#### Shopping模块

Gateway路由控制: com.yoho.gateway.controller.order.shopping.ShoppingController

请求对象定义: com.yoho.service.model.order.request.ShoppingCartRequest

具体请求处理: com.yoho.yhorder.shopping.restapi. ShoppingCartController

Gateway提供向外的服务接口，gateway内部使用rpc调用方式访问具体的服务，具体服务的实现在各自所属模块。

购物车相关的服务请求如下表所示

|  |  |  |
| --- | --- | --- |
| Method | Call rest url |  |
| app.Shopping.add | order.addShopping | 添加购物车 |
| app.Shopping.cart | order.cartShopping | 查询购物车 |
| app.Shopping.increase | order.increaseShopping | 增加数量 |
| app.Shopping.decrease | order.decreaseShopping | 减少数量 |
| app.Shopping.swap | order.swapShopping | 交换 |
| app.Shopping.addfavorite | order.addfavorite |  |
| app.Shopping.remove | order.removeShopping | 删除购物车商品 |
| app.Shopping.submit | order.submitShopping | 提交下单 |
| app.Shopping.compute | order.computeShopping | 计算金额 |
| app.Shopping.selected | order.selectedShopping | 选中商品 |
| app.Shopping.payment | order. paymentShopping | 获取购物车支付信息接口 |

在yoho-service-modol内部也有同样的定义resturl OrderServices定义了所有order模块的服务请求路径。

问题：为什么gateway模块不直接使用yoho-service-modol模块里面OrderServices里面定义的请求路径？（request对象也是在该模块定义），目前使用内部常量定义的方式会不会造成维护上的困难。

具体请求流程：

调用HystrixServiceCaller(com.yoho.core.rest.client)的相应方法

ServiceFinderAdaptor发现服务

Yohobuy-order模块结构:

Dal 数据库持久层(所有模块均采用dal作为数据库持久层)

Common 公共部分

Deploy 发布部署相关

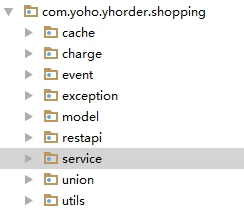
Shopping 购物车相关

Order 订单相关

Audit 黑名单相关

购物车主要针对两张表操作shopping\_car、shopping\_cart\_items。

购物车模块包结构如下图所示:



其中restapi包内 以restful风格定义具体的访问路由，同时以注解ServiceDesc定义具体的服务名，在注册服务和查找服务时通过该注解的服务名作为键值，这部分在框架模块有简单的说明。

Service包定义具体服务接口以及服务实现(impl包内):

IShoppingCartService定义购物车相关的服务接口，比如添加购物车、减少商品、查询、下单、结算等。

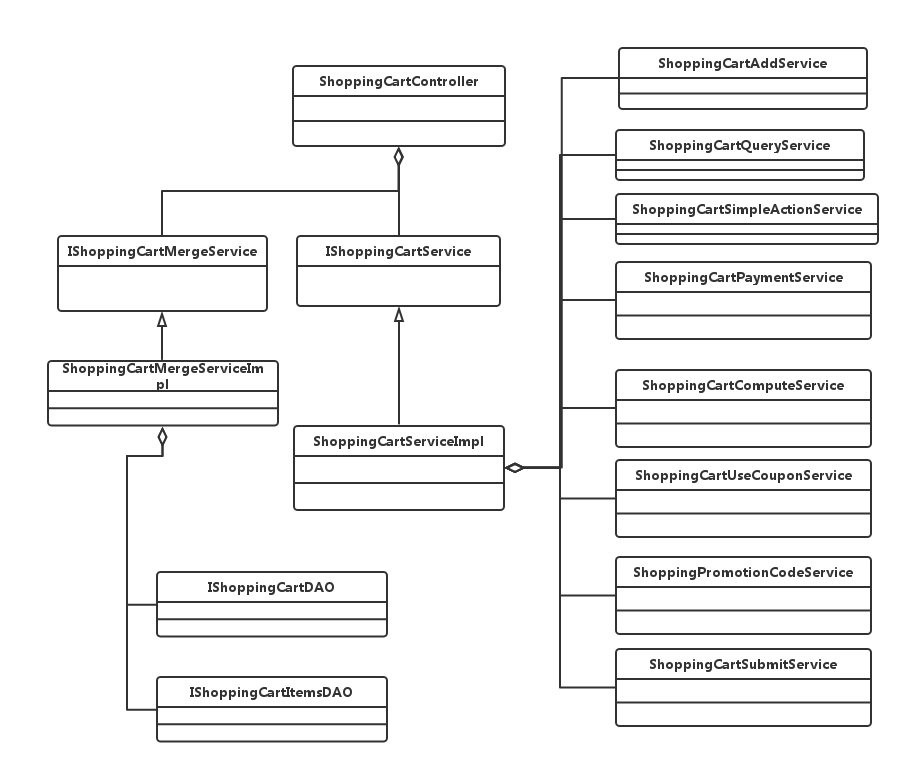
ExternalService服务定义了访问外部接口的方法，比如访问商品模块等。

IOrderCreationService定义下单接口

IShoppingCartMergeService 定义购物车合并接口

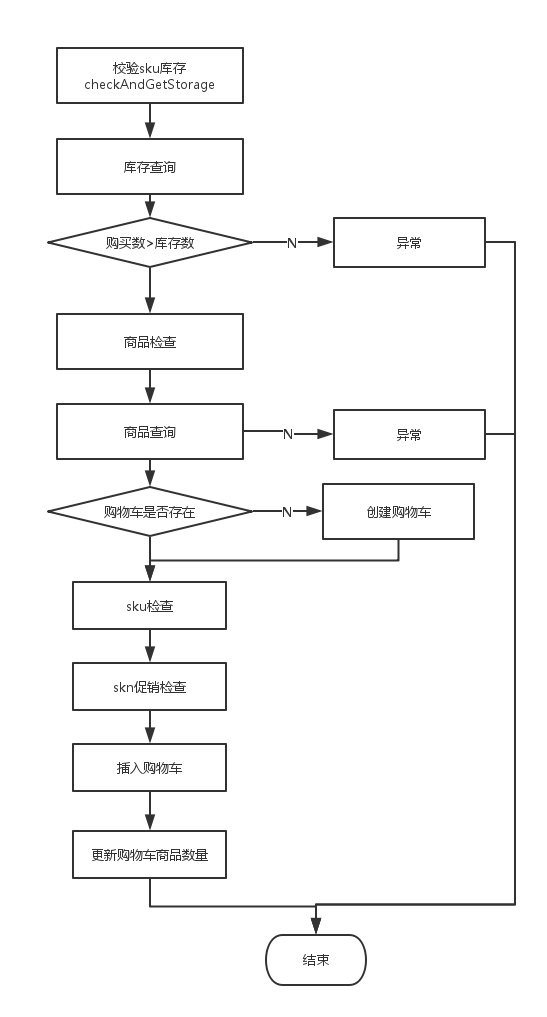
Charge包内为具体的购物车结算相关业务逻辑，包括打折、红包优惠等。

其主要类关系图如下:



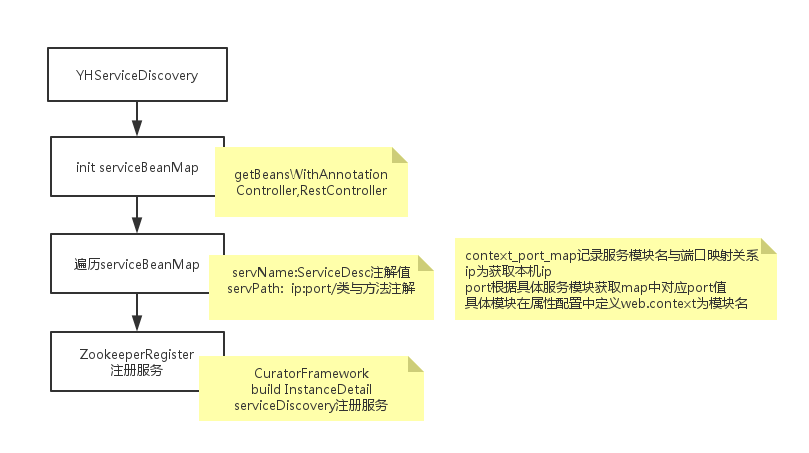
购物车部分业务流程：

添加商品至购物车流程大致如下图所示



购物车金额计算部分：待续

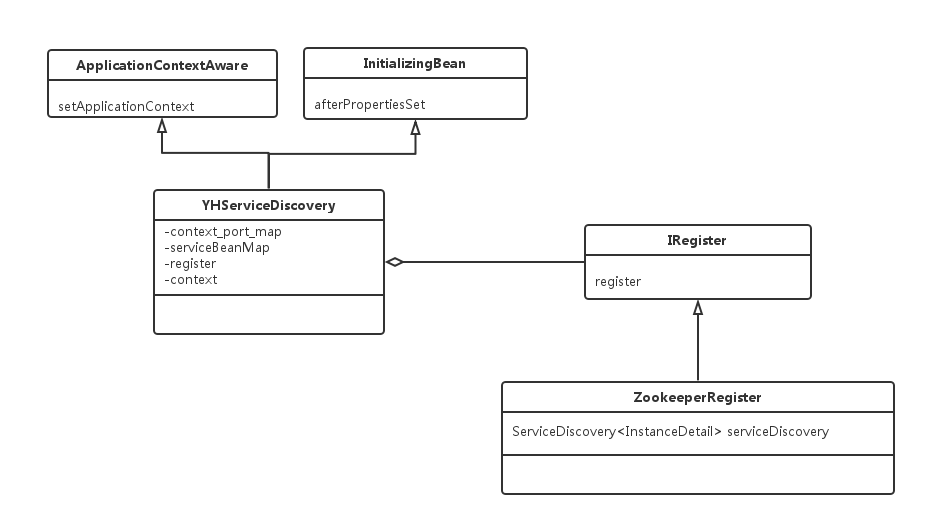
#### 框架学习 Yoho-core模块



服务注册主要通过YHServiceDiscovery类完成，其在模块启动时执行初始化，注册所有的服务。

YHServiceDiscovery

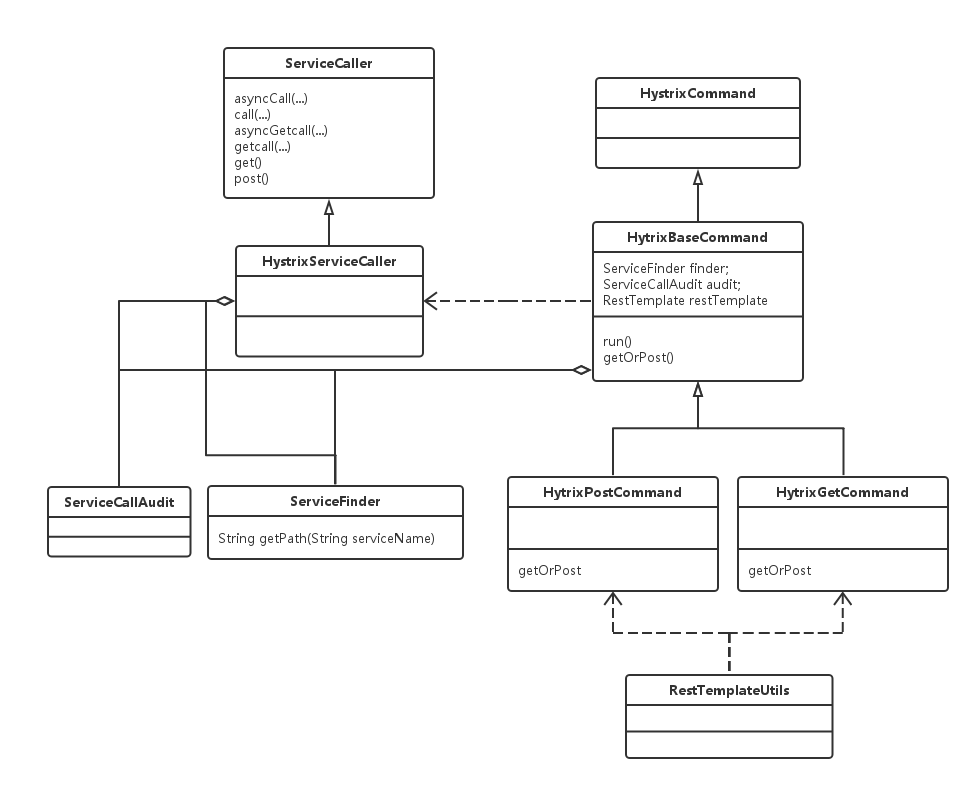
ZookeeperRegister



具体：待续

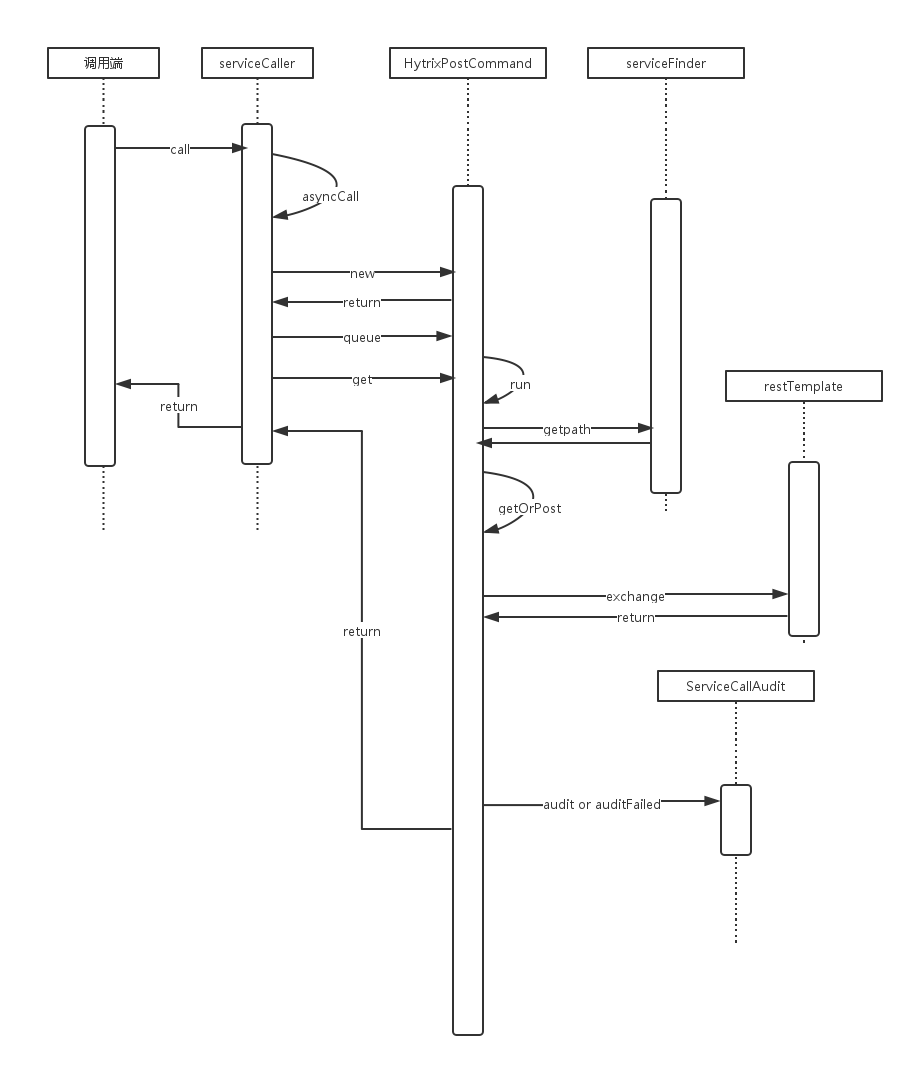
服务发现流程：

类图大致如下:



ServiceCaller是代码直接使用发起调用的接口，其实现类为HystrixServiceCaller，在HystrixServiceCaller内部将请求信息封装到HystrixBaseCommand中，使用Hystrix框架的命令模式进行访问调用。在HystrixServiceCaller内调用HystrixBaseCommand.queue方法，对应的HystrixBaseCommand的run方法被回调，run方法中首先会通过serviceFinder根据服务名查找服务路径，然后发起具体请求getOrPost,该方法本身也是虚方法，尤其具体子类实现(子类通过spring的restTemplate对象进行服务调用，)。ServicecallAudit是一个统计工具类。

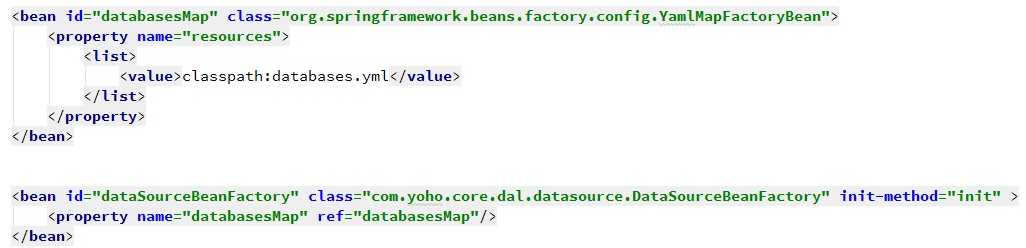
大致时序图如下：



ServiceCaller内部通过是否调用get来区别是否是异步还是同步请求，时序图里面说明的可能不够具体。

#### 数据库持久层学习

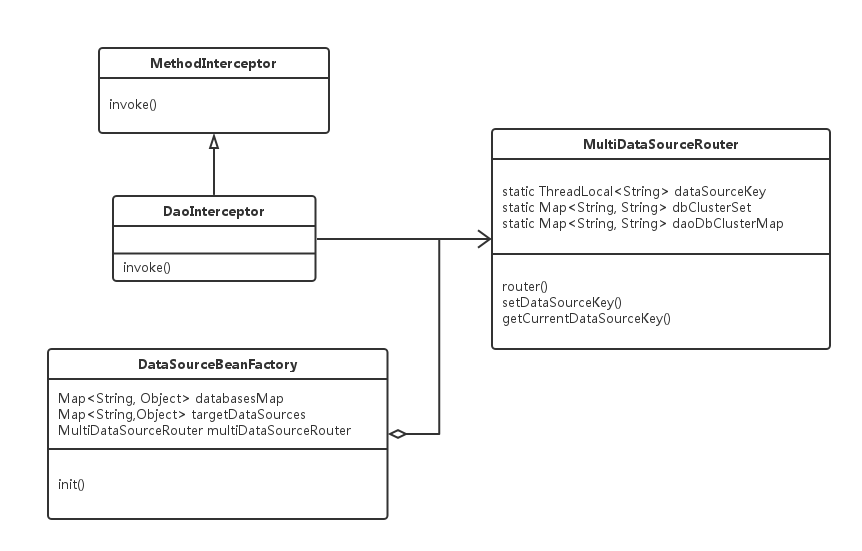
数据库相关配置 spring-mybatis-datasource.xml



Database.yml定义数据源

**DaoInterceptor**为数据库访问拦截器

数据库相关类图如下所示:

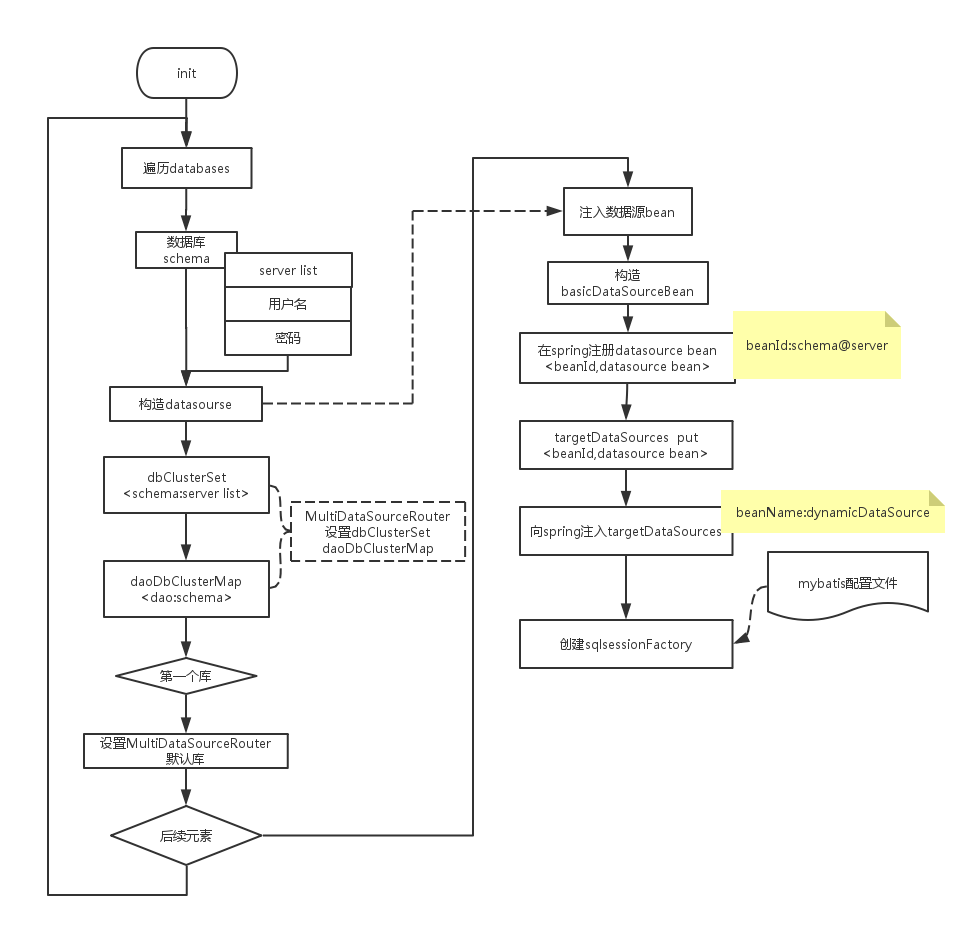


DataSourceBeanFactory在容器启动时执行初始方法init(),其内部解析数据库配置文件databases.yml来初始化数据库相关配置。解析yml文件通过spring注入的**YamlMapFactoryBean**的对象databasesMap。

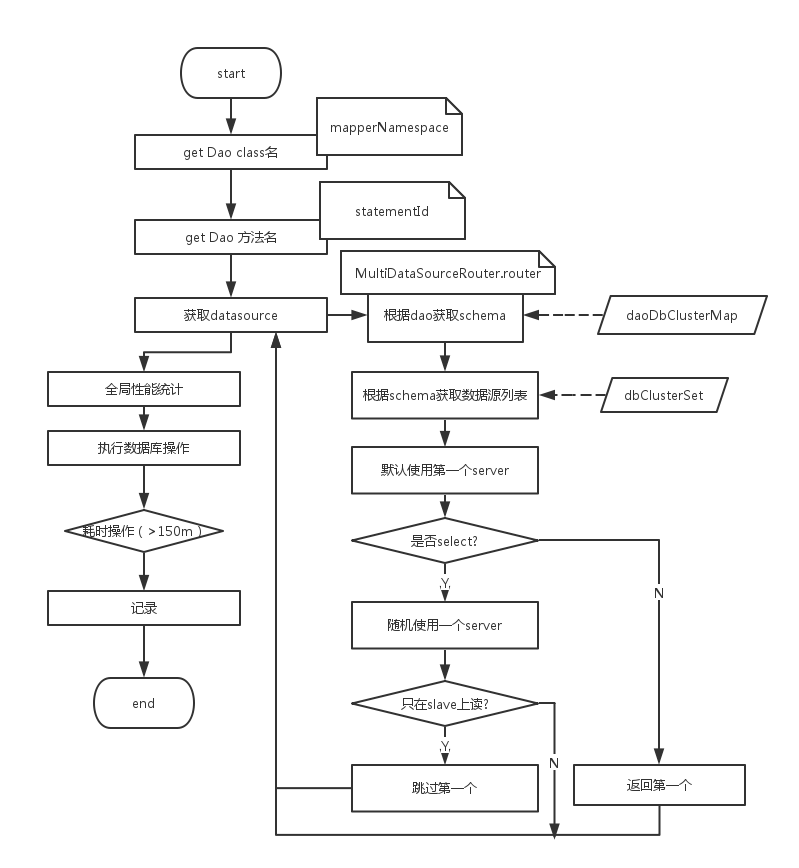
MultiDataSourceRouter 类的作用是登记数据源与数据库操作类的映射关系，在数据库拦截器内部获取当前操作对应的数据源，并设置线程局部数据源：

ThreadLocal<String> ***dataSourceKey***

Init方法大致流程如下图所示:



**DaoInterceptor**工作流程，即重写的invoke方法。



#### 缓存相关

使用:待续

#### MQ相关

使用:待续