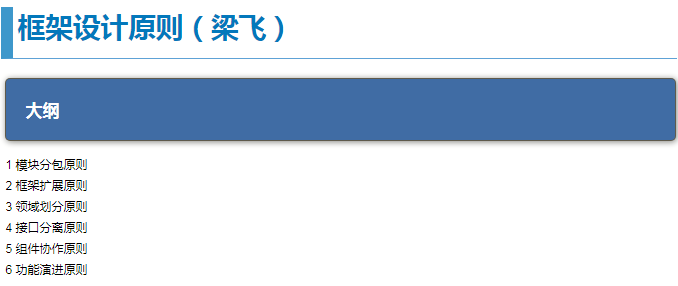
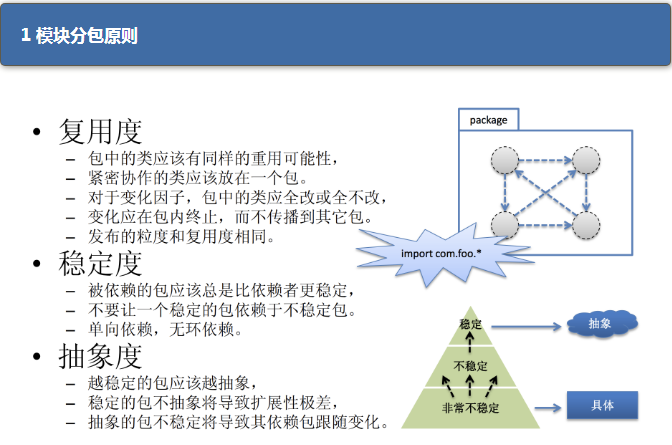
Dubbo框架设计原则



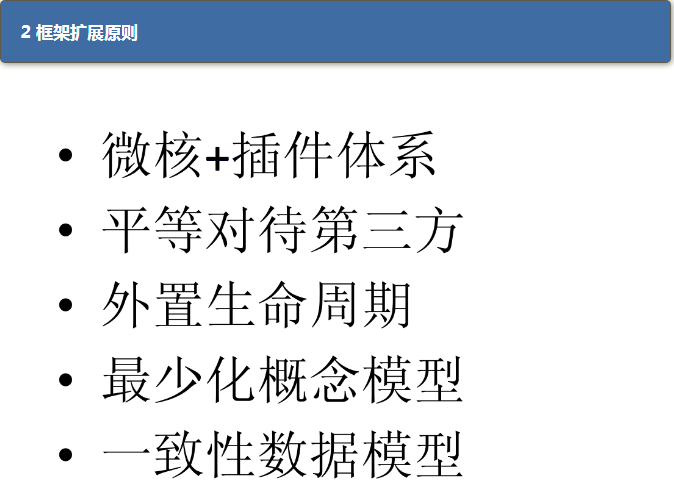


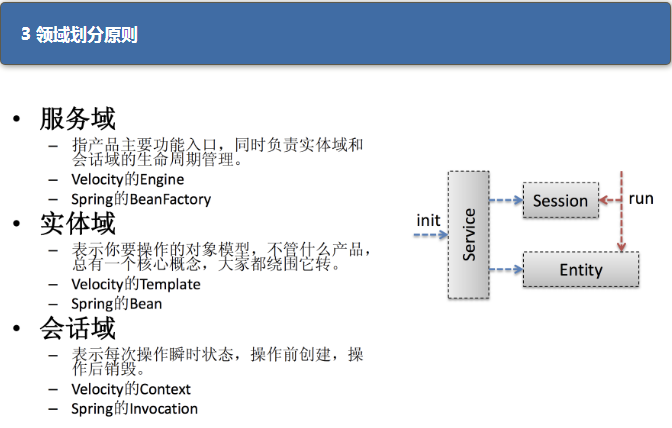
这里其实是从框架结构的角度来解读,这里的包指的是Maven的module.

复用度:指的是maven包的复用,可以理解为工具类.这个工具类不应该变化无常.

稳定度:被依赖的包应该保持稳定,或者说被依赖者应当比依赖者稳定;且不能成环状依赖.如果不稳定,将会影响其他的包.

抽象度:越抽象,越稳定;越具体,越容易变化.





接口分离原则,是单一职责的一种实现.

其中API是面向用户的,SPI是面向开发者的,两者必须分离.

API区分命令和查询,例如不应该有updateAndGet这个方法(不包括原子类),

应该分成2个方法,保证get方法幂等.

对称性接口:就是说有get方法,对应的就应该有set方法,有add方法就有remove方法,我们称之为对称性和完备性.

这样用户能自行推导出接口.

兼容性:如果接口加方法,应该是增加子接口的方式.



我们知道Dubbo是管道式设计.

一个Invoker贯通整个流程.事实上,web服务器都是这么设计的,例如Tomcat,Netty也是如此.

派发类似于Spring MVC框架中的DispatchServlet类

关于状态的共享：

* 分布是什么？即通过行为传递（适合交互性系统）。
* 共享是什么？通过一个固定的点获取，称之为仓库（适合管理状态的系统）。

主过程拦截，还记得 Mybatis 留给我们的插件吗？还记得 Spring 留给我们的拦截器吗？框架要在关键节点留出拦截点供用户扩展。

事件派发：观察者模式，Reactor 模式，另外提到 Proactor 模式，查了一下，通常在 GNU 编程中，由 OS 支持。

Dubbo 暴露、引用、调用事件，都预留了监听器。

关键路径，即在管道使用职责连模式进行拦截，保证每个拦截器职责单一。

非关键路径，需要有监听机制，不能影响主流程运行

关于协作防御，我理解为防御性编程.

1:分离可靠操作和不可靠操作.不可靠操作尽量范围要小.

2:状态分离,尽量无状态.状态要尽可能小.

3:对状态要尽早验证,因为如果失败,通常无人回滚.前后断言验证状态正确性,

4:异常防御,应该是预见性的异常,异常包含环境信息,

5:降低修改成本,防止埋雷:不要根据异常类型做分支判断.保持null和empty一致.



开闭原则,微核心+插件机制能够支持. 软件质量的下降,来源于修改.

加功能的姿势:应该是增量式,而不是扩充式;即不在原有基础上修改,而是新增加功能.

关于高阶:顶层接口尽量抽象,且不能依赖底层实现.这样,当底层实现变化时,高层无需变化.

例如Dubbo 泛化,在顶层就足够抽象,底层实现方式不影响高层.



框架设计和大部分人喜欢说的架构设计不同.

框架设计更能考验一个程序员对程序的抽象.

关于一个系统的设计,这里指的是框架的设计,首先要知道用户需求,

根据需求抽象出模型,再变成代码,且是可扩展,可复用的代码.

这里提到的 6 个原则，应该算是比较成熟的原则了.

1: 微核 + 插件，非常理想化，例如 SOFA，也有自己的扩展机制。

2: 关于领域模型设计，这 3 个模型的职责一定要划分清楚，同时实现无锁编程，这个对于系统的性能非常重要.

3: 关于组件协作，一个系统有多个组件，通常需要进行状态的共享，在 Dubbo 中，使用行为进行传递，也就是会话域.

4: 关于功能演进，请遵循开闭原则，但前提通常是有一个好的内核.

5 :关于接口分离和模块分包，通常在后期重构能够达到更好的效果?