首先 看整个构建顺序：

从`main`里构建`viewcontroller`和`view`， `viewcontroller`在构造方法中创建`blserviceimpl` 然后`bl层`的`impl`在构造方法中创建`dataserviceimpl` `dataserviceimpl`在构造方法中创建`dao`和`factory`直接初始化`data层`持有的数据信息（数据库不一定是这样吧）

界面层

view

```
    应该按照模块分ui包
```

viewcontroller (实现接口`ProcessOrder`)：

```
    在界面层内进行了部分逻辑操作（？），委托任务给不同的service
    拿到po再new VO
    按照用例分的
```

vo部分:

```
    把po作为构造函数的参数
    很特别的 实现了vector接口 直接add

-----
```

逻辑层

blservice： `orderservice` 和 `userservice` 符合我们的分配方法 然后下层`impl`分别继承

bl：由`impl`组成

```
    发现成员变量很少 几乎都是直接拿到po 没有vo操作（？）
    主要功能是传数据po给界面层及接受界面层的指令下调数据层代码
    整个demo没有界面层向下传递vo 有那种传两个参数的 所以具体对vo的操作demo里面并没有体现出在哪里解包什么的 应该是把vo传给逻辑层解包把
```

数据层

dataservice： `data`层接口 也是对应于`logic层`的`impl`

dataserviceimpl：实现`dataservice`

> 注意里面有factory和dao都在构造函数中初始化 factory在构造方法中直接new dao由factory new

dao层： `dao`接口和`daoimpl` 提供方法供`dataserviceimpl`调用

> daoimpl 与数据库的交互 我们具体实现的时候再研究

最后

代码没有用到rmi呀 这个我们也要考虑一下放在哪里