ArchGuard 架构治理

ArchGuard

开源架构治理平台

ArchGuard 是一个主要针对于分布式场景下的架构治理平台。它可以在开发过程中，帮助架构师、开发人员分析系统间的远程服务依赖情况、数据库依赖、API 依赖等；根据架构评估模型，对于整体架构进行评估并提出改进建议，达到架构治理的目标。

微信公众号

1.

架构可视化

（基于 C4 模型可视化系统现状）

2.

架构分析

（代码、服务、数据库、代码变更等）

3.

架构治理

（基于规则与表达式持续守护系统架构）

Roadmap

2020.02

2022.04~

1.0 单体应用分析（内部）

单体系统依赖分析与可视化

代码调用链分析、模块化分析、单体架构指标、代码坏味道

1.5~ 分布式可视化

服务地图、数据库地图、变更影响分析

中间件：JDBI、JPA、MyBatis。
Web 框架：Spring Boot、Dubbo（进行中）

2.0 可扩展的分布式分析

Scanner 插件化、持续集成

Scanner 插件化、框架分析插件化、持续集成守护等

2.x / 3.0 分布式治理

自定义规则、架构适应度函数

设计系统的架构适应度函数，结合架构的一系列规则，以持续演进系统架构

ArchGuard 现状

社区发展

当前版本主要贡献者

张大步
（DoubleDuckDuck）

1.x 核心开发

刘航
(isixline)

1.x **核心开发。**主要打杂，补充文档，写了🫳代码

廖桉冬
(Anddd7)

主要维护Scanner、Chapi（Kotlin写的比较多一点 🐳）

刘云志
(Yunzhi-Liu)

主要解决DevOps相关的问题

黄峰达
(phodal)

主要维护 Scanner、Chapi（Antlr 语法解析）等

黄雨青
(YuqingHuang)

主要提issue🦫、设计模型、💬💭

架构可视化

ArchGuard 现状

现状：基于 C4 模型的软件可视化

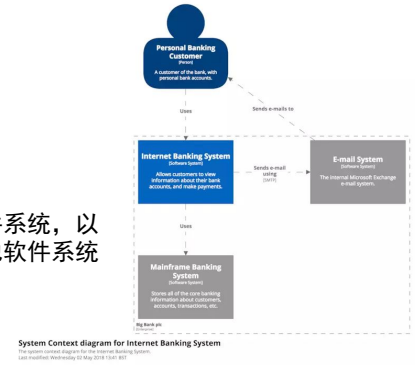
- 单个组件（或服务）总览
- 服务地图（HTTP API 分析）
- 代码级分析（模块、包、类、函数）
- 数据库地图（代码到数据库表）
  当前还没有关联到多个系统
- 变更影响分析（代码变更的影响范围）
  当前还没有关联到多个系统
- 消息系统地图
  还没有开始设计

架构分析

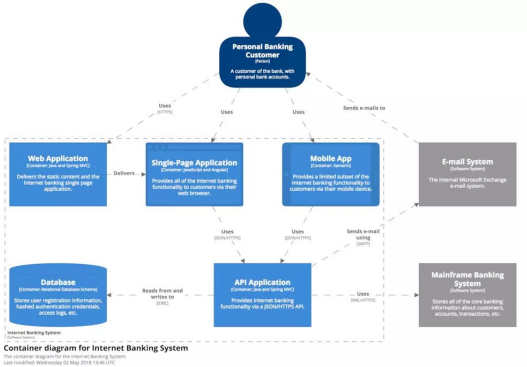
可视化现有技术架构：C4 模型

系统上下文图

显示正在构建的软件系统，以及系统与用户及其他软件系统之间的关系。

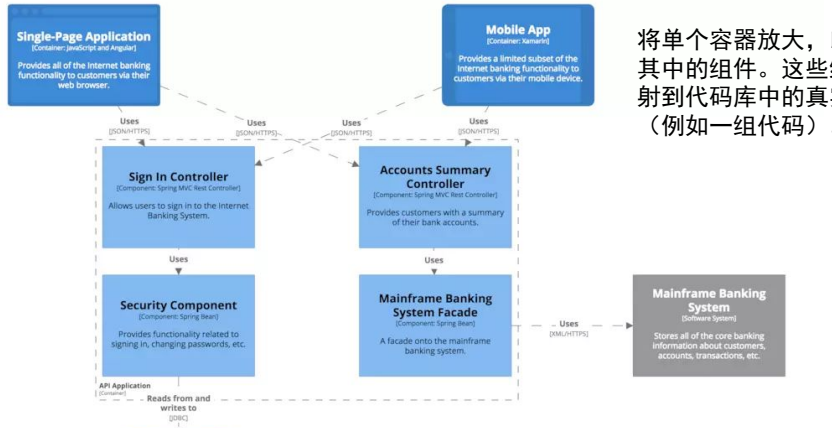


容器图



将软件系统放大，显示组成该软件系统的容器（应用程序、数据存储、微服务等）。

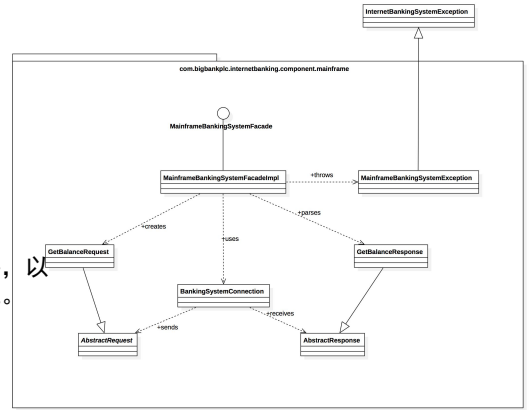
组件图



将单个容器放大，以显示其中的组件。这些组件映射到代码库中的真实抽象（例如一组代码）。

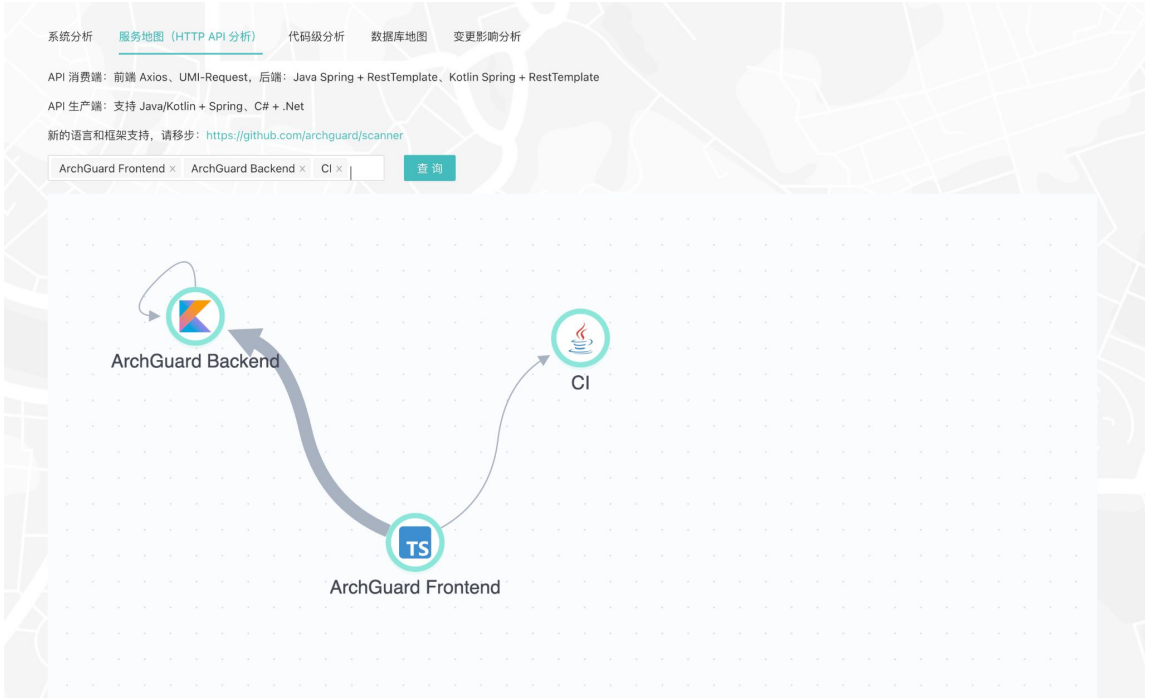
代码图

按需可以放大个别组件，以显示该组件的实现方式。



服务地图

手动定义的容器图

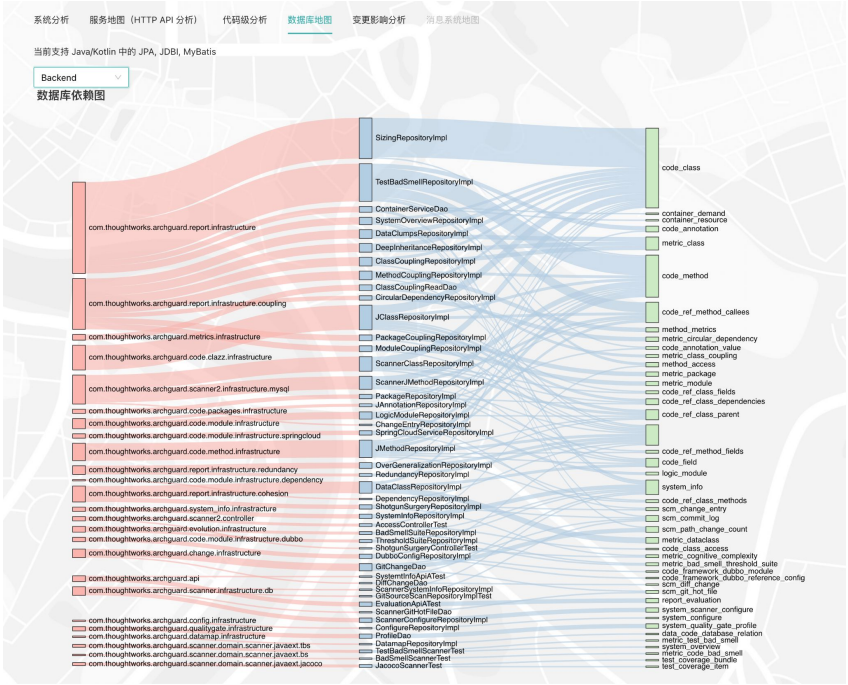


代码级分析

分析系统内部的调用关系

数据库地图

结合代码分析的容器图扩展



TODO：

需要其它依赖于些数据库表的服务

变更影响分析

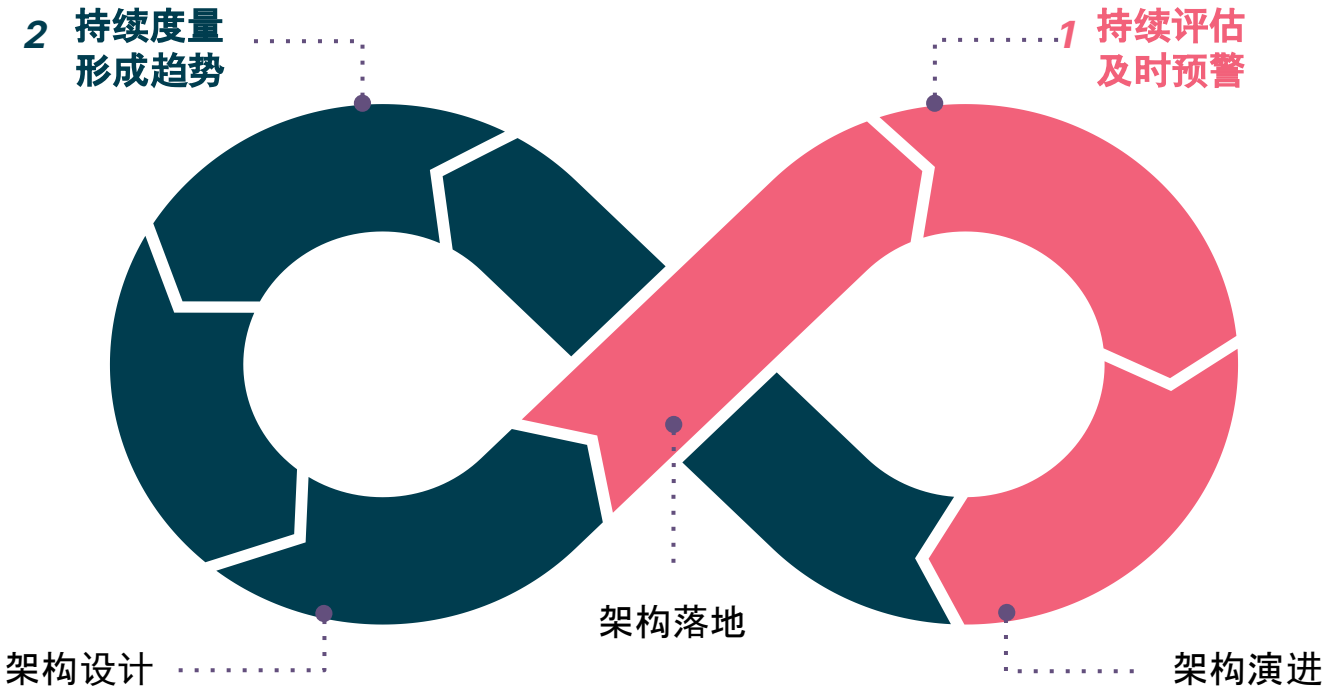
分析代码提交的影响范围

TODO：

回溯到 API 声明 时，需要依赖于这个 API 的服务

架构治理
（未来）

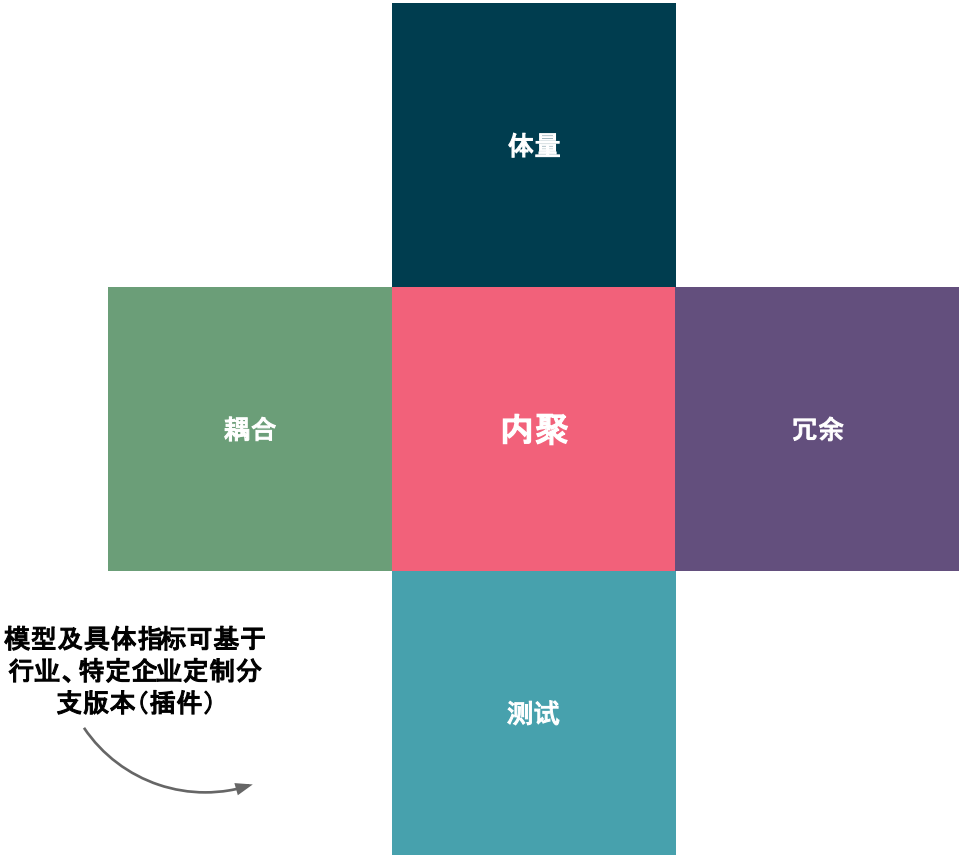
架构治理思路 - 双环守护



1. 架构评估及预警

通过持续的架构评估，可以在开发态对于架构实现的效果进行评定，并且对于一些不合理的实现进行预警。
架构评估的关键在于评估模型的建立以及阈值区间的设置，现有评估模型的一级维度如右图所示，源于行业经验沉淀而构建的，仍需完善：

1. 从业界架构设计的原则、规范与最佳实践出发，如：高内聚、低耦合、开闭原则、依赖倒置等
2. 对代码架构坏味道进行了体系化的归纳总结，如数据泥团、枢纽化模块、深继承、循环依赖都可能导致过高耦合
3. 充分重视测试保护对于架构的贡献及影响
4. 同时兼顾考虑了业界实践与企业内部规范的融合



TODO：评估模型完善，对应功能开发

1. 架构评估及预警

预警规则可与阈值相关，也可以制定专家规则：

请选择合适您系统的指标阈值：

架构评估优化指标　　选择

评估维度	坏味道	判断条件	指标阈值
体量维度	过大的包	包中包含的类个数 > 指标阈值	20
	过大的包	包的总代码行数（不含import行数） > 指标阈值	12000
	过大的模块	模块中包含包的个数 > 指标阈值	20
	过大的模块	模块的总代码行数 > 指标阈值	24000
	过大的类	类中包含的方法个数 > 指标阈值	20
	过大的类	类的代码行数 > 指标阈值	600
	过大的方法	方法代码行数（含空行） > 指标阈值	30
耦合纬度	枢纽模块	出向依赖或入向依赖 > 指标阈值	8
	枢纽包	出向依赖或入向依赖 > 指标阈值	8
	枢纽类	出向依赖或入向依赖 > 指标阈值	8
	枢纽方法	出向依赖或入向依赖 > 指标阈值	8
	数据泥团	缺乏内聚指标（LCOM） > 指标阈值	4
	过深继承	继承深度 > 指标阈值	4

预警示例(待实现)：

架构守护预警

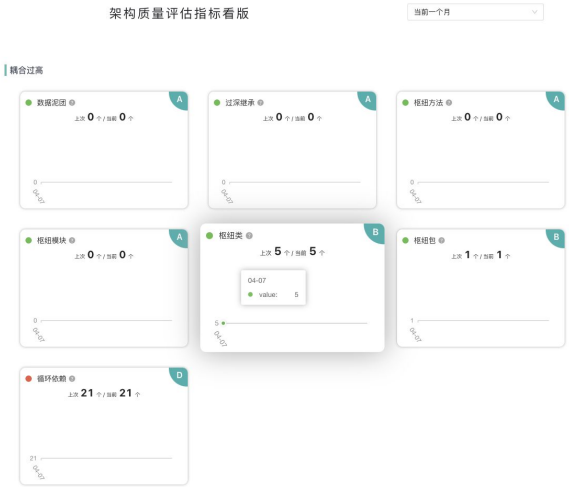
- #13迭代#243版本出现：贷款服务破坏了分层，贷款服务不应依赖外部系统
- #13迭代#243版本出现：分润服务破坏了分层，分润服务不应依赖账务服务

TODO：预警规则库完善；预警信息实现；接入流水线产生预警

2. 架构度量及趋势分析

定期快照架构评估结果中的目标数据(可定义)，形成架构变化趋势。架构师及管理层可以基于变化趋势，来判断架构变化是否符合预期，作为架构治理的数据基础，最终达成“架构落地不变形”等期望。

现有的度量及趋势部分还较为简单：



TODO:架构变化趋势图;趋势分析;趋势预测🤩

分布式架构治理

To be continue

……🤫

https://github.com/archguard/archguard/issues/45

开源协议

代码库

可修改，可商用，可私有化

功能	地址	开源协议	其它
后端	https://github.com/archguard/archguard	MIT	Spring
前端	https://github.com/archguard/archguard-frontend	MIT	React
Scanner	https://github.com/archguard/scanner	MIT	
文档	https://github.com/archguard/archguard.org	-	Jekyll
语法分析	https://github.com/modernizing/chapi	**MPL 2.0**	Antlr

Chapi 不属于 ArchGuard，所以协议是不一样的

MIT x MPL

在所有常用的开源许可协议中，MIT许可协议最为简短，可能也最为广泛。它的条款非常松散，比起大部分其它许可协议来说更加宽松。其基本条款如下：

1. 使用者可以随意使用，复制，修改这个软件。没有人能够阻止你在任何工程里使用它，你可以复制任意次数、以任何形式，或按你的愿望修改它。
2. 使用者可以向外免费发放，或出售。你可以随意的分发它，没有任何限制。
3. 唯一的限制是使用者必须接受协议条款。即软件必须附带版权和许可协议。
4. MIT 协议是目前最少限制的协议。它基本上就是任何人可以对这个协议下的软件的做任何的事情，只要你能认可这个协议。

MPL 是 The Mozilla Public License 的简写，是 1998 年 初 Netscape 的 Mozilla 小组为其开源软件项目设计的软件许可证。MPL 许可证出现的最重要原因就是，Netscape 公司认为 GPL 许可证没有很好地平衡开发者对源代码的需求和他们利用源代码获得的利益。同著名的GPL 许可证和 BSD 许可证相比，MPL 在许多权利与义务的约定方面与它们相同（因为都是符合 OSIA 认定的开源软件许可证）。但是，相比而言MPL 还有以下几个显著的不同之处：

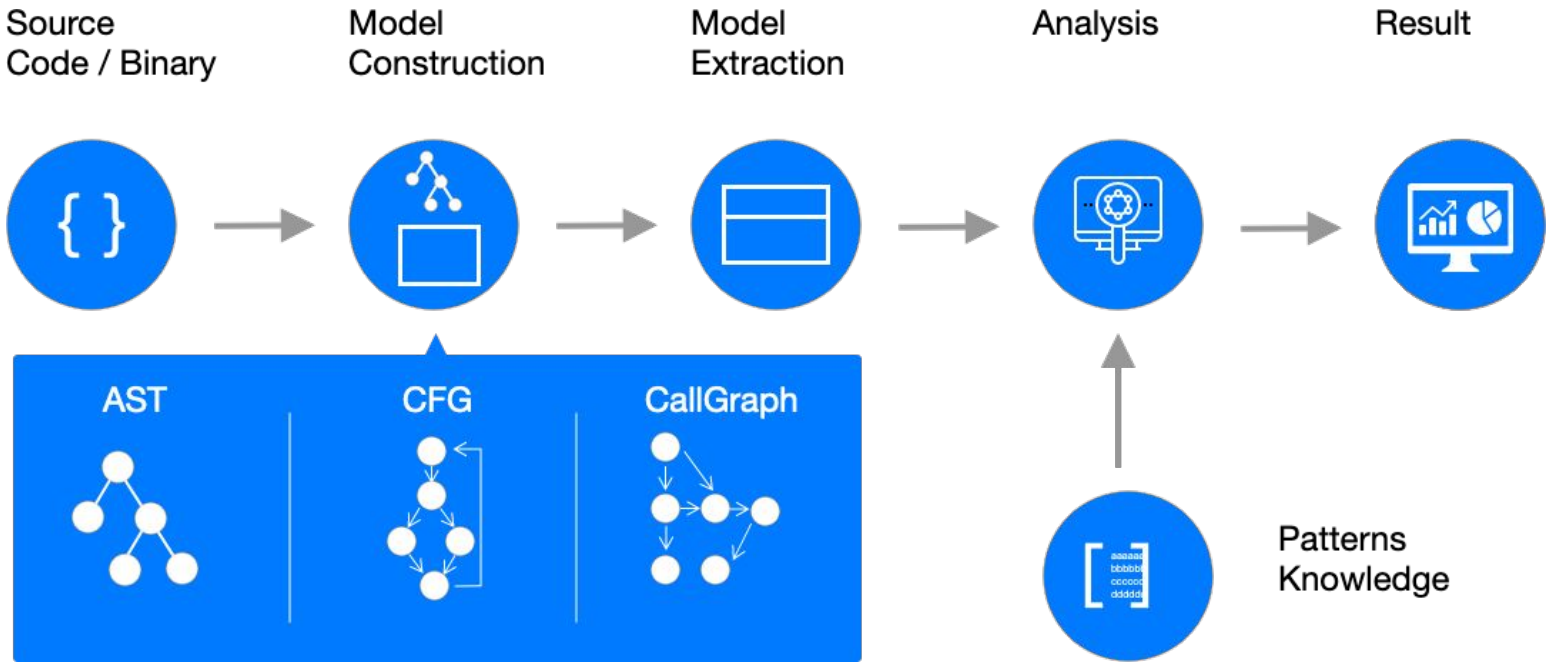
1. MPL允许免费重发布、免费修改，但要求**修改后的代码版权归软件的发起者**。这种授权维护了商业软件的利益，它要求基于这种软件得修改无偿贡献版权给该软件。这样，围绕该软件得所有代码得版权都集中在发起开发人得手中，但MPL 是无偿使用得。
2. MPL 要求对于经 MPL 许可证发布的源代码的修改也要以MPL 许可证的方式再许可出来，以保证其他人可以在MPL 的条款下共享源代码。但是，在 MPL 许可证中对“发布”的定义是“以源代码方式发布的文件”，这就意味着 MPL 允许一个企业在自己已有的源代码库上加一个接口，除了接口程序的源代码以 MPL 许可证的形式对外许可外，源代码库中的源代码就可以不用MPL 许可证的方式强制对外许可。（**非原软件的源码不需要开源**）
3. MPL 许可证第三条第 7 款中允许使用者将**经过 MPL 许可证获得的源代码同自己其他类型的代码混合得到自己的软件程序**。（**允许混合协议**）
4. 对软件专利的态度，MPL 许可证不像 GPL 许可证那样明确表示反对软件专利，但是却明确要求源代码的提供者不能提供已经受专利保护的源代码（除非他本人是专利权人，并书面向公众免费许可这些源代码），也不能在将这些源代码以开放源代码许可证形式许可后再去申请与这些源代码有关的专利。
5. MPL 许可证第 3 条有专门的一款是关于对源代码修改进行描述的规定，就是要求所有再发布者都得有一个专门的文件就对源代码程序修改的时间和修改的方式有描述。

简单来说：语法分析比较复杂，问题比较多，期望能回馈到上游。

出处：http://jhjguxin.github.io/blog/2012/04/11/kai-yuan-xu-ke-xie-yi/

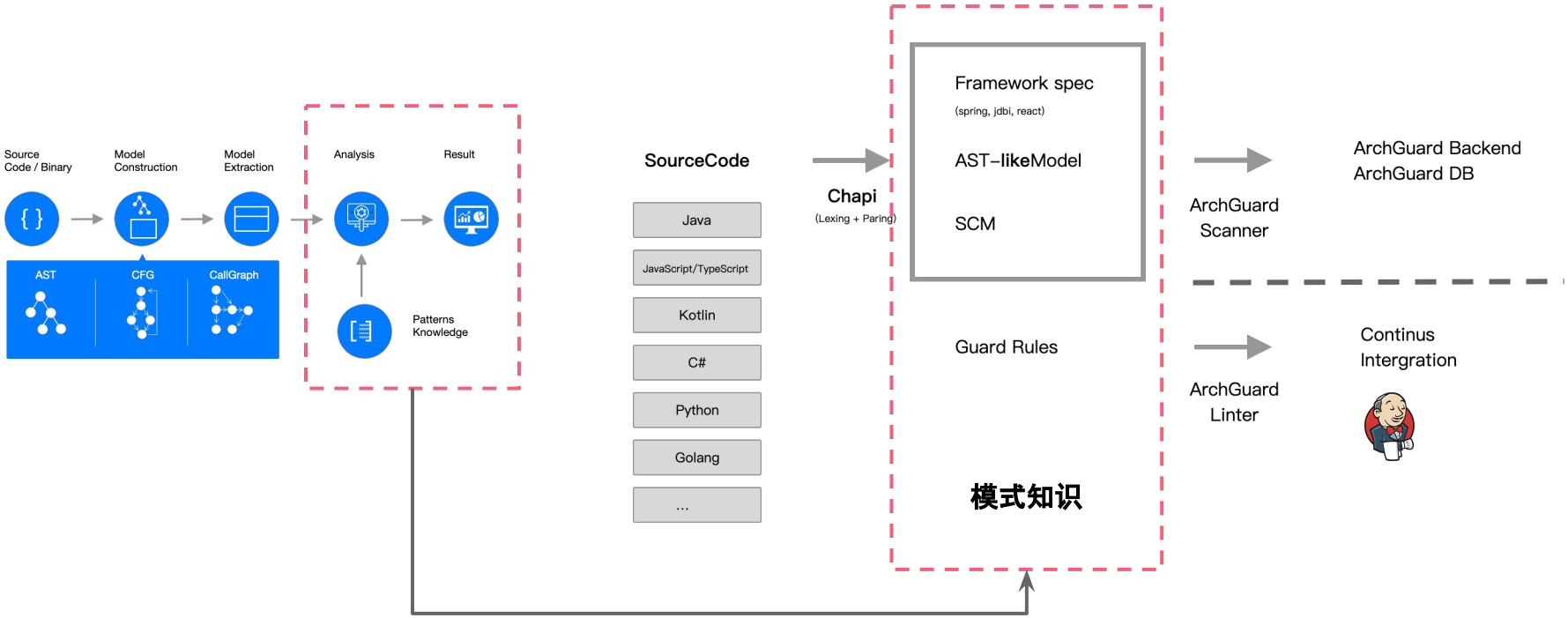
How ArchGuard works ?

静态代码分析的基本模式



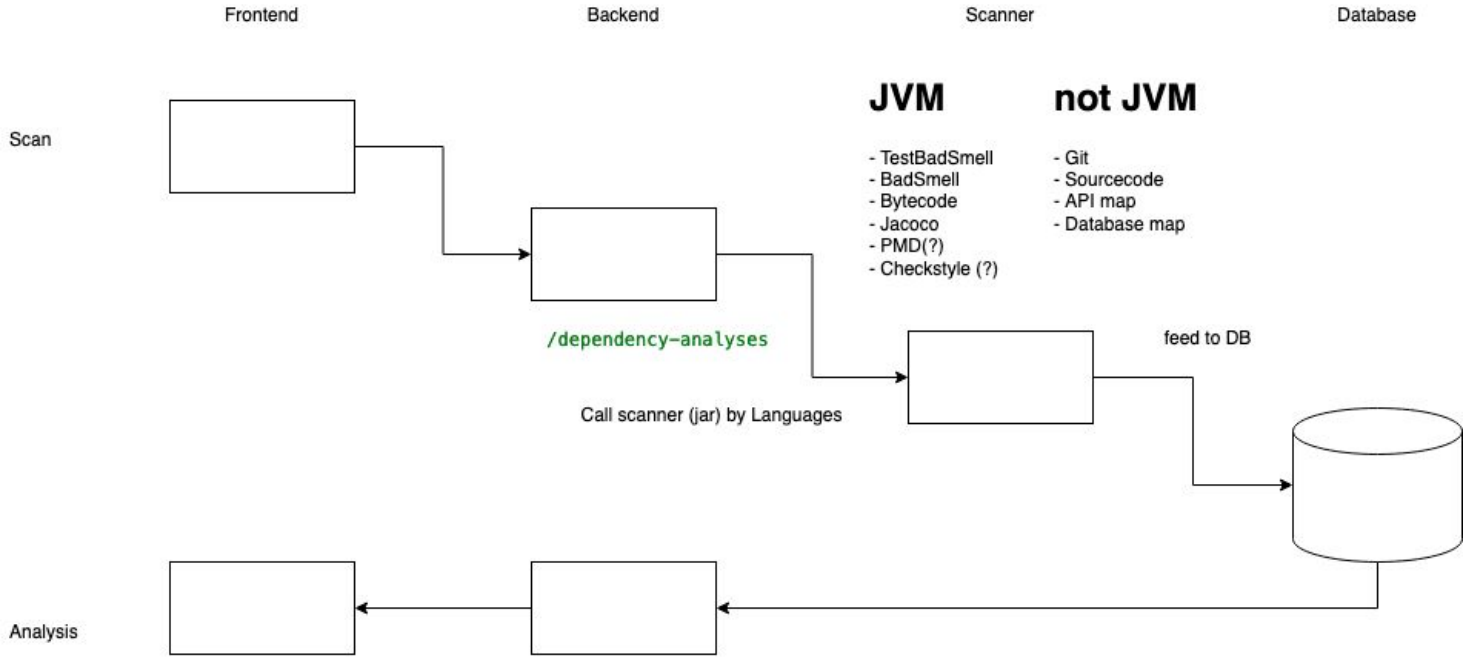
https://github.com/modernizing/modernization

ArchGuard 数据流



组件关系

前端 -> 后端 -> Scanner -> 数据库



下个版本：**Scanner 重构**

Scanner 重用

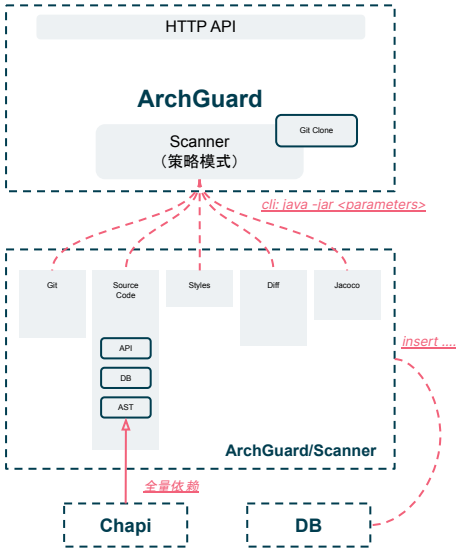
从我们接触到的用户来看, Scanner 的重用是比较重要的

已知的 ArchGuard 使用场景:

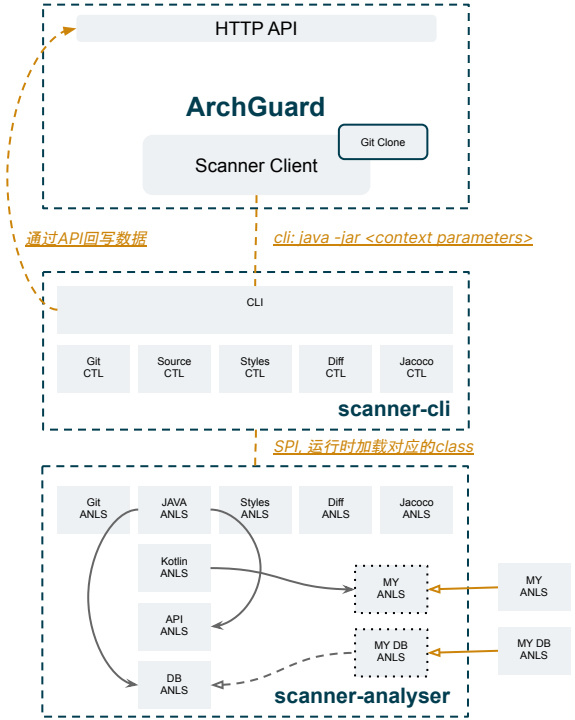
1. 基于 ArchGuard 前端、后端进行分析或者定制。
2. 基于 Scanner 配合持续集成使用。(自定义规则)
3. 将 Scanner 集成到自己现有的系统中。(定制一些插件)

Scanner 组件化：可插拔与自定义扩展

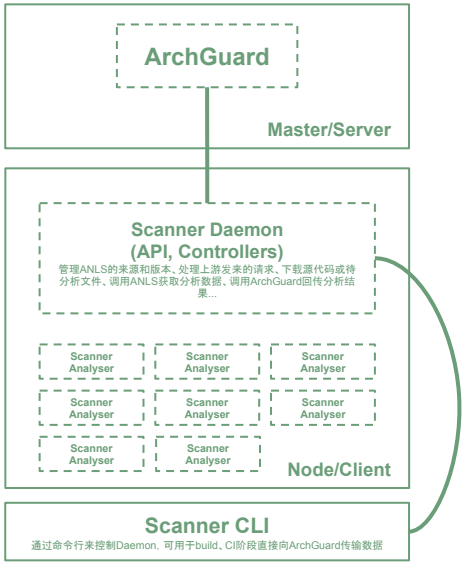
拆分scanner，并允许使用自定义的组件替换官方组件



现状：依赖打包好的scanner，DB直连



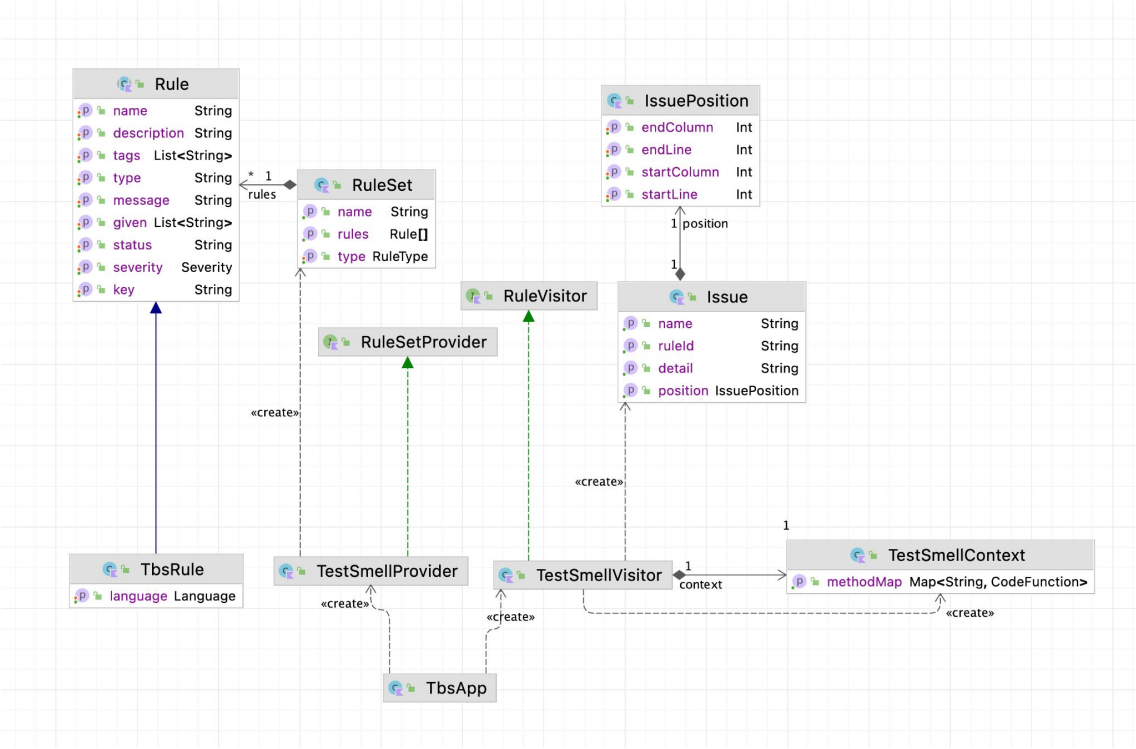
下一步：标准化的Analyser，可配置调用关系，通过API回写数据



未来?：分布式部署，CI支持

Scanner Linter 插件化：框架插件化 API

通过抽象与复用，将现有的测试坏味道、数据库地图、API 分析等做成标准 API



- 测试坏味道（已完成）
- 数据库
- API 分析（测试中）
- 其它

其它

1. 依赖组件清单（如：Gradle、Maven、NPM）
2. API Linter 示例
3. RPC 调用
4. All in One 版本包
5. ……

Welcome to board！