**租户与订阅**

**文档概述**

|  |  |
| --- | --- |
| 版本历史 | 1.0 |
| 参与人员 | fansea |
| 目标读者 | 产品经理，产品研发，产品运营 |

**背景与业务需求**

**概述**

在设计“租户与订阅”模块时，我们面临着一项独特且复杂的挑战：如何在SaaS系统架构中有效地支持多种用户类型（个人免费版、个人收费版、企业版），同时保证数据隔离和服务的个性化。这一挑战不仅涉及技术层面的实现，还关系到用户体验的优化和商业模型的可持续性。

此模块的核心目标是构建一个灵活且健壮的架构，能够适应不同用户群体的特定需求，同时保持代码的一致性和维护的简便性。个人免费版用户将接触到基础功能和有限的用量，这不仅满足了他们的基本需求，同时也为升级到付费版本创造潜在机会。对于企业版用户，我们的策略是提供更高级的功能和按席位收费的灵活性，以满足他们更为复杂的需求。

在技术层面，关键在于设计一个高效的用户管理和订阅管理系统，同时集成灵活的计费机制。这意味着系统需要能够处理各种不同的订阅方案，支持动态的功能访问控制，并确保每个租户的数据隔离和安全。此外，鉴于目标用户群体的多样性，从学生到企业，我们的系统设计还需要考虑易用性和可访问性，确保所有用户都能高效地利用我们的服务。

|  |
| --- |
| 因此，需要进行一次架构设计，决定后面产品的基本架构形态。 |

**业务背景**

**市场与用户需求**

在分子生物学领域，我们面临着多元化的用户群体，包括个人用户（学生、科研人员）和企业用户（公司、科学机构）。个人用户强调工具的便捷性和易用性，而企业用户则更重视项目管理和团队协作。这一多样化需求推动了我们对租户与订阅管理模块的架构设计，旨在通过单一的系统平衡这些不同的用户需求。

**架构设计驱动因素**

为满足这些复杂且多变的需求，我们采用了统一的代码基础来服务所有用户类型。这种策略要求我们的架构设计必须具备高度的灵活性和可扩展性，以支持不同用户之间的无缝切换和个性化服务。模块的设计重点在于实现有效的租户隔离和灵活的订阅管理机制，同时保持系统的整体性能和稳定性。

**长期业务视角**

这种架构设计不仅回应当前市场的未满足需求，而且为公司在分子生物学领域的长期战略定位打下基础。通过这种综合考虑用户需求和技术可行性的方法，我们旨在构建一个既能满足当前市场需求，又能适应未来发展的系统。

**目标用户**

**1、个人用户（如学生、科研人员）**

* 个人用户主要分为免费版和收费版，根据不同订阅类型所需要控制的维度，设计合适的架构。

**2、企业用户（如公司、教研室）**

* 架构需要支持更高级的数据管理和安全性要求，如数据隔离和角色基础的访问控制。
* 需要支持企业/项目的协作。
* 需要考虑扩展性和模块化设计，以便企业用户可以根据需求扩展或定制功能。
* 对系统的稳定性和可靠性有更高的要求，因此需要更强大的错误处理和故障恢复机制。
* 根据不同订阅类型所需要控制的维度，设计合适的架构。

**3、企业用户与个人用户的转换**

* 支持以下模式的转换

|  |  |  |
| --- | --- | --- |
| 转换前/转换后 | 个人版 | 企业版 |
| 个人版 | ✅ | ✅ |
| 企业版 | ❌ | ✅ |

* 系统需要灵活地支持用户的转变，这要求架构能够适应用户状态的变化，如权限调整和资源重新分配。
* 需要提供平滑的数据迁移和账户转换机制，以免在用户转变过程中造成数据丢失或服务中断。

**功能需求**

参照：[用户注册流程](https://zwer55qw58u.feishu.cn/wiki/S61kwDrz1iOjinkel2Ncjehenwg)

**1、订阅类型**

1. 个人免费用户：非盈利机构（科研院所、高校机构）个人用户
2. 个人付费用户：企业里以个人名义购买使用的用户
3. 企业用户：以企业、机构（包括非赢利机构）名义批量采购的用户

**2、注册流程**

1. 个人免费用户：
2. 注册
3. 点击链接激活账号
4. 试用
5. 试用到期前提交证明材料(材料：邮箱后缀)
6. 衍因运营人员审核
7. 设置有效期
8. 到期前重复d
9. 个人付费用户：
10. 注册
11. 点击链接激活账号
12. 试用
13. 到期前在线支付，系统延长有效期
14. 到期前系统提醒，重复d
15. 企业线下采购：
16. 系统演示
17. 签订合同
18. 衍因运营人员后台创建管理员账号，分配席位，设定有效期
19. 衍因开票
20. 企业付款
21. 到期前联系企业续费
22. 企业在线采购：
23. 注册企业账号（发票抬头）
24. 点击链接激活账号（默认3席位）
25. 试用
26. 试用到期前在线支付，系统延长有效期
27. 衍因开票
28. 到期前系统提醒续费，重复d
29. 个人付费用户升级为企业用户：
30. 系统里提交变更，录入企业信息
31. 按照企业账号最少席位数计算差价
32. 在线支付，完成变更
33. 衍因开票

**3、租户管理**

* 支持以上订阅类型，以及注册流程。

**假设与限制**

**假设**

1. 用户行为一致性：假设不同用户群体（个人用户和企业用户）的基本行为模式相对一致，使用统一的架构来满足他们的需求是可行的。
2. 市场环境稳定：假设市场环境在可预见的未来保持稳定，不会出现剧烈变化，影响用户需求。
3. 个人版和企业版只是大方向的划分，实际可能还会划分为个人免费版，个人收费版，企业XX版等不同运营策略。

**限制**

1. 付费模块的独立性：

* 虽然付费策略对于整个系统至关重要，但具体的付费策略和实施细节不包括在本文档的讨论范围内。付费模块将作为独立模块开发，其策略和实施与订阅方案可能不完全一致。

1. 订阅付费方式的补充：

* 线上付费能力将在后续的架构设计中详细讨论和实现，目前不包括在本阶段的讨论范围内*。*

1. 非核心功能的排除：

* 一些辅助性功能，如运营系统、推广工具或市场分析模块，并不属于租户与订阅管理模块的核心功能，因此这些内容不包括在当前的设计范围内。

1. 定制化解决方案的限制：

* 鉴于SaaS产品的特性及资源和时间限制，我们目前不打算为每个用户提供完全定制化的解决方案。

1. 扩展性的实际限制：

* 尽管我们的架构设计注重可扩展性，但在技术和资源上仍有一定限制。我们将在现有的资源和技术能力范围内最大化系统的可扩展性。

1. 集成范围的界定：

* 考虑到技术兼容性和优先级，我们将不包括所有潜在的第三方系统或服务（例如财务系统等）的集成在本次设计范围内。

**架构设计**

**架构决策**

在设计我们的系统时，我们考虑了以下三种架构选型以满足不同的业务需求：

**选型一：双架构策略**

* 定义：为个人版和企业版分别制作两套完全不同的架构。
* 特点：这种方法允许每个版本专注于特定用户群体的需求，但可能增加维护和开发的复杂性。

**选型二：混合架构策略**

* 定义：在业务功能层面采用统一架构，屏蔽个人用户和企业用户之间的差异；而在用户运营和管理层面使用不同的架构。
* 特点：这种方法平衡了统一性和灵活性，适合在不牺牲核心业务功能的情况下，针对不同用户群体提供定制化服务。

**选型三：标准化SaaS模式**

* 定义：所有版本（包括个人版和企业版）统一使用一套标准的SaaS架构，不区分个人用户和企业用户。
* 特点：这种方法强调了架构的统一性和简洁性，有利于减少开发和维护成本，同时提高系统的可扩展性和可维护性。

**决策**

深入分析上述背景和考虑以下关键因素，得出决策。

**用户需求的一致性**：我们没有发现明显的证据表明不同用户群体（个人用户与企业用户）之间的功能需求存在显著差异。这意味着一个统一的架构能够有效地满足所有用户的基本需求。

**当前产品发展阶段和资源考量**：考虑到我们产品目前所处的发展阶段，以及公司的规模和人力资源状况，采用一个统一的架构将更加合适。这将有助于我们集中资源，提高开发效率。

**后续维护和扩展性**：从长远来看，一个标准化的SaaS架构将更容易维护和扩展。这种架构的选择有助于未来的功能迭代和技术升级，同时也降低了长期的技术债务风险。

|  |
| --- |
| 决策使用「标准化SaaS模式」。 |

**基本架构设计**

基于以上的决策，进行一些基本设计。

**租户隔离模型**

|  |
| --- |
| 在我们的系统中，租户是实现数据隔离的基本单位。这种模型确保了即使在底层数据存储中没有物理上的隔离，用户也能体验到数据上的完全隔离。对用户来说，不同租户的数据在逻辑上是完全隔绝的，从而在保障数据安全和隐私的同时，提供了一个清晰、高效的数据管理环境。 |

**租户订阅模型**

我们的系统采用灵活的租户模型来区分个人版和企业版，以及它们之间的转换机制：

1. 个人版：

* 定义：在个人版中，单个用户被视为一个独立的租户。
* 特点：这样的设计允许每个个人用户在系统中拥有独立的空间和配置。

1. 企业版：

* 定义：在企业版中，一个公司或组织构成一个单独的租户。
* 特点：这使得企业能够在一个共享的环境中管理其员工和资源。

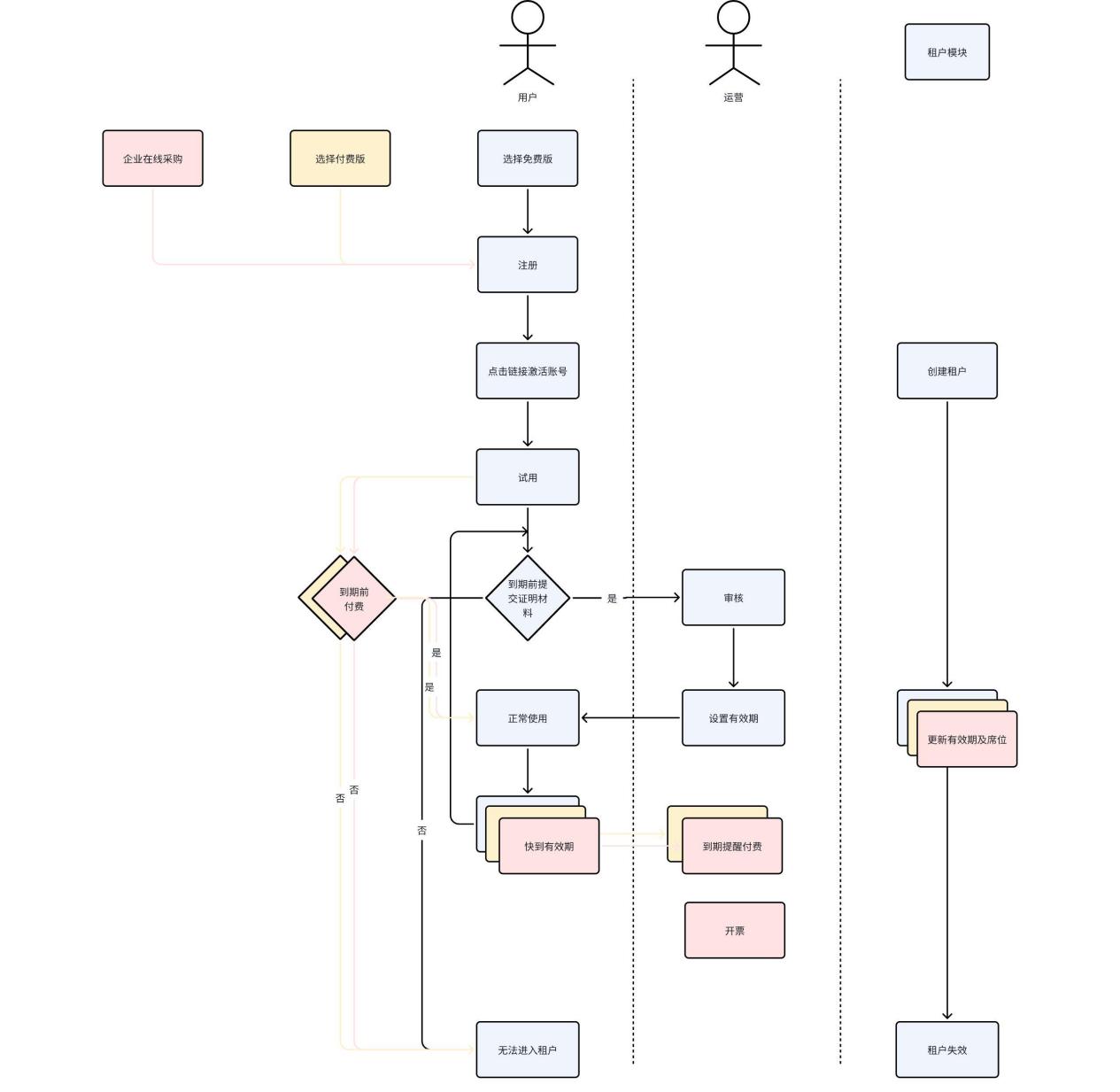
1. 版本之间的转换：

* 个人版到个人版：一般情况下，用户在个人版之间的转换仍旧保持在同一个租户内。具体的租户变更细节，在需求明确后由产品经理补充。
* 企业版到企业版：类似地，企业版用户在不同企业版之间的转换通常保持在同一个租户内。具体细节待后续需求确定。
* 个人版到企业版：这种转换涉及从一个租户到另一个租户的跨越。用户在登录后需要选择相应的租户。如有特殊需求，可以在转换过程中提供一次性的数据迁移或初始化服务。

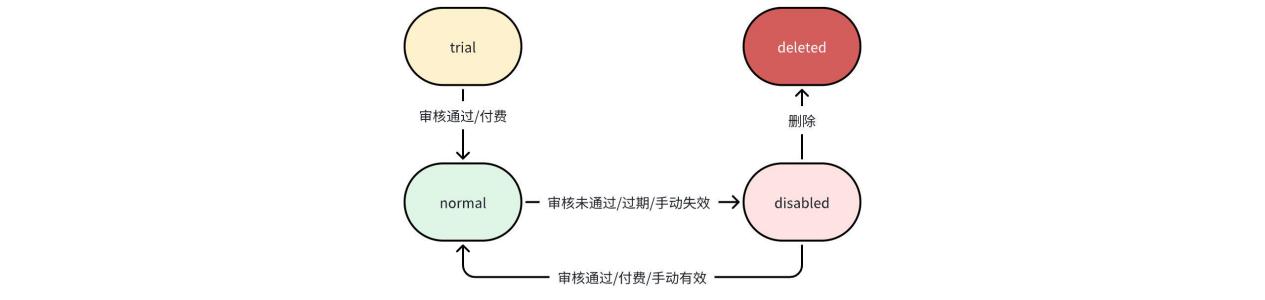
通过这种灵活的租户模型，我们的系统能够支持不同用户类型的需求，并且灵活地处理用户身份和权限的变更。

**租户生命周期**

梳理用户注册流程，如下图。



由此整理出租户的生命周期如下。



**订阅维度设计**

各个订阅类型对比如下。

|  |  |  |  |
| --- | --- | --- | --- |
| 维度/版本 | 个人免费版 | 个人收费版 | 企业版 |
| 模块级别（如角色管理） | ❌ | ❌ | ✅ |
| 功能级别（如文件-复制） | ❌ | ✅ | ✅ |
| 用量限制（如素材容量） | 1G | 5G | 20G |
| 成员限制（如坐席管理） | ❌ | ❌ | ✅ |
| 沟通方式 | 帮助文档 | 帮助文档  反馈与建议 | 帮助文档  反馈与建议  客服 |

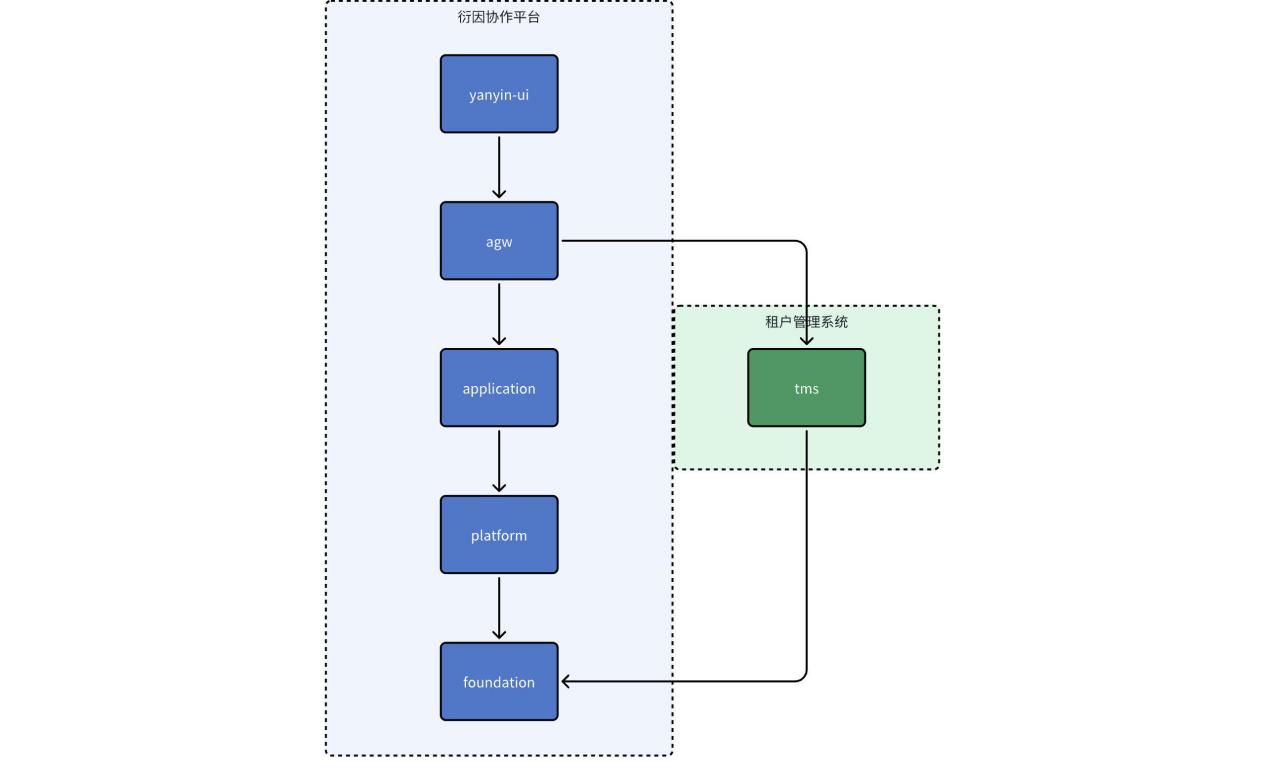
对应的设计方案如下。

|  |  |
| --- | --- |
| 维度 | 架构方案 |
| 模块级别（如角色管理） | 根据租户的订阅类型，决定是否启用特定模块。通过模块管理，从而在模块级别上控制访问权限。 |
| 功能级别（如文件-复制） | 根据租户的订阅类型，决定是否启用特定功能。通过功能开关来控制对特定接口或页面的访问，实现在功能级别上的细粒度控制。 |
| 用量限制（如素材容量） | 根据订阅类型设置租户的功能限额。这一设置直接影响租户可使用的资源量。 |
| 成员限制（如坐席管理） | 根据订阅类型设置租户的功能限额。这一设置直接影响租户可创建的席位数。 |
| 沟通方式 | 提供功能与入口 |

**决策**

|  |
| --- |
| 为了高效地管理系统功能，我们计划在租户管理中引入三个关键层级的控制机制：模块开关、功能开关以及功能限额。此外，面向系统内各种订阅版本（例如个人版、企业版等），我们将设计一系列可编辑的版本模板。这些模板将在租户初始化时进行配置（Provisioning），以此确保各版本在功能可用性和资源使用上均符合其特定的服务协议和用户需求。 |

**组件设计**



**Foundation 服务组件设计**

Foundation 服务作为系统的核心支撑组件，担负着以下关键功能：

1. 租户管理：

* 实现租户模型（Server层），提供租户数据结构和逻辑。
* 承担租户管理职责（Server层），处理租户的创建、更新和删除。
* 负责租户的provisioning过程（Server层），确保租户初始化和配置。
* 提供运行时租户隔离的库（Lib层），支持上层服务在运行时实现租户隔离。

1. 模块与功能管理：

* 维护模块与功能管理模型（Server层），定义模块和功能的数据结构。
* 提供运行时模块开关的库（Lib层），用于控制模块的激活与禁用。
* 提供运行时功能开关的库（Lib层），用于管理功能级别的访问控制。
* 提供运行时功能限制的库（Lib层），用于实施功能的使用限额。

1. 订阅管理：

* 构建订阅模型（Server层），用于管理订阅计划的数据结构。
* 实施订阅管理功能（Server层），包括订阅计划的创建和更新。
* 管理租户的订阅变更（Server层），处理订阅升级、降级和变更。

**Platform 服务组件**

Platform 服务利用 Foundation 提供的库（Lib），负责管理租户的上下文信息。

**Application 服务组件**

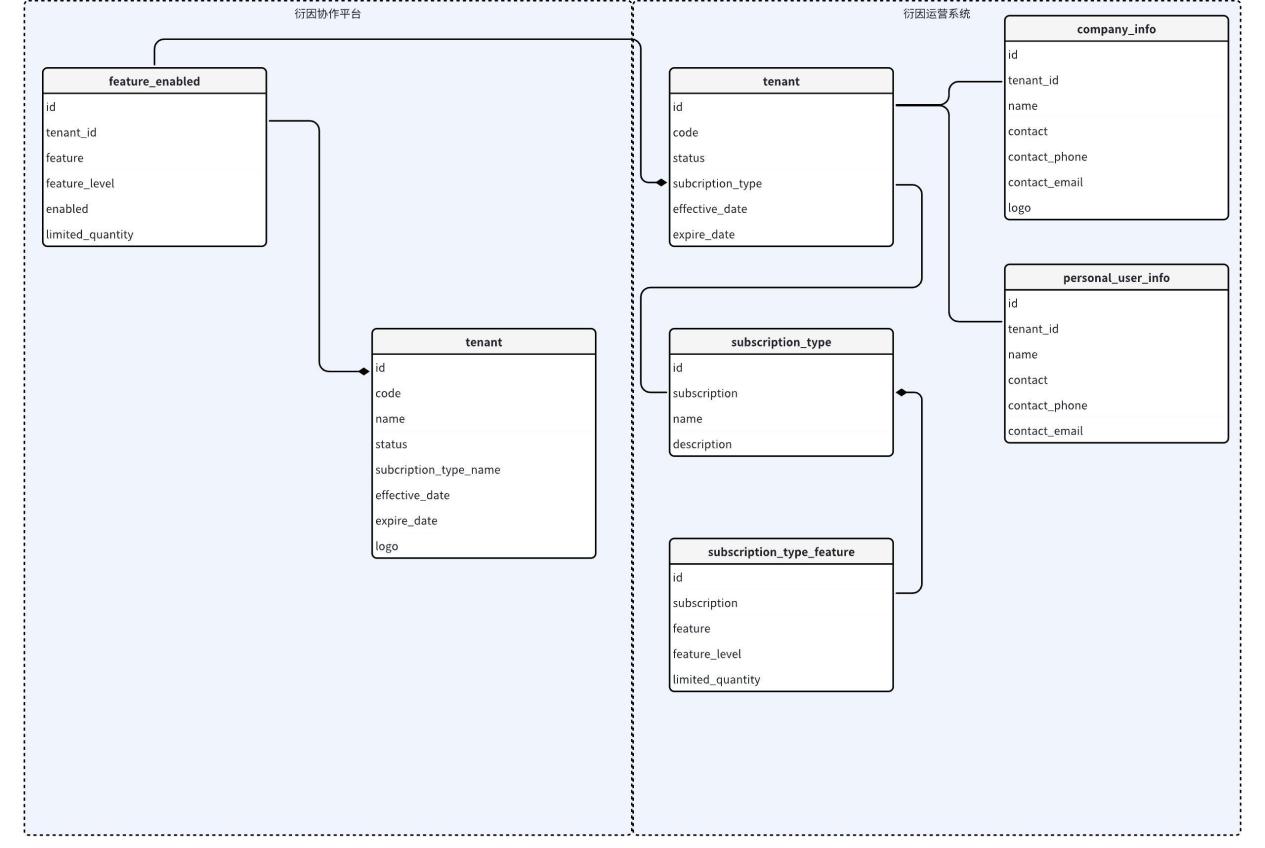
Application 服务同样使用 Foundation 提供的库，以确保在应用层面上正确管理租户上下文。

**Agw 服务组件**

Agw 服务使用Foundation 提供的Lib，负责以下任务：

* 管理租户上下文。
* 控制模块开关上下文，确保正确的模块可用性。
* 管理功能开关上下文，实施功能级别的访问控制。
* 实施功能限制上下文，确保功能使用符合预定限额。

**数据模型设计**



**安全性考虑**

**租户id生成机制**

|  |
| --- |
| 采用非连续的ID生成策略，避免使用连续数字序列。这种方法可以减少可预测性，增加系统的安全性。 |

**租户id保护机制**

|  |
| --- |
| 对于所有外部可见的场合（如用户界面、API调用、集成方或其他外部系统交互），使用加密形式的租户ID，或是另一套id体系。这种加密机制确保在租户ID需要公开时，仍然保持数据的安全性和不可预测性。 |

**可扩展性与弹性设计**

1. 核心功能集中维护：

* 所有主要功能均在 Foundation 服务中维护，确保了功能的集中管理和高效运作。此设计允许后续其他产品能够直接利用这些功能，增强了系统的可复用性和一致性。

1. 无状态服务设计：

* 本服务采用无状态设计理念，使其具备优秀的水平扩展能力。这意味着可以根据需求动态增减服务实例，从而在用户负载增加时保持高性能和响应速度。

1. 并发处理与弹性设计：

* 鉴于该模块的并发水平预计不会过高，因此我们选择不实施复杂的弹性设计。这种决策有助于保持系统架构的简洁性，同时满足当前的性能需求。

**运维策略**

**部署策略**

为了确保系统整体的一致性和稳定性，我们将采取统一的发版策略。该模块将与系统的其他模块一同进行版本更新和部署。通过这种同步发版的方法，我们可以确保所有系统组件的兼容性，同时减少由于部署不同步带来的潜在问题。这种策略有助于维持系统的整体性能和可靠性，同时简化维护和升级过程。

**监控与日志**

为了确保我们的系统能够实时监控性能并有效地记录关键事件，我们将采用阿里云提供的日志服务作为我们的主要监控和日志记录工具。这项服务将使我们能够：

* 实时监控：持续追踪系统的运行状态和性能指标，及时发现和响应潜在的问题。
* 日志管理：自动收集和存储系统日志，方便进行错误分析、审计和历史数据回溯。
* 故障诊断：利用详细的日志数据快速定位和解决系统问题。
* 性能优化：基于日志分析的见解，持续优化系统性能和用户体验。

通过整合阿里云的日志服务，我们能够提高系统的透明度和可靠性，同时减轻内部管理的负担，使团队能够专注于核心业务和创新。

**灾难恢复与备份**

为了确保我们的系统在面临任何不可预见的事件时都能快速恢复，并保护关键数据免受损失，我们将依赖于云服务提供商的高级恢复与备份能力。这项策略包括：

* 自动化数据备份：定期自动备份所有关键数据，确保数据的完整性和可靠性。
* 灾难恢复能力：在发生系统故障或数据丢失的情况下，能够迅速启动灾难恢复流程，减少业务中断时间。

通过整合云服务提供商的备份和恢复解决方案，我们不仅提高了系统的韧性，还确保了即使在极端情况下，也能保障业务连续性和数据安全。

**测试策略**

**测试计划**

详细的测试策略和范围将由测试团队在后续阶段确定并补充。这个计划将包括各类测试，旨在全面验证系统的功能、稳定性和用户体验。一旦测试细节确定，将立即进行更新和实施。

**性能测试**

关于性能测试，当前我们的决策是暂时不将其列为优先任务。这一决策基于对当前系统设计、业务需求和资源分配的综合评估。随着项目的发展，我们可能会根据新的业务需求和系统性能指标重新评估并考虑未来实施性能测试。

**路线图与里程碑**

**开发计划**

* Sprint 6：
* 重点将放在对 Foundation 服务的调整上，目的是增强其能力，使之能够提供更加强大和灵活的 Server 和 Lib 功能。这一步骤是为了提升整体架构的效率和可扩展性。
* Sprint 7：
* 任务将聚焦于完成 Platform、Application 和 Agw 的重构工作，以及将 TMS（任务管理系统）服务独立化。这些变更旨在优化各个组件的性能和互操作性，同时提升系统的模块化和维护便利性。

以上规划的实施将确保我们的系统不仅在技术层面上保持先进，而且在功能和运维上更加高效和可靠。

**风险评估与缓解**

**风险识别**

**技术风险**

* 在当前阶段，由于产品尚未正式进入市场，重构过程预计不会面临重大的技术风险。这一情况为我们在进行必要的架构调整和优化时提供了相对稳定和安全的环境，使我们能够更加自信地推进技术改进。

**资源风险**

* 我们目前面临的主要挑战是潜在的人力资源短缺。考虑到团队成员可能同时参与其他任务和故事，这种资源限制可能会对项目的开发进度和最终质量产生直接影响。因此，对人员分配进行持续的监控和灵活调整将是确保项目按计划进行的关键因素。

**缓解策略**

为了有效应对项目中可能遇到的资源风险，我们将采取一种动态评估和调整的方法。这一策略的核心是在每个迭代周期（Sprint）中进行细致的风险评估：

* 定期评估：在每个Sprint的开始和结束时，我们将评估当前面临的风险，包括技术风险和资源风险。
* 适时调整：基于评估结果，我们将及时调整项目计划和资源分配，以缓解潜在风险的影响。
* 持续监控：在整个项目周期内，我们将持续监控风险因素，确保能够迅速响应任何变化。

这种策略旨在保持项目的灵活性和适应性，确保我们能够及时识别和处理任何可能影响项目成功的因素。

**附录**

**术语表**

|  |  |  |
| --- | --- | --- |
| 术语 | 英文 | 描述 |
| 租户 | tenant | 在多租户架构中，租户指的是使用系统资源的独立实体，通常对应一个用户或一组用户，他们共享系统资源但保持数据隔离。 |
| 订阅 | subscribe | 指用户选择并付费使用软件或服务的过程。在SaaS模型中，订阅通常涉及对特定服务级别的定期付费。 |
| 配置 | provision | 在IT领域，配置通常指为用户或系统设置和准备必要的服务和资源的过程，以使其达到可操作状态。 |
| 软件即服务 | SaaS | 是一种软件分发模型，在这种模型中，软件应用托管在云中，并通过互联网提供给用户使用，通常基于订阅制。 |
| 无状态服务 | Stateless Service | 一种服务架构模式，其中服务不保留任何用户会话信息。这种设计使得服务在每次请求之间不保留数据或状态，有利于提高可扩展性和容错能力。 |