他们这些不直观的值(例如1，2，3，4)，我们更希望返回给前端中文名称(例如启用，冻结，已删除)，所以我们应该对这些数值做一下包装，把他们包装成文字描述

[4.3.1 如何使用](https://www.stylefeng.cn/doc/guns/#/core/?id=_431-%e5%a6%82%e4%bd%95%e4%bd%bf%e7%94%a8)

以查询用户列表的接口为例，不包装的情况下默认的查询结果为这些字段  其中性别，角色，部门，状态都是数值或者id类型，我们需要把他们包装成文字形式返回给前端

1.首先建立UserWarpper类继承BaseControllerWarpper类

/\*\*

\* 用户管理的包装类

\*

\* @author fengshuonan

\* @date 2017年2月13日 下午10:47:03

\*/

public class UserWarpper extends BaseControllerWarpper {

public UserWarpper(List<Map<String, Object>> list) {

super(list);

}

@Override

public void warpTheMap(Map<String, Object> map) {

map.put("sexName", ConstantFactory.me().getSexName((Integer) map.get("sex")));

map.put("roleName", ConstantFactory.me().getRoleName((String) map.get("roleid")));

map.put("deptName", ConstantFactory.me().getDeptName((Integer) map.get("deptid")));

map.put("statusName", ConstantFactory.me().getStatusName((Integer) map.get("status")));

}

}

通过查看BaseControllerWarpper类可了解到被包装的参数必须为Map或者List类型

/\*\*

\* 控制器查询结果的包装类基类

\*

\* @author fengshuonan

\* @date 2017年2月13日 下午10:49:36

\*/

public abstract class BaseControllerWarpper {

public Object obj = null;

public BaseControllerWarpper(Object obj) {

this.obj = obj;

}

@SuppressWarnings("unchecked")

public Object warp() {

if (this.obj instanceof List) {

List<Map<String, Object>> list = (List<Map<String, Object>>) this.obj;

for (Map<String, Object> map : list) {

warpTheMap(map);

}

return list;

} else if (this.obj instanceof Map) {

Map<String, Object> map = (Map<String, Object>) this.obj;

warpTheMap(map);

return map;

} else {

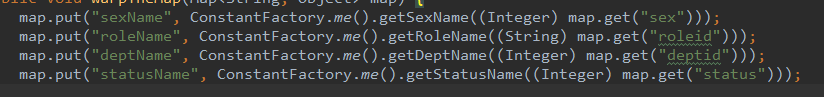
return this.obj;

}

}

protected abstract void warpTheMap(Map<String, Object> map);

}

我们继承BaseControllerWarpper类主要是为了实现warpTheMap()方法，也就是具体的包装过程，warpTheMap()方法的参数map就是被包装的原始数据的每个条目，我们可以在这每个条目中增加一些字段也就是被包装字段的中文名称，如下 

[4.3.2 ConstantFactory](https://www.stylefeng.cn/doc/guns/#/core/?id=_432-constantfactory)

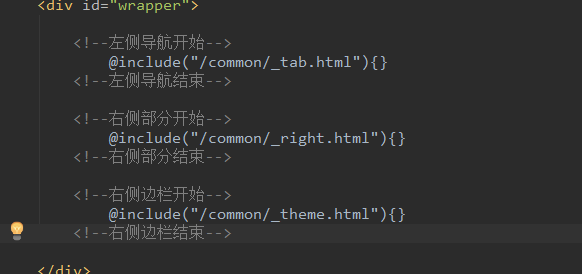
在包装过程中，我们经常会用到ConstantFactory这个类，这个类是连接数据库和包装类的桥梁，我们可以在ConstantFactory中封装一些编辑的查询方法，这些方法通常会被多个包装类多次调用，并且在调用这些方法的时候ConstantFactory.me()的形式静态调用，可以快速的包装一些状态和id，非常方便，在ConstantFactory中我们可以利用spring cache的@Cacheable注解来缓存一些数据，把这些频繁的查询缓存起来

[4.4 前端思想](https://www.stylefeng.cn/doc/guns/#/core/?id=_44-%e5%89%8d%e7%ab%af%e6%80%9d%e6%83%b3)

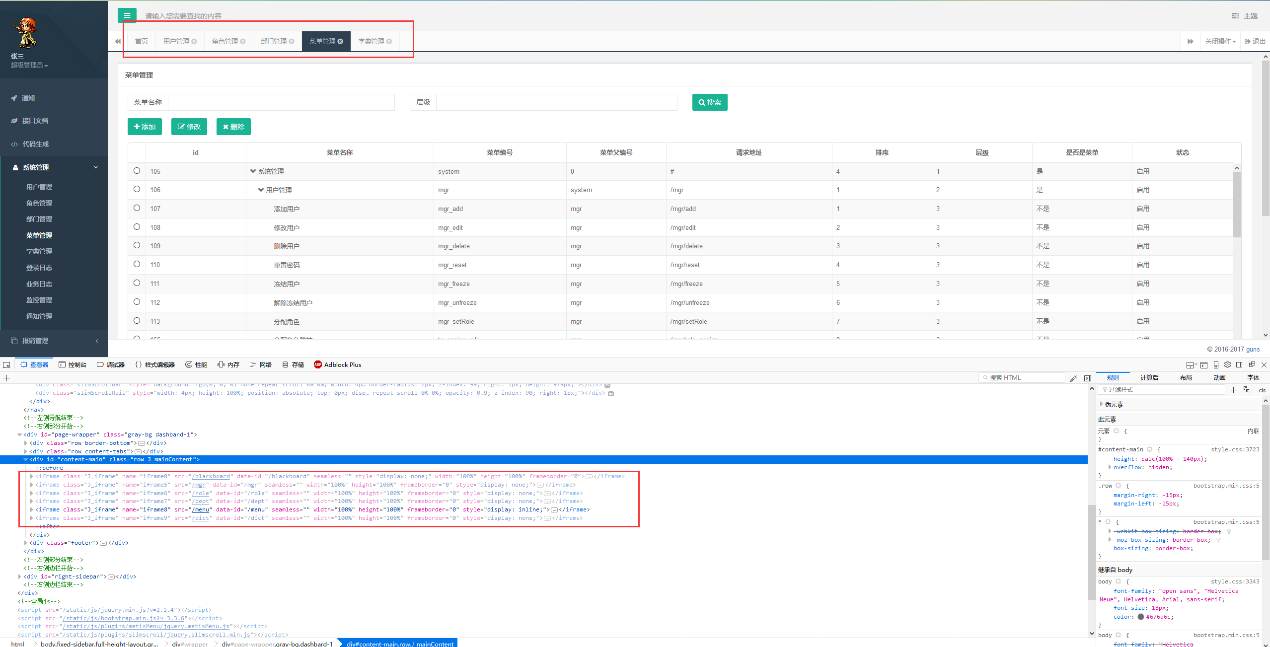
Guns前端采用了beetl模板引擎，beetl包含语法简洁，速度快，文档全，社区活跃等众多优点，所有的beetl语法都以@开头

[4.4.1 布局](https://www.stylefeng.cn/doc/guns/#/core/?id=_441-%e5%b8%83%e5%b1%80)

在用户登录页面后进入的是index.html页面，这个页面加载了整个后台管理系统的框架，我们可以看到index.html源代码中把整个页面分为了三部分，左侧菜单栏，右侧页面和右侧主题栏部分，其实就是利用beetl的@include把整个大的复杂的页面细化了，这样好维护



左侧菜单和右侧主题栏部分在用户登录后会一直不变，除非刷新浏览器页面，动态变化的是页面右侧这部分，我们打开6个标签页，并打开浏览器F12调试



新建和切换标签，页面的地址不会变化，变化的是页面右侧的iframe这部分

下面我们分析一下右侧页面的组成，我们打开菜单管理页面，查看他的代码

@layout("/common/\_container.html"){

<div class="row">

XXXX等html代码...

</div>

<script src="${ctxPath}/static/modular/system/menu/menu.js"></script>

@}

整个页面被@layout所包围，@layout是beetl的引用布局(具体用法文档可以查看beetl的官方文档)，Guns中内置了/common/\_container.html这样一个布局，可以把/common/\_container.html理解为一个html的抽象封装，我们每个页面都继承自这个模板，默认包含了一系列通用的js css引用等，这样写即简化了我们的开发和维护，又使我们的代码简洁有序，在/common/\_container.html中的${layoutContent}就代表我们每个页面不同的html

[4.4.2 手动新增标签页](https://www.stylefeng.cn/doc/guns/#/core/?id=_442-%e6%89%8b%e5%8a%a8%e6%96%b0%e5%a2%9e%e6%a0%87%e7%ad%be%e9%a1%b5)

新版Guns提供了手动新增标签页的方法，如下：

top.layui.index.openTab({

title: '百度',

url: 'https://www.baidu.com',

end: function() {

// table.reload('userTable');

}

});

// 参数说明如下

// title： 选项卡的标题

// url： 打开的页面地址

// end`： Tab关闭的回调事件

也可以直接使用<a ew-href="xxx.html">XXX</a>，注意是ew-href。