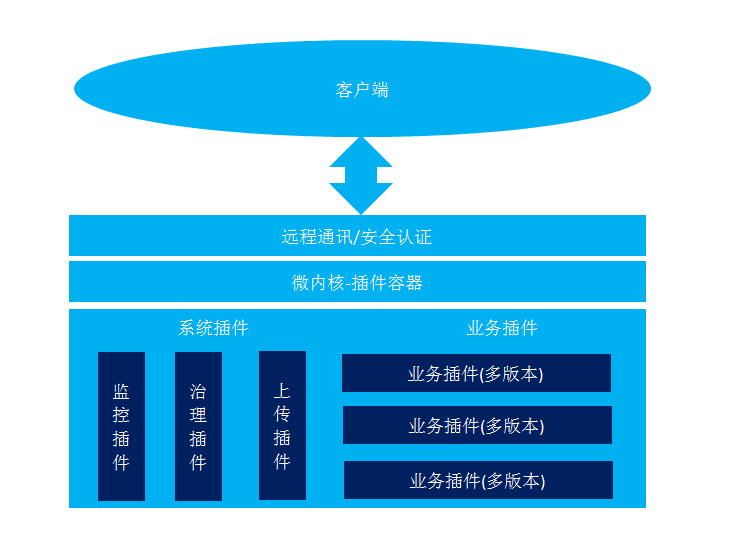
**基于插件化的服务化框架Remoting**

## 架构设计

Remoting是一个 '多服务' ，'多版本' ，‘微内核’，‘插件化’的服务化框架.



多服务：一切都是服务，监控服务，治理服务，上传服务，业务对外提供服务。

多版本：一个服务有多个版本，是真正意义上的多版本，独立类加载器加载管理。

微内核：Remoting本身。

插件化：开发人员按照Remoting发布的规范进行开发。

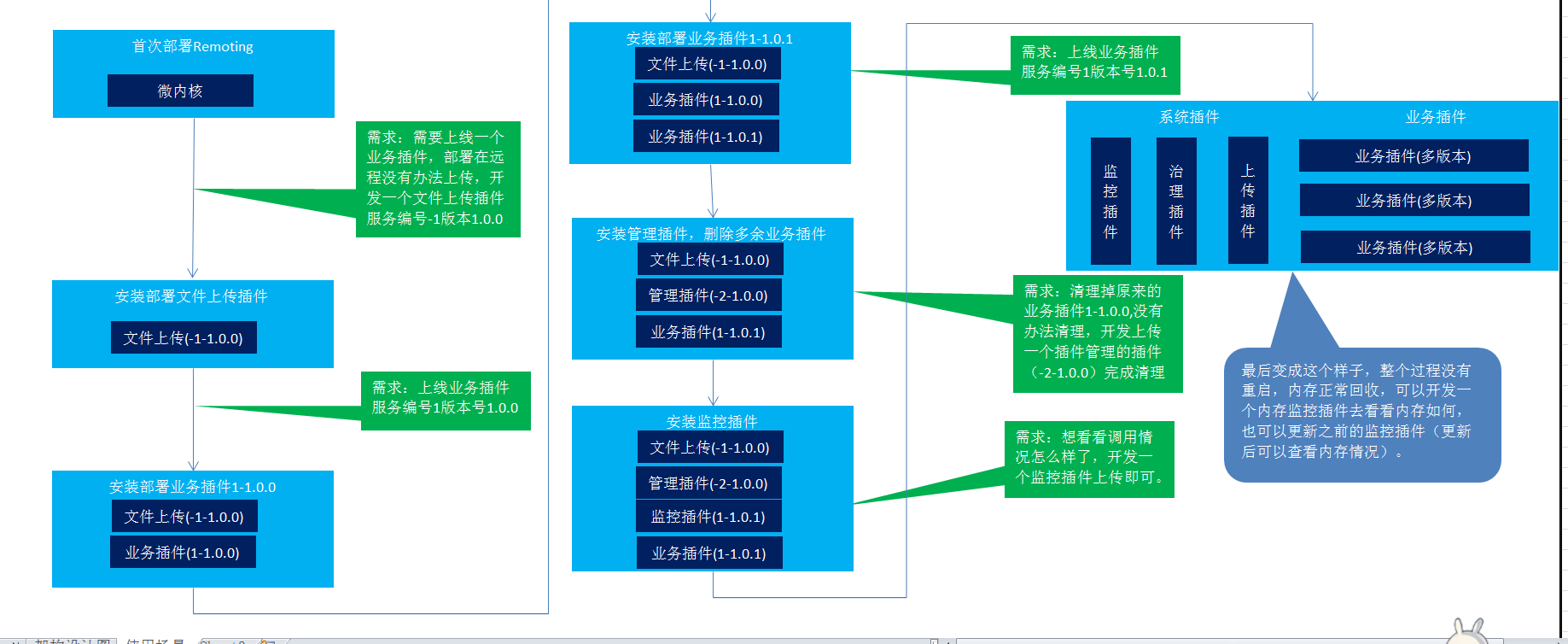
客户端：客户端发起调用，指定调用哪个插件哪个版本。

**动态部署插件**

**插件管理插件**

**插件提供服务**

## 场景演进



## 客户端怎么弄？

如果不考虑客户端多版本相对简单，通过修改请求参数的服务号(sid)和版本号(version)来决定调用哪个插件。

## 技术选型

通讯层:基于当前开源的成熟稳定的通讯框架netty。

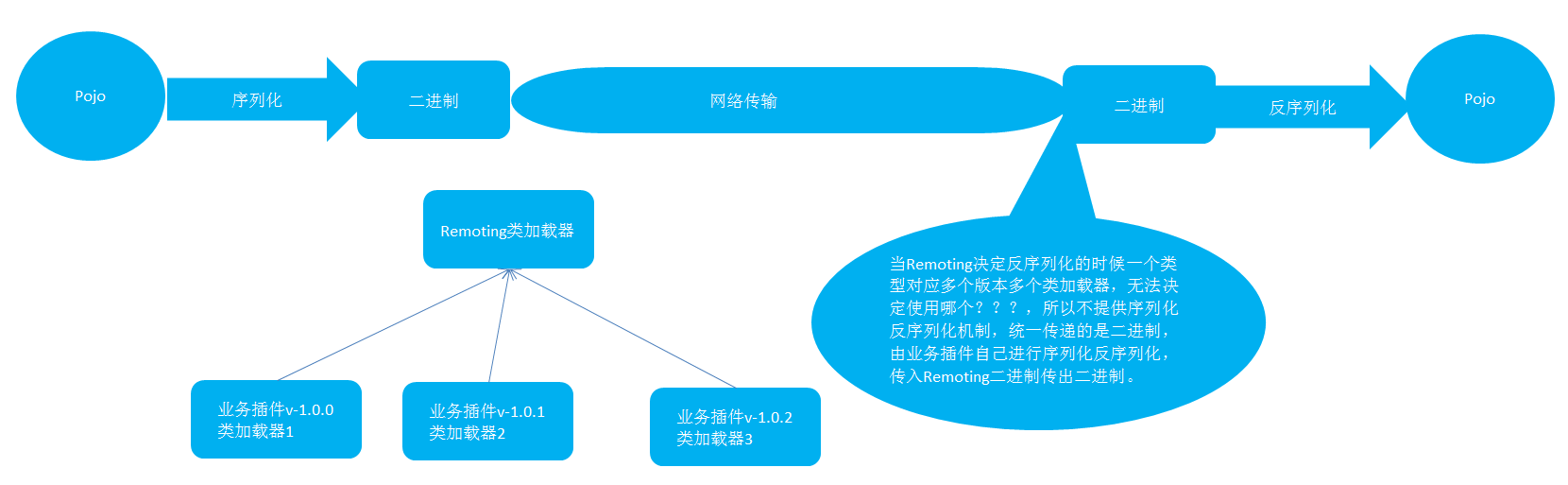
插件容器:基于java的类加载机制自主进行开发(为什么不选择OSGi,上手慢学习成本太高,推广难度大，过多考虑插件之间的依赖关系,最终工作都在bundle化上面，暂不考虑)。

自定义开发插件:除netty外无任何技术选型的限制(微内核处于插件类加载上层，会优先加载，所以自定义插件限制netty，其实和Remoting使用相同版本netty无所谓)。

## 序列化反序列化的思考

本打算把序列化和反序列化封装掉dubbo提供了这样的功能 觉得比较好，可以让用户自己选择比如json，protostuff，hession和其他的方式，让开发人员使用的使用不要关心细节提高生产力。

但是在这里行不通，



当然也可以通过维护版本和类加载决定使用哪个类加载器，这样需要倾入第三方序列化反序列化库，有点得不偿失。

## 模块划分

## Remoting通信层协议

协议格式 <length> <header length> <header data> <body data> 1 2 3 4 协议分4部分，

含义分别如下

1、大端4个字节整数，等于2、3、4长度总和

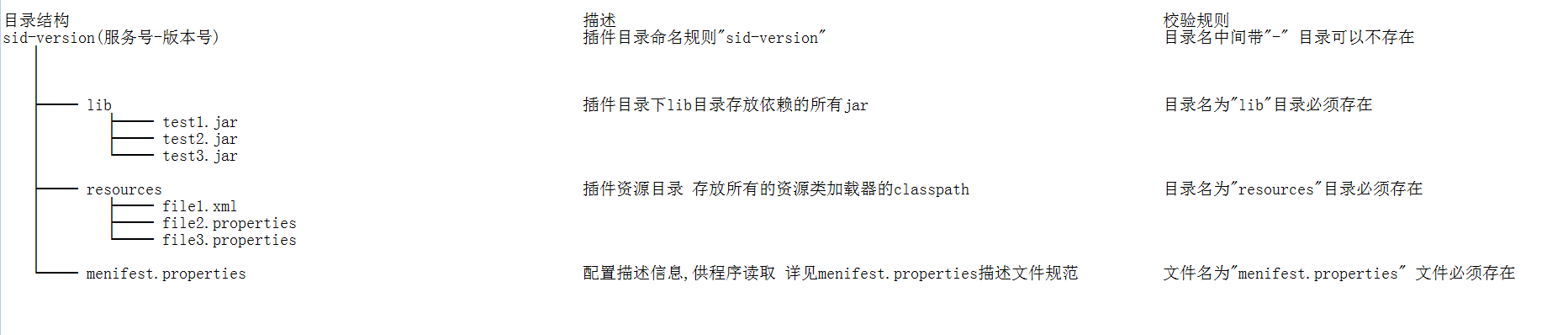
2、大端4个字节整数，等于3的长度

3、使用json序列化数据

4、应用自定义二进制序列化数据

详细见Remoting通信协议 v1.0.0.txt

## Remoting插件定义规范



详细见Remoting插件定义规范 v1.0.0.txt