陈业诚-25a900085-项目申请书

项目申请书

实现 dynamic-tp 的自建管理端功能

项目导师：林彦宏

申请人：陈业诚

日期：2025-6-12

邮箱：eachannchan@qq.com

一.项目背景

Dynamic-Tp是一款轻量级Java动态线程池增强框架，获得了大量开发人员的青睐。由于目前项目主要是基于配置中心来实现相关配置等操作的，从零到一地部署及运维相关环境需要较多地精力，故希望项目拥有相应的管理端来简化部署并减少运维成本。

项目作为通用的基础组件，其出发点是满足不同场景不同用户的需要，希望自建管理端能满足使用者的需要并使Dynamic-Tp真正成为一站式线程池管理框架。

二.项目目标

1. 建立server端和client端，server端负责采集及管理项目线程池的相关数据，client端负责与使用者进行交互展示
2. 保证项目轻量级的特性，引入各种技术来保证引入管理端后不过于臃肿

3. 与server端交互的client端以单独包存在，不过多地侵入现有配置中心的代码
4. 支持docker快速部署，降低框架的使用成本
5. 支持架构：x86_64

三.相关产品调研

1.Nacos

Nacos是阿里巴巴开源的一款动态服务发现、配置管理和服务管理平台，提供了服务注册与发现、动态配置等功能。其支持多种格式的配置方式（如YAML，JSON，Properties），配置变更后，Nacos 可以动态推送更新，无需重启服务。

其逻辑架构如下：

相关地，可借鉴到的特性有通过控制台来方便地进行管理，动态配置，支持namespace命名空间的使用，配置分组，健康检查等。

动态配置的实现原理

Nacos 动态配置的实现原理基于其核心机制，包括长轮询、配置监听和本地缓存等，旨在实现配置的实时更新与高效管理。以下从多个方面详细介绍其实现原理：

核心机制：长轮询与配置监听

Nacos 采用**长轮询（Long Polling）**机制来实现动态配置更新。长轮询是一种优化后的轮询方式，能够减少无效的网络交互，提高效率。其工作流程如下：

1. **建立长连接**：
   客户端向 Nacos 服务端发起请求，请求中包含需要监听的配置信息。服务端不会立即返回结果，而是保持连接状态，等待配置更新或超时。
2. **挂起请求**：
   如果服务端的配置在请求期间没有发生变化，请求会一直挂起，直到以下情况发生：
   - 配置发生变更。
   - 连接超时（默认超时时间为 30 秒）。
3. **配置变更通知**：
   当服务端检测到配置变更时，会立即唤醒挂起的请求，并将最新的配置信息推送给客户端。
4. **配置比较与更新**：
   客户端收到配置后，会将其与本地缓存的配置进行比较。如果发现差异，则更新本地配置并触发监听器的回调方法，通知应用程序重新加载配置。

健康检查的实现原理

在微服务架构中，服务实例可能会因为各种原因（如宕机、网络问题等）变得不可用。如果这些不健康的实例继续留在服务列表中，可能会导致调用方请求失败。因此，健康检查机制的作用是：

- 实时检测服务实例的健康状态：确保服务注册中心中的服务列表准确反映实例的实际可用性。
- 及时剔除不健康的实例：避免调用方请求被路由到不可用的实例上，提高系统的可靠性。

Nacos根据服务实例的类型（临时实例和持久实例）提供了两种健康检查机制：临时实例（Ephemeral）的健康检查和持久实例（Persistent）的健康检查。

在此可以借鉴持久实例的实现机制：

机制：服务端主动探测机制。

工作方式：

- Nacos服务端会定期向持久实例发送探测请求（如HTTP请求），检查实例的健康状态。
- 默认探测间隔为20秒。
- 如果实例在连续几次探测中未响应或响应不正确，Nacos会将其标记为“不健康”，但不会立即移除，而是进行隔离处理。

2.XXL–JOB

XXL–JOB是一款**开源分布式任务调度平台**，其核心设计目标是**开发迅速、学习简单、轻量级、易扩展。**适用于定时任务调度，任务依赖管理等场景。

XXL–JOB 的架构主要由以下几个核心组件构成：

1. **调度中心（Admin）**
   - 负责任务的管理、分配和监控。
   - 提供 Web 界面，用户可以通过该界面进行任务的创建、编辑、删除等操作。
2. **执行器（Executor）**
   - 负责执行调度中心分配的任务。
   - 支持集群部署，可以动态扩展。
3. **注册中心（Registry）**
   - 负责管理执行器信息，供调度中心发现可用的执行器。

底层通信原理

项目需求可以以此作为参照，以下为XXL–JOB的底层通信原理：

其主要基于**HTTP 协议** 和 **Netty 框架**，其通信机制设计上兼顾了高性能、异步处理和灵活性。

通信方式

XXL–JOB 的通信主要采用 **HTTP 协议**，但其底层实现使用了 **Netty** 框架，这使得通信效率更高、性能更优。具体来说：

- **调度中心（Admin）** 与 **执行器（Executor）** 之间的通信是通过 HTTP 协议进行的。
- 虽然 XXL-JOB 支持多种通信方式（如 Mina、Jetty、Netty TCP），但默认使用的是 **Netty HTTP**。
- 执行器在启动时会主动向调度中心注册，调度中心通过 HTTP 接口接收注册信息，并将执行器的信息存储在数据库中

通信流程

XXL-JOB 的通信流程可以分为以下几个步骤：

执行器注册

- **执行器启动时**，会通过 HTTP 请求向调度中心发送注册信息，包括执行器的 IP、端口、AppName 等。
- 调度中心接收到注册请求后，会将这些信息存储到数据库中，以便后续调度任务时使用。
- 执行器会周期性地向调度中心发送心跳，以保持注册状态。如果执行器长时间未发送心跳，调度中心会将其标记为离线。

任务调度

- **调度中心** 根据任务配置（如 Cron 表达式）计算任务的执行时间。
- 当任务到达执行时间时，调度中心会通过 HTTP 请求向对应的执行器发送任务执行指令。
- 执行器接收到请求后，会调用任务执行逻辑（如通过 `@XxlJob` 注解标记的方法）。

任务执行

- 执行器在接收到任务请求后，会将任务信息存储到一个 **异步队列**（如 `LinkedBlockingQueue`）中，由异步线程处理。
- 任务执行完成后，执行器会通过 HTTP 回调将执行结果返回给调度中心。
- 调度中心会记录任务的执行状态（如成功、失败、超时等），并更新到数据库中。

任务结果回调

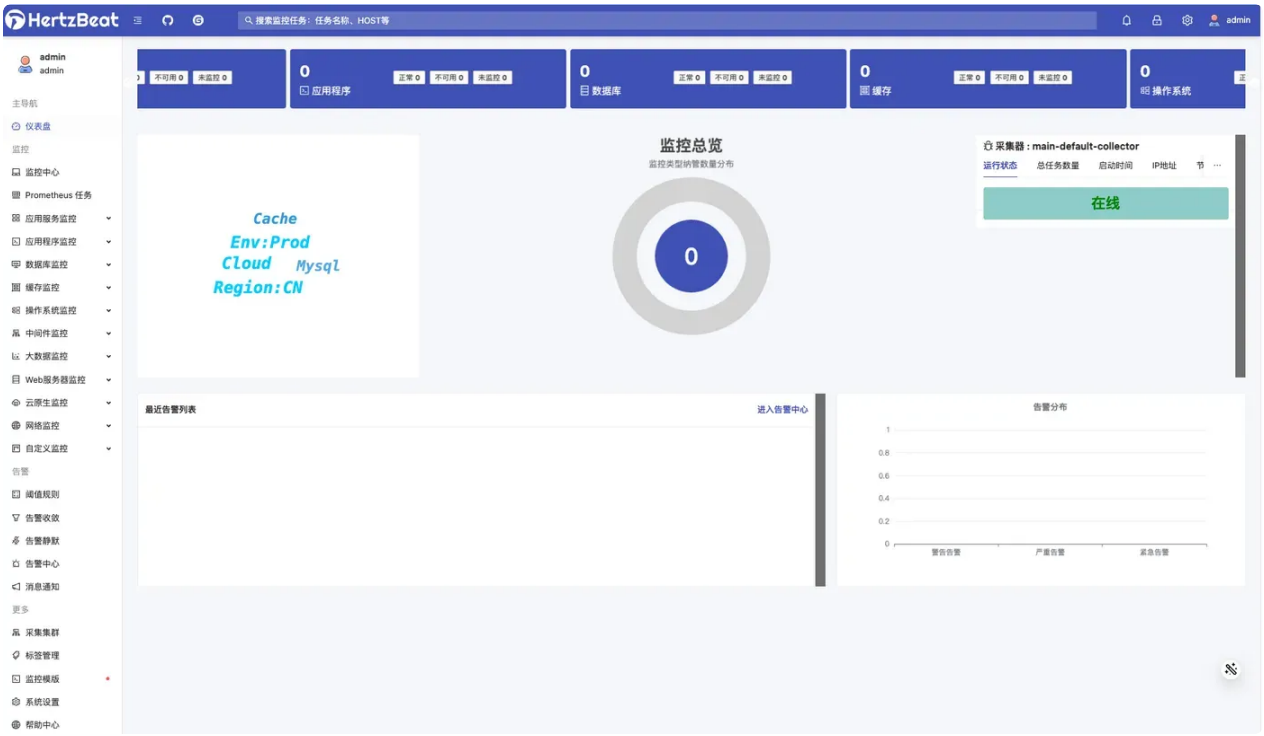
- 执行器在任务执行完成后，会通过 HTTP 回调将结果发送给调度中心。
- 调度中心接收到结果后，会更新任务的执行日志，并通知用户（如通过邮件、短信等方式）。

3.HertzBeat

Apache HertzBeat 是一款开源的，易用友好的实时监控告警工具，无需 Agent，强大自定义监控能力。 HertzBeat 的后端基于 Java 和 Spring Boot 开

发，前端使用 Vue.js 框架，数据可视化部分采用 ECharts，这种技术组合保证了系统的易用性和性能。通过定时任务实现数据的定期采集，保证监控信息的即时性。

其主页大盘效果如下：



可借鉴HertzBeat的监控数据采集模块的实现逻辑以及页面设计。

监控数据采集模块的整体架构

HertzBeat的数据采集模块是其核心功能之一，负责从各种监控对象（如应用服务、数据库、操作系统、网络设备等）实时获取数据。其架构设计具有以下特点：

- **无代理架构**：无需在目标监控对象上安装额外的Agent，通过直接连接目标系统采集数据，降低了部署复杂性和资源占用。
- **模块化设计**：将数据采集、处理和存储功能分离，确保模块间的低耦合和高扩展性。
- **分布式支持**：支持多采集器集群部署，能够横向扩展，满足大规模监控需求。

4.Hippo4j

Hippo4j也是一款聚焦于线程池核心参数动态调整、运行告警和监控等功能并支持docker快速部署的项目，实现了自己的管理端，其对实现需求一定的参考价值。

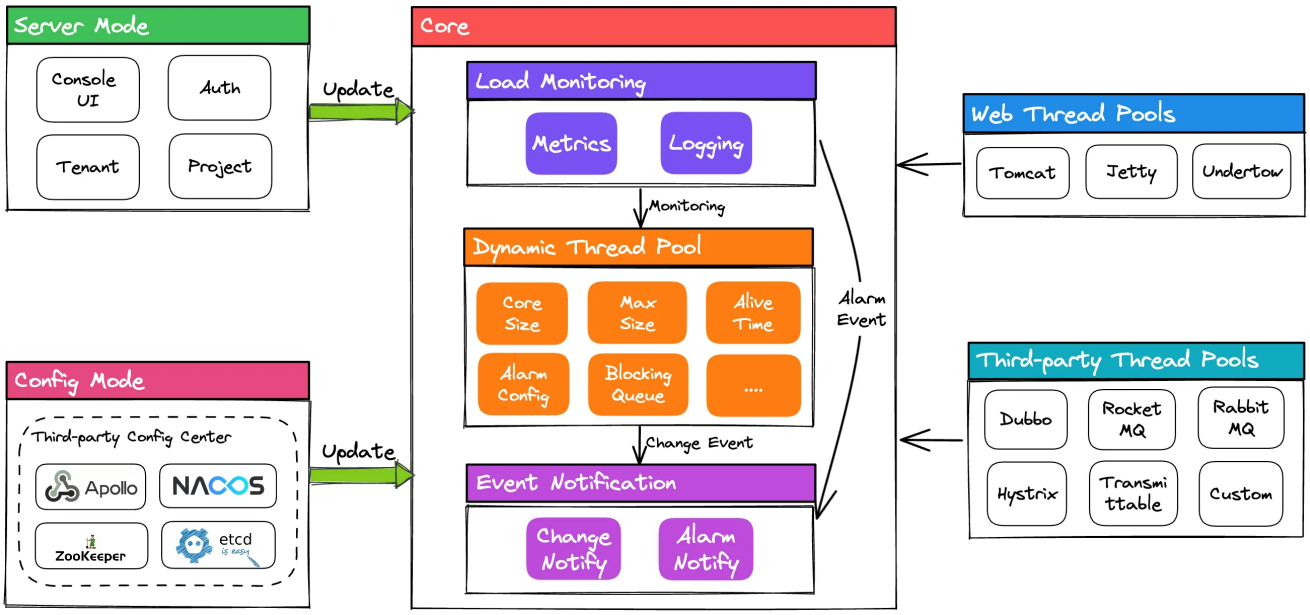
以下为其架构设计：

Hippo4j 从部署的角度上分为两种角色：Server 端和 Client 端。

Server 端是 Hippo4j 项目打包出的 Java 进程，功能包括用户权限、线程池监控以及执行持久化的动作。

Client 端指的是我们 SpringBoot 应用，通过引入 Hippo4j Starter Jar 包负责与 Server 端进行交互。Client端注册自己的信息和线程给服务端，服务端监控线程，通过心跳上报自己的状态线程池参数状态。

服务端改了参数后，客户端通过长轮询能获取到修改的参数信息， 客户端再调线程池api去更改。



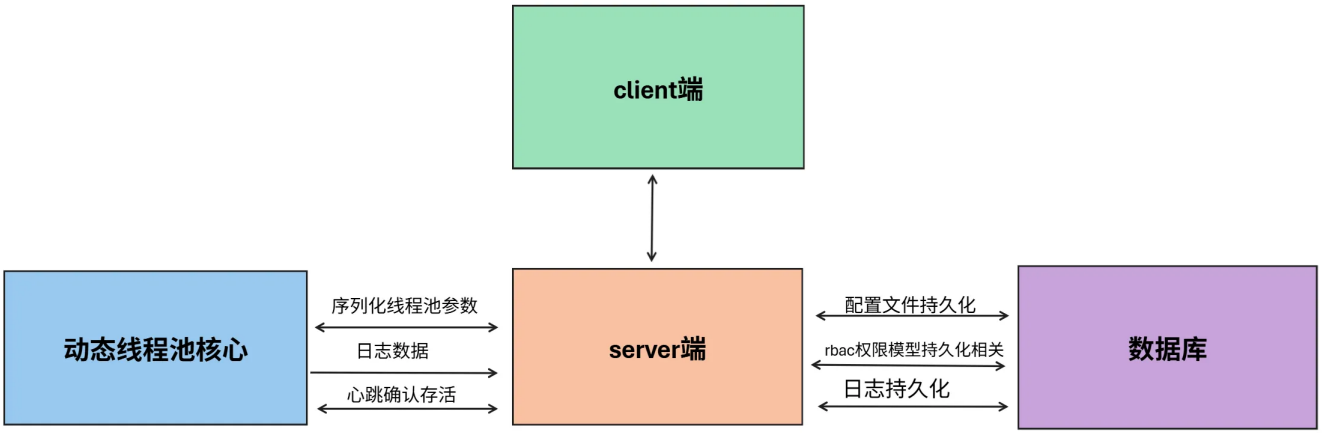
步骤

1. **线程池注册**（通过bean生命周期后置处理器实现）。
2. 服务端**监控线程池**状态，客户端 *（定时调用ThreadPoolExecutor的api）* 上报线程池状态。
3. 用户 在服务端**修改**参数配置。
4. 客户端 **长轮询 获取修改的参**数信息
5. 客户端 调用ThreadPoolExecutor**线程池的api**修改参数（setCorePoolSize）。
6. 客户端下线，停机，线程池bean销毁（通过bean生命周期销毁方法，等待线程池任务执行完毕）。

值得注意的是在Hippo4j中，客户端（即接入的项目）获取到线程池参数信息的方式采用的是推拉模式中的拉模式。在此需求中，为了使其与现有的配置中心逻辑一致以及为及时对修改做出响应，计划实现为管理端的推模式，线程池模块监听相关事件并做出调整。

三.实现方案

admin端作为一个负责监控管理，动态配置的微服务，以下为大致的实现逻辑图：



1.权限管理逻辑

基于RBAC角色权限模型为各用户设置对应的权限，抽象出几个核心概念：

- **角色（Role）**：角色是指在系统中具有一组相关权限的抽象概念，代表了用户在特定上下文中的身份或职能，**例如管理员、普通用户**等
- **权限（Permission）**：权限是指对系统资源进行操作的许可，如**读取、写入、修改等**。权限可以被分配给角色
- **用户（User）**：用户是指系统的实际使用者，每个用户可以被分配一个或多个角色
- **分配（Assignment）**：分配是指将角色与用户关联起来，以赋予用户相应的权限

Client端

a. 设计登陆页面，登陆成功则从server端获取相关权限，展示相关界面及按键

b. 设计权限管理页面，供管理员分配用户权限

Server端

a. 基于sa-token框架验证登陆
b. 对于登陆的用户向数据库请求相关角色及权限并返回前端

数据库

建立五张数据表：用户表，角色表，权限表，用户角色表，角色权限表

实现可基于PanisAdmin作为后台管理手脚架，具体链接为
https://github.com/paynezhuang/panis-admin

2.数据监控

server端定时向动态线程池核心请求线程池当前线程数量及阈值统计窗口等参数；在client端与配置的相关参数于同一个图表中进行展示，并在首页大盘中显示关键数据如高风险阈值，tps等

数据获取方式

项目的监控功能主要在core模块中的monitor包中实现，需求的具体实现为：

- 定义AdminClient类，用于转发Collector的数据序列化后发送至Admin端等
- 定义AdminCollector类，继承AbstractCollector重写相关方法，将收集到的数据传输给AdminClient类
- Admin端获取监控数据，将监控数据进行解析，将数据持久化以及发送给前端展示
- 前端通过刷新按钮可以主动请求数据以监控实时数据

3.配置中心及日志管理

client端对于配置中心有两种实现方式，一为参照现有的配置文件的方式来对线程池和告警渠道等的管理；二为对于每个线程池生成一个可视化的配置页面，告警渠道同理

日志管理的实现为将日志信息可视化展示，可根据日期查询，模糊查询等方式查询相关数据，将日志写入数据库或生成相关log文件

动态配置实现

- 动态线程池核心定义继承AbstractRefresher的AdminRefresher类，用来获取处理admin的更新操作
- admin的server端对client端的更新请求进行处理，转换成线程池核心对应的格式类并发送给线程池核心

4.心跳机制

心跳机制是为了让admin端及动态线程池核心确保对方存活的机制，实现为：

- 动态线程池核心实现相关HeartBeat类，利用`@Scheduled`注解定时发送心跳
- 若admin端做出响应则核心确认admin端正常
- admin端在定期请求数据监控时也相当于发送了心跳，若核心未做出响应则认为核心离线了，可以产生告警

四.开发规划

第一阶段（7月1日至7月31日）：

1. 设计实现client端的界面及功能
2. 设计实现server端的各个功能处理逻辑
3. 搭建相关数据库，实现各数据的持久化
4. 使用docker进行容器化管理部署

第二阶段（8月1日至8月31日）：

1. 编写单元测试类，对需求进行系统的测试，确保无明显bug
2. 编写相关文档，对admin的使用及一些细节进行详细的说明

第三阶段（9月1日至9月30日）：

1. 对导师指出的问题以及自己找到的问题进行重构及优化
2. 协助完成版本发布前的各项准备工作

五.期望

我一直深信，开源是推进软件技术进步的重要力量。

在去年，我虽然未在开源之夏中申请到项目，但是我也参与到了dynamic-tp开源项目的贡献当中，得到了莫大的成长并结识到了各位厉害的社区开发者。相较于去年的议题，今年的议题不局限于项目核心本身，出发点为拓展项目的外围生态，降低用户的引用使用成本，故涉及到的技术及领域将更广泛。我希望可以参与到本次2025年度的开源之夏活动当中来为项目继续做出自己的贡献并在此过程中提高个人的问题解决能力及项目设计能力。