如何优雅的落地中间件

找钢网 刘星辰

- | 如何优雅的落地中间件

CONTENTS

目 录

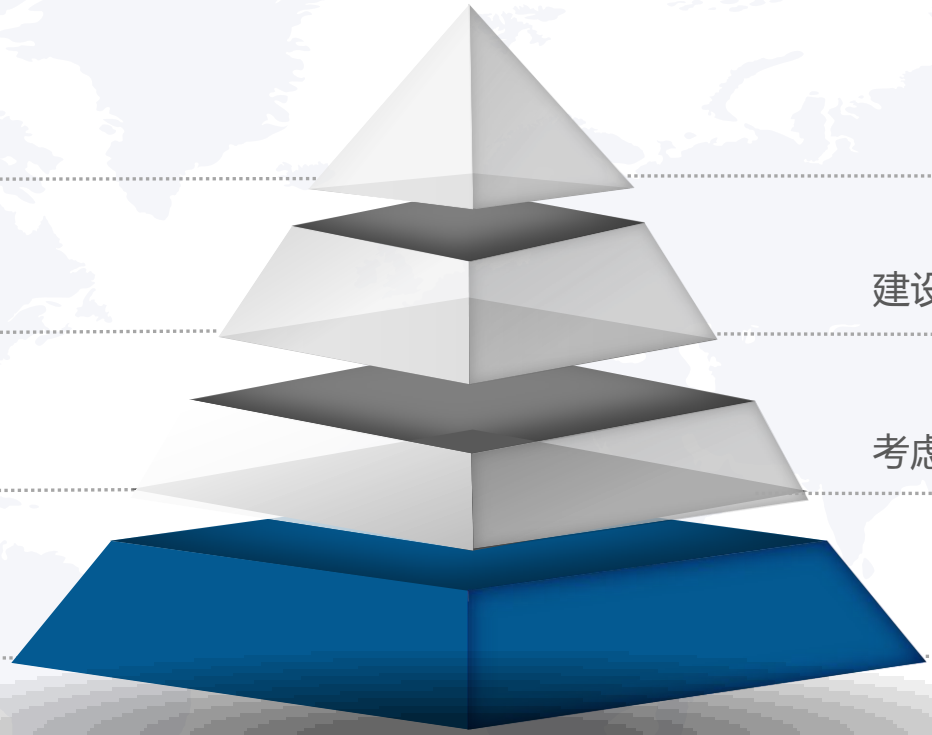
01 | 食物链顶端的大爷们

02 | 叔能忍婶子不能忍

03 | 我们正面对的挑战

04 | 最终的解决方案

1.0 | 架构模型



业务架构 — 实际业务领域，模式

应用架构 — 建设规划应用系统，满足业务需求

数据架构 — 考虑场景中数据存储，读写的方法

技术架构 — 提供上层满足建设需求的资源

中间件是技术架构向上支撑的一块基石。

虽然从宏观架构看，几乎找不到中间件的位置，但实际架构实现的过程中几乎都离不开中间件的支撑。

因为中间件是正是某一类问题的通用技术解决方案

1.1 | 理想中落地的样子

军令如山

>两天完成接入

>一周全部发布

一切掌握之中

>线上运行平稳

>老板喜笑颜开

专心写写代码

>准备下次升级

>构思美好蓝图

1.2 | 可伤心总是难免的

刷脸求接入

大 哥 喝 奶

对方不想你和说话
并向你抛出了一个异常

打脸发布后

大哥给个机会

1.3 | 业务上了高速路

2014 蛮荒时期

业务刚起步，作坊式开发

- 20+ 团队
- 15 - 20 应用
- 20 - 30 节点

2015 开天辟地

业务爆发，团队迅速扩大

- 80+ 团队
- 30-50 应用
- 60 - 80 节点

2016 高歌猛进

业务与团队持续爆发增长

- 200+ 团队
- 300+ 应用
- 600+ 节点

2017 持续发展

异地研发部增多

- 300+ 团队
- 500+ 应用
- 1500+ 节点

未来 拓疆扩土

业务稳定，进军新领域

- ? 团队
- ? 应用
- ? 节点

1.4 | 可高速不给换轮胎

1. 爸爸是技术负责人或老板
2. 组织上以行政命令的方式推进
3. 微服务 —— 部署各种“服务”到机器上

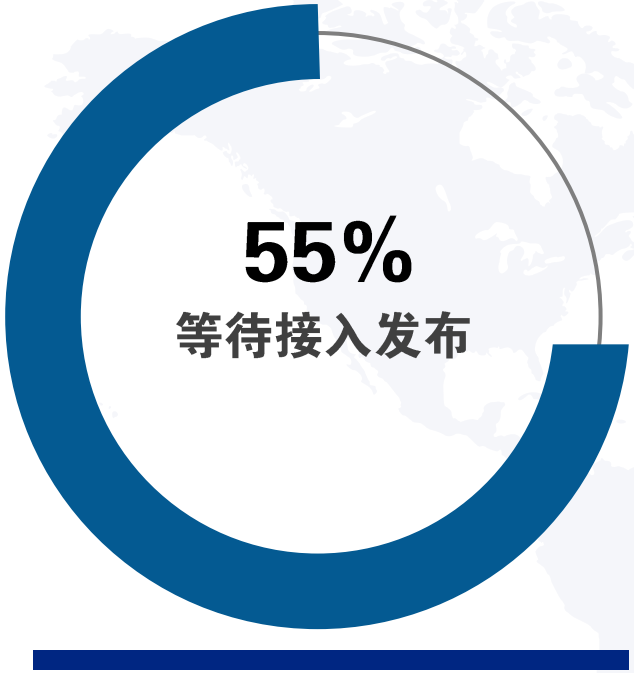
1.5 | 想想我们的架构蓝图

蓝不蓝，难不难你说

如果无法落地，架构什么都是扯的。

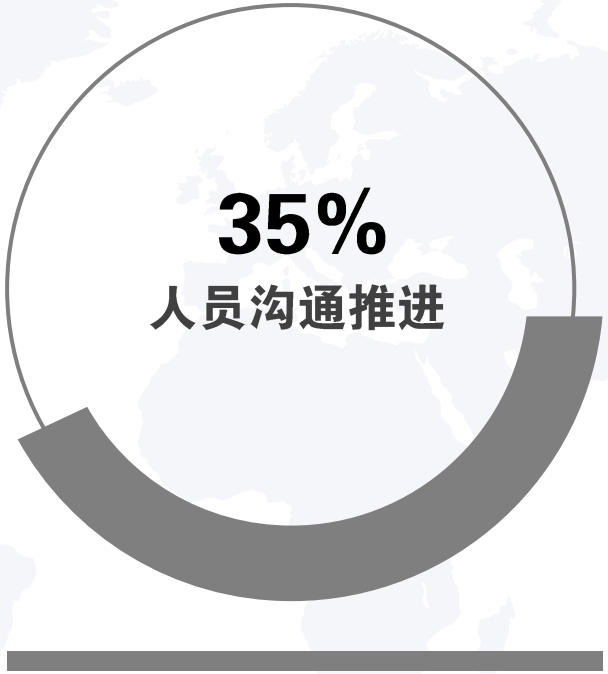
现有的流程都是依赖着应用的发布来完成架构落地，可是他们晃点我，我也很绝望啊~

2.0 | 寻找领域内的潜在问题



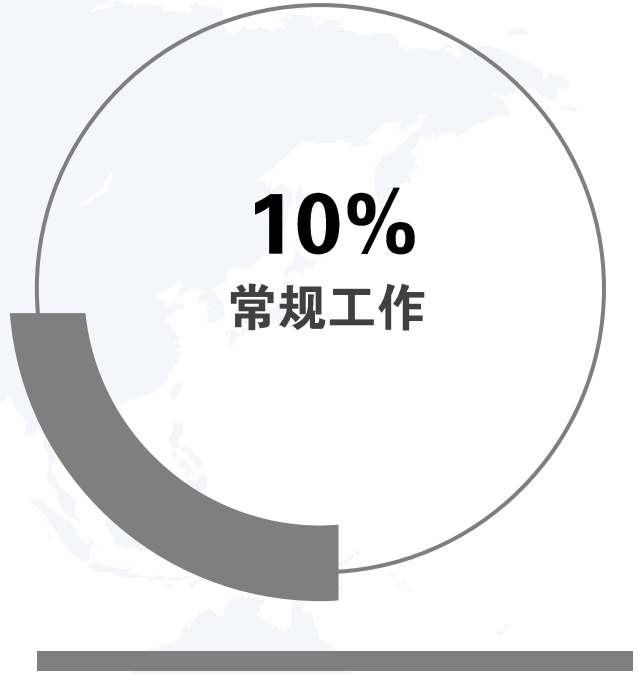
每个应用的发布时间完全不可控

少则两周，多达一月，也有稳定无发布的



沟通困难，而且沟通层次深

从经理->组长->开发，甚至还有测试



关注运行情况、收集运行数据

依据数据继续迭代改进

铺路爪 Pavepaws

无感升级
自动化部署，动态加载

三方依赖隔离
多版本依赖共存无冲突

运行时可控
随时升降级，BUG热修复

2.1 | 铺路爪核心组件

Java Runtime

Instrumentation
利用Java本身的一些特性来
完成类的替换，监听

Class Loader

类加载的必经之路

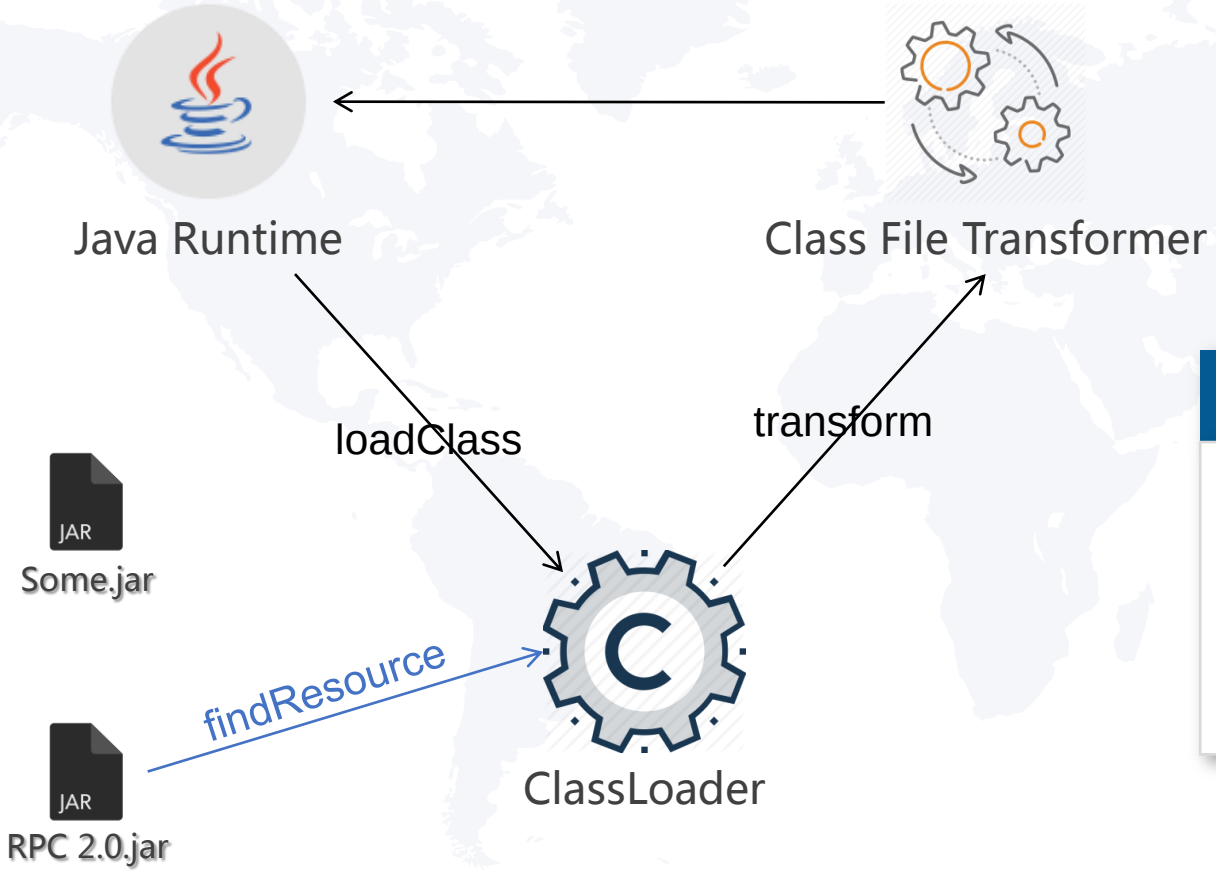
ClassTransformer

字节码转换器
用于在类在被实际Runtime解析
前对字段，方法等进行编辑修改

Shadow Loader

影子加载器
挂钩ClassLoader
通过拦截loadClass等方法
实现自由的类加载逻辑

2.2 | 后庭别院菊花香

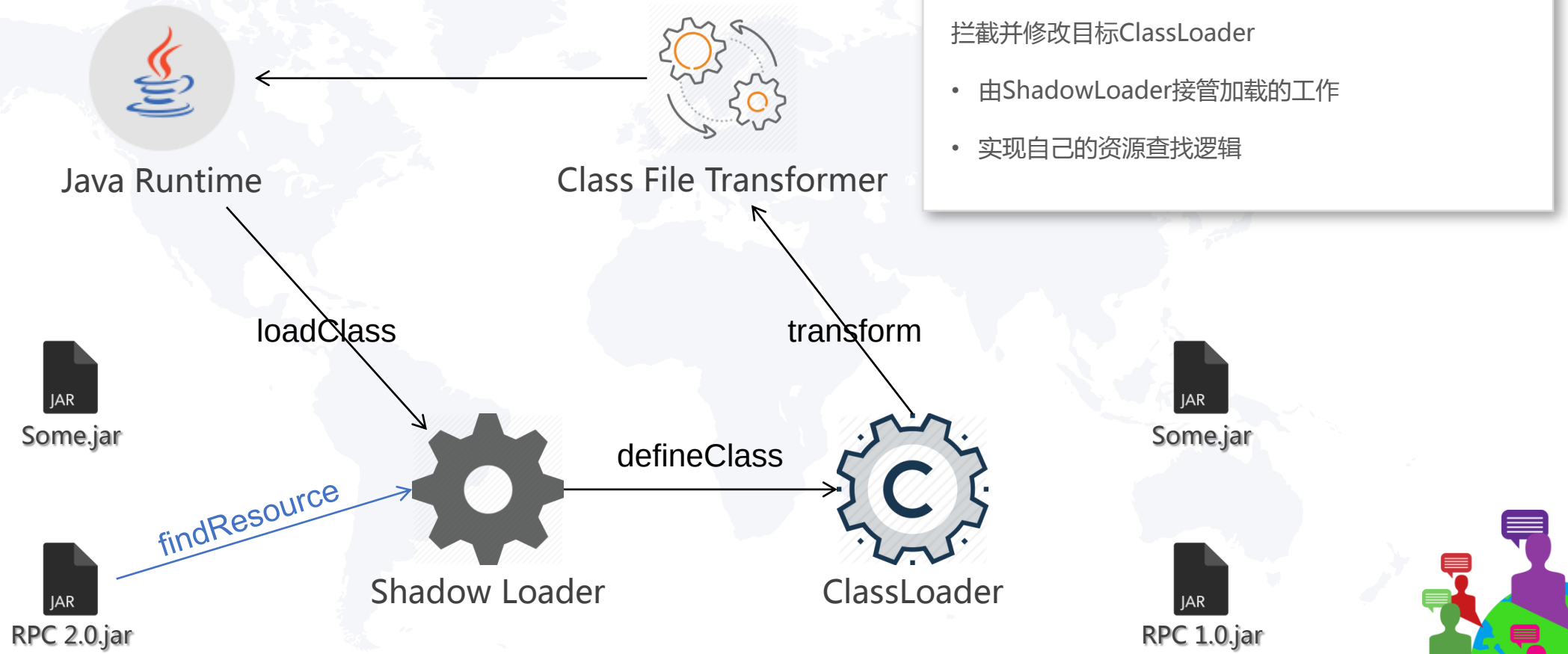


AOP

通过ClassFileTransformer进行基于字节码的方法切面

- 几乎没有性能损失
- 对任意非原生类任意方法操作

2.3 | 偷梁不换柱，移花能接木

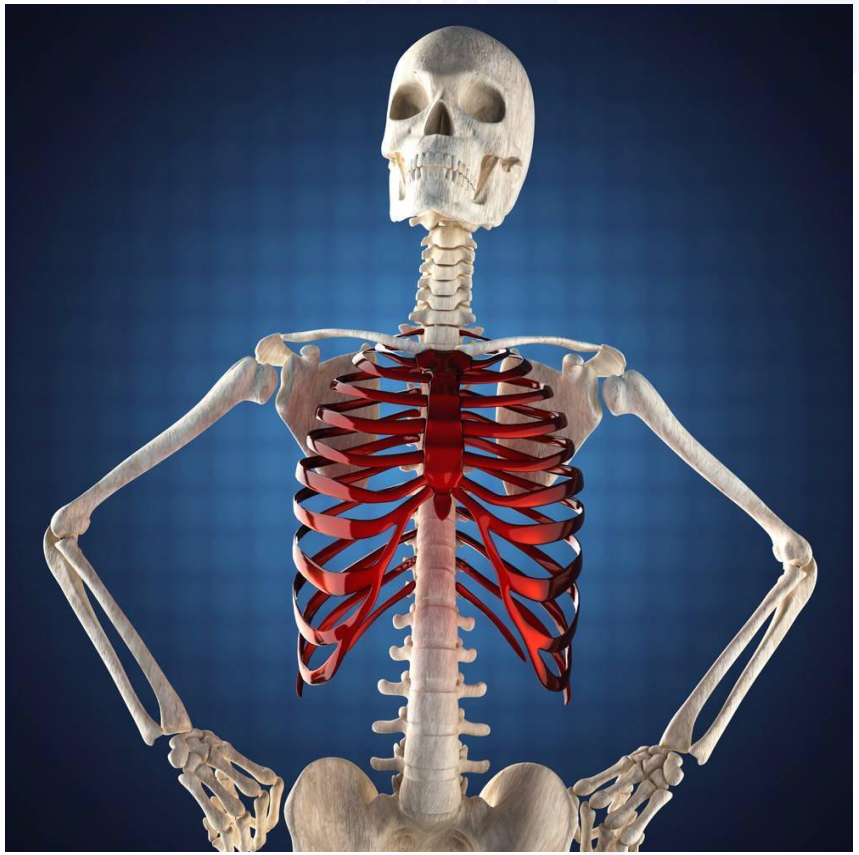


HOOK

拦截并修改目标ClassLoader

- 由ShadowLoader接管加载的工作
- 实现自己的资源查找逻辑

2.4 | 遵守基本的设计原则



I'm Interface !

实现的还可以是不是

2.5 | 运行时为实现独立容器加载

```
public class LogFactory {
    private static final boolean isLogClientPresent;

    static {
        isLogClientPresent = ClassUtils.isPresent(contair
    }

    public static Logger create(String name) {
        if (isLogClientPresent) {
            Logger instance = new GangLogAdapter(name);
            instance.init();
            return instance;
        }
        return new Log4jAdapter(name);
    }
}
```

```
public static Logger create(String name) {

    Isolation container = Isolation.newContainer();
    if (isLogClientPresent) {
        Logger instance = container.single("cloud.appgov.pavepaws.impl.logger.GangLogAdapter", name);
        instance.init();
        return instance;
    }
    return container.single("cloud.appgov.pavepaws.impl.logger.Log4jAdapter", name);
}
```

伪代码

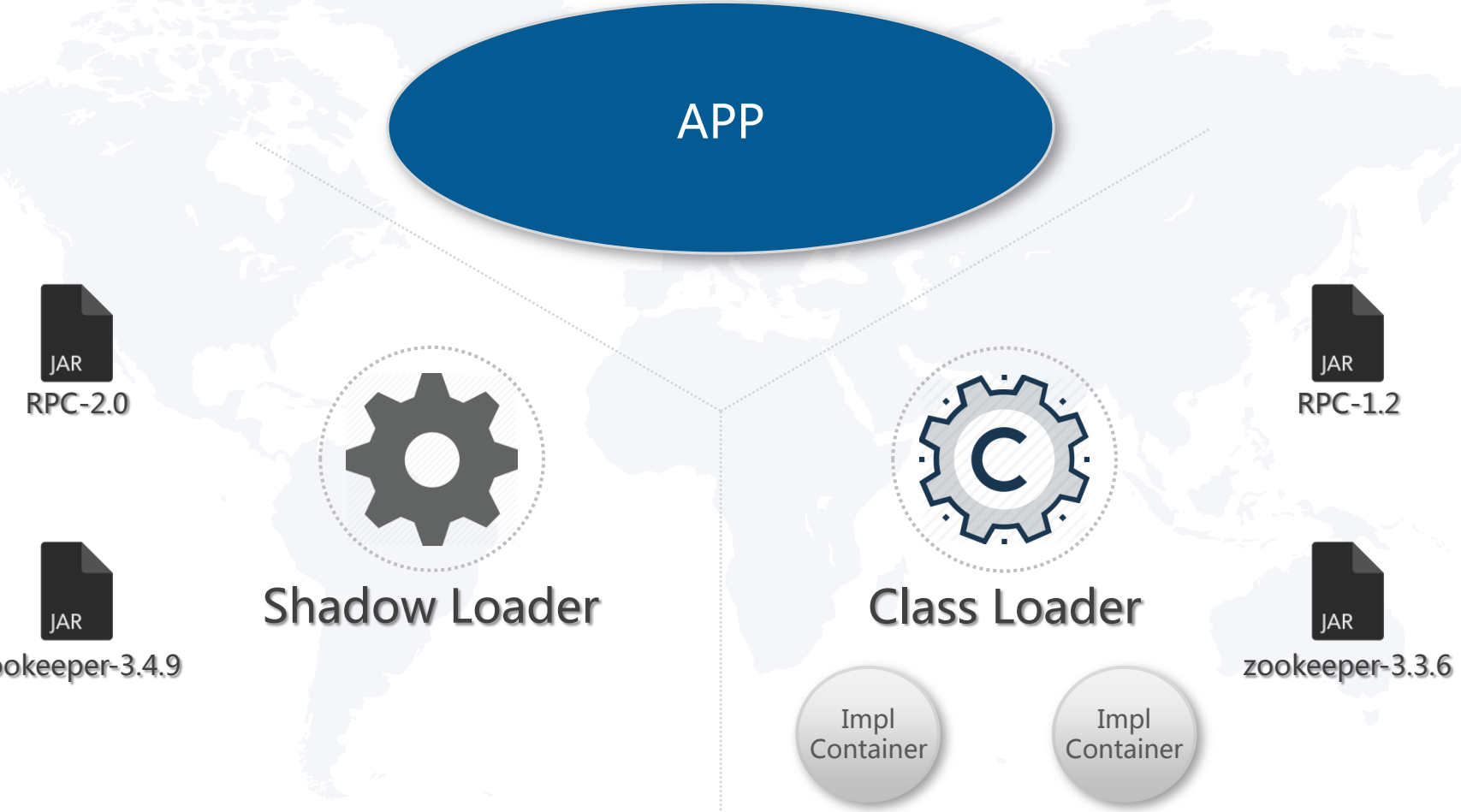
IsolationTransformer.class

```
byte[] transform(String classname,...,byte[] bytecodes){
    ClassEditor editor = new AsmClassEditor(bytecodes);
    for(method in editor.Methods){
        scanAndReplaceInit(method);
    }
}
```

scanAndReplaceInit

在类加载时检查类是否有依赖的服务,将原具体实现类的构建转给负责“隔离”的容器。容器会在单独的ClassLoader中加载，之后创建对应的实例返回。

2.6 | 一山容二虎，一公和一母



2.7 | 进程内的微服务

APP

应用自身

Service Interface　Service Interface

服务接口

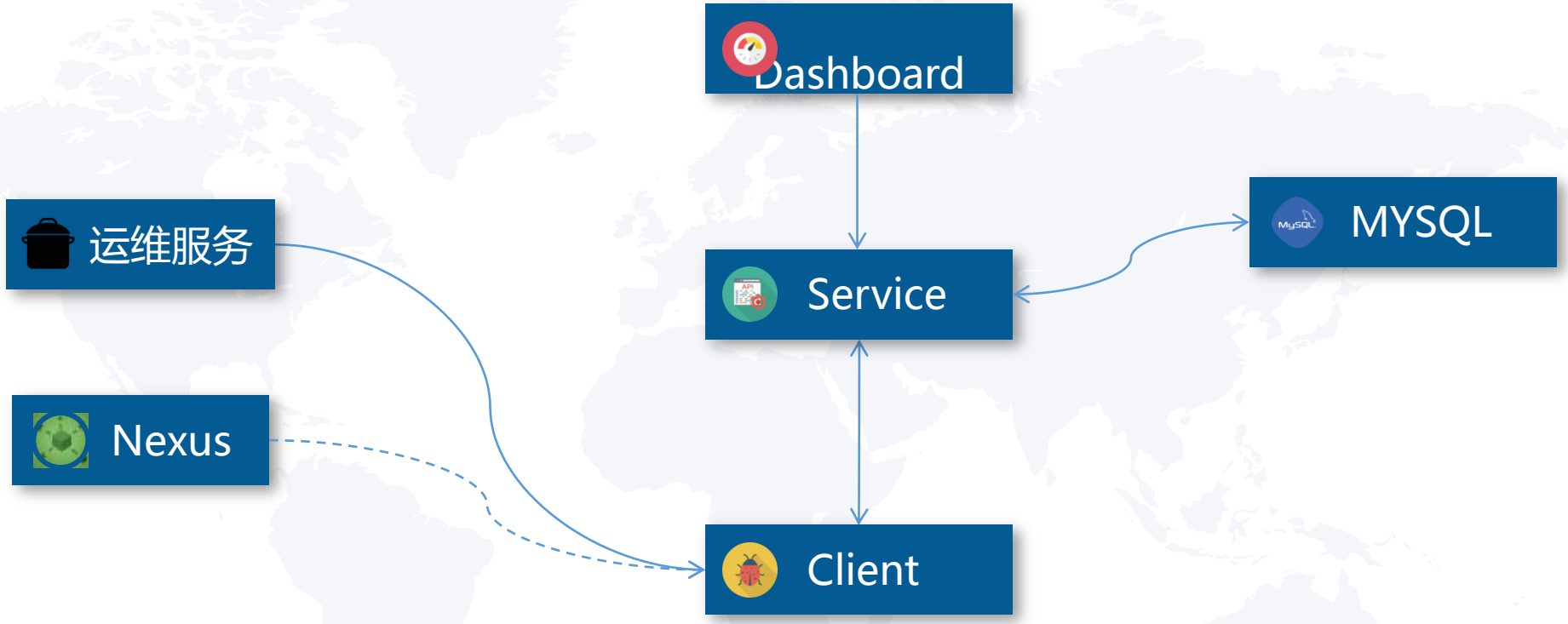
应用只关心和使用接口，不关心背后实现

Service Impl　Service Impl　Service Impl　Service Impl

服务实现

实现可以有多个，单例，在需要时可以切换实现

2.8 | 没啥亮点的前后端



整体结构

非常简单的结构，Service定下规则，通过运维服务推送文件到目标机器。再由Client按照规则执行。

2.9 | 配套实施流程，形成闭合

QA

与QA制定中间件升级流程，将中间件升级发布与业务应用升级发布隔离开

研发

有了QA流程后，研发这一块主要在升级前做到沟通，协调时间避免与大项目冲突

运维

做好使用培训，用于在突发风险时，即时关停或组件降级

流控系统

做好流控为了在风险发生时，能够即时通过调整策略进行止损

3.1 | 大路朝天，各走一边

业务系统 → SIT → UAT

独立发布

中间件升级更新不再受制于业务系统的发布周期

快速部署

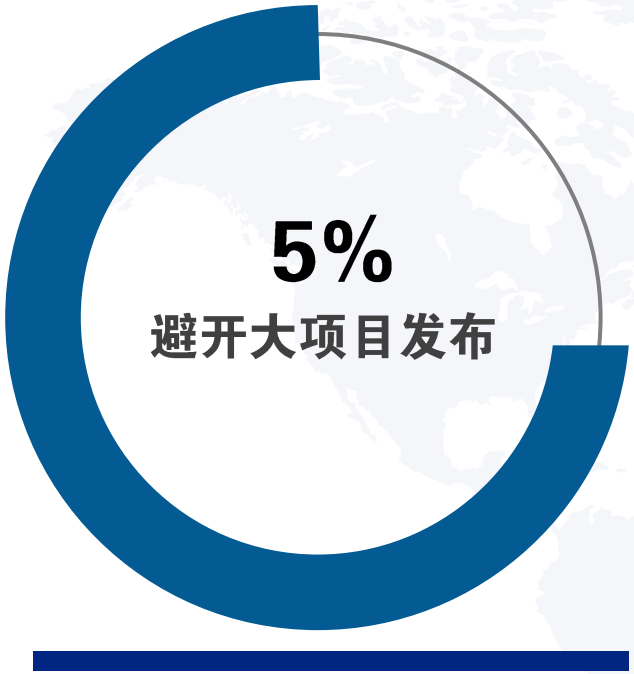
任何环境都可以批量进行组件升级更新操作，而研发是无感知的

容易验证

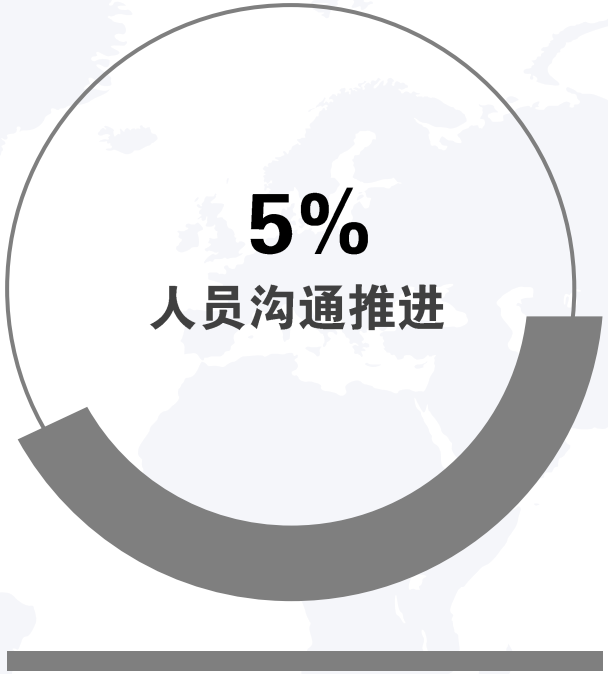
众多的应用系统能够提供更全更多的运行测试，发现未知问题

中间件 → SIT → UAT

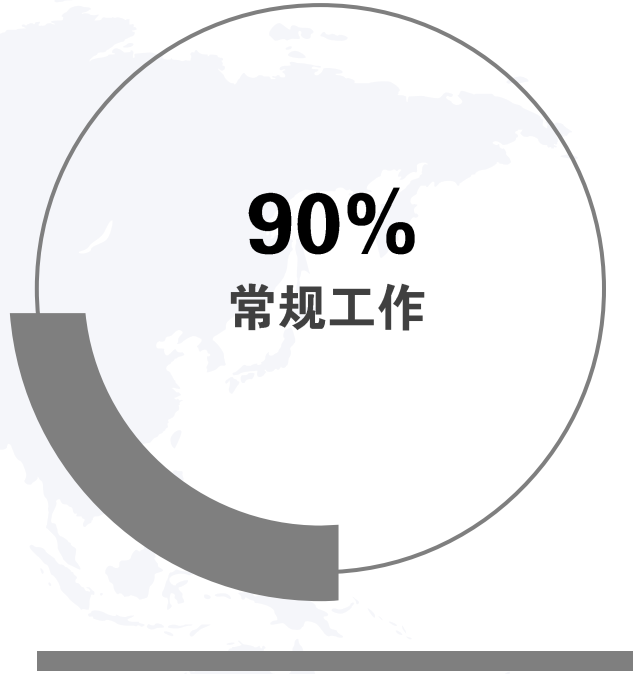
3.2 | 不再需要“脸”的时代



大项目上线时，人员都比较紧张，不合适操作。万一发生问题，也容易影响研发人员判断排查。



不在需要重复解释各种问题，只要和业务线经理沟通确认好时间，最后统一邮件告知即可。



关注运行情况，收集数据，协查疑似问题
以数据为依据，进行迭代，调整设计，重新部署

3.3 | 买一赠一，附加价值

（1）三方包数据：应用加载的所有JAR都会铺路爪被上报，用于后续分析。

（2）统一版本：统一应用使用的三方依赖版本，规范研发使用。进一步降低版本爆炸时带来的依赖冲突和其它问题。

（3）发现安全隐患：例如知名的struts2 漏洞，通过检索依赖包的数据集合，能够快速发现定位到具体应用，立即进行升级改进。