ArchGuard 架构治理

—— 架构工作台

ArchGuard

GitHub：https://github.com/archguard/

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）

AGENDA

- 变化:RFC、规则、洞见功能
- 为什么设计架构工作台？
- 架构工作台实现技术
- 下一阶段

捐赠 **NextArch** 基金会旗下(进行中)

让 ArchGuard 保持所有权的中立性，设计未来的软件架构，构建下一代架构开源生态

2011 年 11 月，Linux 基金会正式成立 NextArch Foundation 下一代架构基金会。该基金会致力于在异构基础设施和多云场景下构建下一代技术架构，并发展适合企业数字化转型的开源生态。

NextArch Foundation 下一代架构基金会将通过聚集 Linux 基金会、TARS基金会的上、下游技术资源，聚合全球 IT 公司、开源原生企业等多维资源，解决异构等场景下框架协议间兼容难度大、组件使用门槛高等问题，以满足企业对未来新业务和新场景的需求。

开源流程：RFC 记录架构决策

随着组织朝着演进式架构的方向发展，记录下围绕设计、架构、技术和团队工作方式的决策是非常重要的。并且希望大家都可以参与到重要决策中来，并且一起参与讨论这些决策。

RFCs 是一种用于收集上下文、设计和架构思想，并与团队协作，最终达成决策以及上下文和结果的方式。

现在我们使用一种轻量级的 RFCs 方法，参考了其他开源项目，使用简单的标准化模板和版本控制来管理 RFCs。

开发者需要创建一个 RFC 文档，向 RFC 仓库提交一个 pull request，然后将社区的反馈包含在提案中。由 core team 做最终决定是否接受这个 RFC。

一份更详细的流程和模版：https://github.com/archguard/rfcs

治理：规则与 Issue

结合持续集成，构建更丰富的架构守护

除了 ArchGuard 社区提供的 Linter，还可以构建自己的 Linter。

SQL

(代码中)

Test

Web API

Layer

（待实现）

Arch

（待实现）

为什么设计工作台？

架构治理是一个复杂的问题

如何将这个复杂问题繁杂化？

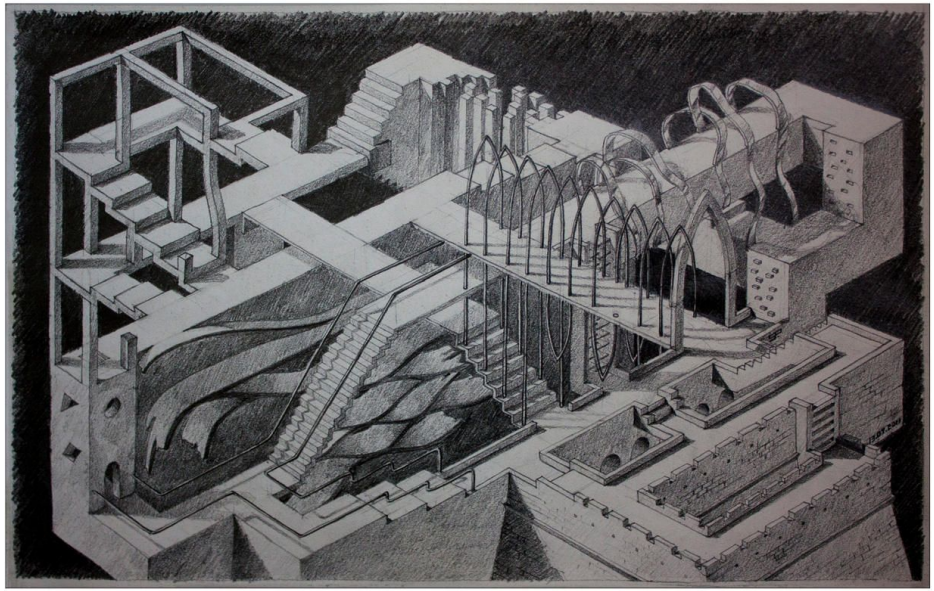
斯诺登教授帮助企业进行决策的“栖息地”框架中译版 v0.2

在不断的尝试之后，模式自然就会浮现出来。

https://www.jianshu.com/p/a961400f0023

如何探索更多的架构风格？

系统间的架构千差万别，如何让人去探索复杂的系统？



1. 架构是多维的。包含技术、数据、安全、运维与系统等
2. 缺乏统一的架构**语言**。用于沟通的人类语言，诸如于什么是组件？
3. 系统的架构千奇百怪。架构风格或模式差异，如微服务架构、插件化架构等。
4. 缺乏业务上下文。作为一个外部架构师，帮助治理时缺乏一些上下文。
5. 细节是魔鬼。架构的世界丰富多彩，没有办法一一展现出来，比如一个小小的接口，可能会反转我们对于理解的假设。
6. 我们（ArchGuard 团队）目前的架构能力有限。

ArchGuard 的三态模型

设计态：目标架构。通过 DSL（领域特定语言）+ 架构工作台来构建 。

开发态：实现架构。关注于：可视化 + 自定义分析 + 架构治理。

运行态：运行架构。结合 APM 工具，构建完整的分析链。

1.

设计态

（现在设计）

2.

开发态

（当前核心关注点）

3.

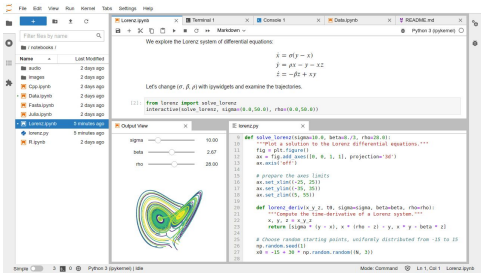
运行态

（未来整合）

架构工作台

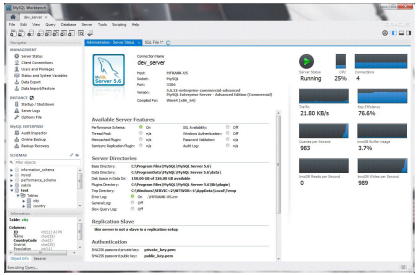
工作台

工作台是一个坚固耐用的桌子，它们设计的范围从简单到非常复杂的平面设计都有，可被视为工具一部分。



Jupyter —— 高级数据工作台

MPS —— 语言工作台

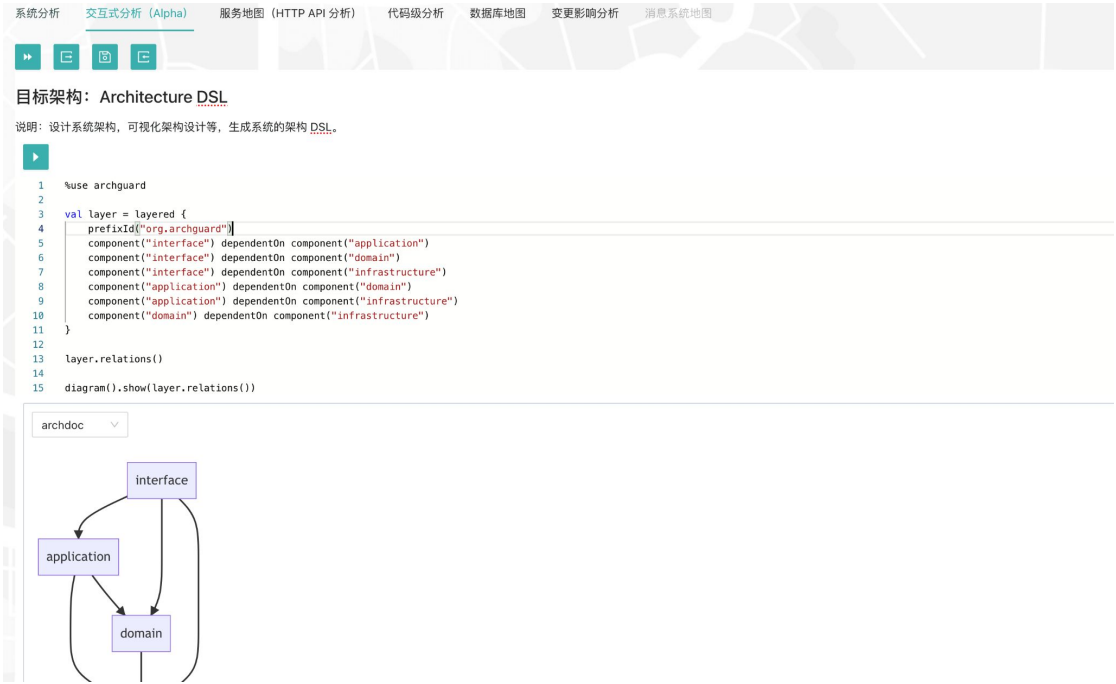


Mysql Workbench ？

工作台预期提供的功能

1. 交互式架构设计(未来)
2. 集成 ArchGuard 现有的功能(进行中)
3. 治理架构(未来)
4. ……

提供类 **Jupyter** 的交互式设计



1. 系统创建 DSL 和导入
2. 执行 Block
3. 基本的 Scan
4. 设计架构

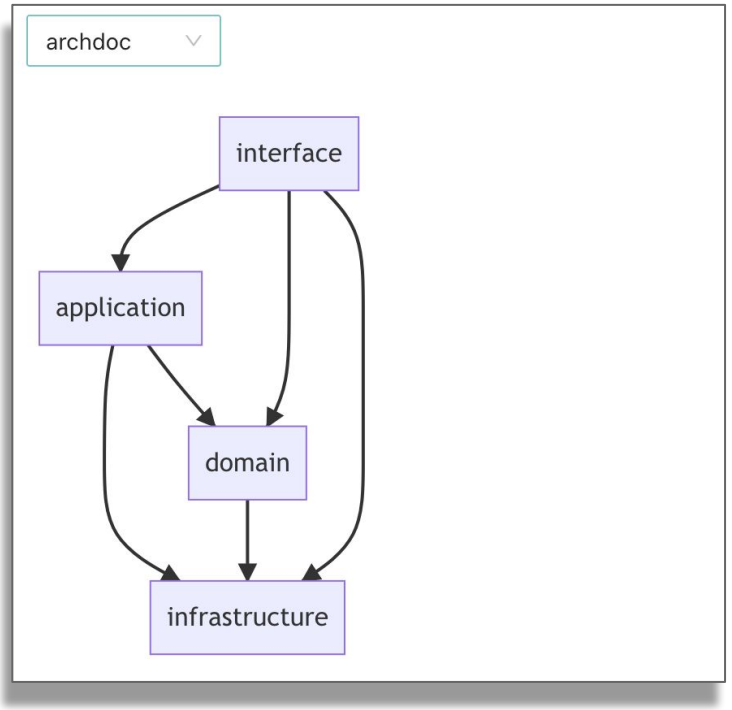
设计 DSL（示例）

基于 Kotlin 的示例

```
 %use archguard

val layer = layered {
    prefixId("org.archguard")
    component("interface") dependentOn component("application")
    component("interface") dependentOn component("domain")
    component("interface") dependentOn
component("infrastructure")
    component("application") dependentOn component("domain")
    component("application") dependentOn
component("infrastructure")
    component("domain") dependentOn component("infrastructure")
}

layer.relations()
diagram().show(layer.relations())
```



ArchGuard DSL（示例）

计划中，对于 ArchGuard 的 CRUD 进行抽象，可能是 Code DSL + 类似 Linq 的方式

当前/现状架构：ArchGuard DSL

说明：基于 ArchGuard Backend，提供 CRUD 封装的 API，如构建系统，查询依赖关系等。

```
1  %use archguard
2
3  repos {
4      repo(name = "Backend", language = "Kotlin", scmUrl = "https://github.com/archguard/archguard")
5      repo(name = "Scanner", language = "Kotlin", scmUrl = "https://github.com/archguard/scanner")
6  }
7
8  context.repos.create()
```

Create

name	language
Backend	Kotlin
Scanner	Kotlin

DSL result:

```
"root" : [...] 2 items
```

治理 DSL（示例）

基于 Kotlin 的示例

治理架构：Analyser/Scanner/Linter DSL

说明：结合 ArchGuard Scanner 中的能力，对系统进行 Scanner、Analyser、Linter 等。

Scan 示例：

```
%use archguard

scan("Backend").create()
```

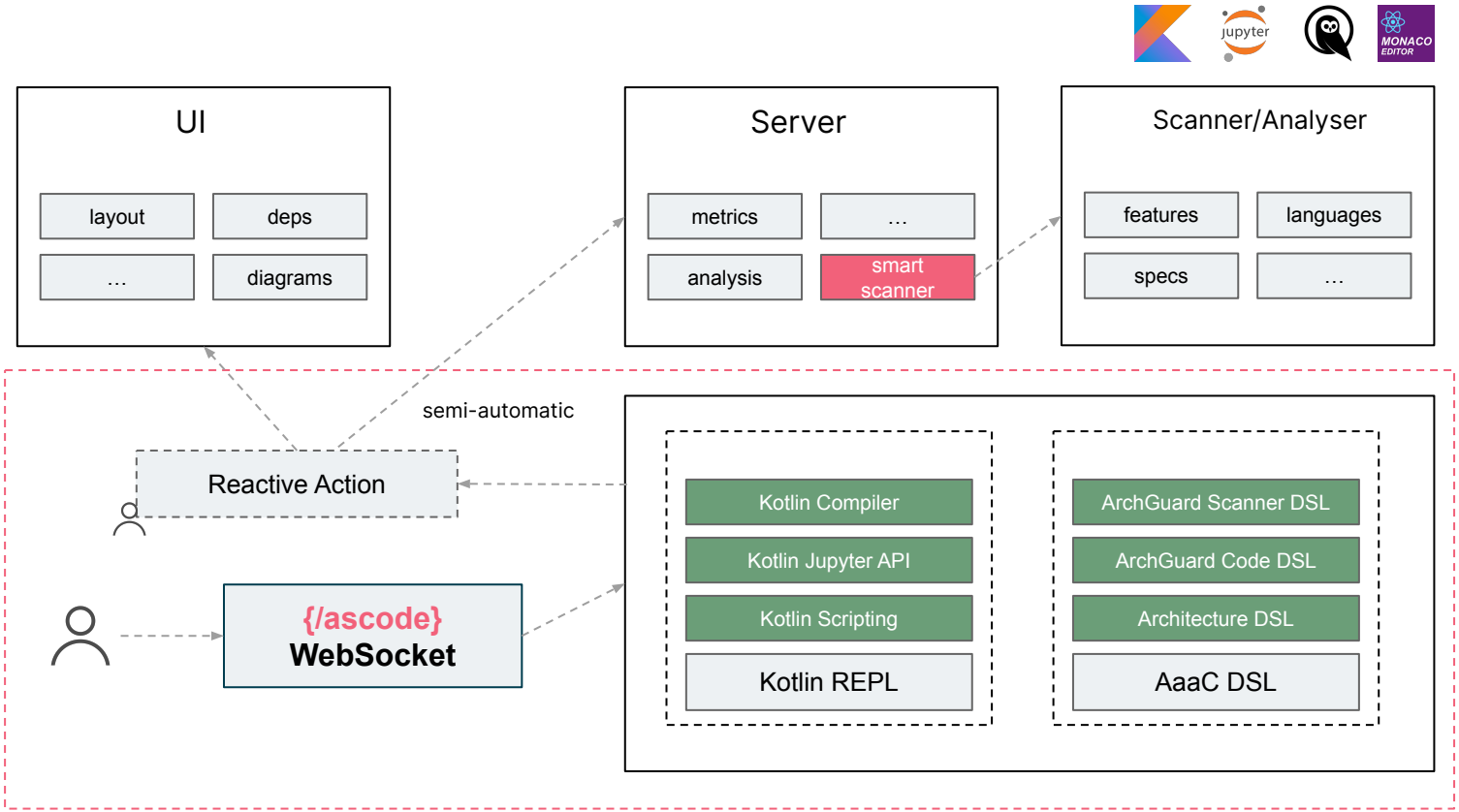
Linter 示例（待实现）：

```
%use archguard

linter("Backend").layer()
```

架构工作台实现技术

Workbench Architecture

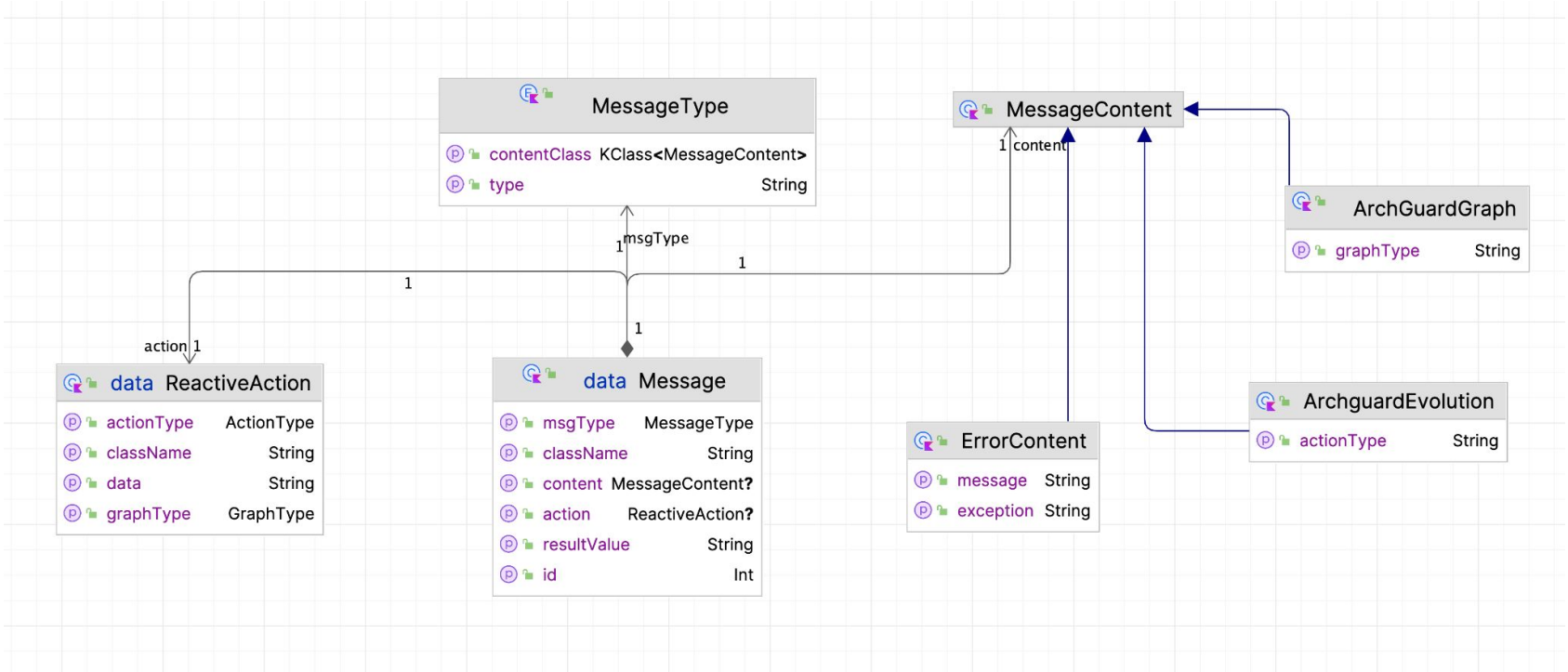


核心组成

1. 通讯协议 -> WebSocket
2. 编辑器 -> ProseMrrior + Monaco Editor
3. DSL -> Kotlin Type-safe Builder API
4. REPL -> Kotlin Jupyter

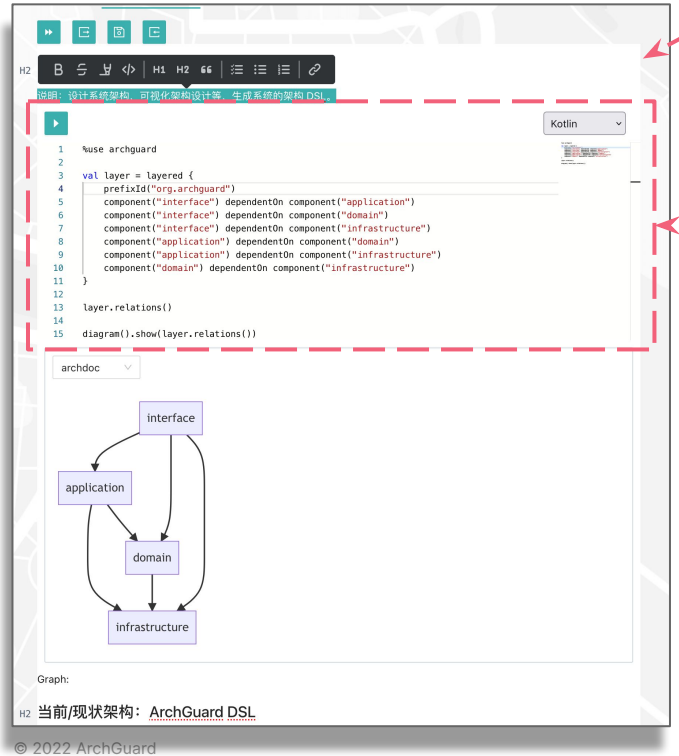
通讯模型

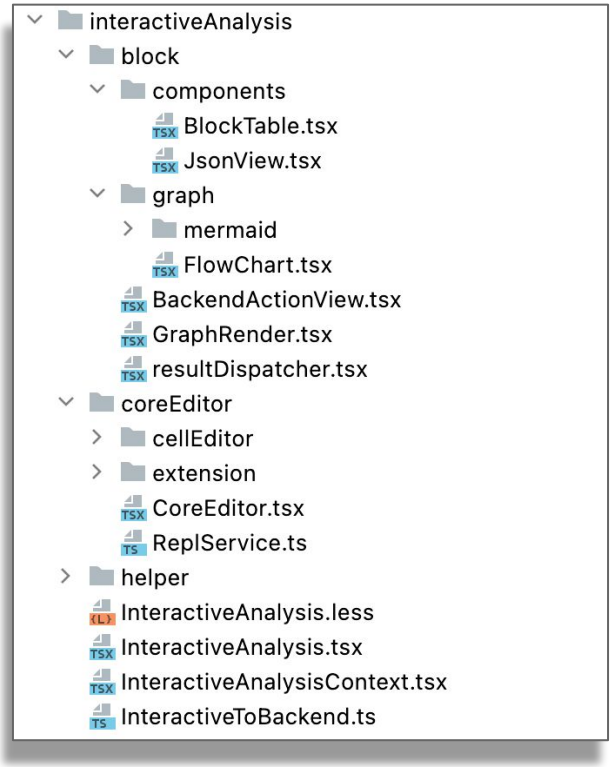
ReactiveAction 用于提供 UI 所需要的信息



UI：ProseMirror + Monaco Editor

WYSIWYG + Markdown + Monaco





DSL 选型：Kotlin Type-safe Builder

基于 Kotlin 的嵌入型 DSL

```kotlin
class ComponentDecl(val name: String) : Element {

    var dependents: List<ComponentDecl> = listOf()

    infix fun dependentOn(component: ComponentDecl) {

        this.dependents += component

    }

    infix fun `依赖于`(component: ComponentDecl) {

        this.dependentOn(component)

    }

}

fun layered(init: LayeredArchDsl.() -> Unit): LayeredArchDsl {

    val layeredDecl = LayeredArchDsl()

    layeredDecl.init()

    context.layered = layeredDecl

    return layeredDecl

}
```

```kotlin
val layer = layered {
    prefixId("org.archguard")
    component("interface") dependentOn component("application")
    component("interface") dependentOn component("domain")
    component("interface") dependentOn component("infrastructure")
    component("application") dependentOn component("domain")
    component("application") dependentOn component("infrastructure")
    component("domain") dependentOn component("infrastructure")
}
```

考虑因素：

- 语法的学习成本。
- 语法的体验设计。
- 语法的编辑器/IDE 支持。

执行 DSL：Kotlin Jupyter

Kotlin Scripting vs Kotlin Jupyter

```
fun makeEmbeddedRepl(): ReplForJupyter {
    val property = System.getProperty("java.class.path")
    var embeddedClasspath: MutableList<File> = property.split(File.pathSeparator).map(::File).toMutableList()

    val isInRuntime = embeddedClasspath.size == 1
    if (isInRuntime) {
        System.setProperty("kotlin.script.classpath", property)

        val compiler = KotlinJars.compilerClasspath
        if (compiler.isNotEmpty()) {
            val tempdir = compiler[0].parent
            embeddedClasspath =
                File(tempdir).walk(FileWalkDirection.BOTTOM_UP).sortedBy { it.isDirectory }.toMutableList()
        }
    }

    embeddedClasspath = embeddedClasspath.distinctBy { it.name } as MutableList<File>
    logger.info("classpath: $embeddedClasspath")

    val config = KernelConfig(
        ports = listOf(8080),
        transport = "tcp",
        signatureScheme = "hmac1-sha256",
        signatureKey = "",
        scriptClasspath = embeddedClasspath,
        homeDir = null,
        libraryResolver = resolveArchGuardLibs(),
        embedded = true,
        resolutionInfoProvider = EmptyResolutionInfoProvider,
    )

    return ReplForJupyterImpl(config, this.replRuntimeProperties)
}
```

考虑因素：

- Kotlin 语言自带的试验性功能：Kotlin Scripting 提供了一种无需事先编译或打包成可执行文件即可将 Kotlin 代码作为脚本执行的技术。因为，对于我们来说，只需要构建我们的 DSL 包，就可以直接执行。
- Kotlin Jupyter 的实现也是基于 Kotlin Scripting 提供了一系列的 API 封装。

工作台（Alpha）只是一个 PoC

TBD 1：设计态架构相关的功能：如 **DSL**

PS：这里的功能仅是考虑中的，具体根据各人感兴趣的方向去实现。

1. 架构设计的线上化 ？
2. 从 DSL 到代码生成 ？
3. 到系统变化的检测 ？

TBD 2:融合现有的功能

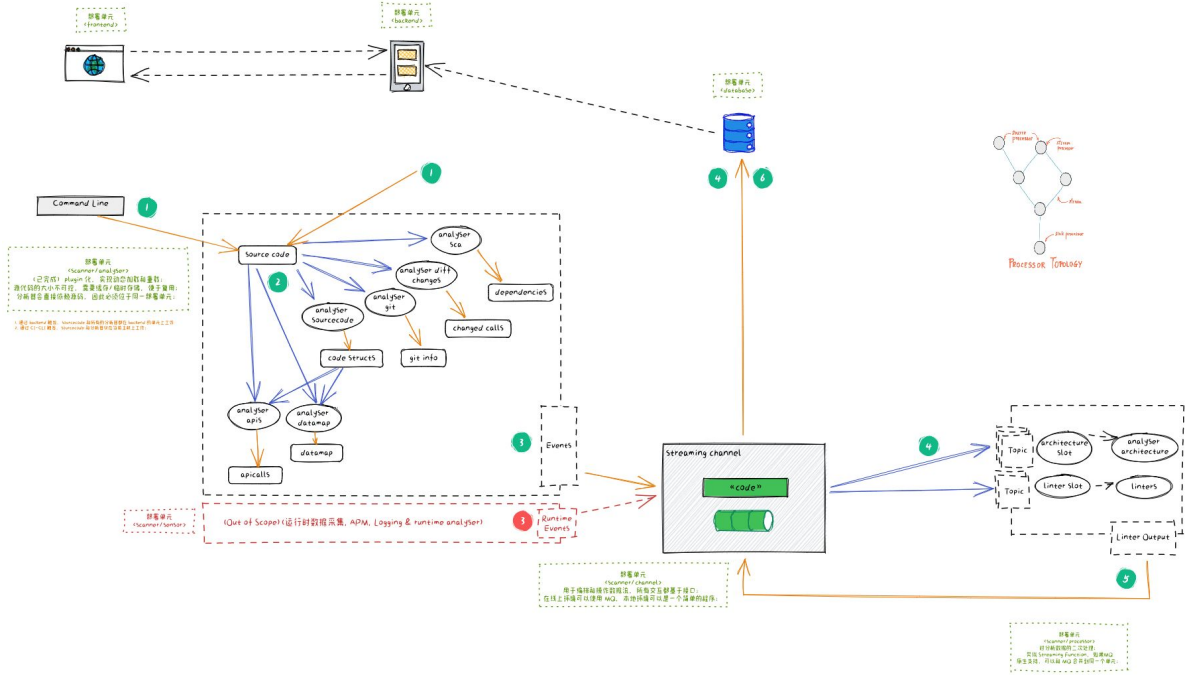
PS:这里的功能仅是考虑中的, 具体根据各人感兴趣的方向去实现。

1. 批量导入 + 扫描 + 自定义扫描功能 ？
2. 构建架构适应度函数？
3. 自定义趋势和洞见 ？

ArchGuard
下一阶段

RFC 0001：MongoDB + MQ 解耦

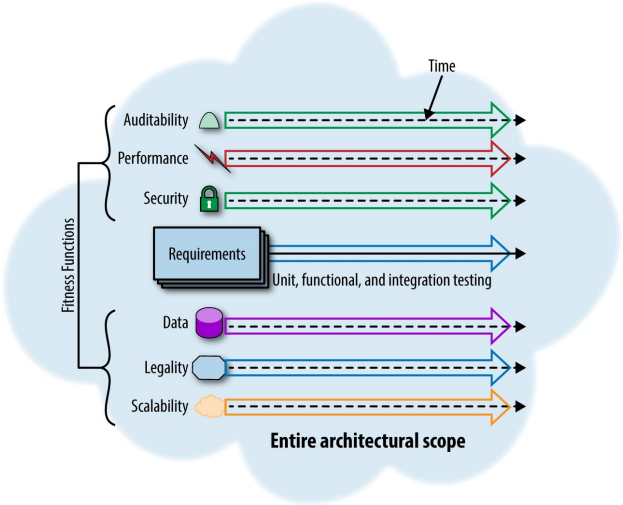
Server 2.0 渐进式重构

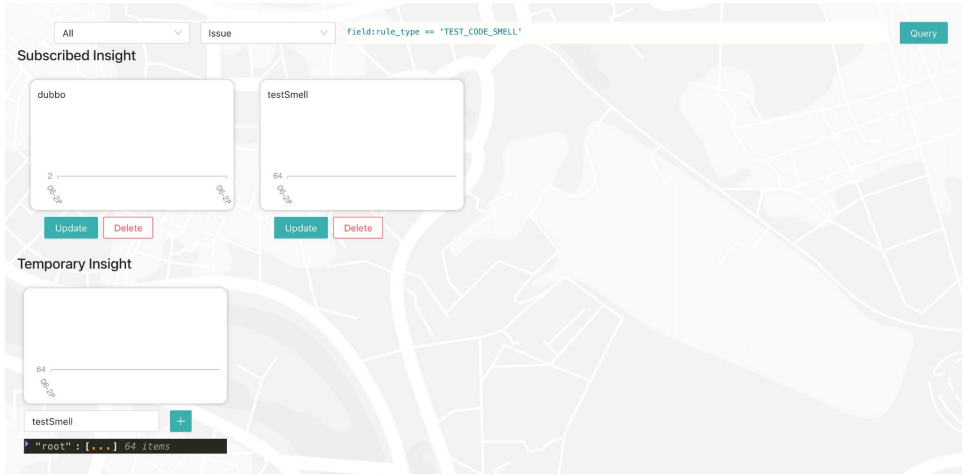


https://github.com/archguard/rfcs/blob/master/text/0001-decoupling-with-mq.md

RFC 0002 : Architecture Insight

构建实时架构洞察，自定义架构适应度函数(最终目标)，丰富架构工作台





https://github.com/archguard/rfcs/blob/master/text/0002-archguard-insights.md

ArchGuard

Welcome to join us~