在遗留 Rails 单体应用中构建 AI 代理

杨岢瑞

Dec 26, 2025

遗留 Rails 单体应用在企业中广泛存在，这些应用往往经历了多年的迭代，积累了丰富的业务逻辑，但也面临代码老化、维护成本高企以及扩展困难等问题。代码库中充斥着过时的 Gem 依赖，数据库模型虽成熟却难以适应现代需求，而集成新技术时常常遭遇安全隐患和性能瓶颈。与此同时，AI 代理作为一种新兴技术迅速崛起，它基于大型语言模型如 OpenAI GPT 或 Anthropic Claude，能够自主感知环境、规划行动并调用工具执行复杂任务，支持多模态交互。这种代理不仅仅是简单的聊天机器人，而是具备决策能力的自治系统。在遗留 Rails 应用中构建 AI 代理，具有显著优势：它能提升开发者和运营团队的生产力，实现现代化改造，同时支持渐进式演进，无需进行破坏性的大规模重构。通过代理，Rails 应用可以自动化处理用户查询、数据分析和后台任务，逐步注入智能能力。

本文的目标是提供一套实用、可操作的指南，帮助有 3 年以上 Rails 经验的中高级开发者，在不破坏现有系统的前提下集成 AI 代理。读者定位于那些熟悉遗留系统维护的工程师，他们可能正为老旧代码烦恼，却希望借助 AI 实现低风险升级。文章将从 AI 代理的概念入手，逐步推进到环境搭建、架构设计、逐步实现、实战案例、挑战应对以及高级主题，最后给出行动建议。整个过程强调实践导向，确保每一步都能在真实项目中落地。

1　AI 代理基础概念

AI 代理本质上是一个基于大型语言模型的自治系统，它能够感知输入环境、通过规划器制定行动方案，并调用专用工具执行任务，最终输出结果或迭代优化。核心组件包括 LLM 作为大脑，提供推理能力；工具集成层，用于连接外部系统如数据库或 API；内存机制，分短期内存用于当前对话上下文和长期内存用于历史知识积累；规划器则采用 ReAct 模式，即反复执行「推理（Reason）+ 行动（Act）」循环，直到任务完成。这种设计让代理从被动响应转向主动解决问题，例如在 Rails 应用中自动查询订单并生成报告。

将 AI 代理集成到传统 Rails 应用具有必要性，因为遗留 Rails 的优势在于业务逻辑完整和数据模型成熟，这些是 AI 训练数据难以匹敌的宝贵资产。然而，挑战同样明显：旧版 Gem 可能不支持现代 Ruby 特性，API 接口不规范，安全漏洞频发。通过代理，我们可以将这些痛点转化为机会，让 AI 作为中间层桥接旧系统与新功能。

推荐的技术栈包括使用 OpenAI API 或 LangChain Ruby 作为 LLM 接入层，因为它们易于集成且稳定性高；LlamaIndex 或 LangGraph 的 Ruby 适配用于构建代理抽象，提供工具调用和状态管理；Rails 侧则依赖 Sidekiq 或 ActiveJob 处理异步任务，确保代理运行不阻塞主应用。

2　准备工作：评估与环境搭建

在着手构建前，首先评估遗留 Rails 应用的状态。检查 Ruby 版本是否达到 2.7 或更高，Rails 版本至少为 5，以确保兼容现代 Gem。识别集成点，如现有 Controller 或 Service 中的业务逻辑、数据库模型以及外部 API

调用。同时，进行安全审计：使用 Rails Credentials 管理 API Key，避免硬编码；实施 Rate Limiting 防止滥用。

项目环境搭建从更新 Gemfile 开始，添加核心依赖。以下是示例 Gemfile 片段，这个配置引入了 OpenAI 客户端、Sidekiq 用于后台任务，以及 LlamaIndex 的 Ruby 适配（或自定义 wrapper）。解读这段代码：`gem 'openai'` 提供官方 Ruby 客户端，支持聊天完成和工具调用 API；`gem 'sidekiq'` 启用 Redis 驱动的队列系统，适合代理的长时任务；`gem 'llama_index'`（假设社区适配）封装了索引和检索功能，便于后续 RAG 集成。安装后运行 `bundle install`，并配置环境变量如 `OPENAI_API_KEY` 和 `RAILS_ENV`。可选地，使用 Docker 容器化应用，提升隔离性和可移植性，例如通过 Dockerfile 定义多阶段构建，确保依赖一致。

```
gem 'openai'
gem 'sidekiq'
gem 'llama_index' # 或自定义 wrapper
```

3 设计 AI 代理架构

AI 代理架构采用分层设计，从高层视角看，Rails 单体应用通过 Controller 或 Webhook 输入任务，流向 AI Agent Service，该服务调用 LLM 并协调工具，最终回写数据到 DB 或外部 API。分层包括输入层负责解析用户请求，代理核心执行规划循环，工具层封装 Rails 服务，输出层生成响应。这种设计确保了松耦合，遗留代码无需改动。

代理组件中，规划器采用 ReAct 模式：代理先观察输入，推理下一步行动，调用工具执行，然后基于结果重复循环，直到任务解决。工具定义是将 Rails 服务封装为可调用函数，例如查询用户数据或发送邮件，这些工具通过 JSON Schema 描述参数给 LLM。内存管理使用 Redis 存储短期上下文（如当前对话），PostgreSQL 的 JSONB 字段存长期记忆，支持复杂查询。

与 Rails 集成有三种模式：在 Service Layer 中注入代理增强业务逻辑；通过 Sidekiq Job 异步运行长任务；新增 API Endpoint 如 `/ai/agent` 支持外部触发。每种模式根据场景选择，确保渐进式引入。

4 逐步实现指南

4.1 第一步：基础 LLM 调用

实现从创建 `Ai::Client` 服务类开始。这个类封装 OpenAI 调用，提供简单接口。以下代码定义了 `chat` 方法，使用 OpenAI 客户端发送提示。详细解读：`OpenAI::Client.new` 初始化客户端，默认从环境变量读取 API Key；`chat` 方法接受 `parameters` 哈希，指定 `gpt-4o` 模型（高效且支持工具调用），`messages` 数组模拟对话，其中 `{role: user, content: prompt}` 是用户输入。调用后返回流式或完整响应，可进一步解析。这个基础封装为后续代理循环奠基，避免在多处重复配置。

```
class Ai::Client
  def chat(prompt)
    OpenAI::Client.new.chat(parameters: {
      model: "gpt-4o",
      messages: [{role: "user", content: prompt}]
```

```
    })
  end
end
```

4.2 第二步：构建工具集

工具集是代理能力的延伸，将 Rails 逻辑封装为独立类。以 `UserQueryTool` 为例，它接受查询参数，从数据库检索用户。代码解读：`call` 方法是工具入口，接收 `query` 字符串，使用 ActiveRecord 的 `where` 子句以 `ILIKE` 实现模糊匹配（忽略大小写），`limit(10)` 防止结果过多。这个工具后续通过 LLM 的函数调用机制触发，LLM 会根据任务生成参数如 `{query: John}`，工具执行后返回结构化数据如用户列表 JSON，提升代理的精确性。类似地，可构建 `EmailTool`，调用 ActionMailer 发送通知。

```
class UserQueryTool
  def call(query)
    User.where("name␣ILIKE␣?", "%#{query}%").limit(10)
  end
end
```

4.3 第三步：组装完整代理

完整代理在 `AiAgent` 类中组装，集成 LLM、工具和 ReAct 循环。以下 Controller 示例展示集成：在 `process` 动作中，实例化代理传入工具数组，调用 `run` 执行任务，返回 JSON 结果。解读代理内部：`run` 方法初始化 LLM 客户端，进入循环 —— 发送当前状态给 LLM，解析工具调用（如 `{name: UserQueryTool, arguments: {...}}`），执行对应工具，更新观察状态，直至 LLM 输出「任务完成」。Controller 捕获结果渲染，异常时 fallback 到默认响应。这个设计让代理自包含，可轻松测试和扩展。

```
class AiAgentsController < ApplicationController
  def process
    agent = AiAgent.new(tools: [UserQueryTool.new])
    result = agent.run(params[:task])
    render json: { result: result }
  end
end
```

4.4 测试与调试

测试分层进行：使用 RSpec 编写单元测试，mock 工具输出验证逻辑；端到端测试结合 Capybara 模拟用户交互，mock LLM 响应确保确定性。日志采用 Lograge 精简输出，并记录代理的决策链，如「Observe: 用户查询订单 → Think: 调用 OrderTool → Act: 执行查询」。

5 实战案例：遗留 Rails 中的 AI 客服代理

考虑一个遗留电商 Rails 应用，用户通过 Telegram 或 Slack Webhook 咨询订单状态。AI 客服代理接收消息作为输入，工具包括 `OrderLookupTool`（查询订单表）、`RefundTool`（处理退款）和 `NotifyAdminTool`（Slack 通知管理员）。代理运行 ReAct：先检索订单，若异常则通知管理员，最后生成自然语言回复如「您的订单 #123 已发货，预计 3 天到达」。这个案例将代理部署为 Sidekiq Job，输入 Webhook 触发异步处理。
性能优化使用 Redis 缓存常见查询，如订单状态哈希键 TTL 1 小时，减少 DB 负载；批量 LLM 调用合并多工具请求，降低 Token 消耗。部署上，Heroku 或 Railway 支持一键上线，New Relic 监控 Rails 指标，结合 LLM 观测工具追踪代理成功率和延迟。

6 挑战与最佳实践

遗留代码兼容是首要挑战，可用 Monkey Patch 临时扩展旧类，或 Adapter Pattern 包装接口。成本控制通过 Token 限额和规则基 fallback 实现，例如查询超 1000 Token 时切换关键词匹配。安全性强调输入 Sanitize 和工具 RBAC，仅授权必要操作。幻觉问题通过 RAG 缓解：检索 Rails 文档验证输出，并加验证层检查事实准确性。
最佳实践包括渐进集成，从简单数据查询起步逐步到决策任务；部署 Prometheus + Grafana 监控代理成功率；A/B 测试对比 AI 与人工路径；使用 Git 分支隔离 AI 代码，便于回滚。

7 高级主题

多代理协作引入 Supervisor Agent，协调子代理如查询代理和通知代理，通过状态机分发任务。RAG 集成使用 PG Vector 扩展 PostgreSQL，存储 Rails 模型文档作为向量，提升查询准确性：嵌入用户问题，检索相似文档注入提示。未来可将代理迁移为独立微服务，作为单体拆分的过渡。开源资源如 LangChain Ruby 提供现成工具链，Rails AI Gems 加速集成。

8 结论与下一步

AI 代理为遗留 Rails 现代化提供了低风险切入点，通过实践从 MVP 迭代，能显著提升系统智能。行动号召：fork 示例仓库，实现首个工具如用户查询，并在生产中测试。欢迎分享你的遗留 Rails + AI 案例，推动社区进步。
资源链接包括 GitHub 示例代码仓库（假设 https://github.com/example/rails-ai-agent），OpenAI Tools 文档和 LangChain Ruby 指南。

9 附录

完整代码仓库见 GitHub。术语表：AI Agent 为自治 LLM 系统；ReAct 为推理行动循环；Tool Calling 为 LLM 函数调用。FAQ 示例：Rails 4 处理通过兼容 Gem 和 Ruby 2.5+ 升级路径。