1. 设计过多程序会变得没有秩序，我们的目标是实现当前需求，不应该过早的考虑以后的需求，毕竟未来谁也说不定。
2. 过度自由的程序会使设计变得混乱，我们应该裁剪那些用不到的功能。
3. 有些功能是无法基于业务逻辑做设计（如权限系统），只能根据业务需要自己去逆向推导设计。
4. 尽量不要双向依赖。这里说的不只是关联，任何模型上显示的或者被隐藏的关系都尽量不要双向依赖。如A模型保存了B模型的一些数据（如ID或者Code等），B模型又保存了A模型的一些数据（如Name等），这就带来的双向依赖。
5. 在分布式系统中应该明确系统的依赖方向。不能A系统保存了B系统的数据，B系统又保存了A系统的数据。

6. 展示的与逻辑的分开考虑（其实这是一个关注点分离）。我们的开发以逻辑为主，展示功能的在开发完逻辑功能后再考虑

- 一个故事：

需求：有一个订单系统A和一个面单系统B，目前的依赖是B依赖于A，现在要求订单要能够绑定面单，并且展示它绑定的面单信息，而且订单要能够取消绑定面单。

方案1：

最开始我的方案是订单绑定面单时，我们生成新的面单，然后将面单的信息（如我们面单的快递公司，面单的单号等）保存到订单中，订单要取消面单只需要发送其对应的面单标识到面单系统B上进行取消就行了，展示面单时我们直接获取订单中的面单信息进行展示。

这个方案是可行的，但实施起来却给我带来了不小的麻烦。

问题1：这个方案造成了A与B系统的双向依赖。

问题2：面单系统不敢随意修改，它总是在担心这样会不会给订单带来影响。

问题3：订单要绑定面单时，我需要去面单系统申请面单，取得面单信息之后再将面单信息保存到订单中。

问题4：订单要取消面单时，我需要去面单系统取消这个面单，然后再去订单系统取消订单绑定的面单。

方案2：

其实这个需求一开始关注的点就错了，我需要关注的是订单能够绑定面单以及订单能够取消面单，等实现的该功能之后我们再考虑面单的展示。

所以方案2是订单绑定面单时，我们生成新的面单，面单保存订单ID，订单要取消面单时，只需要发送订单ID过来就可以取消了。而订单要展示其绑定的信息，只需要将订单ID发送过来，我们再返回其对应的面单信息即可，这样我们就完美解决的方案1的4个问题

- 这个故事告诉我们，要先考虑逻辑功能，再考虑展示功能，不应该同时考虑