系统精简之道：如何以极低风险，高效清理线上无用代码

去哪儿旅行　基础架构部/资深研发工程师　马阳阳

目录

01 背景介绍

为什么去哪儿网会启动系统瘦身项目？

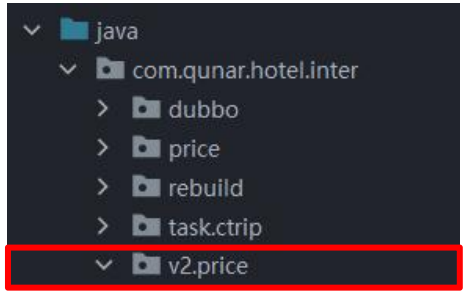
系统瘦身背景

业务历史悠久

短周期业务多

复杂度单调递增

人员流动频繁



系统瘦身背景 & 痛点

人均应用：5个

人均代码：10w

一年内无变更应用：20%

线上方法行覆盖率：40%

维护成本高

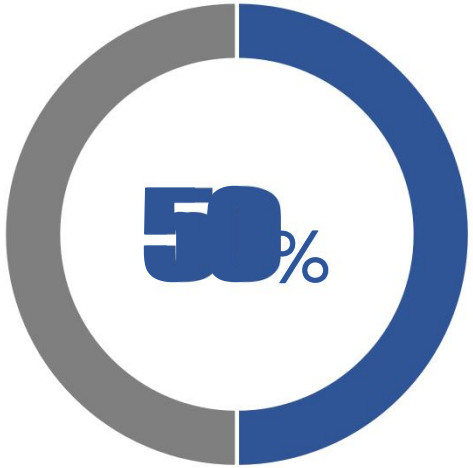
估时误差大

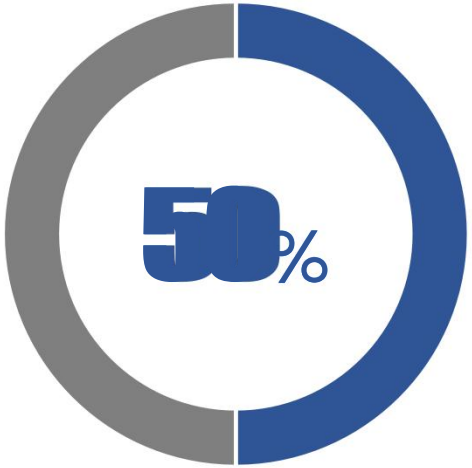
开发进度慢

效率下降

系统瘦身目标 & 挑战

代码精简





服务精简

目标高

范围广

无参考

系统瘦身时间及组织架构规划

系统瘦身两步走

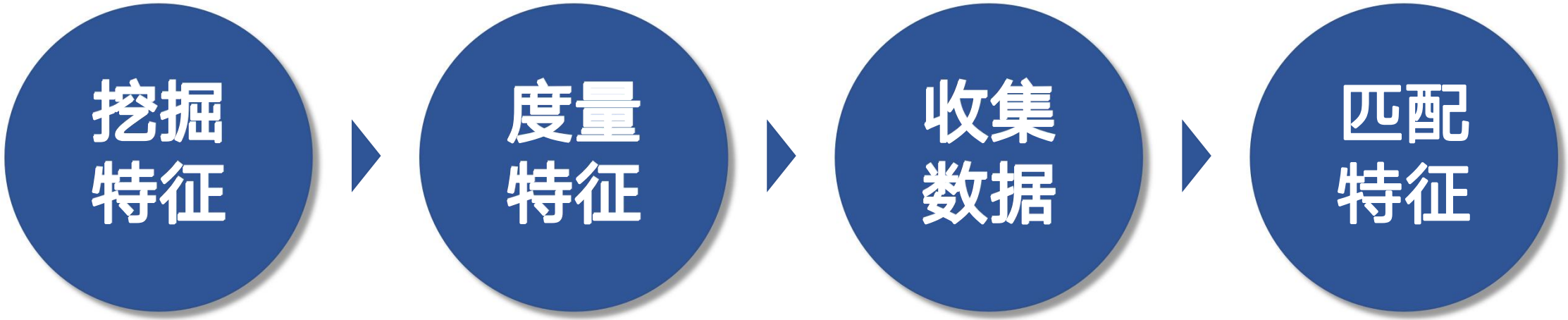
找得到

能筛选出目标对象

删得好

准、全、快

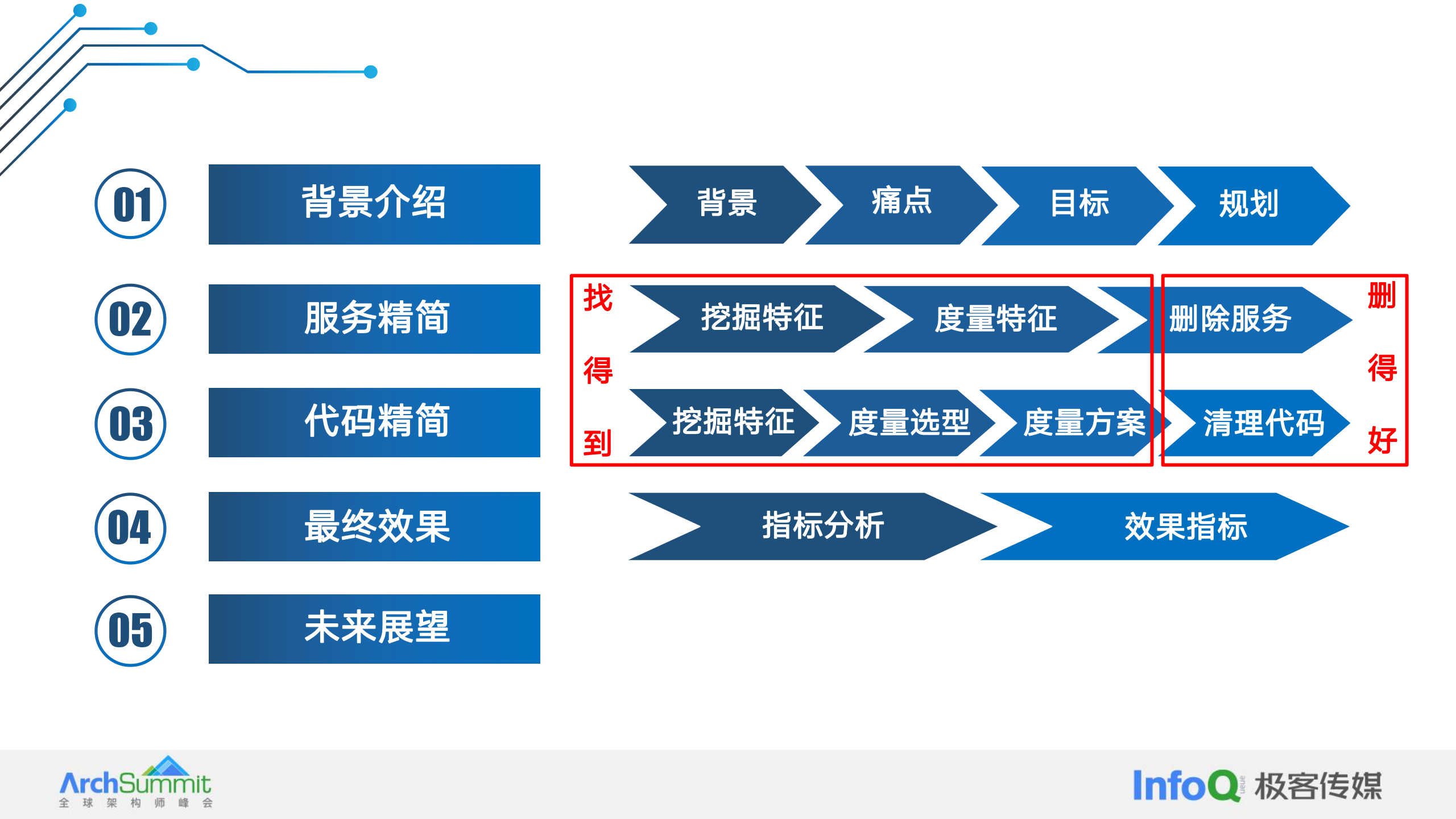
筛选模型（方法论）



从具体入手

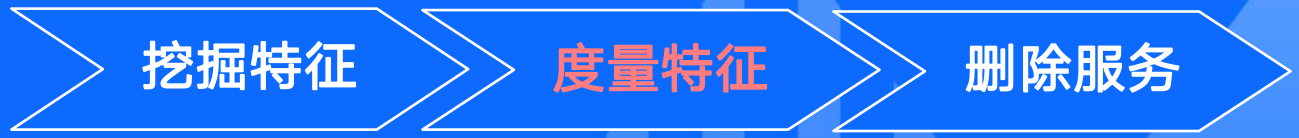
明确如何度量

自动化


01
背景介绍
背景
痛点
目标
规划
02
服务精简
找得到
挖掘特征
度量特征
删除服务
删得好
03
代码精简
挖掘特征
度量选型
度量方案
清理代码
04
最终效果
指标分析
效果指标
05
未来展望

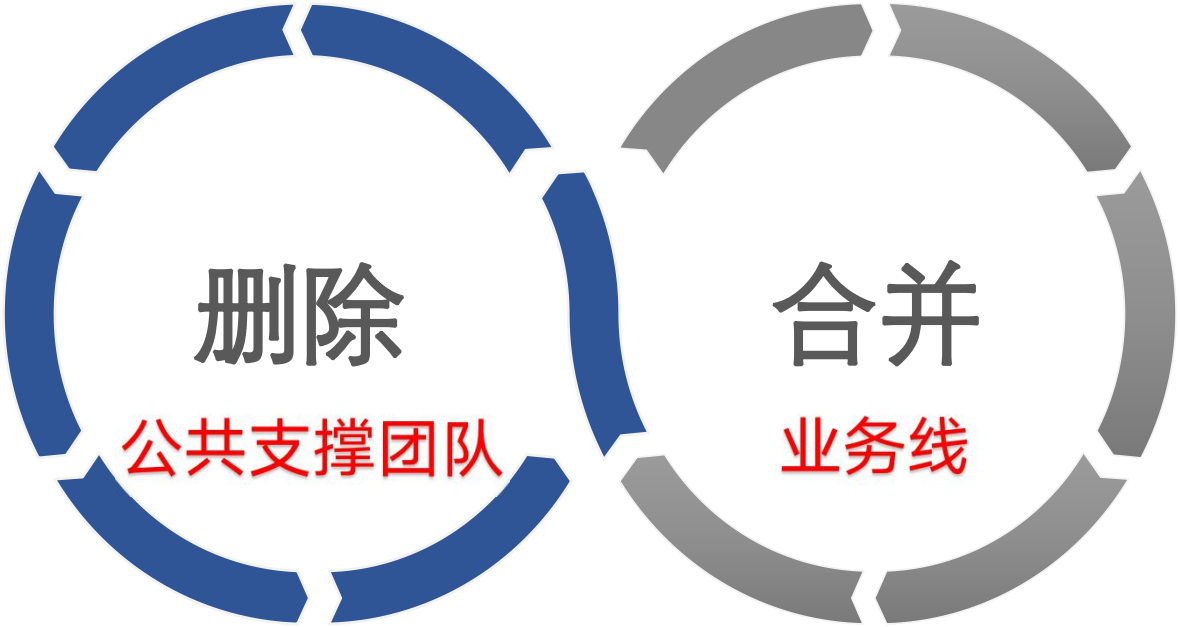
02 服务精简

如何自动化、低风险地精简服务？



可精简服务特征分析

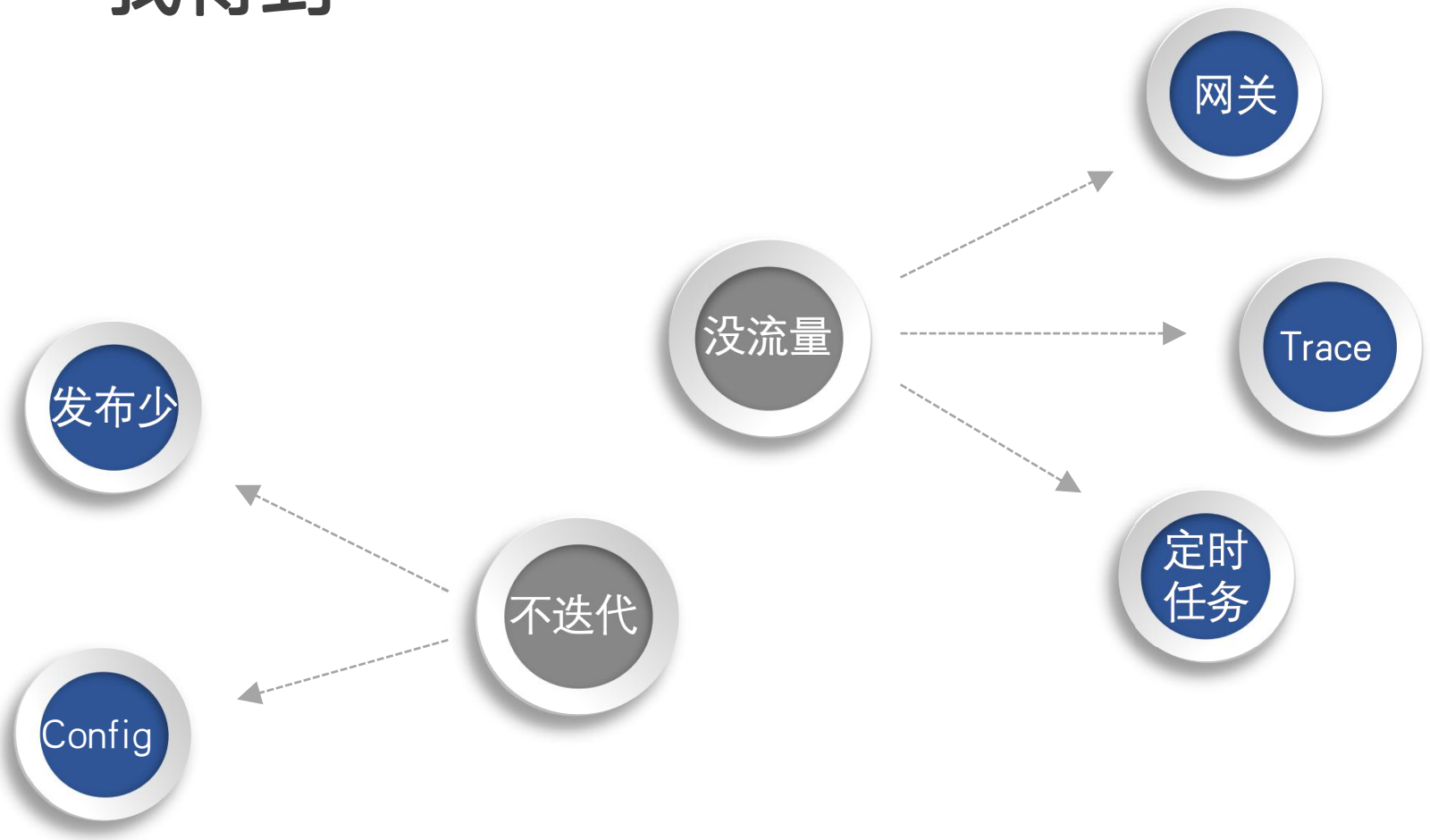
低价值：
- 没流量
- 不迭代

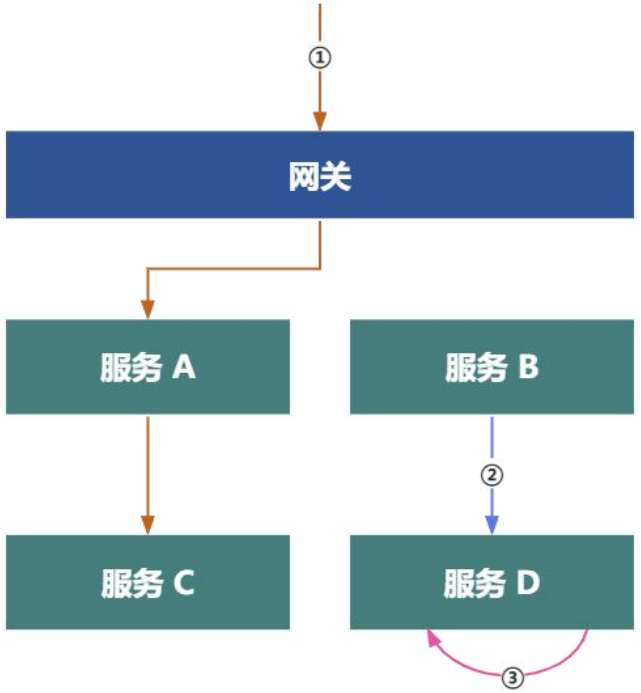


业务维度：
- 业务域
- 业务流程
- 重要性

质量维度：
- 性能
- 稳定性
- 可用性
- 安全边界
- 异构

“找得到”



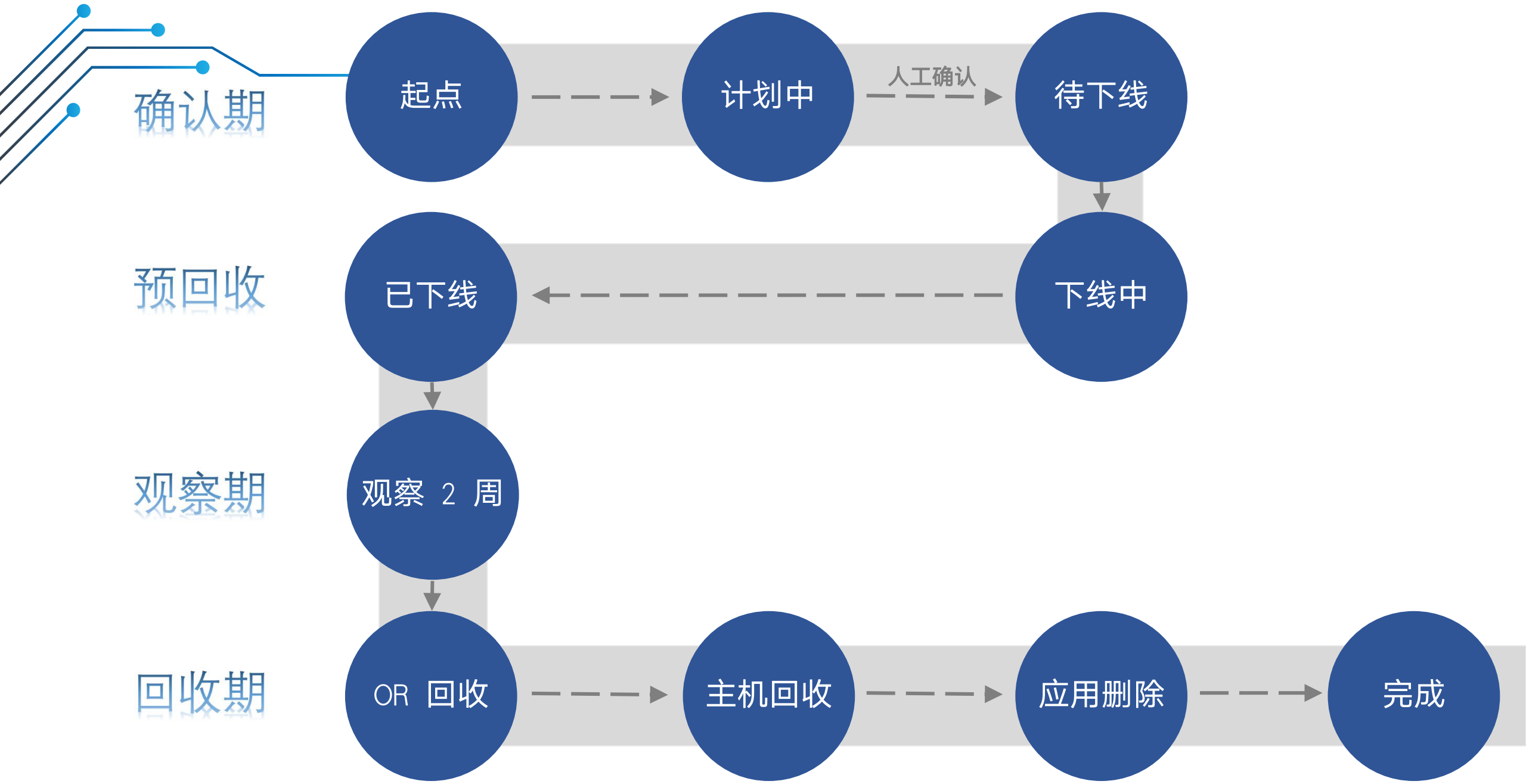


“删得好” – 准全快

标准化

平台化

自动化


确认期
起点
计划中
人工确认
待下线
预回收
已下线
下线中
观察期
观察 2 周
回收期
OR 回收
主机回收
应用删除
完成

03 代码精简

如何精准找到线上无流量的方法？

挖掘特征 → 度量选型 → 度量方案 → 清理代码

可精简代码特征分析

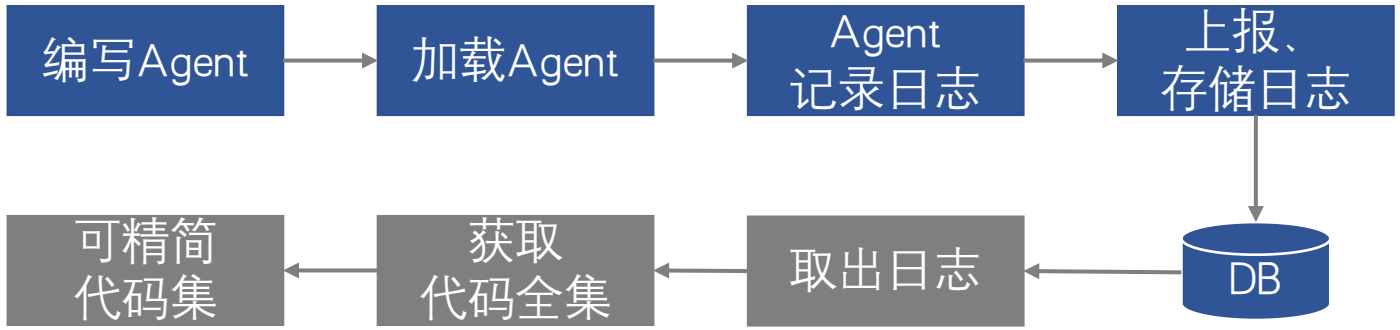
	量大（效果）	通用性（效率）	
未被引用的方法（静态）	✗	✓	
没有流量的方法（运行时）	✓	✓	公共支撑团队
重构	✓	✗	业务线

“找得到”方案选型

方案一：AOP

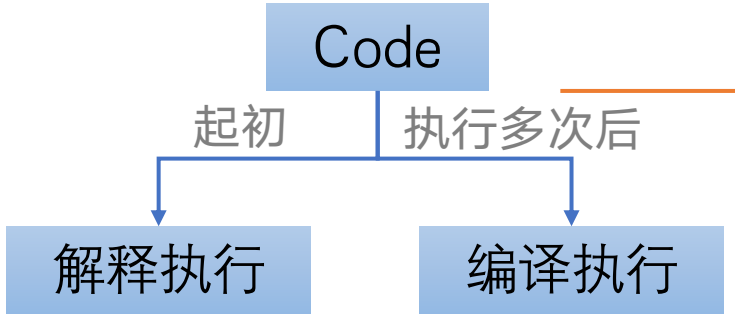
```java
@Aspect
@Component
public class LoggingAspect {

    private static final Logger logger = LoggerFactory.getLogger(LoggingAspect.class);

    @Before("execution(* com.qunar.noah.kvm.*.*(..))")
    public void logBefore(JoinPoint joinPoint) {
        logger.info("Entering " + joinPoint.getSignature().getName() + " method");
    }

}
```

方案二：Agent 字节码插桩



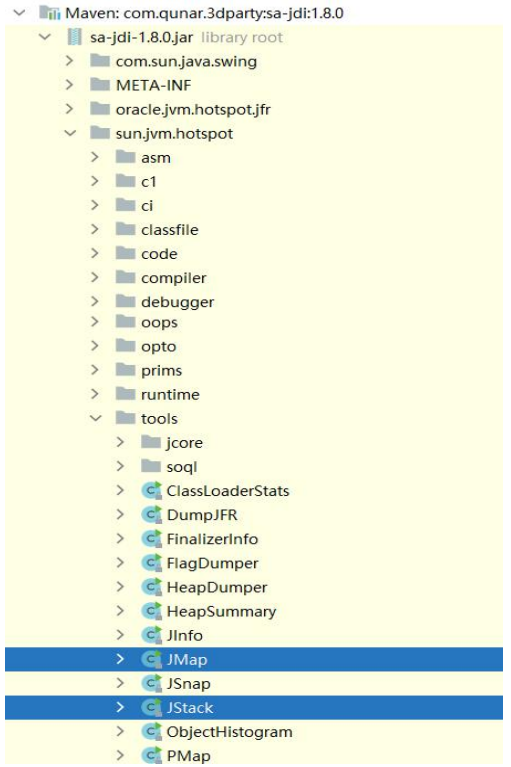
“找得到”方案选型

方案三：基于 SA 工具



1、猜测：有方法执行次数的计数器

```
// A Method* represents a Java method.
//
// |------------------------------------------------------|
// | header                                               |
// | klass                                                |
// |------------------------------------------------------|
// | ConstMethod*                 (oop)                   |
// |------------------------------------------------------|
// | methodData                   (oop)                   |
// | methodCounters                                       |
```



2、有办法用 java 代码读到方法计数值吗？

Serviceability Agent

The Serviceability Agent(SA). The Serviceability Agent is a Sun private component in the HotSpot repository that was developed by HotSpot engineers to assist in debugging HotSpot. They then realized that SA could be used to craft serviceability tools for end users since it can expose Java objects as well as HotSpot data structures both in running processes and in core files.

“找得到”方案选型

方案三：基于 SA 工具

跑数：探测 JVM 中方法计数，并保存结果的过程

- STW
- 消耗内存

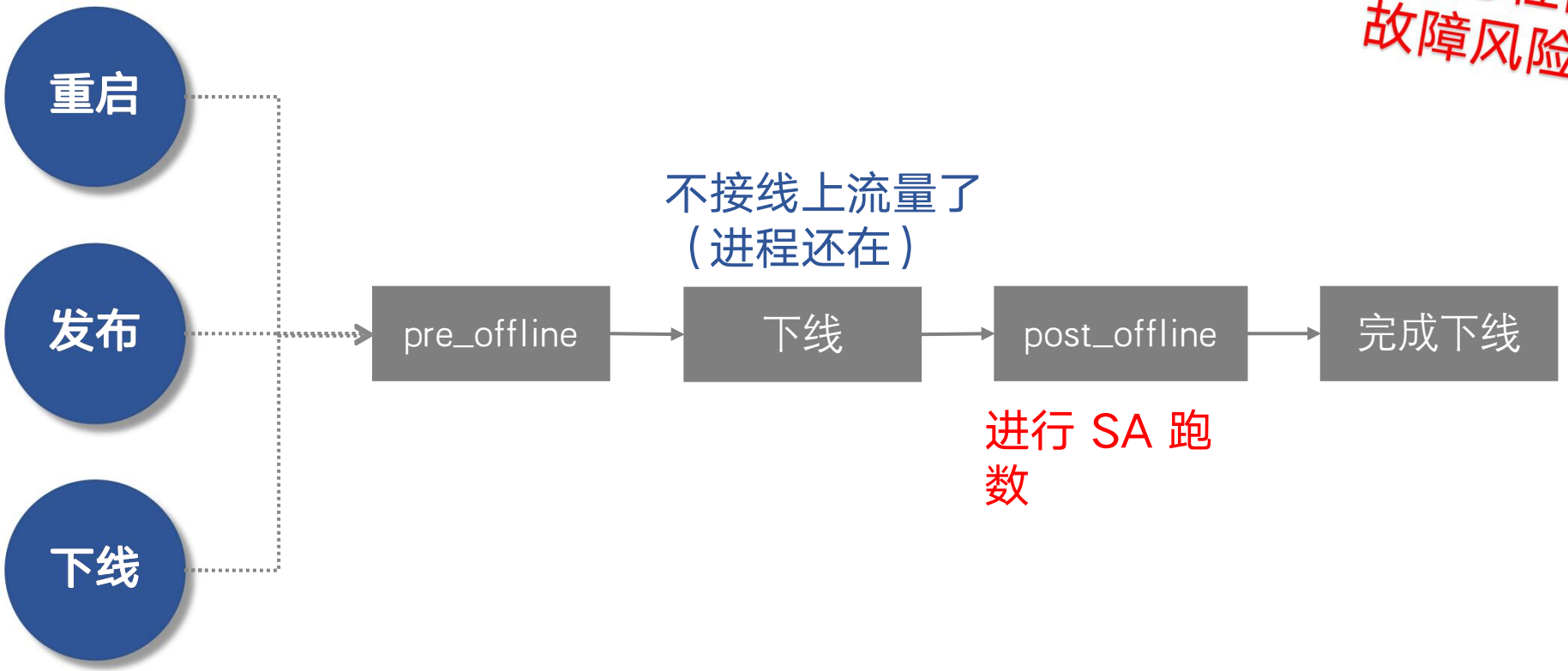
“找得到”方案选型

方案选型

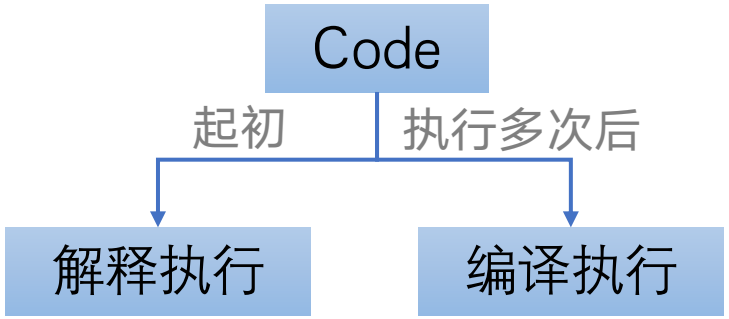
方案	性能损耗	故障风险	实现复杂度
AOP 拦截器	高	低	高
Agent 字节码插桩	低	中	高
基于 SA 工具	无	无	低

SA 方案详细设计 – 性能无损跑数

对业务性能无影响
故障风险为零！



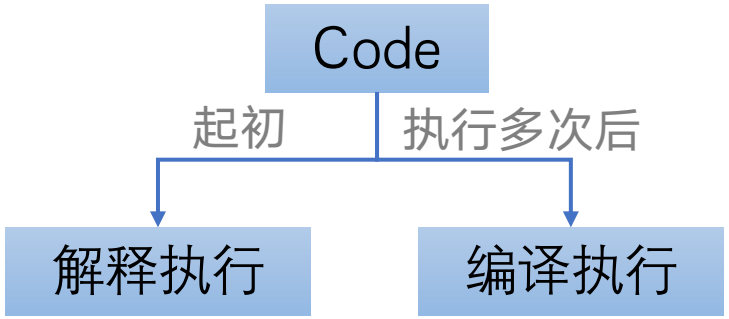
SA 方案详细设计 – 跑数代码实现




```
private static void scanAllClasses(Set<MethodDefinition> result) {
    // 扫描所有未被编译的类
    VM.getVM().getSystemDictionary().allClassesDo(new InvocationCounterVisitor(result));

    // 扫描所有已被编译成 codeBlob 的类
    VM.getVM().getCodeCache().iterate(new CompiledMethodVisitor(result));
}
```

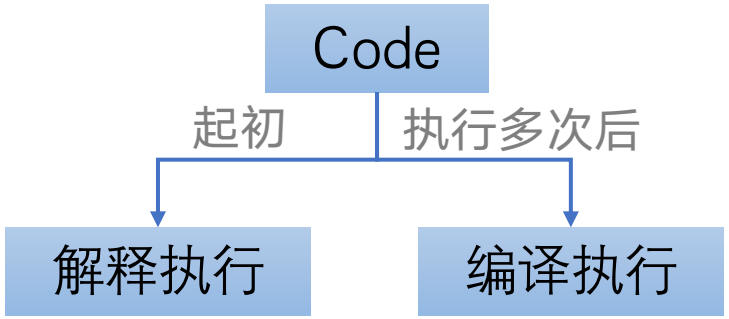
SA 方案详细设计 – 解释执行方

```
public class InvocationCounterVisitor implements SystemDictionary.ClassVisitor
                                                                    ①
    private final Set<MethodDefinition> result;

    public InvocationCounterVisitor(Set<MethodDefinition> result) {
        this.result = result;
    }

    @Override
    public void visit(Klass klass) {
        if (klass instanceof InstanceKlass) {
        ② final MethodArray methods = ((InstanceKlass) klass).getMethods();
            for (int i = 0; i < methods.length(); i++) {
                final Method method = methods.at(i);
            ③ if (method.getInvocationCount() > 0) {
                    result.add(convert(method));
                }
            }
        }
    }
```



SA 方案详细设计 – 编译执行方法

```
public class CompiledMethodVisitor implements CodeCacheVisitor {
                                                              ①
    private final Set<MethodDefinition> result;

    public CompiledMethodVisitor(Set<MethodDefinition> result) {
        this.result = result;
    }

    @Override
    public void visit(CodeBlob codeBlob) {
        if (codeBlob == null) {
            return;
        }
    ② final Method method = codeBlob.asNMethodOrNull().getMethod();
        if (method == null || method.isNative()) {
            return;
        }
    ③ result.add(convert(method));
    }
```



SA 方案详细设计 – 计算可精简方法集

聚合，取并集

有流量方法集

取

差

集

可精简方法集

工程方法全集

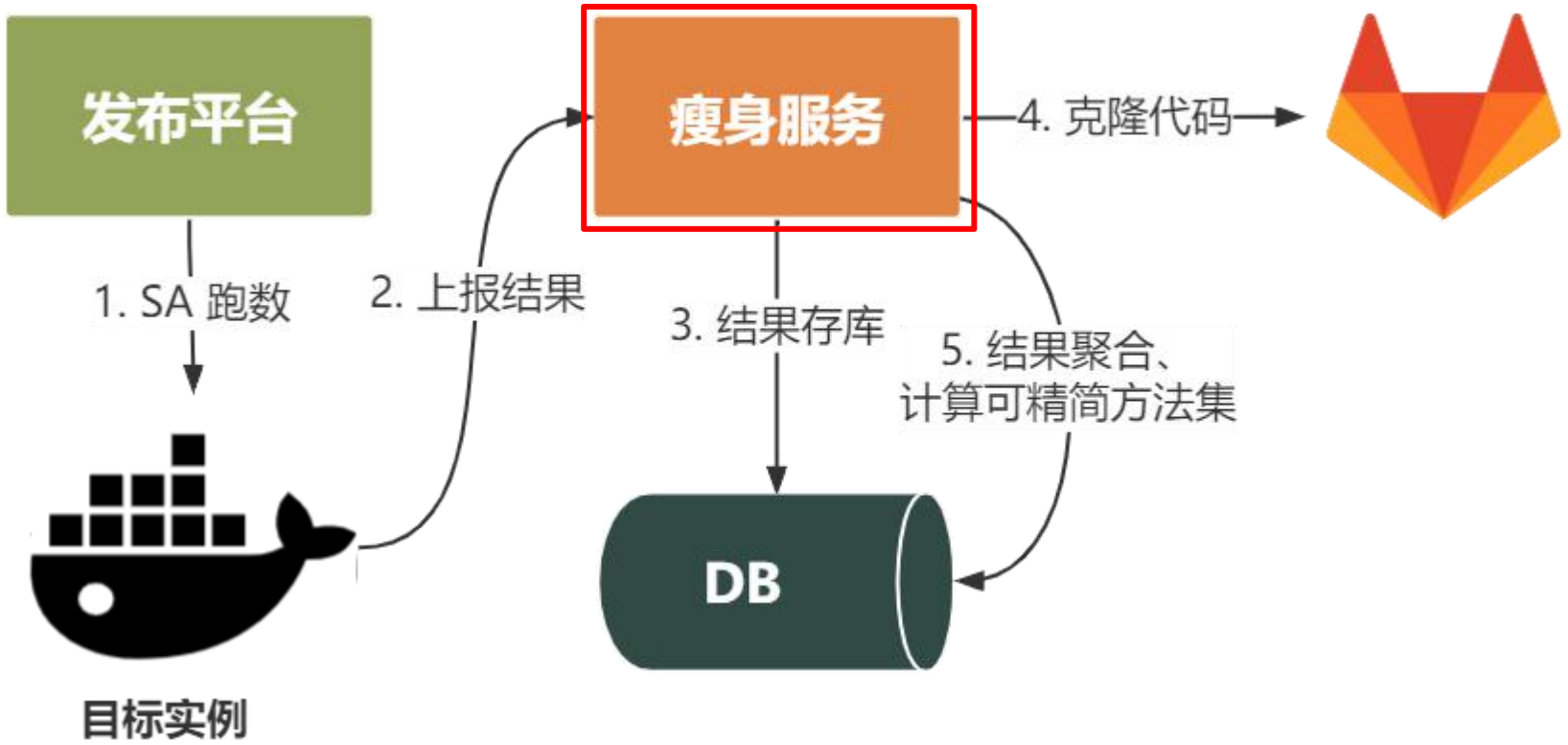
GitLab

通过 Spoon

Spoon是一个基于Java语言的代码转换和分析工具，它可以将Java程序的源代码转换成抽象语法树（AST），并支持对AST进行修改和分析。

Spoon可以帮助我们进行各种代码转换和重构操作，例如修改Java程序的语法结构、添加或删除代码、提取代码片段等等。Spoon还支持多种输出格式，包括Java源代码、JAR文件、XML文件等等。

SA 方案业务流程图

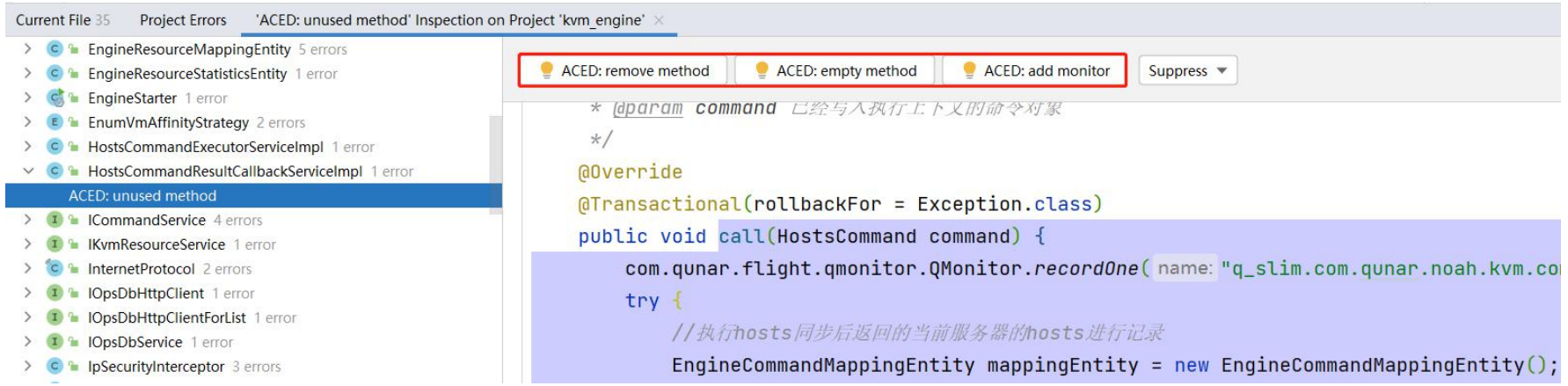


“删得好”- 多种手段

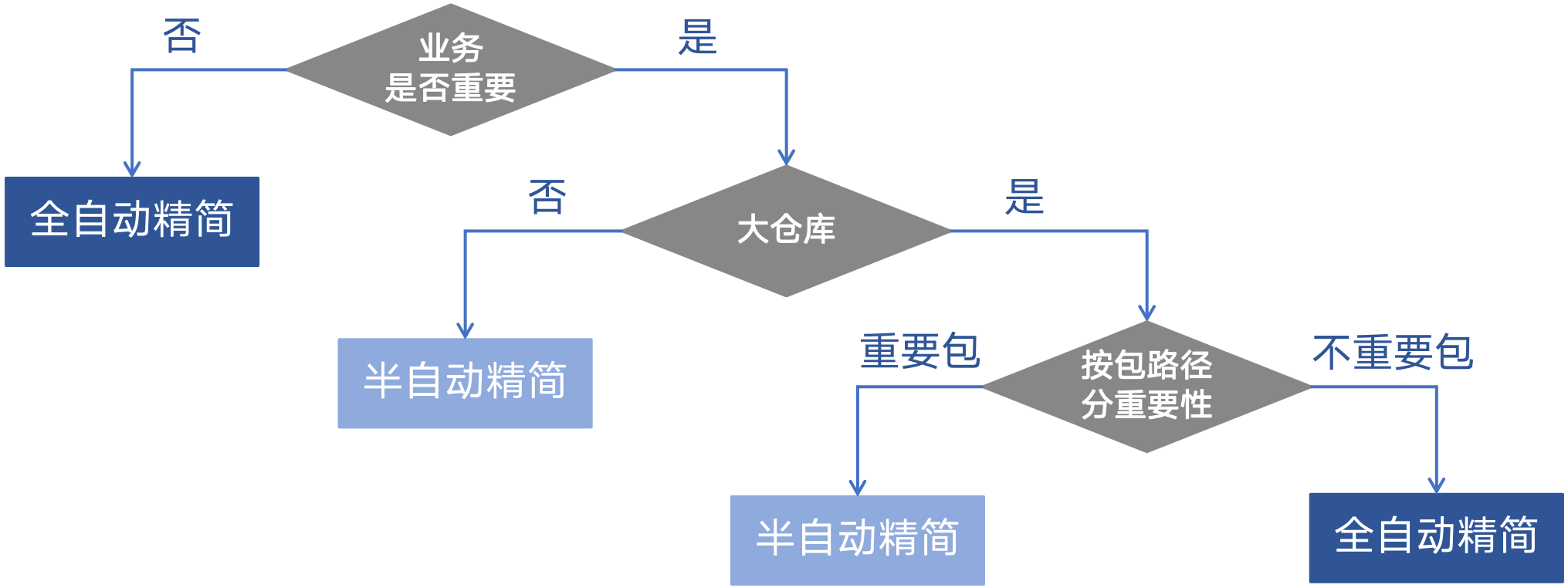
全自动手段

注意代码可编译

半自动手段



“删得好”- 最佳实践



验证“删得好”

和业务上线流程一致

04 最终效果

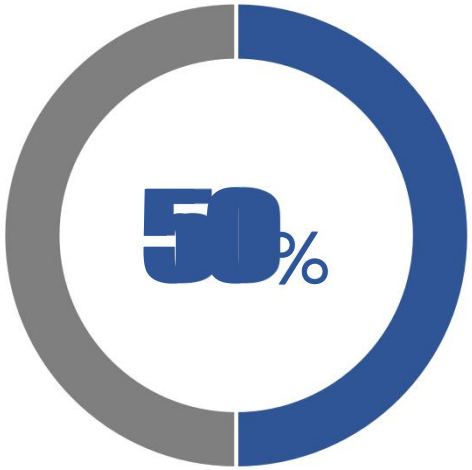
代码少了多少？带来了哪些收益？

精简成果 → 指标分析 → 效果指标

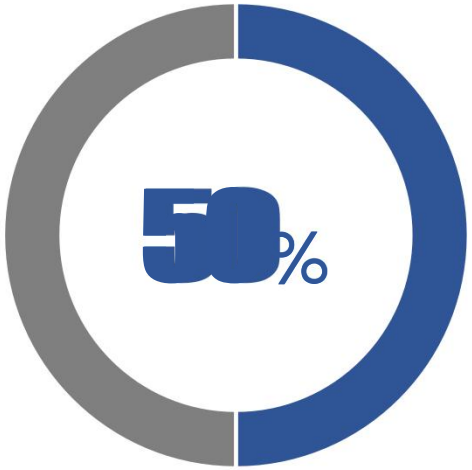
精简成果

目标回顾

代码精简



服务精简



达成情况

代码减少 50%
（数千万行）

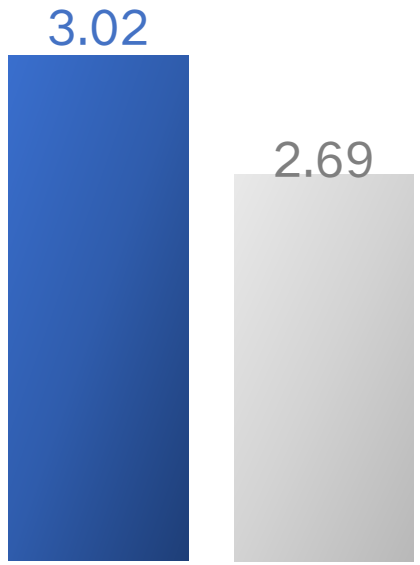
服务减少 26%

无 P1、P2 故障

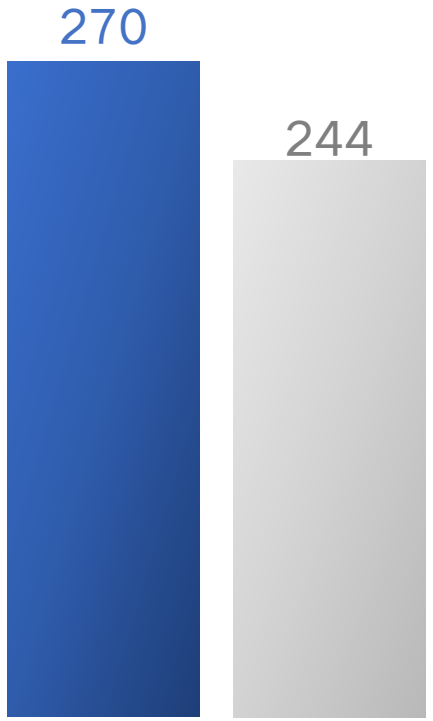
效果指标分析

开发域	测试域	交付域（CI/CD）	运维域
新人：熟悉成本	环境构建/更新时长	编译时长	告警量
需求估时		发布耗时	
需求耗时			

最终效果



需求耗时（天）



发布耗时（秒）

[illegible] %

[illegible] %

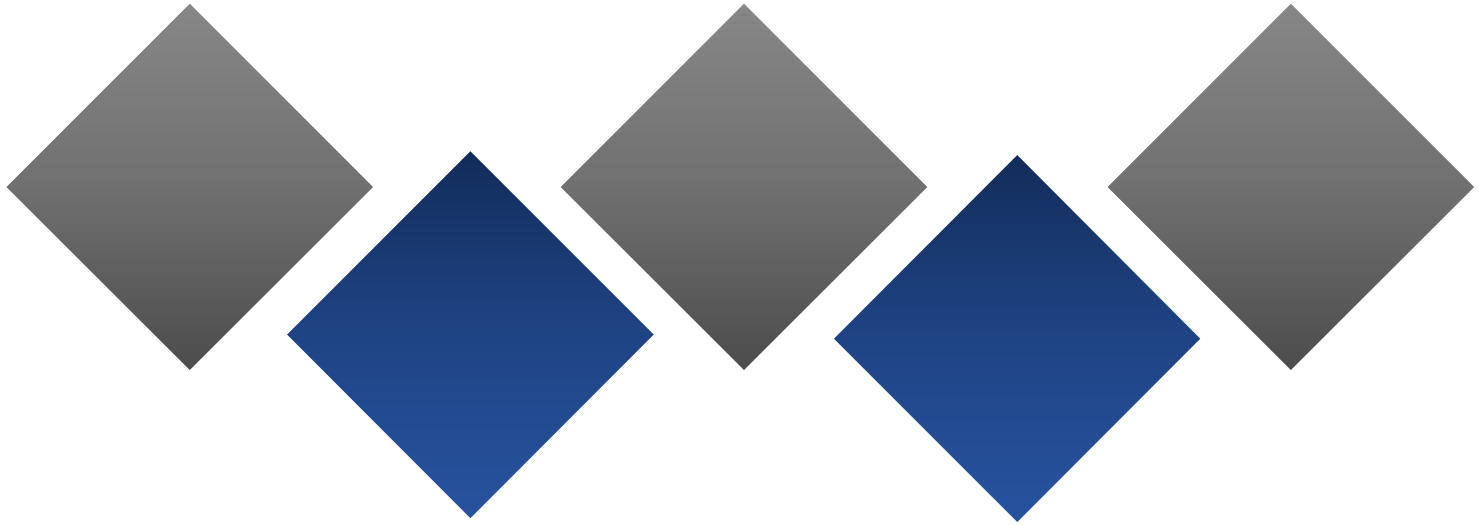
9

05 展望未来

之后还有哪些方向和技术值得探索？

未来展望

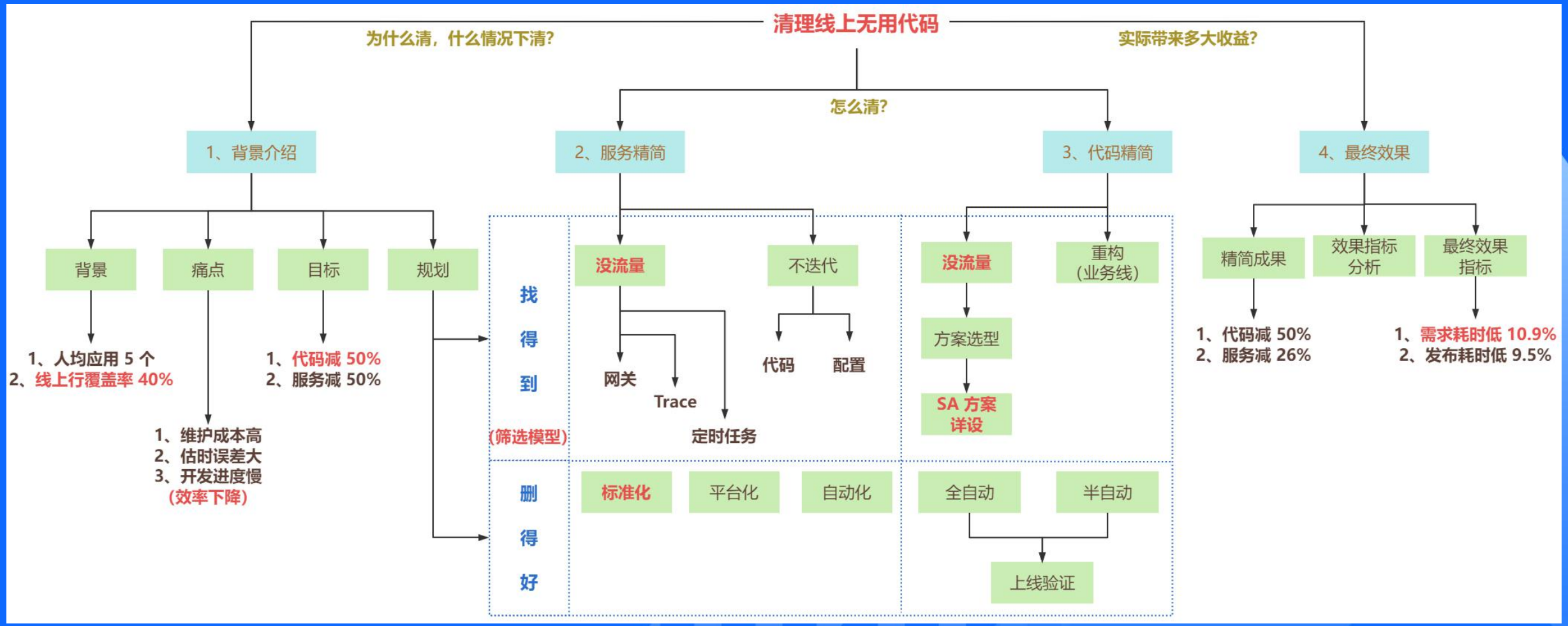
代码行粒度　　配置精简　　**腐化治理**



依赖精简　　服务合并

Q & A

给我一个 API，我可以干掉一半代码



想一想，我该如何把这些技术应用在工作实践中？

THANKS