|  |  |
| --- | --- |
| **ArchGov系统设计** | |
|  | |
| 图片包含 标牌  已生成极高可信度的说明 | |
|  | |
|  | |
|  | |
| **组长** |  |
| **成员** |  |
| **日期** | 2023-11-30 |

目录

[一、 前期调研 3](#_Toc31513)

[1. SonarQube 3](#_Toc28483)

[2. ArchGuard 3](#_Toc7712)

[3. CodeScene 4](#_Toc24425)

[4. ArchUnit 5](#_Toc8739)

[二、 功能描述 7](#_Toc13734)

[1. 对源代码的分析 7](#_Toc22971)

[2. 版本维护历史挖掘 7](#_Toc15504)

[5. 导出分析报告 7](#_Toc9728)

[三、 架构设计过程 8](#_Toc30294)

[1. 可行性阶段 8](#_Toc22546)

[2. 概要设计阶段 8](#_Toc4248)

[3. 详细设计阶段 8](#_Toc3079)

[4. 规划阶段 10](#_Toc12324)

[四、 架构设计结果 14](#_Toc12355)

[1. 架构风格 14](#_Toc7023)

[2. 架构可视化 14](#_Toc17536)

[3. 组件介绍 14](#_Toc16899)

[4. 连接件介绍 15](#_Toc30216)

[5. 架构评估 15](#_Toc20541)

[五、 架构缺陷 16](#_Toc11006)

1. 前期调研
2. SonarQube

SonarQube框架包含以下四个部分：Project、SonarQube Scanner、SonarQube Server、SonarQube Database。其中：

Project是需要被分析的源码；SonarQube Scanner负责执行代码分析，在Project的根目录下执行，其中指定了SonarQube Server的地址，分析完毕后上报给Server；SonarQube Server显示分析结果，可以在Server上设置代码质量管理相关的各种配置，如设置代码检查的规则（Rule）和质量门限（Quality Gate）等。

总的来说，SonarQube遵循分层架构风格。

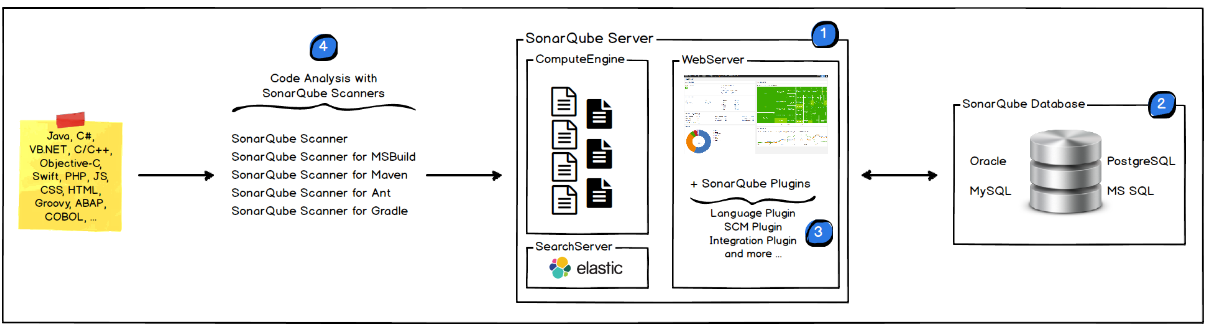


图 1 SonarQube 架构图

1. ArchGuard

ArchGuard 是一个由 Thoughtworks 发起的面向微服务（分布式场景）下的开源架构治理平台。它可以在设计、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。并根据一些架构治理模型，对现有系统提出改进建议。

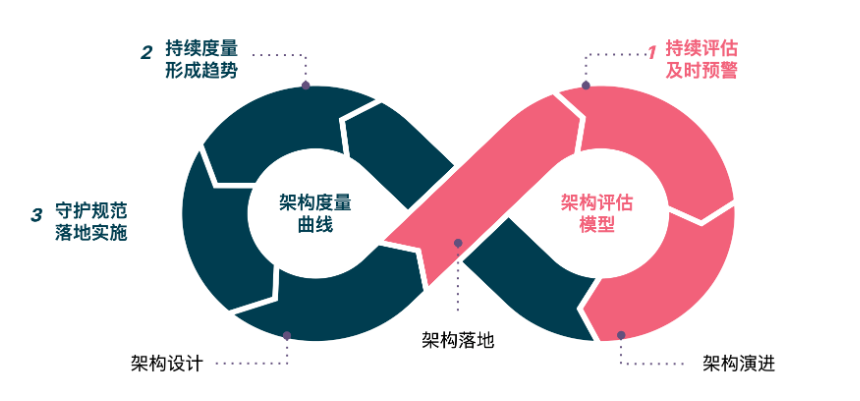
ArchGuard的核心理念是三态模型+双环守护。三态包括：设计态、开发态和运行态。设计态指向目标机构，通过DSL（领域特定语言）和架构工作台来构建。开发态指向实现架构，关注于可视化+自定义分析+架构治理；运行态指向运行架构，结合APM工具，构建完整的分析链。

图 2 双环守护

ArchGuard基于c4架构模型进行分层化的分析，在 System Context（上下文）， Container（容器）, Component（部件）, Code（代码）四个不同的架构视图上，它们是不同的抽象级别，对应于不同的受众，如团队内开发人员关心代码内的依赖，架构师关心组件、窗口间的依赖。ArchGuard遵循微服务架构风格和分层架构风格。

1. CodeScene

CodeScene主要遵循微服务架构。它基于版本控制的行为代码分析，支持各类主流编程语言。它更倾向于分析代码变化的趋势，而不仅仅是代码本身。

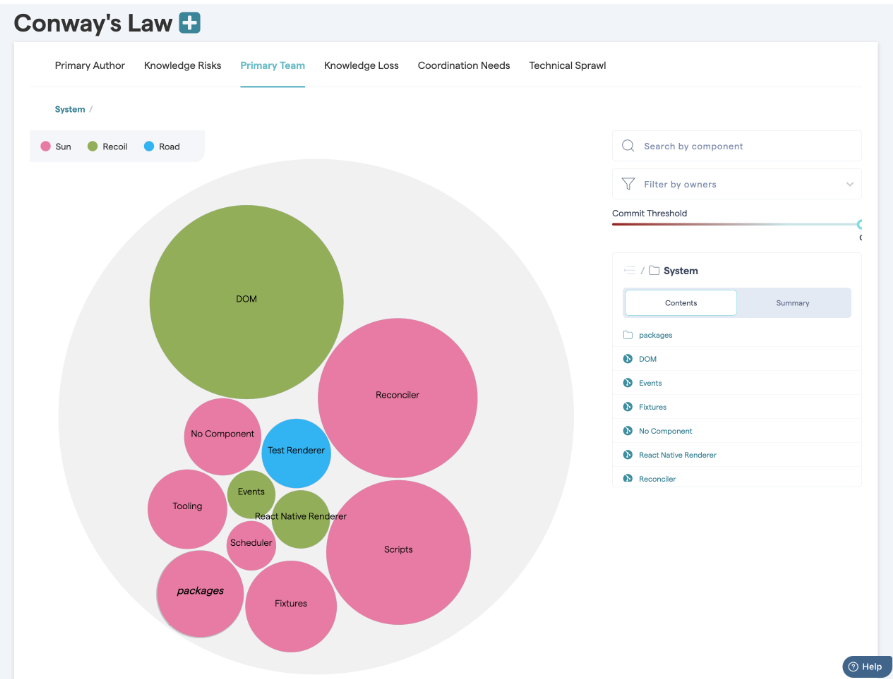


图 3 CodeScene 架构图

1. ArchUnit

ArchUnit利用反射和字节码技术获取所有的包、类、字段等信息，并通过特定的分析来确定对象之间的访问关系。ArchUnit使用ASM作为字节码分析的工具，但是代价是ArchUnit的许多规则不是类型安全的。

ArchUnit支持的检查特性有：包依赖检查；类依赖检查；类和包的位置约定检查；继承检查；分层依赖检查；循环依赖检查。

ArchUnit遵循分层架构风格，分为三层：

Core：核心层，处理一些基本的类、字节码等操作，用于 import 一组类进行断言。

Lang：处理各种规则的语法和架构逻辑，以及一些基本的检查器。

Library：定义了一些更为复杂的预定义规则。

综合分析上述各种代码管理系统，可以发现，它们都以各种方式将功能模块进行拆分，但拆分的方式和依据有各自的区别，而维护和管理的重点也各自不同。比如ArchUnit注重对各种依赖的检查，CodeScene统计代码版本变更的情况等等。

1. 功能描述
2. 对源代码的分析
   1. 实体检测：检测代码中存在的实体（实体类）；
   2. 依赖检测：检查包之间、类之间的依赖关系；
   3. 组件检测：分析代码中的逻辑构建块，比如将一层视为一个组件；
   4. 接口检测：检测代码中的接口部分，比如java中的interface接口类；
   5. 继承检测：检测代码中实体类对接口的继承。
3. 版本维护历史挖掘

从commit中分析重构操作、issue信息和bug信息。

1. 大规模项目的扫描
2. 可视化

可视化展示扫描日志、扫描结果和度量统计。可以通过编译器的终端或者zeppelin等可视化工具进行呈现。

1. 导出分析报告

将可视化展示的结果以文件的形式输出，比如txt或pdf等格式。

1. 架构设计过程
2. 可行性阶段

根据对需求的分析，项目需要具备可视化功能，需要前端框架；需要对源码进行分析，需要逻辑控制层；对版本历史进行挖掘和比较，需要一个数据库记录系统信息。

前端采用vue框架，控制层采用java编写，数据利用git存储mysql数据库。

1. 概要设计阶段

ArchGov的功能支持对源码的分析，可以获取代码中的实体、依赖、接口等信息；同时支持对代码版本历史的爬取、比对和分析，同时具备良好的交互功能，满足用户的使用需要。

从总体设计上看，ArchGov分为三个层次，分别负责视图交互、逻辑控制和数据存储。用户通过视图交互界面上传代码文件或者git地址，逻辑控制部分对代码进行解析获取相关信息，并将分析结果以文件的形式返回给前端交互界面，并将数据传输至数据存储模块。

1. 详细设计阶段

用户在进入系统之后，首要功能是在交互界面上上传代码文件或者git地址，交互界面将数据传输到后端进行分析，如果传输的是git地址，控制端需要先爬取代码文件，然后解析。控制端对文件包结构进行检查，提取抽象语法树，解析代码中的实体、依赖、接口等信息，将信息返回交互界面，并将解析结果存储至数据存储模块。

在版本维护历史挖掘的功能中，控制端需要从git中获取commit信息，分析每次提交的变更内容，提取重构操作、issue信息和bug信息，将这些信息与代码分析结果进行关联，形成一个完整的版本历史分析报告，同时也将这些信息返回交互界面，并存储至数据存储模块。

在大规模项目的扫描的功能中，控制端需要对多个项目进行并行处理，利用多线程或者分布式计算的方式，提高扫描效率和准确度，同时也需要考虑内存和网络的限制，避免出现资源不足或者超时的情况，将扫描结果返回交互界面，并存储至数据存储模块。

在可视化的功能中，交互界面需要根据用户的选择，展示不同的视图，比如扫描日志、扫描结果和度量统计等，可以采用表格、图表、树形图等方式，增强用户的体验和理解，同时也提供一些交互功能，比如筛选、排序、放大等，让用户可以自由地探索和分析数据。

在导出分析报告的功能中，交互界面需要根据用户的需求，将可视化展示的结果以文件的形式输出，比如txt或pdf等格式，方便用户保存和分享，同时也需要保证文件的格式和内容的清晰和完整，避免出现乱码或者缺失的情况。

1. 规划阶段

前端交互当中主要实现的是代码上传和分析结果呈现两个功能。代码分析需要支持对多语言源码的分析，在解析代码的阶段需要相对复杂的代码实现。挖掘版本控制时，需要对commit进行爬取和访问。

在规划阶段，我们需要根据架构设计的结果，制定具体的开发计划和时间表，分配好各个组件和连接件的开发任务和责任人，确定好开发工具和环境，以及测试和部署的策略和方法。我们还需要考虑到可能遇到的风险和挑战，以及应对的措施和备选方案。以下是我们的规划阶段的主要内容：

- 前端视图组件的开发计划：

- 使用vue框架，结合element-ui和echarts等组件库，实现交互界面的布局和样式，以及数据的可视化展示。

- 使用axios库，实现与后端的数据交互，以及文件的上传和下载功能。

- 使用webpack打包工具，优化前端代码的性能和兼容性。

- 使用jest和cypress等测试工具，进行单元测试和端到端测试，保证前端功能的正确性和稳定性。

- 使用nginx服务器，部署前端项目到云端，提供访问地址和接口文档。

- 逻辑控制组件的开发计划：

- 使用java语言，结合spring boot和spring cloud等框架，实现微服务架构，将不同的功能模块拆分为不同的服务，提高系统的可扩展性和可维护性。

- 使用antlr等工具，实现对多种编程语言的源码的解析，提取抽象语法树，分析代码中的实体、依赖、接口等信息。

- 使用gitlab api等工具，实现对git仓库的爬取，获取commit信息，分析每次提交的变更内容，提取重构操作、issue信息和bug信息，将这些信息与代码分析结果进行关联，形成一个完整的版本历史分析报告。

- 使用spark等工具，实现对大规模项目的扫描，利用多线程或者分布式计算的方式，提高扫描效率和准确度，同时也需要考虑内存和网络的限制，避免出现资源不足或者超时的情况。

- 使用junit和mockito等测试工具，进行单元测试和集成测试，保证后端功能的正确性和稳定性。

- 使用docker和kubernetes等工具，部署后端服务到云端，提供访问地址和接口文档。

- 数据存储组件的开发计划：

- 使用mysql数据库，存储代码分析结果和版本历史分析报告，设计合理的数据表和索引，优化数据的查询和更新性能。

- 使用redis数据库，存储用户的会话信息和缓存数据，提高系统的响应速度和并发能力。

- 使用mongodb数据库，存储扫描日志和度量统计，利用其灵活的文档结构，方便数据的存储和检索。

- 使用mybatis和spring data等工具，实现与后端的数据交互，简化数据操作的代码和配置。

- 使用dbunit和flyway等测试工具，进行数据库的测试和迁移，保证数据的正确性和一致性。

- 使用阿里云等云服务商，部署数据库到云端，提供访问地址和接口文档。

- 可能遇到的风险和挑战：

- 对多种编程语言的源码的解析可能存在一些兼容性和准确性的问题，需要不断地测试和优化。

- 对git仓库的爬取可能受到网络和权限的限制，需要考虑使用代理或者授权的方式，或者提供本地上传的选项。

- 对大规模项目的扫描可能消耗大量的资源和时间，需要考虑使用分布式计算或者异步处理的方式，或者提供扫描进度和中断的选项。

- 对数据的存储和交互可能存在一些安全性和可靠性的问题，需要考虑使用加密或者备份的方式，或者提供数据恢复和清理的选项。

- 对前后端的交互可能存在一些兼容性和稳定性的问题，需要考虑使用统一的数据格式和错误处理的方式，或者提供友好的提示和反馈的选项。

1. 架构设计结果
2. 架构风格

ArchGov采用三层架构风格和微服务架构风格，将前端视图、逻辑控制和数据存储分开。

1. 架构可视化

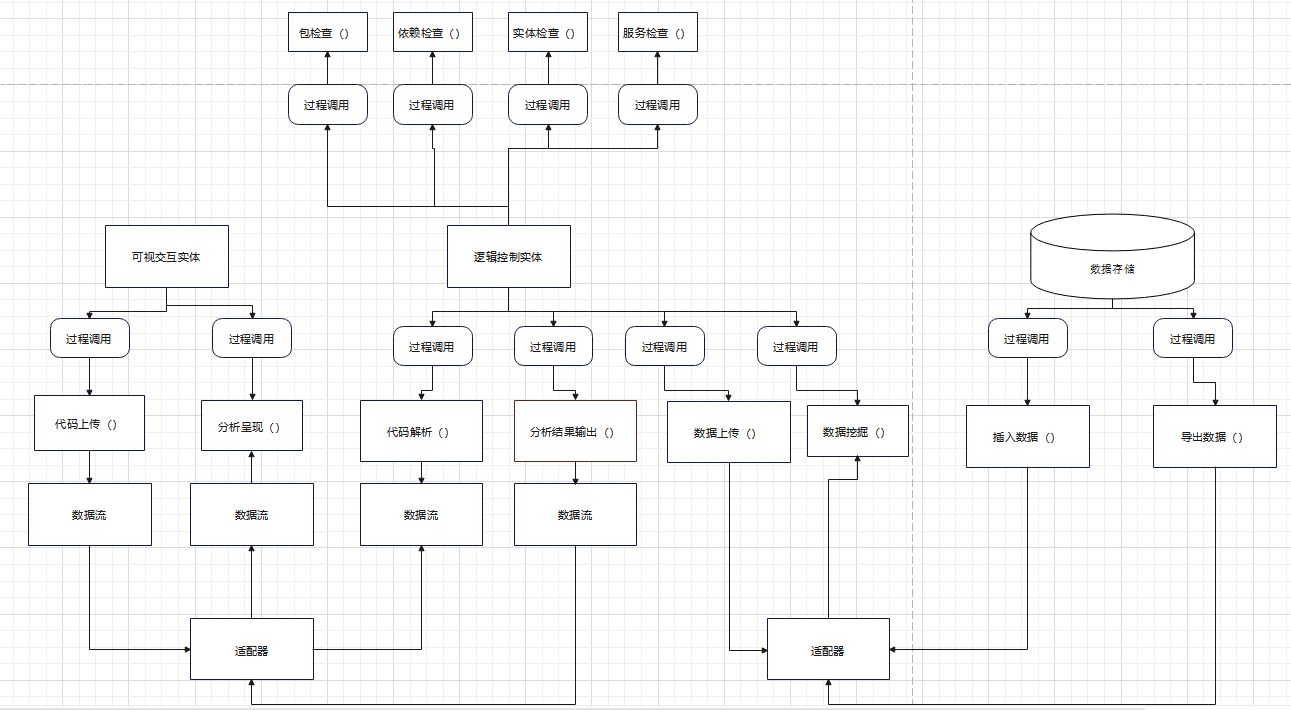


图 4 ArchGov架构图

1. 组件介绍

该架构主要包括三个组件：视图组件、逻辑组件和状态组件。顾名思义，视图组件主要负责对外交互和结果呈现，它对外提供的API应为代码项目、代码文件或者git地址，它所需要的API应为数据流文件；逻辑组件主要负责对软件项目的解析，它对外提供的API应为经过解析后的抽象语法树，它所需要的API应为解析后生成的分析文件；状态组件负责数据的存储，它所提供的API为一定格式的分析报告。

1. 连接件介绍

该架构中主要采用了过程调用连接器、流连接器和适配器连接器。首先是实体对函数调用、参数传递的要求比较常见，选用流连接器；其次不同实体之间需要频繁传输数据，且它们之间数据格式的要求存在不同，因此采用流连接器和适配器连接器进行转换。

- 过程调用连接器主要承担coordination；

- 流连接器主要承担communication和facilitation；

- 适配器主要承担conversion。

1. 架构评估

视图组件与流连接件、过程调用连接件实现可视化；

逻辑控制组件与过程调用连接件共同实现代码分析和版本历史挖掘；

视图组件与适配器连接件、逻辑控制组件共同实现分析报告的输出。

1. 架构缺陷

该系统对于版本历史挖掘的具体实现部分没有制定好相对明确的概念，对于这部分信息的处理和使用需要更加细致的规划设计。例如，如何识别和提取重构操作、issue信息和bug信息，如何与代码分析结果进行关联，如何评估版本之间的质量变化等。此外，该系统还需要考虑一些非功能性的需求，比如安全性、可用性、可扩展性等，以及如何进行性能优化和容错处理等。以下是我们发现的一些架构缺陷：

- 视图组件与逻辑组件之间的通信可能存在一些延迟和不稳定的情况，需要考虑使用消息队列或者WebSocket等技术，提高通信的效率和可靠性。

- 逻辑组件中的微服务可能存在一些服务发现和负载均衡的问题，需要考虑使用Eureka或者Zookeeper等工具，实现服务的注册和发现，以及使用Ribbon或者Feign等工具，实现服务的负载均衡和熔断。

- 状态组件中的数据库可能存在一些数据一致性和冗余性的问题，需要考虑使用分布式事务或者最终一致性等方案，保证数据的一致性，以及使用数据清洗或者去重等方案，减少数据的冗余。