项目文档

github链接： GitHub - JiCode-TJSE/JiCode-backend: JiCode后端代码

GitHub - JiCode-TJSE/JiCode-frontend

演示视频链接： 同济大学软件学院微服务架构项目演示视频 JiCode敏捷开发管理工具_哔哩哔哩_bilibili

一、需求分析

软件研发过程管理，是一条复杂的管理链条，在当今的商业环境中，敏捷项目管理 方法已经成为了高效的项目管理方式之一。敏捷方法特征包括：强调小规模的可交付结 果为导向，采用迭代和增量的开放方式，鼓励与客户紧密联系以理解需求、获取反馈， 具有高度适应性和灵活性。因此，越来越多的团队和组织采用敏捷方法组织、协调和管理项目。在敏捷开发方式当中，软件是通过一系列的小型迭代开发的，团队成员可以将多个工作项移入一次迭代当中完成，每个迭代都会产生一个可工作的软件版本。

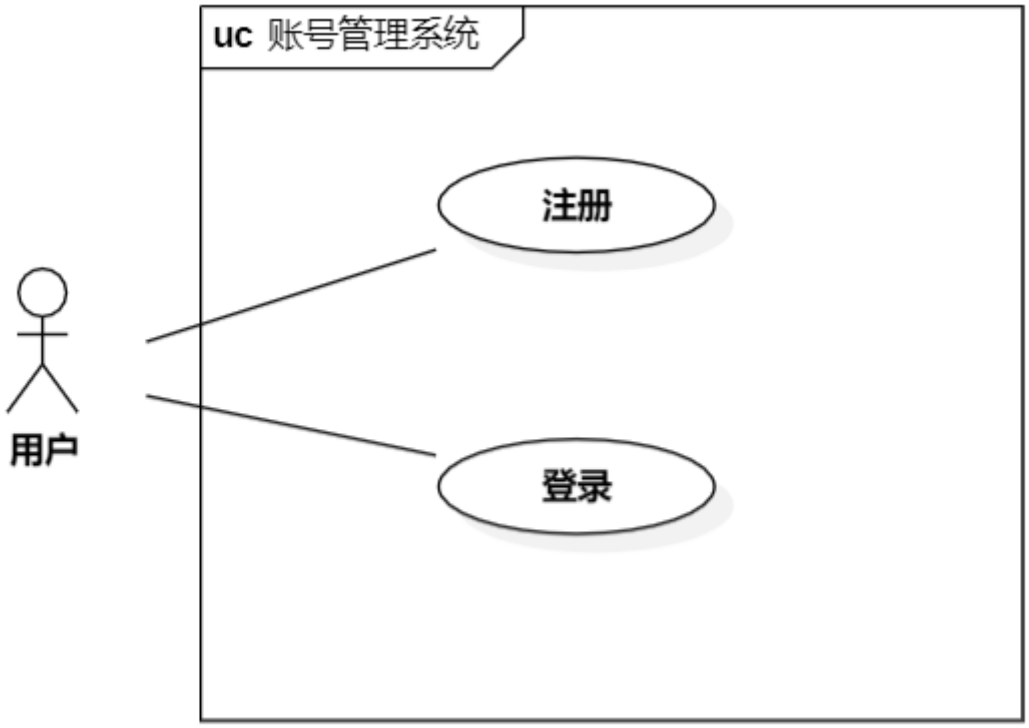
JiCode是一个管理Scrum敏捷开发方法的平台，具有账号管理、项目管理、产品管理三个模块功能，确保工具提供敏捷项目管理的核心功能，如需求获取、迭代管理、团队协作等，帮助产研团队：

- 快速建立标准化、规范化研发管理工作流程，有节奏的持续交付价值；
- 轻松规划和应对需求变化，提高项目可预测性，降低风险；
- 提升项目信息透明度，协作更顺畅，过程统计和改进有据可依。

通过小步快跑的方式，能够有效帮助企业持续规划和交付，打破协作壁垒和信息孤岛，让项目管理更加轻松且高效。

功能性需求

账号管理子系统

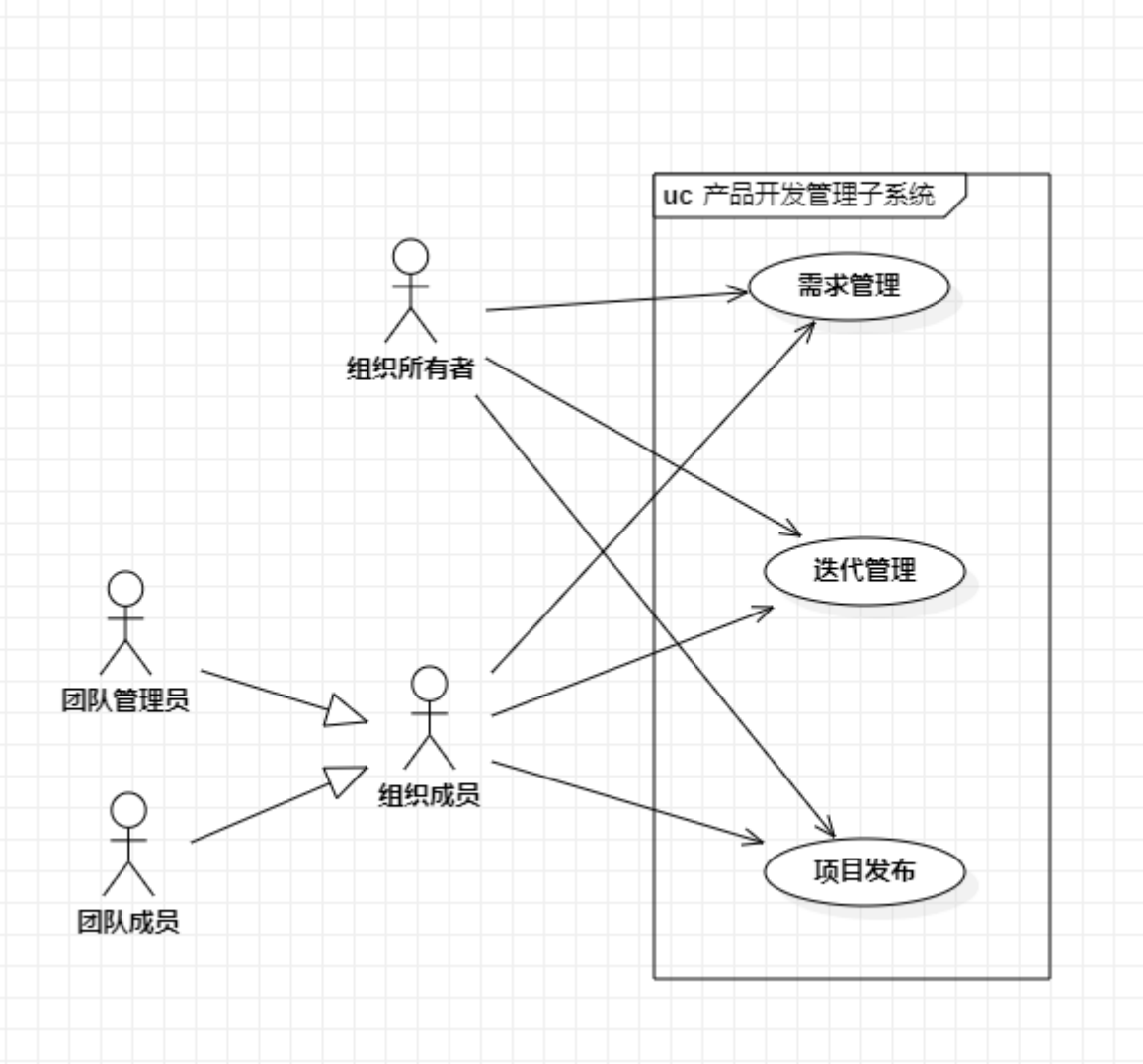


名称	注册
编号	UC01
说明	用户可以用邮箱进行账号的注册
参与者	用户
频度	中
状态	进行中
前置条件	无
基本事件流	用户进入注册页面，可能有以下操作： 为组织注册： a. 填写邮箱：用户填写自己的邮箱 b. 修改个人昵称和头像等基本信息 c. 设置用户名和密码 添加成员时为成员注册： a. 输入姓名，用户名，邮箱，初始密码等信息
扩展事件流	1. 若该邮箱在该团队当中已经被注册了，则显示错误信息

后置条件	1. 成功注册账号

名称	登录
编号	UC02
说明	用户使用用户名和密码进行登录
参与者	用户
频度	高
状态	进行中
前置条件	组织所有者在添加成员时已经使用这个邮箱注册了账号
基本事件流	用户进入登录页面，可能有以下操作： 1. 输入用户名： 用户输入自己的用户名 2. 输入密码： 用户输入自己的密码
扩展事件流	1. 若密码错误，则将错误显示给用户并提示用户需要重新输入 2. 若用户选择忘记密码，则调用邮箱发送验证码 3. 若用户是初次登录（组织管理员添加账号），则需要重置密码 4. 若用户选择修改信息，则对自己的个人资料和账号信息进行修改
后置条件	1. 用户成功登录 2. 用户成功修改自己的密码

项目管理子系统



名称	需求管理
编号	UC03
说明	对团队当前项目需求进行管理，包括与迭代、发布同步进行规划，做好需求的版本记录等
参与者	团队所有成员
频度	高
状态	进行中
前置条件	登录账号为该团队成员，且成功创建项目
基本事件流	团队成员进入需求管理，可能有以下操作： 1. 创建用户故事，填写用户故事信息，确定新建成功 2. 用户故事之间可以进行关联。 3. 将用户故事移入一次迭代或者一次发布当中。
扩展事件流	

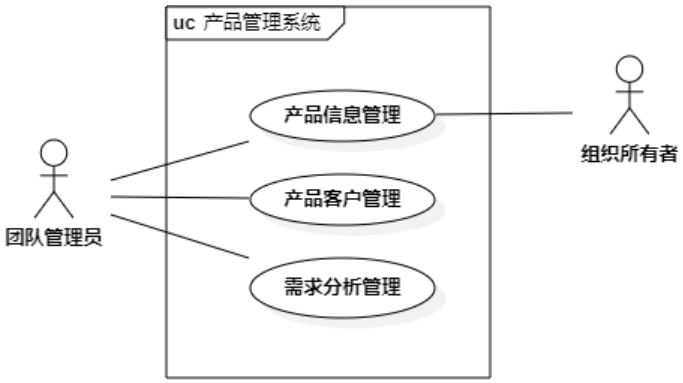
后置条件	需求相关信息的更新

名称	**迭代管理**
编号	UC04
说明	管理项目迭代相关信息，是敏捷开发中普遍的项目管理方式
参与者	团队所有成员
频度	高
状态	进行中
前置条件	用户登录为团队成员，项目开始迭代开发
基本事件流	团队成员进入迭代管理，可能有以下操作： 1. 新建迭代，输入迭代相关信息及目标后确认，成功新建迭代 2. 需求、迭代、发布关联，选择该项目各工作项相互关联
扩展事件流	工作项完成或者被误操作移入迭代管理时，用户对工作项重新规划。
后置条件	项目迭代更新，满足团队项目目前状态

名称	**项目发布**
编号	UC05
说明	设置团队规划的未来发布信息，便于团队做版本记录、发布阶段监管等
参与者	团队所有成员
频度	低
状态	进行中
前置条件	用户登录为团队成员，项目计划开始发布
基本事件流	团队成员进入项目发布，可能有以下操作： 1. 修改发布阶段，设置发布时间、状态，确认团队项目目前阶段 2. 需求、迭代、发布关联，选择该项目各工作项相互关联

扩展事件流	发布阶段设置错误或时间错误时，可以选择之前的发布阶段，重新选择时间和发布阶段
后置条件	记录团队项目目前发布阶段

产品管理子系统



名称	产品信息管理
编号	UC06
说明	产品经理可以管理所有的产品以及指定产品基础信息。
参与者	产品经理
频度	低
状态	进行中
前置条件	以产品经理身份登录并进入产品管理界面
基本事件流	产品经理进入产品管理界面，可能有以下操作： 1. 新建产品：填写产品基础信息即可新建 2. 删除产品：可以在产品管理界面删除指定的产品 3. 查看产品信息：选择进入指定产品的页面即可查看 4. 修改产品信息：选择进入指定产品的页面可以修改产品的基础信息
扩展事件流	1. 删除产品等敏感操作时，需要进行额外的邮箱验证确认 2. 新建产品时产品标识不能重复，重复了会弹窗提醒
	产品经理

后置条件	1. 成功新建了产品 2. 成功修改了产品信息 3. 成功删除了产品

名称	**产品客户管理**
编号	UC07
说明	产品经理可以记录产品用户的信息，包括联系方式、行业、分类等基本信息
参与者	产品经理
频度	低
状态	进行中
前置条件	以产品经理身份登录并进入产品管理界面，并选择一个产品进入指定产品的管理界面，点击客户板块
基本事件流	产品经理进入指定产品的管理界面，可能有以下操作： 1. 新建客户：填写客户基本信息新建 2. 修改客户信息：点进一个客户，可以修改其基本信息 3. 查看客户信息：点进一个用户即可查看基本信息 4. 删除客户：可以删除已建立的用户信息 5. 客户可视化：可以根据用户的分类等可视化分析
扩展事件流	1. 删除客户等敏感操作时，需要进行额外的邮箱验证确认 2. 客户标识不能重复，重复了会弹窗提醒
后置条件	产品经理： 1. 成功新建需求 2. 成功管理需求 3. 成功删除需求

名称	**需求分析管理**
编号	UC08
说明	产品经理可以记录产品的需求规划，包括状态、优先级、需求来源、工作量、价值等
参与者	产品经理

频度	中
状态	进行中
前置条件	以产品经理身份登录并进入产品管理界面，并选择一个产品进入指定产品的管理界面，点击需求板块
基本事件流	产品经理进入指定产品的管理界面，可能有以下操作： 1. 新建需求：填写需求的基本信息新建 2. 修改需求信息：点进一个需求，可以修改其信息 3. 查看需求需求：点进一个需求即可查看 4. 删除需求：可以删除已建立的需求 5. 需求可视化：可以查看已建立需求的可视化统计图标
扩展事件流	1. 删除需求等敏感操作时，需要进行额外的邮箱验证确认 2. 需求标识不能重复，重复了会弹窗提醒
后置条件	产品经理： 1. 成功新建需求 2. 成功修改需求 4. 成功删除需求

非功能性需求

1. 安全需求

敏捷开发项目管理工具需要具有安全性，确保项目数据和用户信息的保密性、完整性和可靠性，包括以下方面：

身份验证：系统应该有一个强大的身份验证机制，确保只有经过授权的用户才能访问系统。这包括用户名和密码验证，以及二次验证（如邮箱验证码或者安全令牌）。

访问控制：系统应该有一个详细的访问控制策略，确保用户只能访问他们应该访问的数据和功能。这包括基于角色的访问控制（RBAC）或者基于属性的访问控制（ABAC）。

数据保护：系统应该保护存储和传输的数据的安全。这包括数据加密（包括数据库内容加密），以及使用安全的协议（如HTTPS）来保护数据在网络中的传输。

审计和日志：系统应该记录重要操作，以便在发生安全事件时进行审计和调查。日志应该包括足够的信息，但不应该包括敏感数据。

错误处理和异常管理：系统应该能够正确地处理错误和异常，防止它们导致安全漏洞。系统应该避免向用户显示过多的内部信息，以防止信息泄露。

安全更新和补丁管理：系统应该定期接收和安装安全更新和补丁，以保护系统免受已知的安全威胁。

备份和恢复：系统应该定期备份数据，并有一个恢复计划，以防止数据丢失或者系统故障。

2. 可维护性和可扩展性

敏捷开发项目管理工具需要具备良好的可维护性和可扩展性，以便能够方便地进行功能扩展和系统更新。这包括清晰的代码结构、模块化设计、易于理解和修改的代码，以及支持插件和扩展的架构设计。

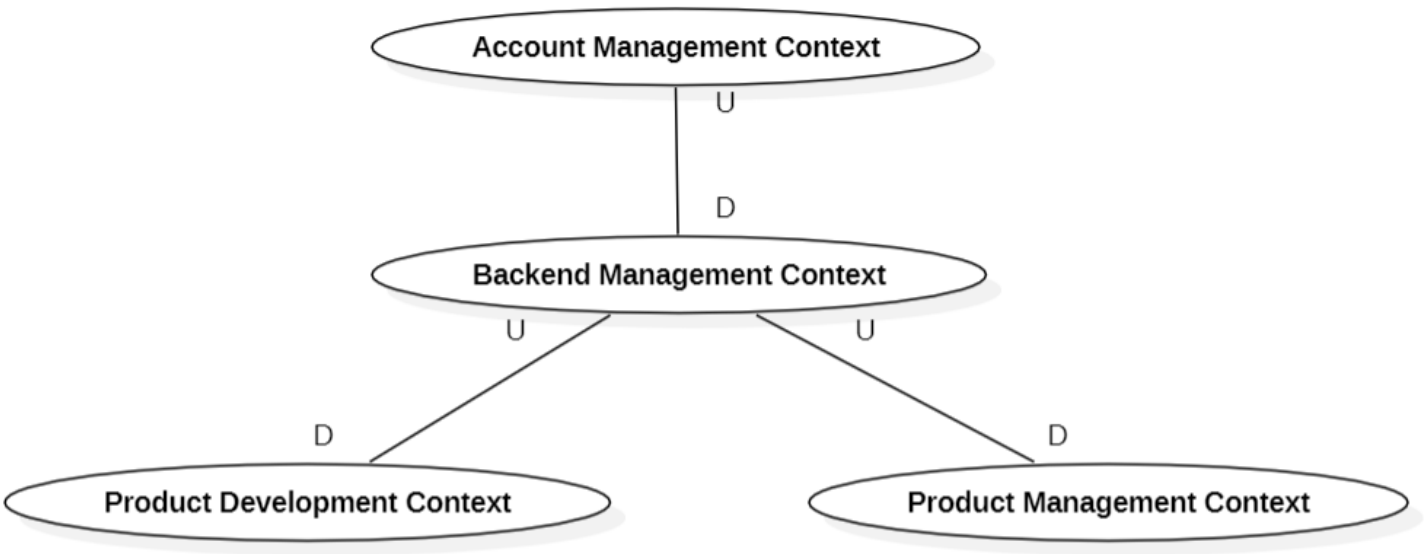
二、系统设计

本项目当中，我们使用领域驱动设计的理念，对具体的业务逻辑进行划分，创建领域模型和定义领域语言，为每一个领域模型进行软件架构的设计，包含实体、值对象、聚合和领域事件等，最后进行领域模型的实现和持续改进。

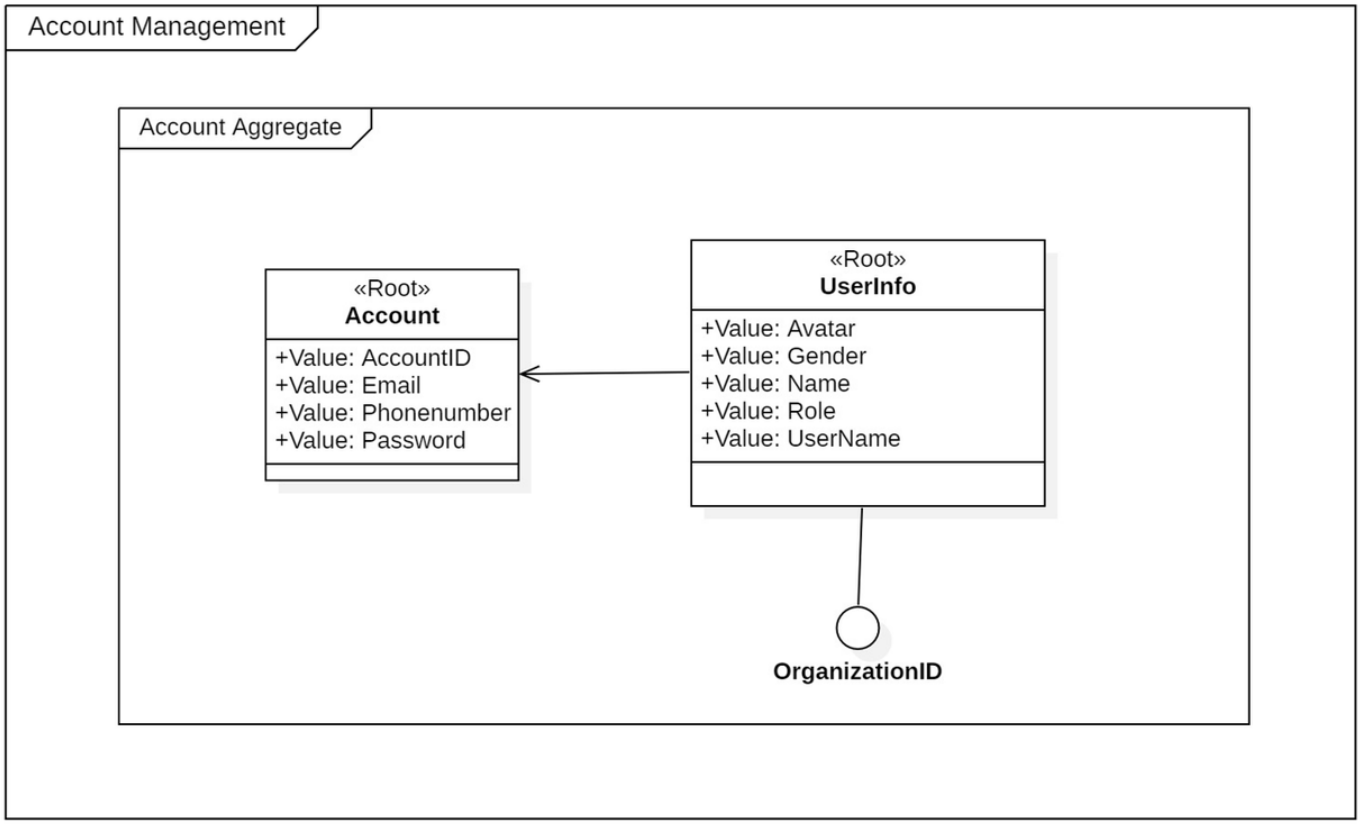
领域划分

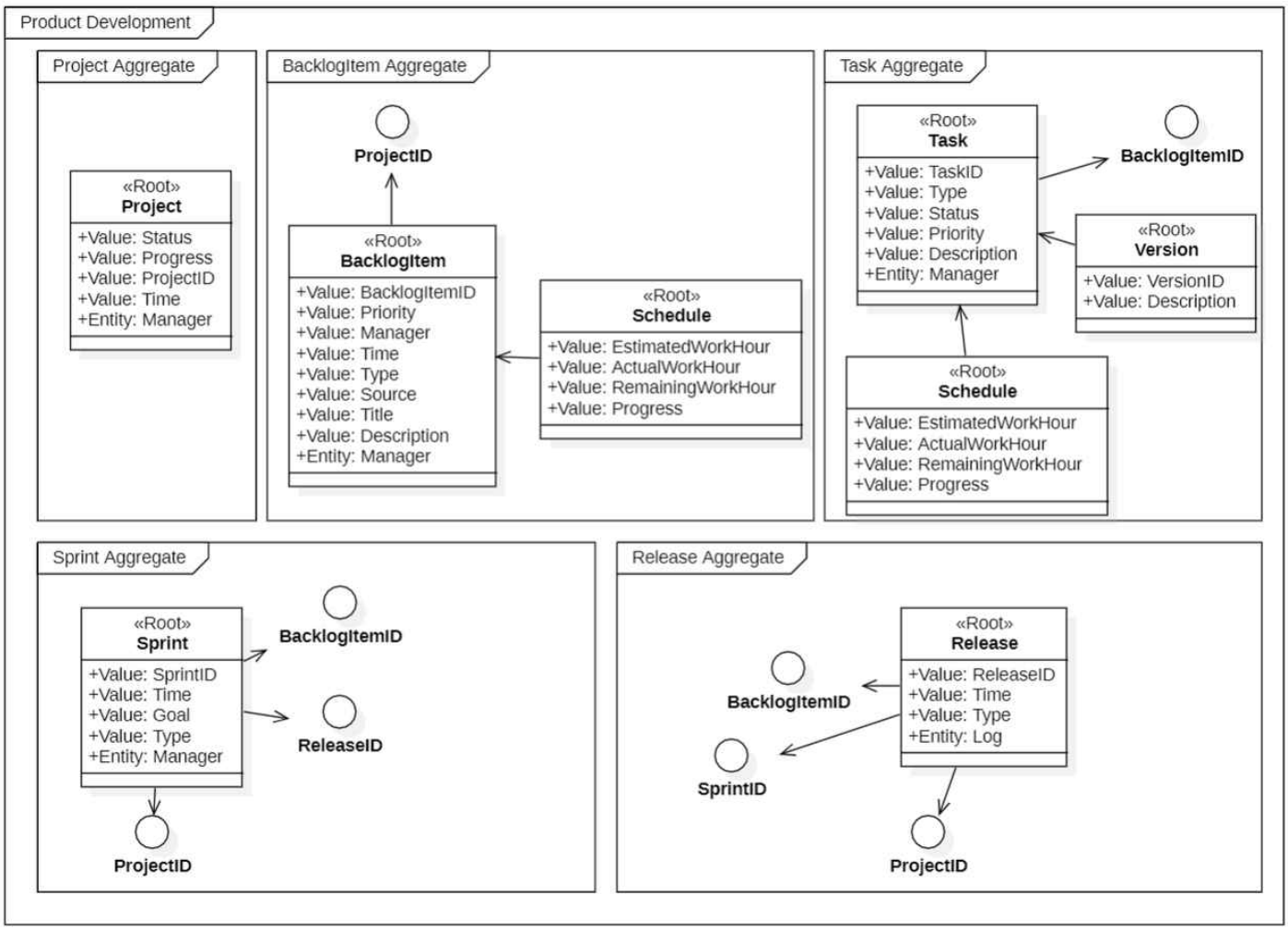
- 核心子域：项目管理
- 支撑子域：需求管理、项目迭代、产品发布
- 通用子域：个人信息

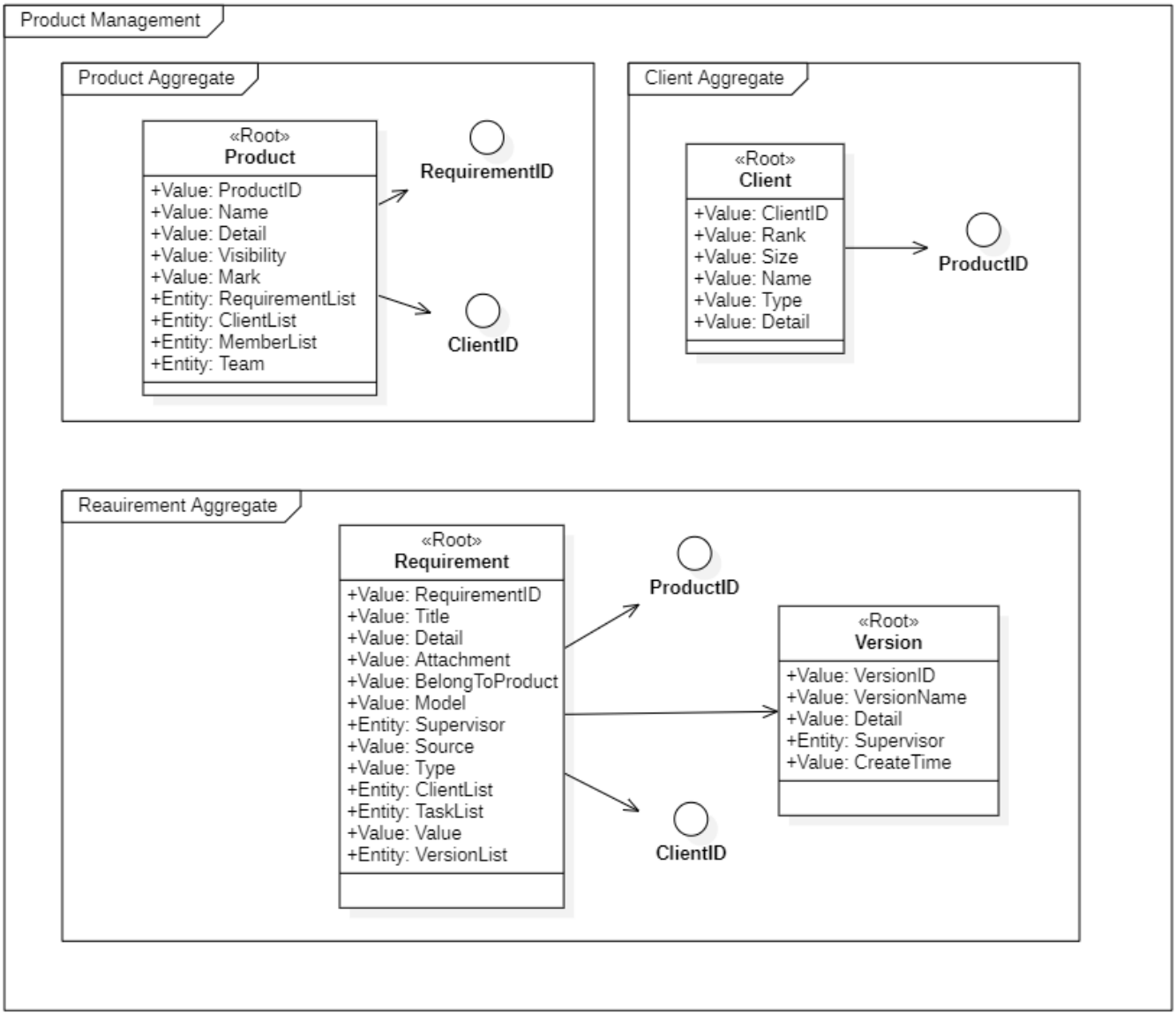
限界上下文划分



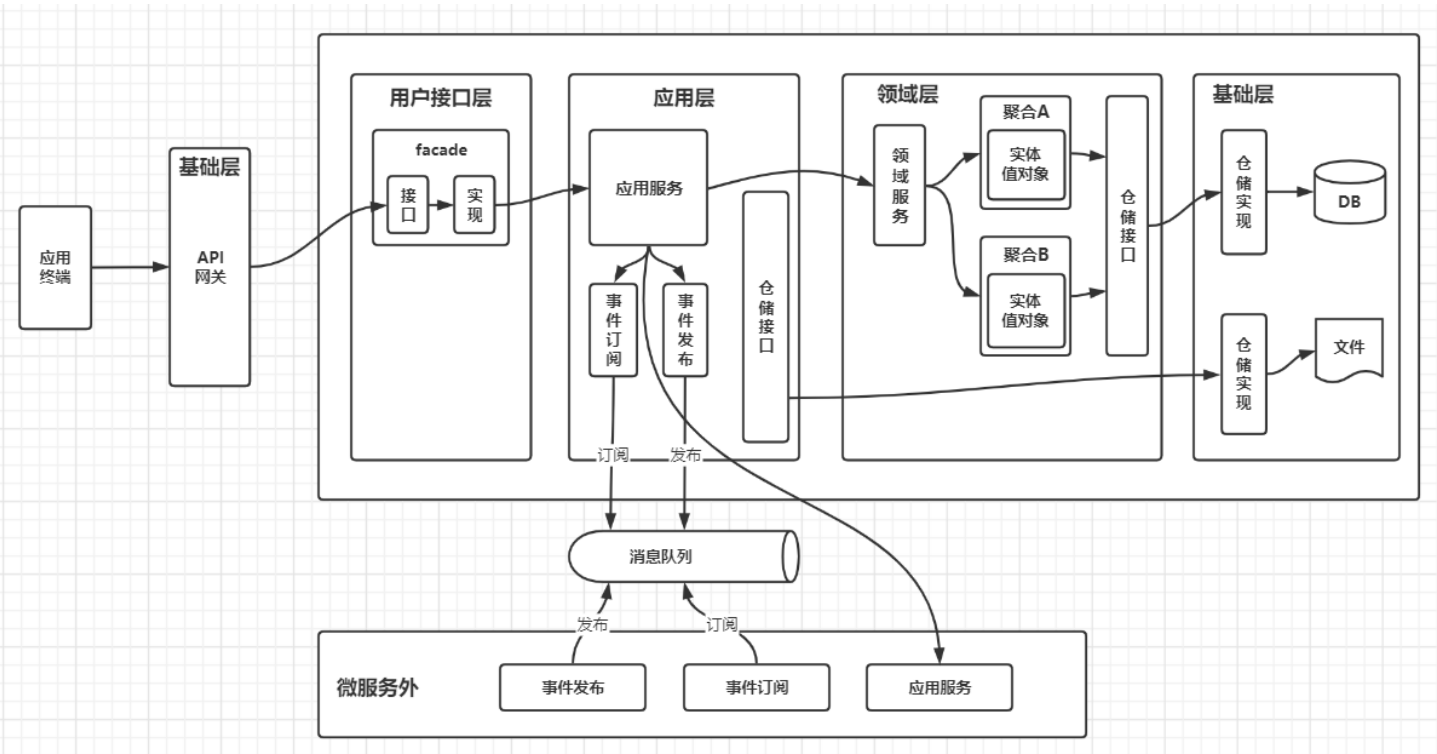
领域模型


Account Management
Account Aggregate
«Root»
Account
+Value: AccountID
+Value: Email
+Value: Phonenumber
+Value: Password
«Root»
UserInfo
+Value: Avatar
+Value: Gender
+Value: Name
+Value: Role
+Value: UserName
OrganizationID


Product Development
Project Aggregate
«Root»
Project
+Value: Status
+Value: Progress
+Value: ProjectID
+Value: Time
+Entity: Manager
BacklogItem Aggregate
ProjectID
«Root»
BacklogItem
+Value: BacklogItemID
+Value: Priority
+Value: Manager
+Value: Time
+Value: Type
+Value: Source
+Value: Title
+Value: Description
+Entity: Manager
«Root»
Schedule
+Value: EstimatedWorkHour
+Value: ActualWorkHour
+Value: RemainingWorkHour
+Value: Progress
Task Aggregate
«Root»
Task
+Value: TaskID
+Value: Type
+Value: Status
+Value: Priority
+Value: Description
+Entity: Manager
BacklogItemID
«Root»
Version
+Value: VersionID
+Value: Description
«Root»
Schedule
+Value: EstimatedWorkHour
+Value: ActualWorkHour
+Value: RemainingWorkHour
+Value: Progress
Sprint Aggregate
«Root»
Sprint
+Value: SprintID
+Value: Time
+Value: Goal
+Value: Type
+Entity: Manager
BacklogItemID
ReleaseID
ProjectID
Release Aggregate
«Root»
Release
+Value: ReleaseID
+Value: Time
+Value: Type
+Entity: Log
BacklogItemID
SprintID
ProjectID


Product Management
Product Aggregate
«Root»
Product
+Value: ProductID
+Value: Name
+Value: Detail
+Value: Visibility
+Value: Mark
+Entity: RequirementList
+Entity: ClientList
+Entity: MemberList
+Entity: Team
RequirementID
ClientID
Client Aggregate
«Root»
Client
+Value: ClientID
+Value: Rank
+Value: Size
+Value: Name
+Value: Type
+Value: Detail
ProductID
Reauirement Aggregate
«Root»
Requirement
+Value: RequirementID
+Value: Title
+Value: Detail
+Value: Attachment
+Value: BelongToProduct
+Value: Model
+Entity: Supervisor
+Value: Source
+Value: Type
+Entity: ClientList
+Entity: TaskList
+Value: Value
+Entity: VersionList
ProductID
ClientID
«Root»
Version
+Value: VersionID
+Value: VersionName
+Value: Detail
+Entity: Supervisor
+Value: CreateTime

三、架构描述


应用终端
基础层
API网关
用户接口层
facade
接口
实现
应用层
应用服务
事件订阅
事件发布
仓储接口
领域层
领域服务
聚合A
实体值对象
聚合B
实体值对象
仓储接口
基础层
仓储实现
DB
仓储实现
文件
订阅
发布
消息队列
发布
订阅
微服务外
事件发布
事件订阅
应用服务

项目整体采用springboot+vue进行开发，使用Spring Cloud作为微服务框架，基于springboot并提供了一系列组件和工具，用于构建分布式系统和微服务架构。

使用MySQL作为数据库，并搭建了主从集群。使用Mybatis作为持久层框架，提供了灵活的SQL映射配置和强大的SQL执行能力，通过与数据库进行交互实现高效的数据访问和操作。

使用Nacos进行服务注册和发现，提供了服务注册、服务发现和健康检查等功能，可以帮助微服务实现自动化的服务注册和发现。

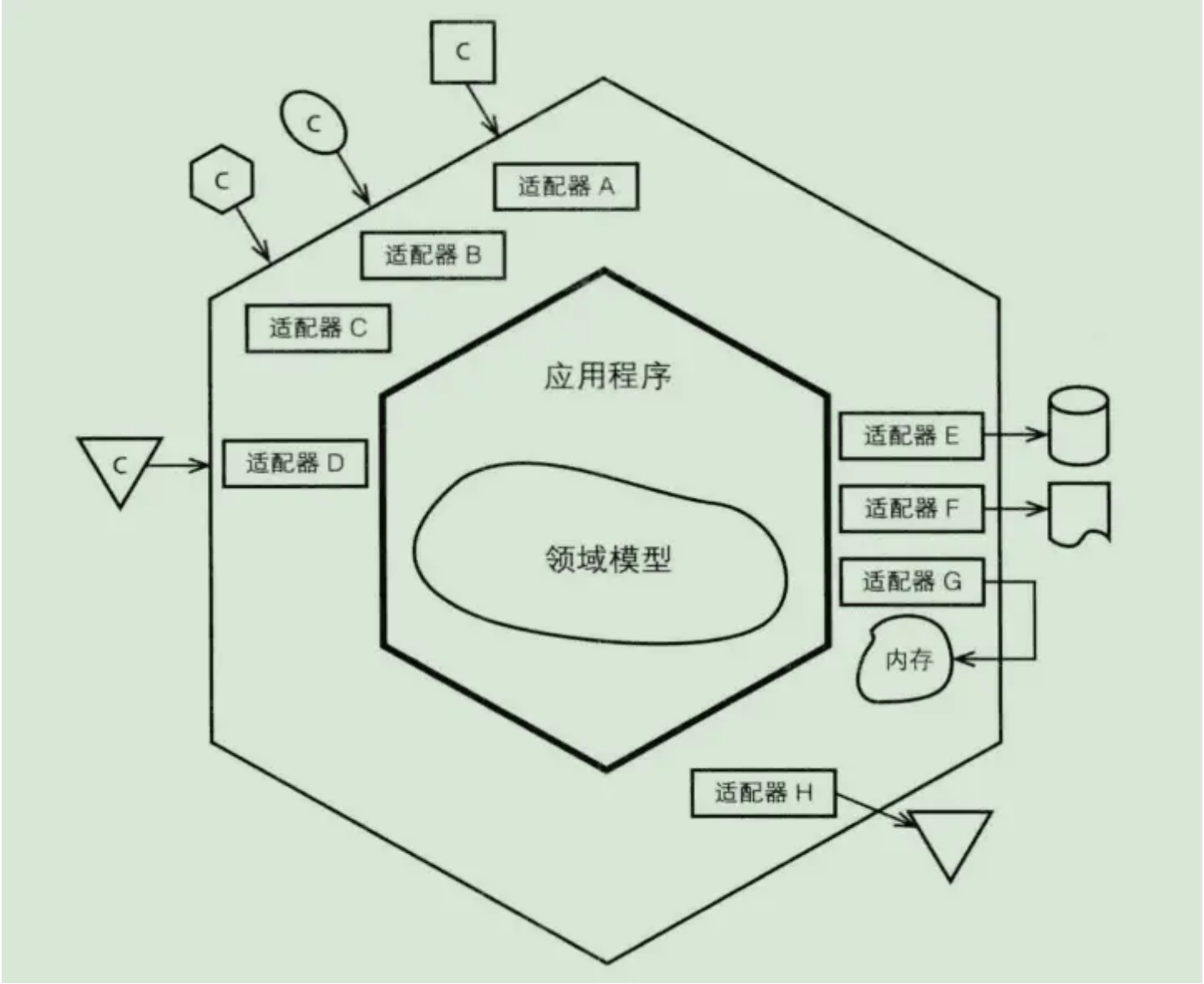
在网络/数据安全性方面使用了ngnix、数据库敏感数据哈希加密、ssl网络通信加密。

在CICD方面使用Jenkins+github的方式进行部署和交付。（详情参照《部署文档》）

项目采用六边形架构，将应用程序的业务逻辑与外部关注点（如用户界面，数据库，网络等）分离，使得应用程序能够独立于这些外部关注点进行开发和测试。使用六边形架构可以将应用程序核心和外部关注点进行解耦。

六边形架构的主要组成部分有：

1. **应用程序核心**：这是应用程序的中心，包含所有的业务逻辑。它不依赖于任何外部关注点，只依赖于抽象的端口。
2. **端口**：端口是应用程序核心与外部关注点之间的接口。它们定义了应用程序核心需要的服务（驱动端口），以及应用程序核心提供的服务（驱动端口）。
3. **适配器**：适配器是端口的具体实现。它们将应用程序核心与外部关注点连接起来。例如，一个适配器可能将应用程序核心与一个特定的数据库连接起来，另一个适配器可能将应用程序核心与一个特定的用户界面连接起来。

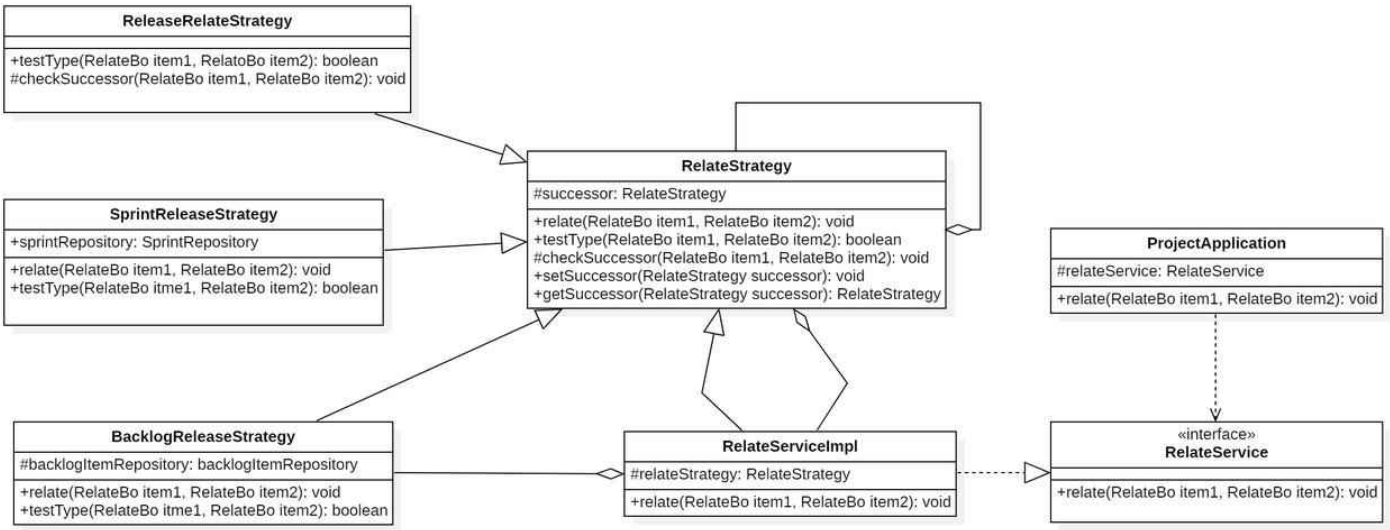


四、解决方案

1. 设计模式

在项目实现的过程中使用了单例模型、工厂模式、责任链模式、策略模式、脏标记模式等，体现在架构设计以及具体业务逻辑实现当中。

在关联业务逻辑的实现当中，工作项可以与工作项、迭代和发布进行关联，在这里使用了策略模式、代理模式，实现灵活、可扩展且可控制的处理流程，允许在运行时选择不同的策略，控制和管理对对象的访问，并实现链式的处理逻辑。



2. 基础设施

Starter公共依赖包

通过使用自己编写的Starter公共依赖包，我们实现了Spring Boot数据库的主从分离读写和数据源的自动配置。我们将该包发布在GitHub仓库中，以便让所有微服务可以将其添加到本地Maven仓库中，并享受以下好处：

1. 便捷的配置管理：通过将Starter依赖包发布在GitHub仓库中，团队成员可以方便地将其添加到他们的项目中，并从本地Maven仓库中获取依赖。这样可以简化项目的配置管理，并提高开发效率。
2. 统一的数据库主从分离读写配置：Starter依赖包提供了自动配置和管理数据库的主从分离读写的功能。通过将该Starter应用到微服务中，团队可以实现统一的数据库配置，而无需在每个微服务中单独配置数据库的主从分离读写。这样可以减少配置错误的可能性，并提高数据库配置的一致性和可维护性。
3. 降低开发复杂性：Starter依赖包提供了封装和抽象，隐藏了数据库主从分离读写的底层细节。团队成员只需要在项目中添加依赖并进行简单的配置，就能够轻松地使用该功能，而无需深入了解主从分离读写的复杂性。这简化了开发人员的工作，提高了开发效率。
4. 提高代码重用性：通过将Starter依赖包发布在GitHub仓库中，其他团队成员可以在需要的时候轻松地使用它。这可以促进代码重用，避免重复编写类似的功能和配置代码。这样一来，团队可以更加专注于业务逻辑的开发，而不必花费过多时间和精力在基础设施的搭建和配置上。

Maven父子项目

Maven的父子项目结构是一种组织和管理多个相关项目的方式。在这种结构中，一个项目充当父项目，其他项目作为子项目。父项目可以定义公共依赖、插件配置和构建设置。子项目可以继承这些配置，并且还可以自定义和覆盖某些配置。我们采用这样的结构来管理我们的各个微服务。

在这个结构中，每个微服务都有一个共同的父项目，即项目的主分支。同时，每个微服务也作为父项目的子项目。这意味着每个微服务的分支都是从主分支继承而来的，并且在主分支有更新时，会与主分支合并以更新父项目。

这种项目结构有以下几个好处：

1. 代码重用和模块化：父子项目结构允许将相关的功能和模块组织在一起，并在不同的子项目中进行代码重用。这样可以提高代码的可维护性和可复用性，减少代码冗余。
2. 依赖管理和版本控制：父项目可以定义公共的依赖项和版本号，子项目可以继承这些依赖项，避免在每个子项目中重复定义相同的依赖项。这样可以简化依赖管理，确保项目中使用的依赖项保持一致和更新。
3. 构建一致性：父项目可以定义构建的通用设置、插件配置和构建过程。子项目继承这些配置，确保构建过程的一致性。这样可以减少配置错误和构建问题，提高项目的可靠性和稳定性。
4. 统一的发布和部署：父项目可以定义公共的发布和部署配置，子项目可以继承这些配置。这样可以确保所有子项目都遵循相同的发布和部署流程，简化发布过程，提高效率。
5. 多模块管理和跨项目协作：父子项目结构可以方便地管理多个相关项目，使得团队成员可以更好地协同工作。每个子项目可以专注于自己的功能，同时可以在父项目的范围内进行整体协作和管理。

3. 聚合优化

聚合优化是在设计和实现大型聚合根时常用的一种策略，旨在提高系统的性能和可扩展性。具体来说，我们项目的聚合优化包括以下两个主要方面：

1. 子聚合和分离实体（Sub-aggregates & Separate entities）：大聚合根可以根据业务需求被分解为多个子聚合或实体。子聚合是更小、更具体的聚合根，负责处理一部分业务逻辑。通过将聚合根分解为子聚合，可以更好地适应更细、更小的业务场景。对于简单的业务操作或者不需要加载整个聚合根属性的读请求，可以选择只加载部分分离的实体，而不必加载整个聚合根。这样可以减少加载的数据量和数据库操作的开销，提高系统的响应速度。
2. 脏标记模式（Dirty Flag）：在大聚合根的更新操作中，通常采用脏标记模式。脏标记是一种标记记录是否已被更新的机制。在聚合中，每个实体或子聚合根都可以被标记为"脏"，表示其数据已被修改。当对整个聚合根进行更新操作时，只会更新被标记为"脏"的实体对应的数据库表。这种方式可以减少不必要的数据库操作，提高性能。此外，脏标记模式还可以延迟加载实体的属性，只有在需要访问具体属性时才执行加载操作，从而减少数据库查询的次数。

通过聚合优化，可以更好地处理大型聚合根的复杂性和高并发负载，提高系统的性能和可伸缩性。然而，聚合优化需要在设计和实现阶段仔细考虑业务需求和性能要求，以确保正确性和一致性。同时，也需要权衡聚合的复杂性和维护成本，避免过度优化导致代码复杂度的增加。