《框架设计原则》

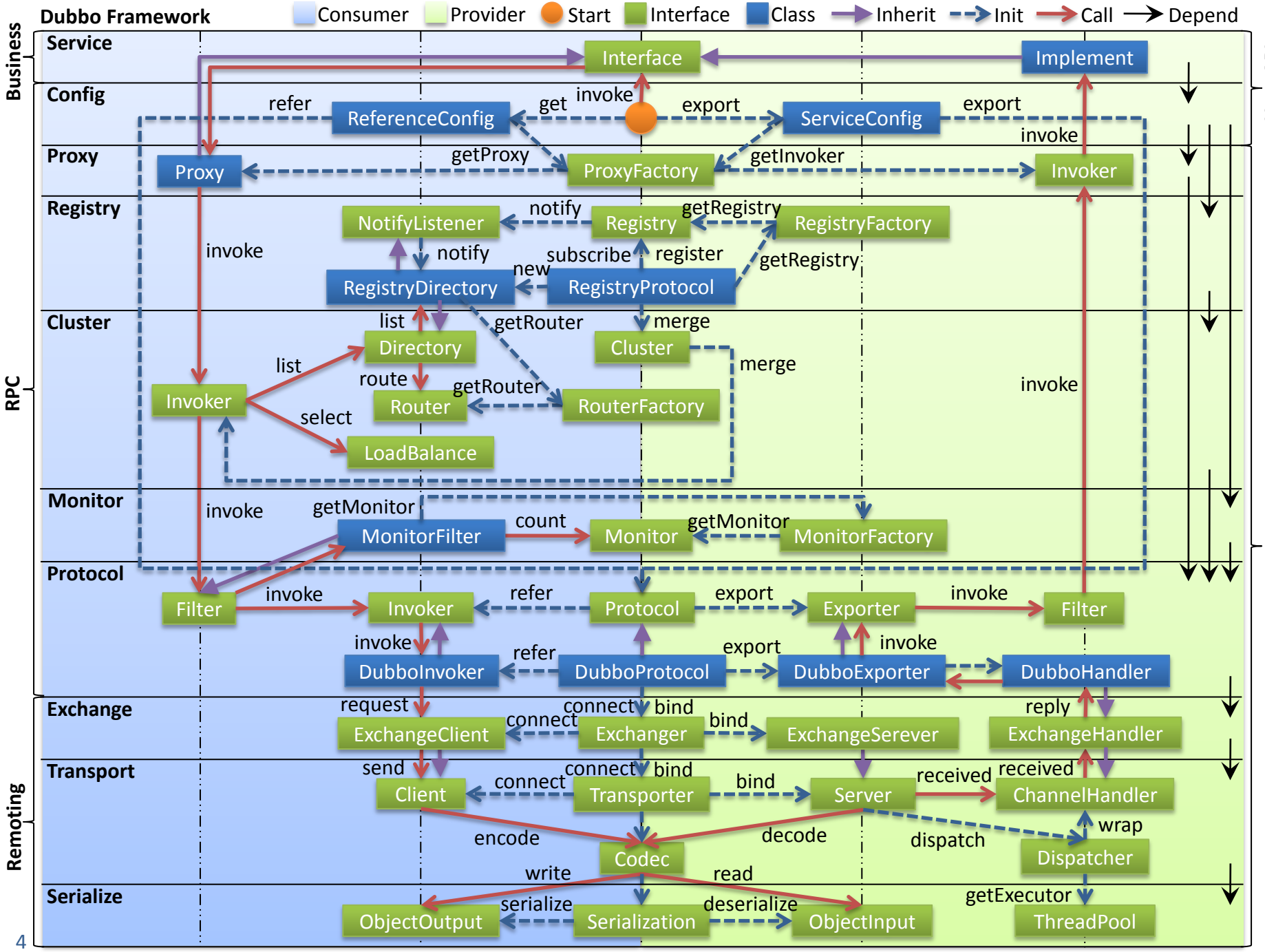
梁飞 (2012-03)

课程说明

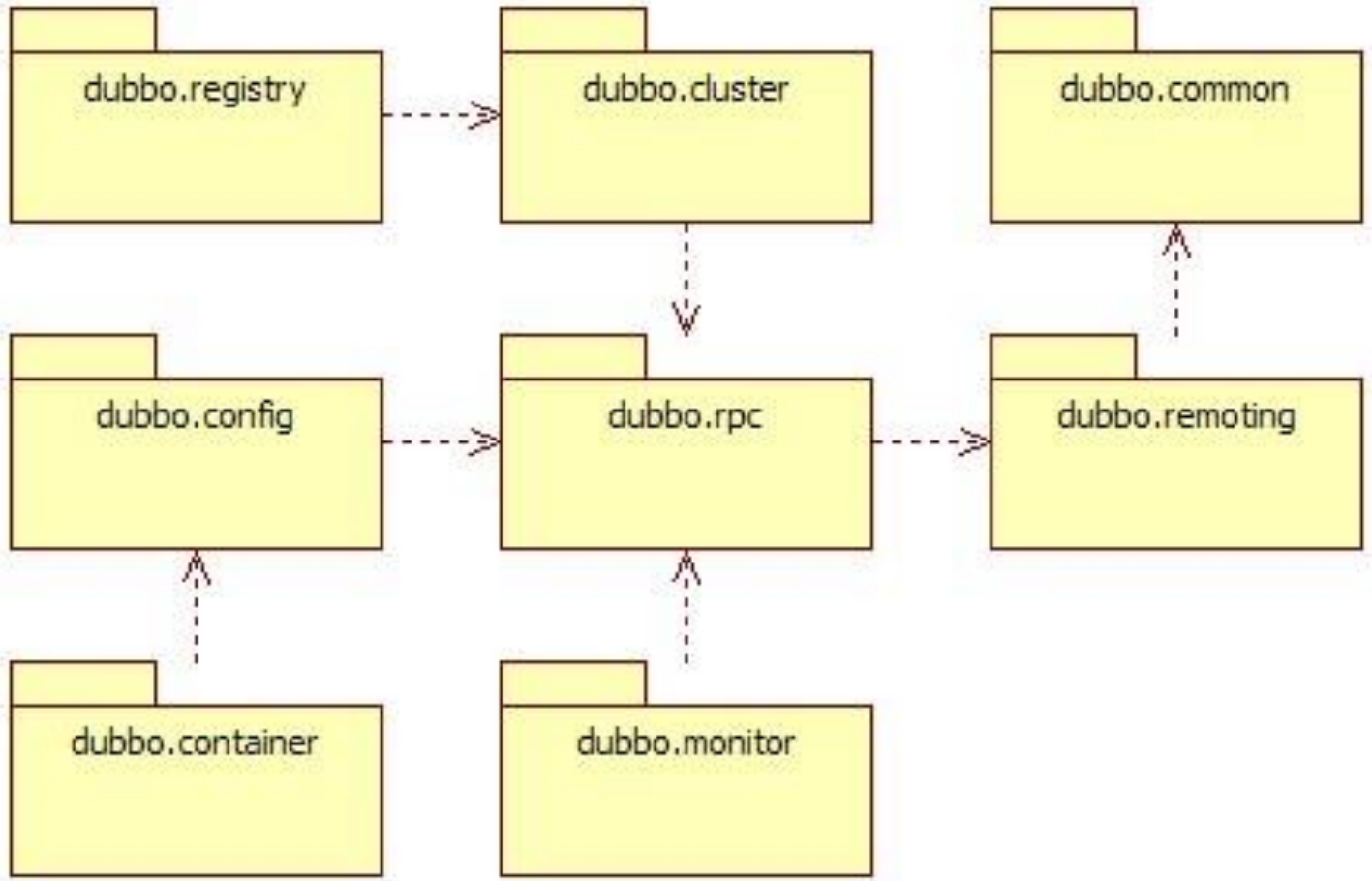
- 内容：
  - 主要讲在Dubbo设计过程中积累的一些经验，
  - 以及一些设计理论在Dubbo中的应用，
  - 并且只讲实践原则，不谈设计模式。
- 目的：
  - 希望给其它产品的设计起一些借鉴作用。

大纲

- 模块分包原则
- 框架扩展原则
- 领域划分原则
- 接口分离原则
- 组件协作原则
- 功能演进原则

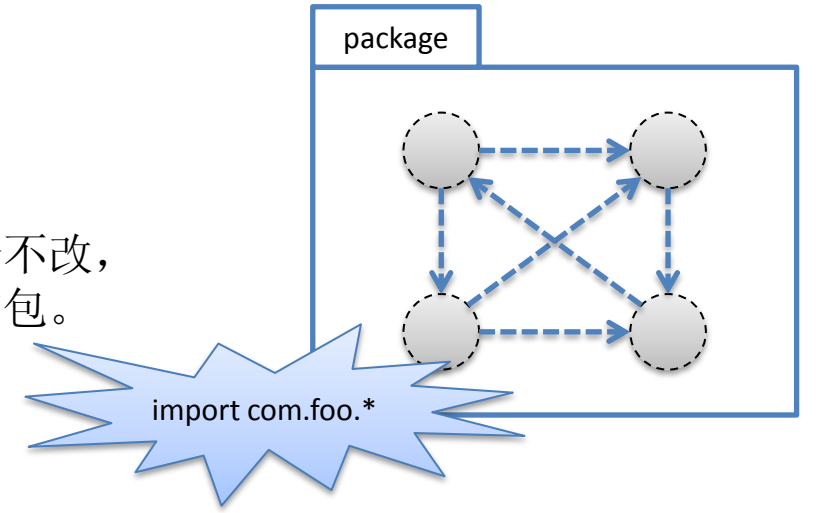

Dubbo Framework
Consumer
Provider
Start
Interface
Class
Inherit
Init
Call
Depend
Business
RPC
Remoting
User API
Contributor SPI
Service
Config
Proxy
Registry
Cluster
Monitor
Protocol
Exchange
Transport
Serialize
Interface
Implement
refer
ReferenceConfig
get
invoke
export
ServiceConfig
export
invoke
getProxy
Proxy
ProxyFactory
getInvoker
Invoker
notify
NotifyListener
Registry
getRegistry
RegistryFactory
invoke
notify
subscribe
register
getRegistry
new
RegistryDirectory
RegistryProtocol
list
getRouter
merge
Directory
Cluster
merge
list
route
getRouter
Invoker
Router
RouterFactory
select
LoadBalance
invoke
getMonitor
count
MonitorFilter
Monitor
getMonitor
MonitorFactory
invoke
refer
export
invoke
Filter
Invoker
Protocol
Exporter
Filter
invoke
refer
export
invoke
DubboInvoker
DubboProtocol
DubboExporter
DubboHandler
request
connect
bind
reply
ExchangeClient
connect
Exchanger
bind
ExchangeSerever
ExchangeHandler
send
connect
bind
received
received
Client
connect
Transporter
bind
Server
ChannelHandler
encode
decode
dispatch
wrap
Codec
Dispatcher
write
read
getExecutor
serialize
deserialize
ObjectOutput
Serialization
ObjectInput
ThreadPool

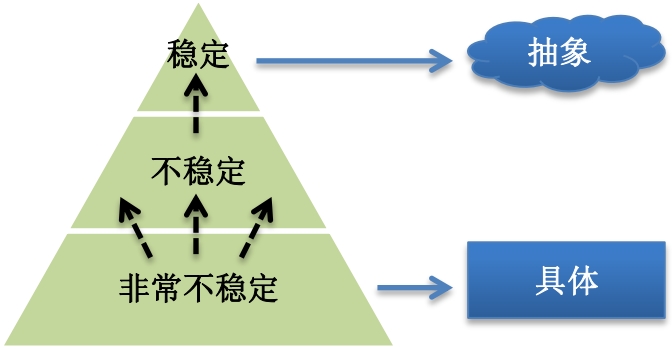
Dubbo的模块包



模块分包原则

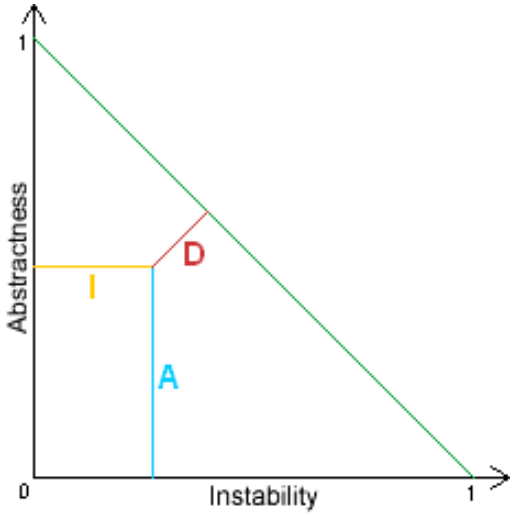
- 复用度
  - 包中的类应该有同样的重用可能性，
  - 紧密协作的类应该放在一个包。
  - 对于变化因子，包中的类应全改或全不改，
  - 变化应在包内终止，而不传播到其它包。
  - 发布的粒度和复用度相同。
- 稳定度
  - 被依赖的包应该总是比依赖者更稳定，
  - 不要让一个稳定的包依赖于不稳定包。
  - 单向依赖，无环依赖。
- 抽象度
  - 越稳定的包应该越抽象，
  - 稳定的包不抽象将导致扩展性极差，
  - 抽象的包不稳定将导致其依赖包跟随变化。

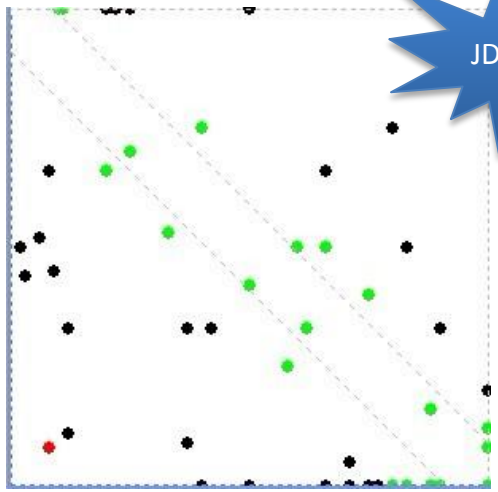




不稳定度与抽象度矩阵

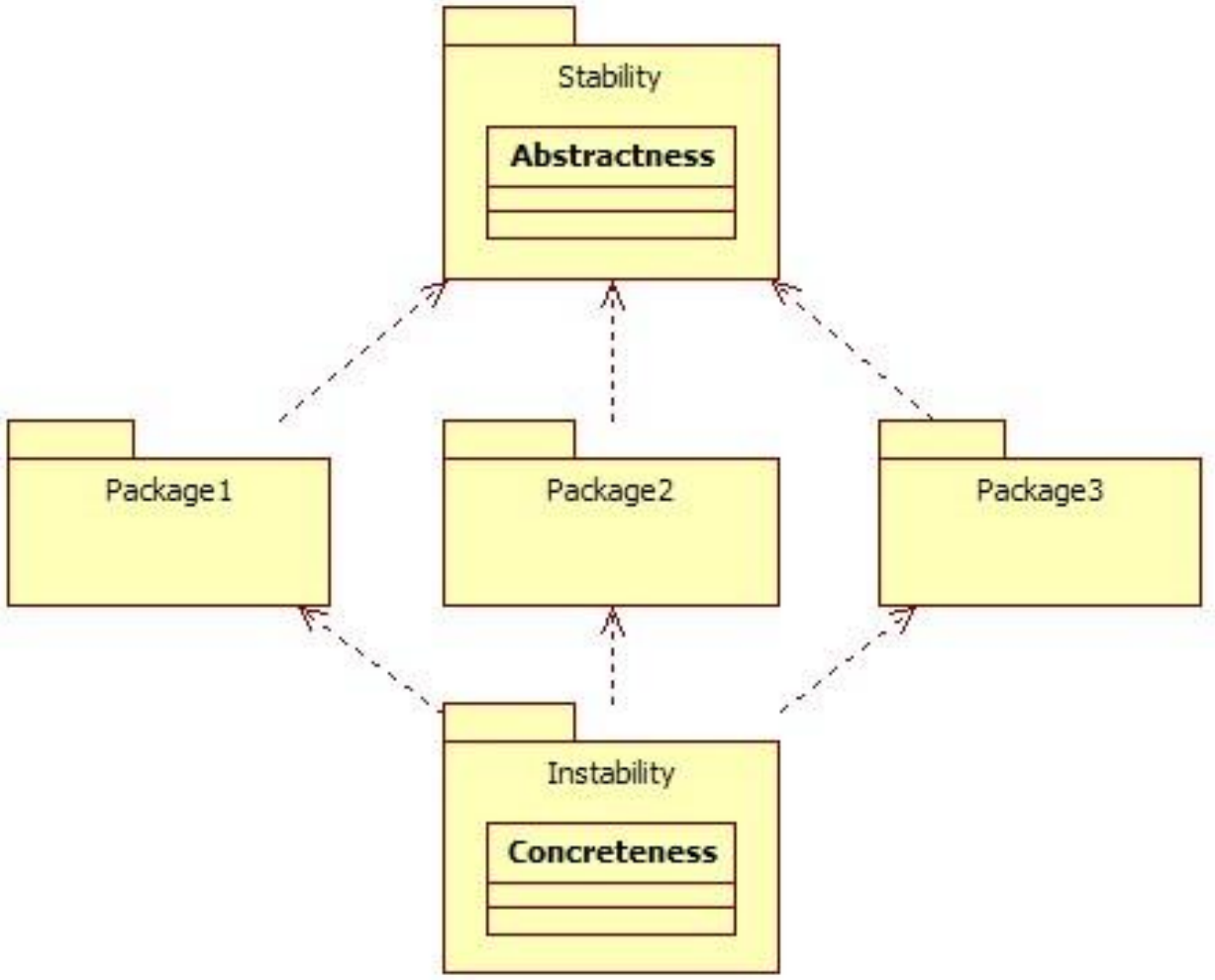
- **I = Ce / (Ca + Ce)**
  I: Instability
  Ca: Afferent Coupling (Used by packges)
  Ce: Efferent Coupling (Depend upon packges)
- **A = Na / Nc**
  A: Abstractness
  Na: Number of abstract classes
  Nc: Number of classes (Include abstract classes)
- **D = abs(1 - I - A) * sin(45)**
  D: Distance
  I: Instability
  A: Abstractness





JDepend

不稳定度与抽象度类图


Stability
Abstractness
Package1
Package2
Package3
Instability
Concreteness

大纲

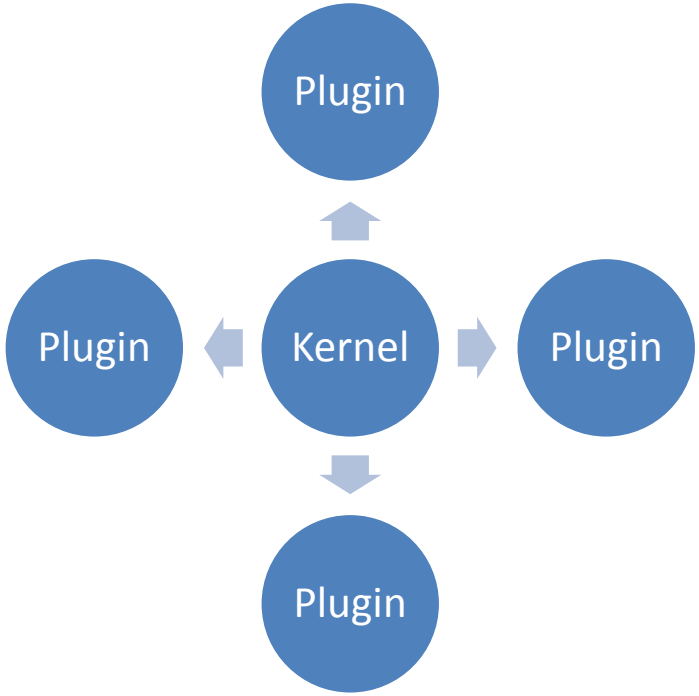
- 模块分包原则
- 框架扩展原则
- 领域划分原则
- 接口分离原则
- 组件协作原则
- 功能演进原则

框架扩展原则

- 微核+插件体系
- 平等对待第三方
- 外置生命周期
- 最少化概念模型
- 一致性数据模型

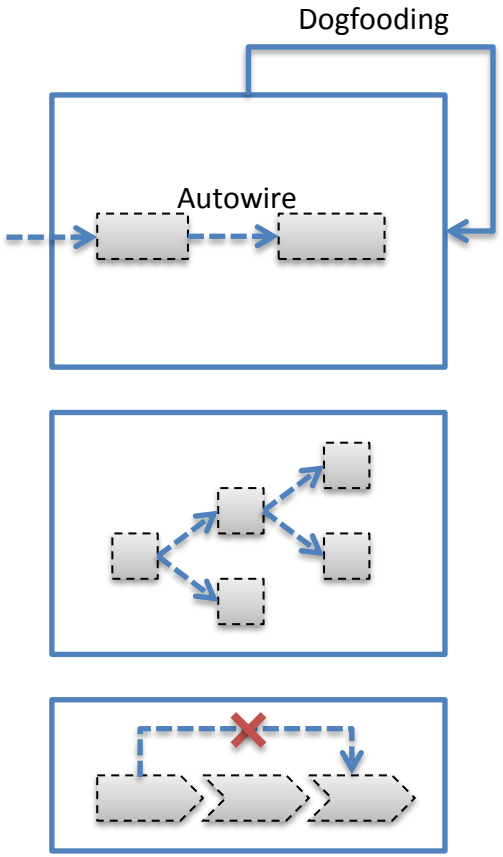
微核+插件体系

- OSGi
  - Equinox
  - Eclipse, HSF
  - META-INF/MANIFEST.MF
- IoC
  - Spring, Guice
  - META-INF/spring/beans.xml
- SPI
  - java.util.ServiceProvider
  - JDBC, MessageDigest, ScriptEngine
  - META-INF/services/com.xxx.Xxx



平等对待第三方

- Dogfooding
  - 框架自己的功能也要扩展点实现
  - 甚至微核的加载方式也可以扩展
- Autowire
  - 装配逻辑由扩展点之间互助完成
  - 杜绝硬编码的桥接和中间代码
- Cascading
  - 层叠扩展粒度，逐级细分
  - 由大的扩展点加载小的扩展点
- Law of Demeter
  - 只与触手可及的扩展点交互，间接转发
  - 保持行为单一，输入输出明确

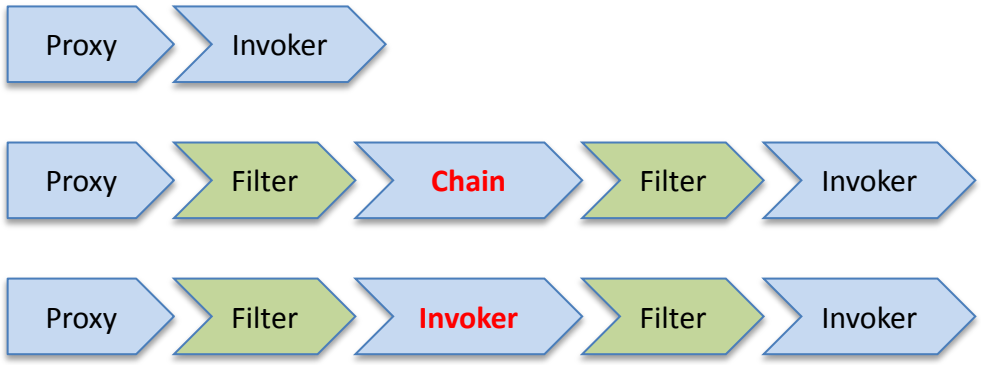


外置生命周期

- API传入参数，SPI扩展点实例
- 尽量引用外部对象的实例，而不类元
  - 正确：
    - userInstance.xxx();
  - 错误：
    - Class.forName(userClass).newInstance().xxx();
- 尽量使用IoC注入，减少静态工厂方法调用
  - 正确：
    - setXxx(xxx);
  - 错误：
    - XxxFactory.getXxx();
    - applicationContext.getBean(“xxx”);

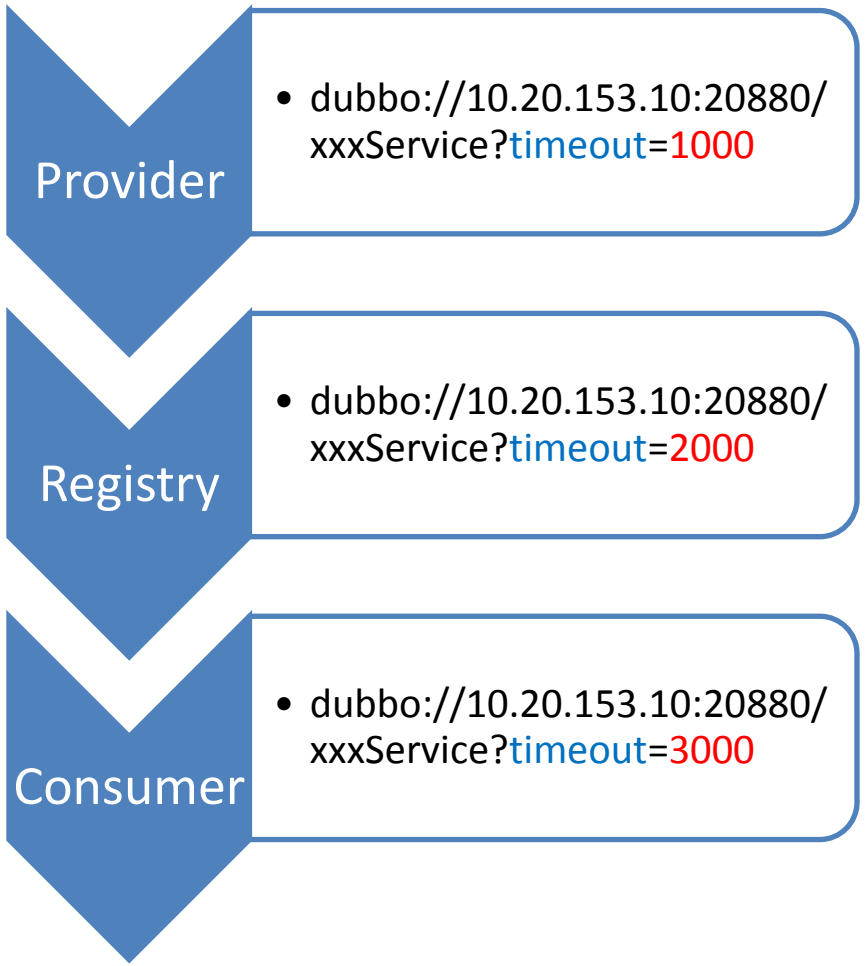
最少化概念模型

- Filter -> Chain?
  - Result filter(Chain c, Invocation i);
    - return c.doNext(i);
  - Result invoke(Invoker v, Invocation i)
    - return v.invoke(i);



一致性数据模型

- Dubbo统一URL模型：
  - 所有配置信息都转换成URL的参数
  - 所有的元信息传输都采用URL
  - 所有接口都可以获取到URL

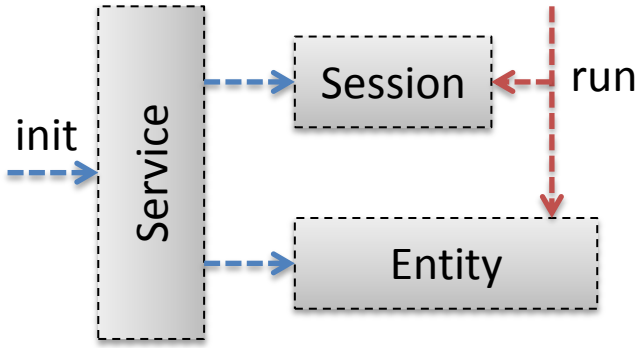


大纲

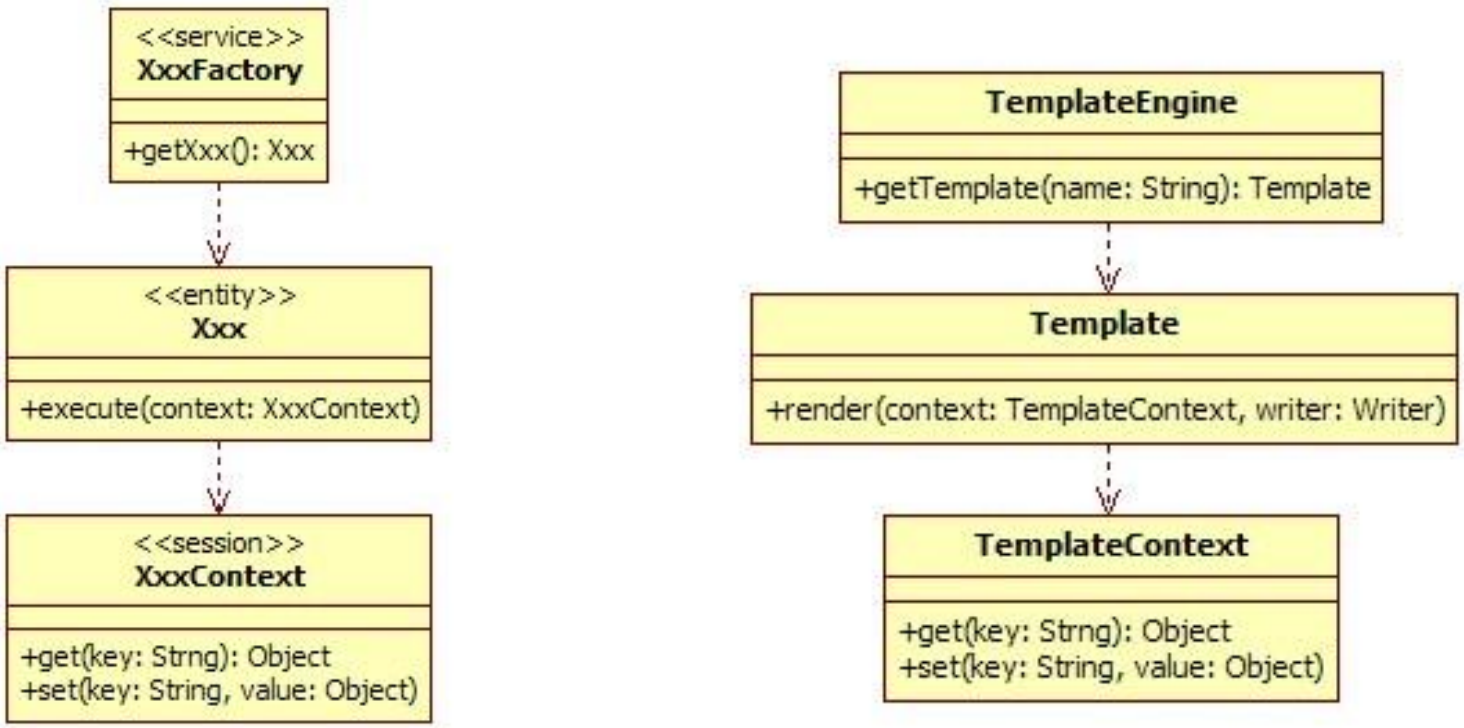
- 模块分包原则
- 框架扩展原则
- 领域划分原则
- 接口分离原则
- 组件协作原则
- 功能演进原则

领域模型划分

- **服务域**
  - 指产品主要功能入口，同时负责实体域和会话域的生命周期管理。
  - Velocity的Engine
  - Spring的BeanFactory
- **实体域**
  - 表示你要操作的对象模型，不管什么产品，总有一个核心概念，大家都绕围它转。
  - Velocity的Template
  - Spring的Bean
- **会话域**
  - 表示每次操作瞬时状态，操作前创建，操作后销毁。
  - Velocity的Context
  - Spring的Invocation



领域模型类图


<<service>>
XxxFactory
+getXxx(): Xxx
<<entity>>
Xxx
+execute(context: XxxContext)
<<session>>
XxxContext
+get(key: Strng): Object
+set(key: String, value: Object)
TemplateEngine
+getTemplate(name: String): Template
Template
+render(context: TemplateContext, writer: Writer)
TemplateContext
+get(key: Strng): Object
+set(key: String, value: Object)

Dubbo的核心领域模型

- 服务域： Protocol
  - 它是Invoker暴露和引用的主功能入口，它负责Invoker的生命周期管理。
- 实体域： Invoker
  - 它是Dubbo的核心模型，其它模型都向它靠扰，或转换成它，它代表一个可执行体，可向它发起invoke调用，它有可能是一个本地的实现，也可能是一个远程的实现，也可能一个集群实现。
- 会话域： Invocation
  - 它持有调用过程中的变量，比如方法名，参数等。

领域模型划分优势

- 结构清晰，可直接套用
- 充血模型，实体域带行为。
- 可变与不可变状态分离，可变状态集中
- 所有领域线程安全，不需要加锁

领域模型线程安全性

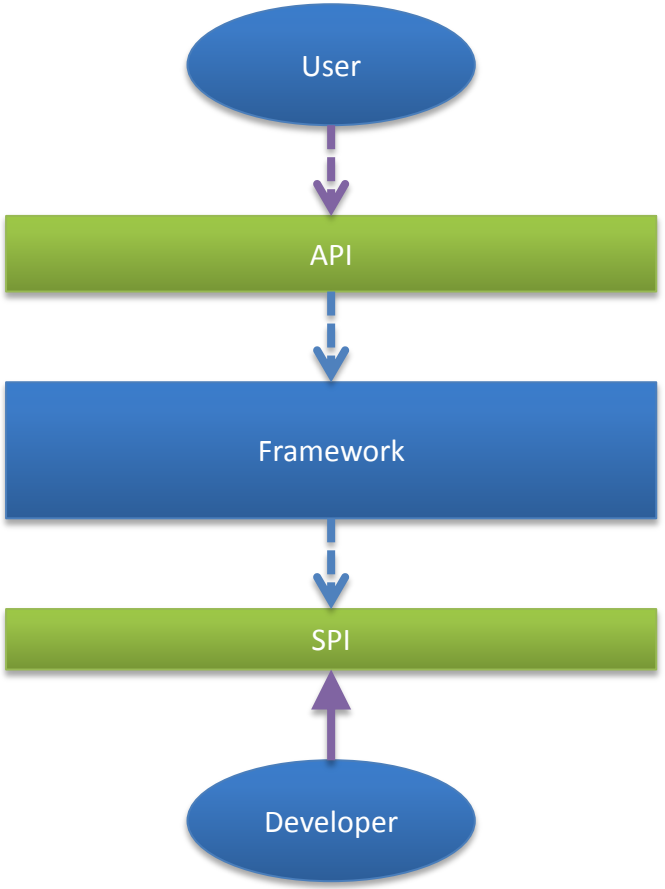
- 服务域
  - 通常服务域是无状态，或者只有启动时初始化不变状态，所以天生线程安全，只需单一实例运行。
- 实体域
  - 通常设计为不变类，所有属性只读，或整个类引用替换，所以是线程安全的。
- 会话域
  - 保持所有可变状态，且会话域只在线程栈内使用，即每次调用都在线程栈内创建实例，调用完即销毁，没有竞争，所以线程安全。

大纲

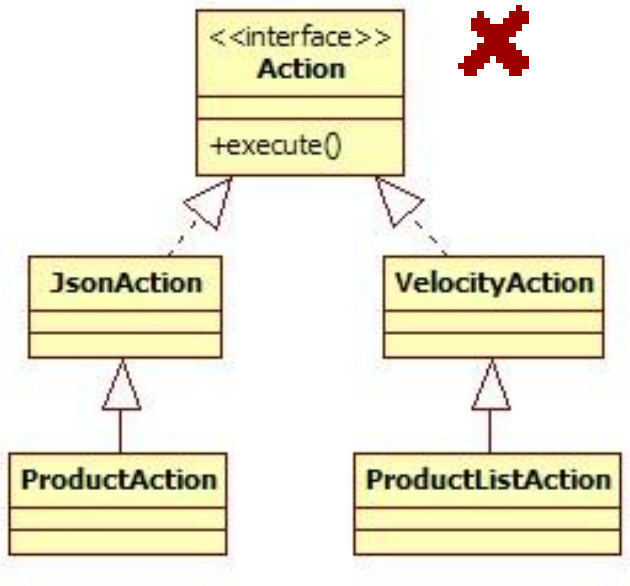
- 模块分包原则
- 框架扩展原则
- 领域划分原则
- 接口分离原则
- 组件协作原则
- 功能演进原则

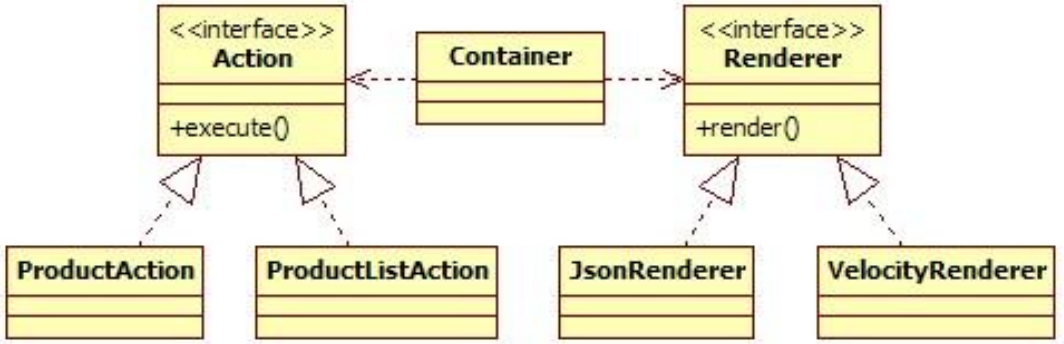
API & SPI

- Dubbo API
  - ServiceConfig
  - ReferenceConfig
  - RpcContext
- Dubbo SPI
  - Protocol
  - Transporter
  - LoadBalance



API与SPI分离


<<interface>>
Action
+execute()
JsonAction
VelocityAction
ProductAction
ProductListAction


<<interface>>
Action
+execute()
Container
<<interface>>
Renderer
+render()
ProductAction
ProductListAction
JsonRenderer
VelocityRenderer

声明式API，过程式SPI

- 声明式API
  - 描述需要什么
- 过程式SPI
  - 描述怎么实现

- 声明式事务
  - @Transactional
  - begin, commit, rollback
- Maven v.s. Ant
  - mvn install
  - ant target1 target2

API可配置，一定可编程

- 配置用于简化常规使用

```
<dubbo:service
   interface="com.alibaba.xxx.XxxService“
   version="1.0.0”
   ref="xxxService"
/>
```

- 编程接口用于框架集成

```
ServiceConfig service = new ServiceConfig();
service.setInterface("com.alibaba.xxx.XxxService");
service.setVersion("1.0.0");
service.setRef(xxxService);
service.export();
```

API区分命令与查询

命令：

- void set(Xxx xxx); // 无返回值表示命令，有副作用

查询：

- Xxx get(); // 有返回值表示查询，保持幂等，无副作用

不建议：

- Object delete(); // 删除并返回 

建议：

- Object get(); // 先查询
- void delete(); // 再删除

对称性接口

- 保持接口的对称性和完备性：
  - add & remove
- 用户能基于常用接口推导其它接口：
  - export & refer => exported & referred

API & SPI 兼容性

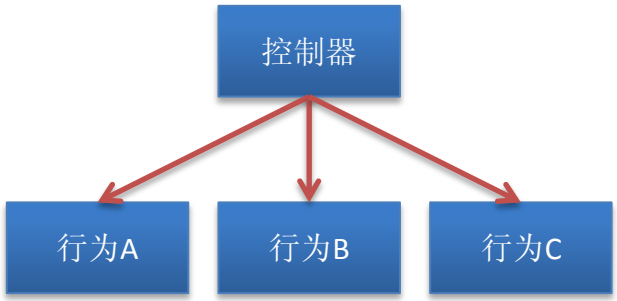
- 二进制兼容 v.s. 源码兼容
  - DEFAULT = 1 => DEFAULT = 2 // 编译时内联
  - set(Child c) => set(Parent p) // 里氏代换原则
- 接口 v.s. 抽象类
  - 保持接口方法是接口名的完备集
  - 抽象类更容易兼容，但对继承树的侵入大
  - API接口，SPI接口+抽象基类
- 规则 v.s. 硬编码
  - 以某符号开头的特殊处理
  - 指定关键字特殊处理
- 包名调整
  - 让旧接口继承调整后的接口
- 接口加方法
  - 增加子接口，通过instanceof识别新方法

大纲

- 模块分包原则
- 框架扩展原则
- 领域划分原则
- 接口分离原则
- 组件协作原则
- 功能演进原则

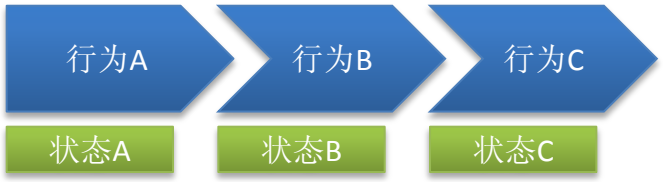
管道 v.s. 派发

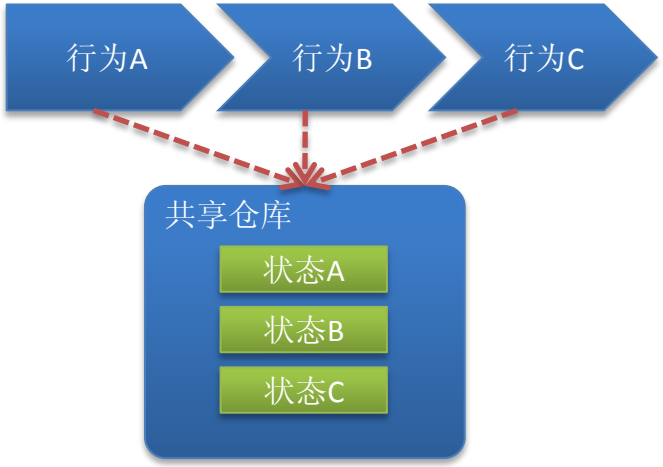
- 管道
  - 组合行为
  - 主功能以截面实现
  - 比如：Servlet
- 派发
  - 策略行为
  - 主功能以事件实现
  - 比如： Swing



分布 v.s. 共享

- 分布
  - 在行为交互为主的系统是适用
  - 状态通过行为传递
- 共享
  - 在以管理状态为主的系统中适用
  - 状态通过仓库共享

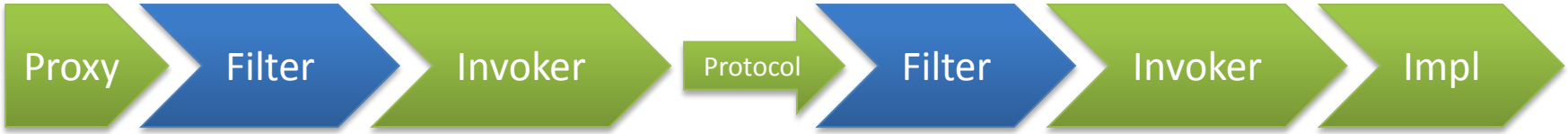




主过程拦截

- Web框架的请求响应流
- ORM框架的SQL执行
- Service框架的调用过程
- 反例：
  - IBatis2在SQL执行过程中没有设拦截点，导致添加安全或日志拦截，执行前修改分页SQL等，不得不hack源代码。

Dubbo调用过程拦截

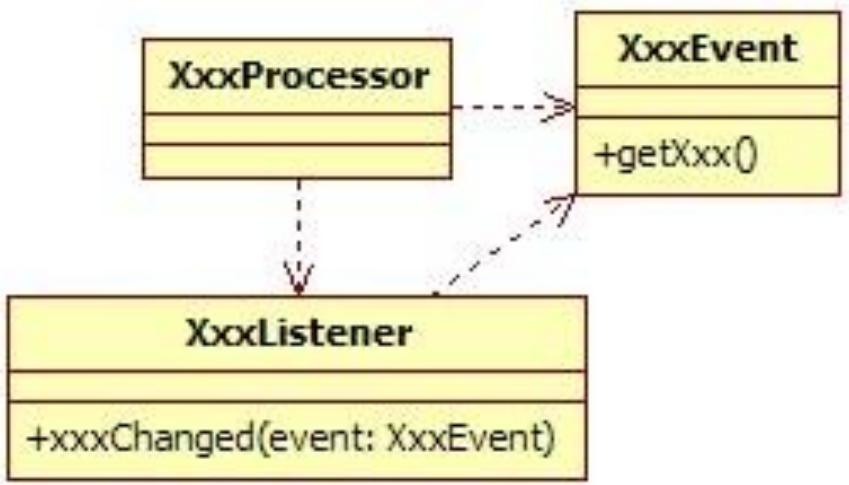


context
deprecated
collect
generic
activelimit
monitor
future
…

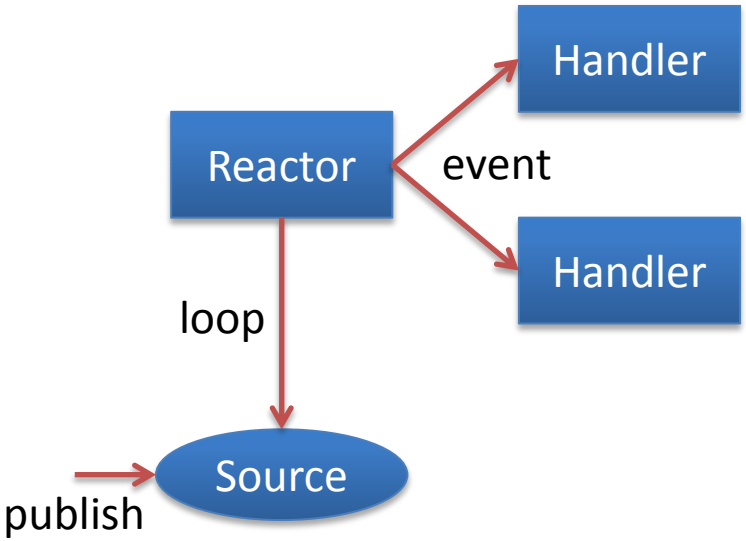
token
exception
echo
accesslog
trace
classloader
executelimit
…

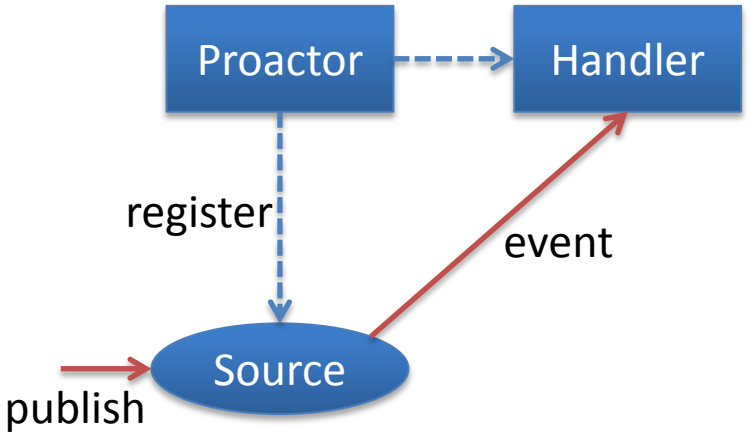
事件派发

- 过程
  - 执行前后
  - 触发附带非关键行为
- 状态
  - 值的变化
  - 触发状态观察者行为

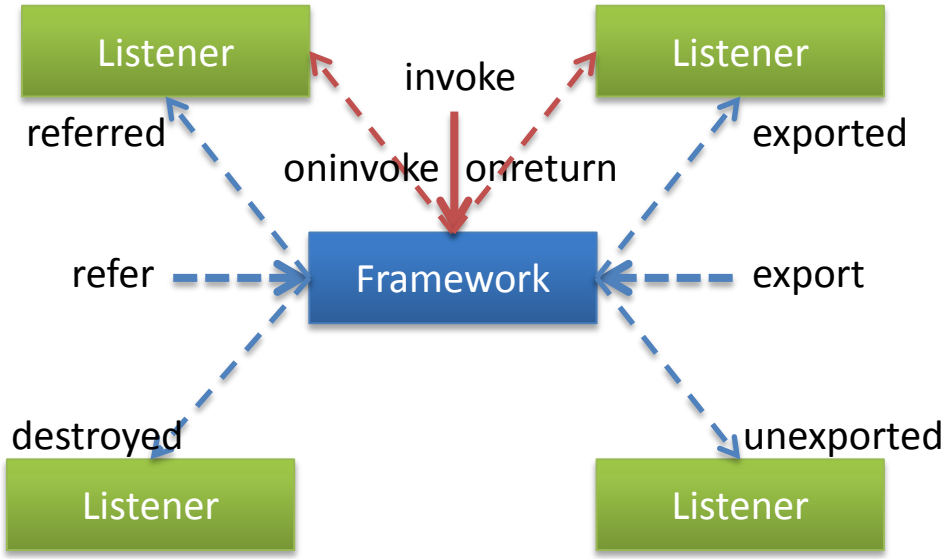


Reactor v.s. Proactor





Dubbo暴露/引用/调用事件



关键路径

- 关键路径
  - 采用拦截链分离职责
  - 保持截面功能单一，不易出问题
- 非关键路径
  - 采用后置事件派发
  - 确保派发失败，不影响主过程运行

协作防御

- 可靠性分离：
  - 可靠操作
  - 不可靠操作 (尽量缩小)
- 状态分离：
  - 无状态
  - 有状态 (尽量缩小)
  - 不可变类 (尽量final)
- 状态验证：
  - 尽早失败
  - 前置断言 + 后置断言 + 不变式
- 异常防御，但不忽略异常
  - 异常信息给出解决方案
  - 日志信息包含环境信息
- 降低修改时的误解性，不埋雷
  - 避免基于异常类型的分支流程
  - 保持null和empty语义一致

协作契约断言

- 前置断言
  - assert(arg != null);
  - if (arg == null) throw new IllegalArgumentException();
- 后置断言
  - assert(result != null);
  - if (result == null) throw new IllegalStateException();
- 不变式
  - assert(a + b = c);
  - if (a + b != c) throw new IllegalStateException();

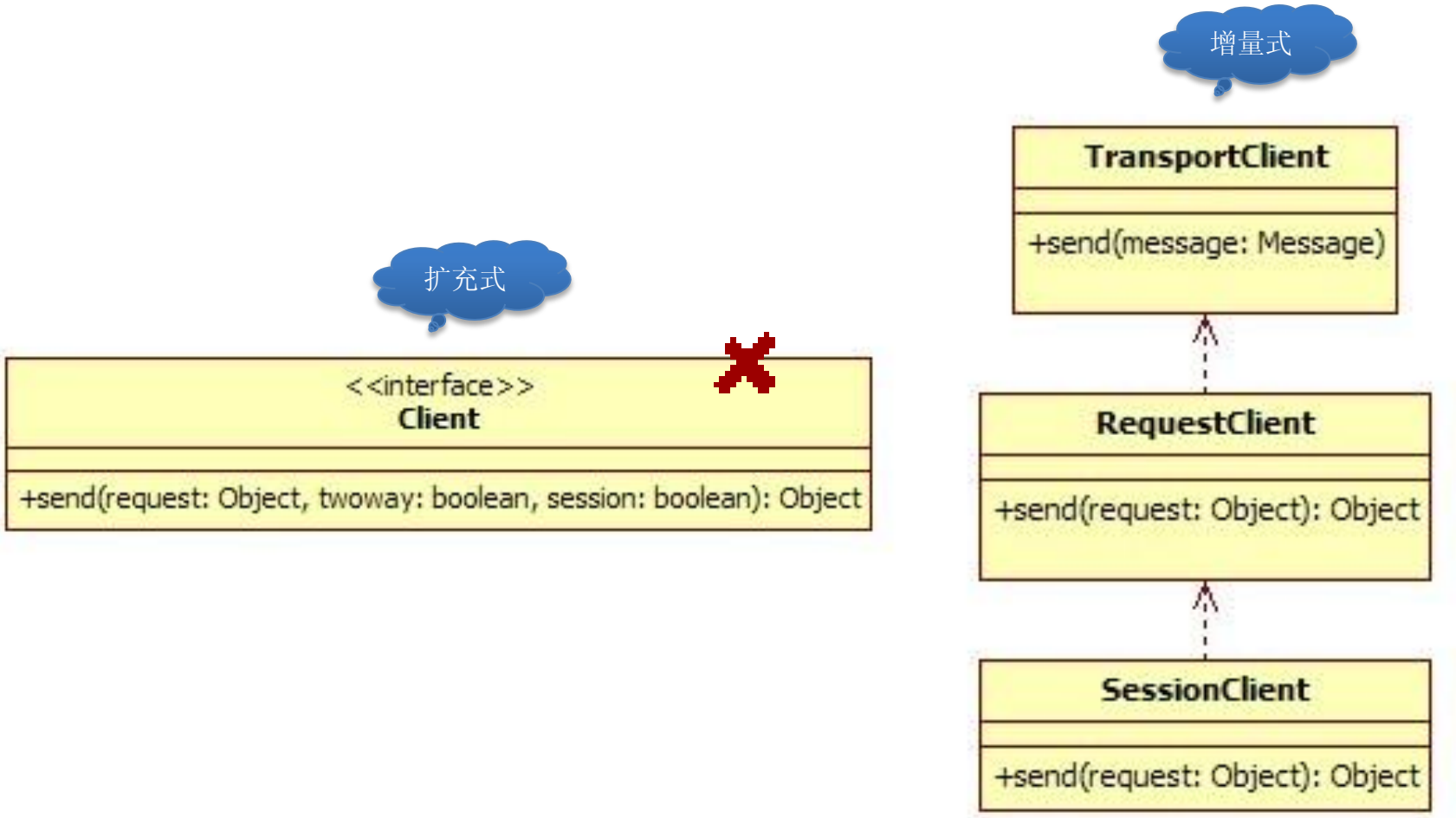
大纲

- 模块分包原则
- 框架扩展原则
- 领域划分原则
- 接口分离原则
- 组件协作原则
- 功能演进原则

开闭原则

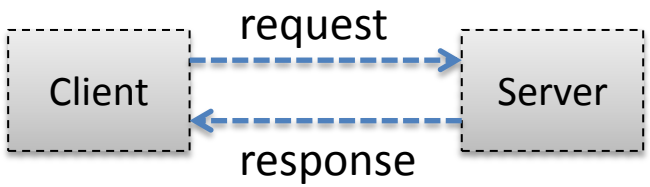
- 对扩展开放。
- 对修改关闭。
- 软件质量的下降，来源于修改。
- 替换整个实现类，而不是修改其中的某行。

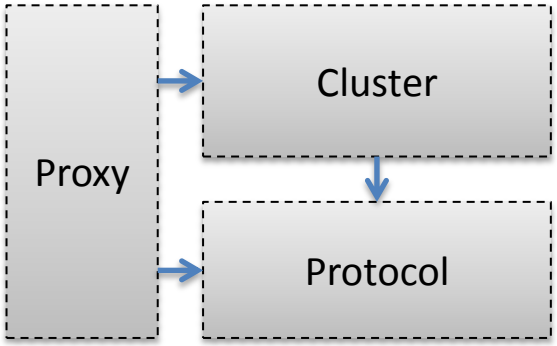
增量式 v.s.扩充式


增量式
TransportClient
+send(message: Message)
RequestClient
+send(request: Object): Object
SessionClient
+send(request: Object): Object
扩充式
<<interface>>
Client
+send(request: Object, twoway: boolean, session: boolean): Object

Dubbo增量式扩展

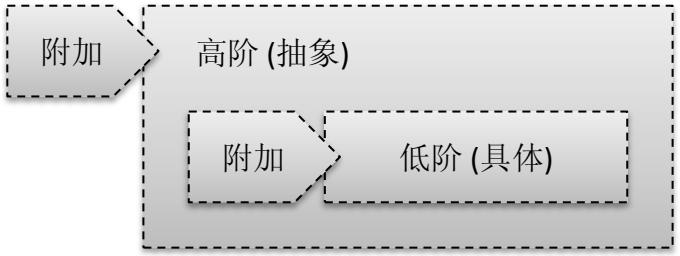
- Remoting
  - Transport：
    - 单向消息发送，抽象Mina/Netty
  - Exchange：
    - 封装Request-Respose语义
    - 调用两次单向消息发送完成
- RPC
  - Portocol：
    - 协议实现，不透明，点对点
  - Cluster：
    - 将集群中多个提供者伪装成一个
  - Proxy：
    - 透明化接口，桥接动态代理



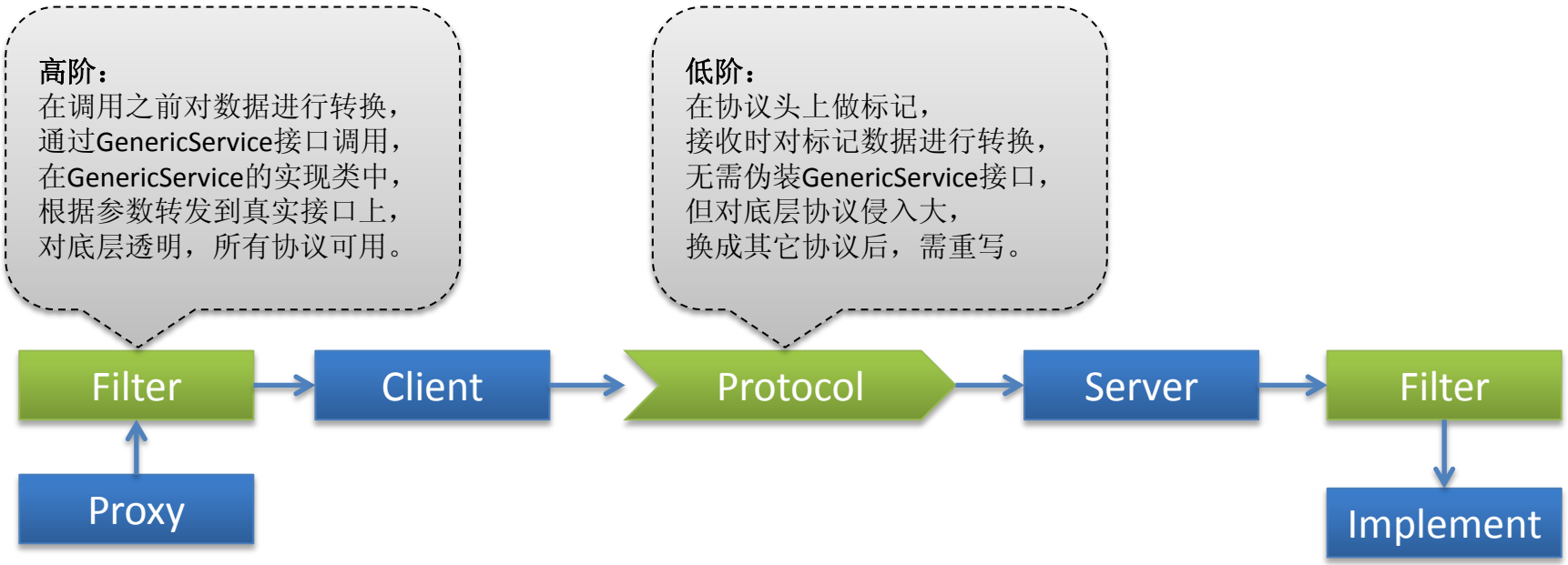


在高阶附加功能

- 尽可能少的依赖低阶契约，用最少的抽象概念实现功能。
- 当低阶切换实现时，高阶功能可以继续复用。



Dubbo高阶泛化调用



总结

模块分包原则

- 复用度 + 稳定度 + 抽象度

接口分离原则

- 声明式API + 过程式SPI + 分离

框架扩展原则

- 微核 + 插件 + 平等 + 一致

组件协作原则

- 拦截 + 事件 + 共享 + 防御

领域划分原则

- 服务域 + 实体域 + 会话域

功能演进原则

- 开闭 + 增量 + 高阶

Q & A

- Q & A
  - http://code.alibabatech.com/wiki/display/dubbo