架构设计文档

一、引言

1.1 项目背景

基于七牛云1024创作节议题，对Github公开数据进行分析和处理，从而实现对开发者的技术能力评估和信息整理。

2.2 项目概述

Gitgle是一个基于Github公开数据API构建的开发者技术能力评估以及信息整理应用。旨在通过数据库缓存技术突破Github API速率同时计算开发者能力以及通过信息网络推测开发者信息。

二、需求分析

2.1 功能需求

1. 实现用户登陆和注册，以及查看自己个人信息，包括Github相关信息
2. 实现对Github开发者的技术能力评估，评估结果使用一个得分TalentRank表示（如：45.12578）
3. 实现对未填写Loaction的开发者进行Nation推测，并给出置信度，置信度过低需要以N/A的形式展示
4. 实现对开发者领域的推测，并给出置信度，置信度过低领域使用N/A表示
5. 实现根据领域以及Nation对开发者的搜索，并根据TalentRank对开发者排序
6. 实现用户查看Github开发者的详细信息
7. 实现Github热门仓库排行以及数据统计

2.2 非功能需求

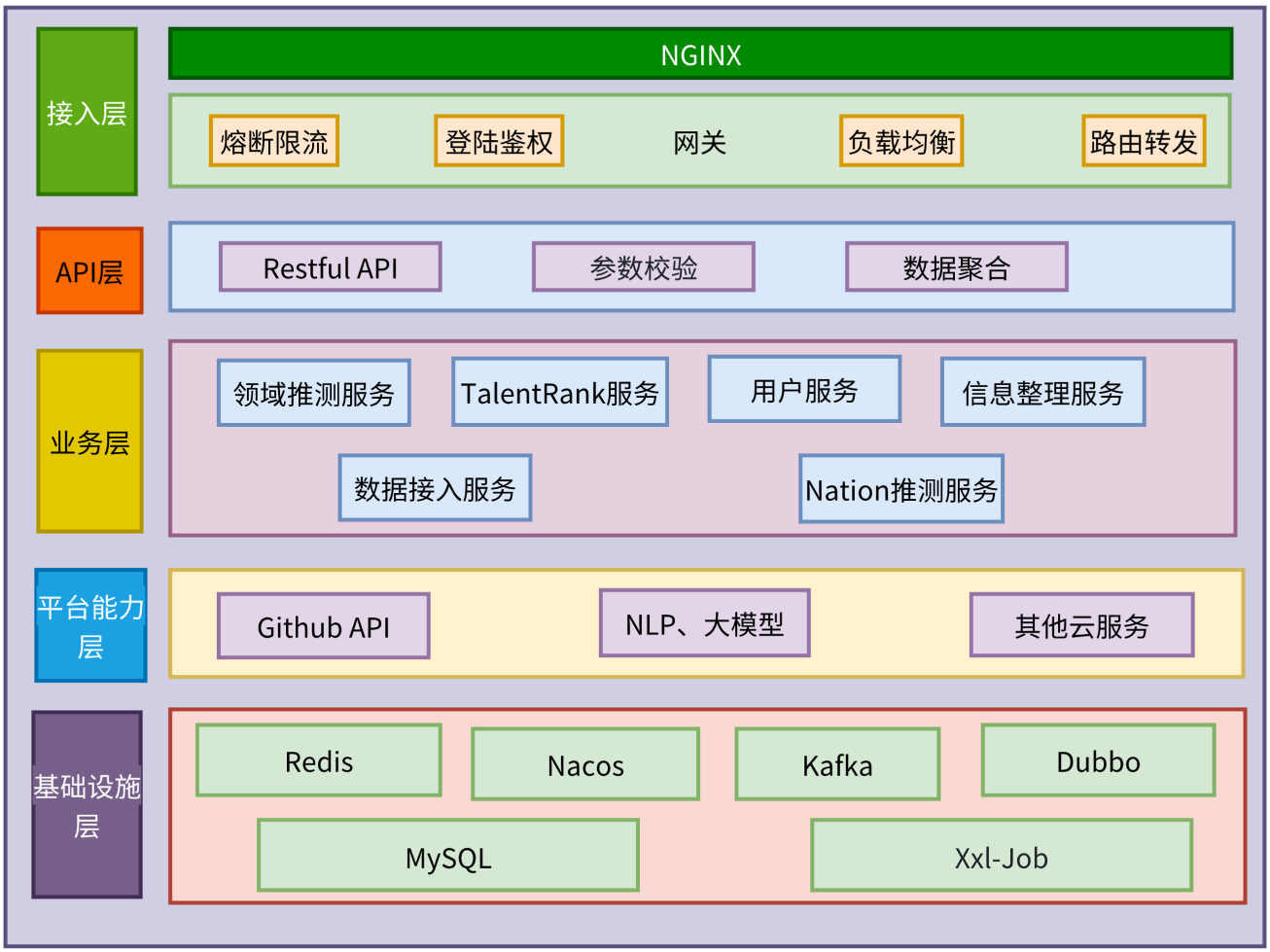
1. 实现系统侧对Github数据的缓存，避免Github API速率限制，同时提高获取数据速率
2. 提高用户请求的响应速度
3. 使用并行计算提高开发者信息计算和推测的速度

三、总体设计

3.1 架构概述

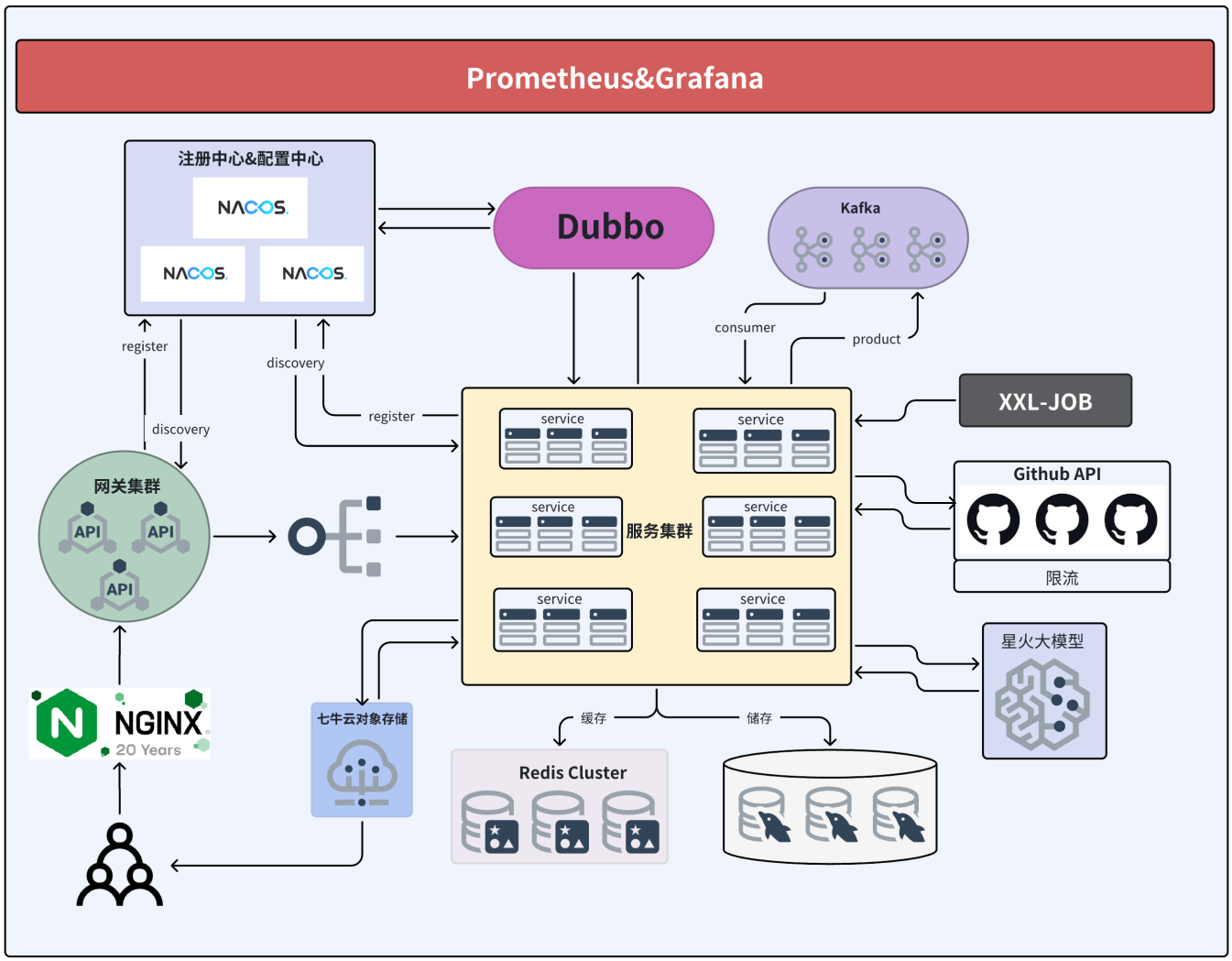
Gitgle应用整体采用微服务架构风格，使用Restful API和用户交互，内部服务通过Rpc以及分布式消息队列进行交互。

3.2 业务架构



Gitgle 使用微服务架构。接入层使用Nginx做后端服务流量网关，并使用SpringCloud Gateway做微服务业务网关，对请求进行负载均衡，统一鉴权等操作。API层则使用SpringMVC框架，专注于提供Restful API与前端交互并对后端Rpc接口进行聚合。业务层也是核心层，采用Springboot + Dubbo简化开发，通过Rpc接口和Kafka进行服务间的通信。平台能力则使用了Github API以及星火大模型以及其他云服务。基础设施则由我们维护缓存、数据库、消息队列、任务调度等组件。

3.3 系统架构



Gitgle 整体采用Springboot开发，使用网关对请求进行代理和分发，并使用Nacos进行服务的发现和注册，以及作为配置中心。后端服务通过Xxl-Job任务调度通过Github API获取Github数据，同时也会由一些被动获取数据的触发策略，保证业务数据获取的效率。推测服务依赖星火大模型。整个系统使用Redis对热门数据进行缓存，提升接口访问性能，增强用户体验。同时我们使用Prometheus和Grafana实现了系统运行状态的可视化监控和报警，旨在提前发现系统问题，及时修复以避免损失。

四、技术选型

- 前端框架：Vue.
- 后端编程语言：JAVA,JDK-8
- 依赖管理工具：maven-3.9.6
- 开发框架：SpringBoot-2.3.12.RELEASE、Dubbo-3.1.2
- 数据库：MySQL8.0
- 缓存中间件：Redis
- 消息队列：Kafka

- 注册中心：Nacos-2.4.3
- 配置中心：Nacos-2.4.3
- 任务调度组件：Xxl-Job-2.3.0
- 流量网关：Nginx
- 微服务网关：SpringCloud Gateway
- 监控中心：Prometheus&Grafana
- 大模型API：星火大模型

五、模块设计

5.1 模块概述

- gateway-service：负责统一接口鉴权、请求路由、熔断限流、负载均衡等
- api-service：负责提供restful api以及数据聚合
- user-service：负责用户注册登录和管理账户信息，以及负责接收计算和推测模块的结果消息进行数据更新
- data-service：负责获取github的数据，以及定时刷新数据，为系统提供数据支持
- talentrank-service：负责计算开发者的TalentRank，并将结果下发给用户服务
- nation-service：负责推测开发者的nation，并将结果下发给用户服务
- domain-service：负责推测开发者的领域，并将结果下发给用户服务
- common：提供工具类和公共数据

5.2 设计与实现

5.2.1 gateway-service

gateway-service是基于Spring Cloud Gateway定制化开发的Gitgle系统的微服务网关，主要负责对微服务集群的统一鉴权、负载均衡、路由转发、熔断限流等。微服务网关承载流量网关代理的请求，保证了与前端解藕以及对服务的保护和统一的操作。

5.2.2 api-service

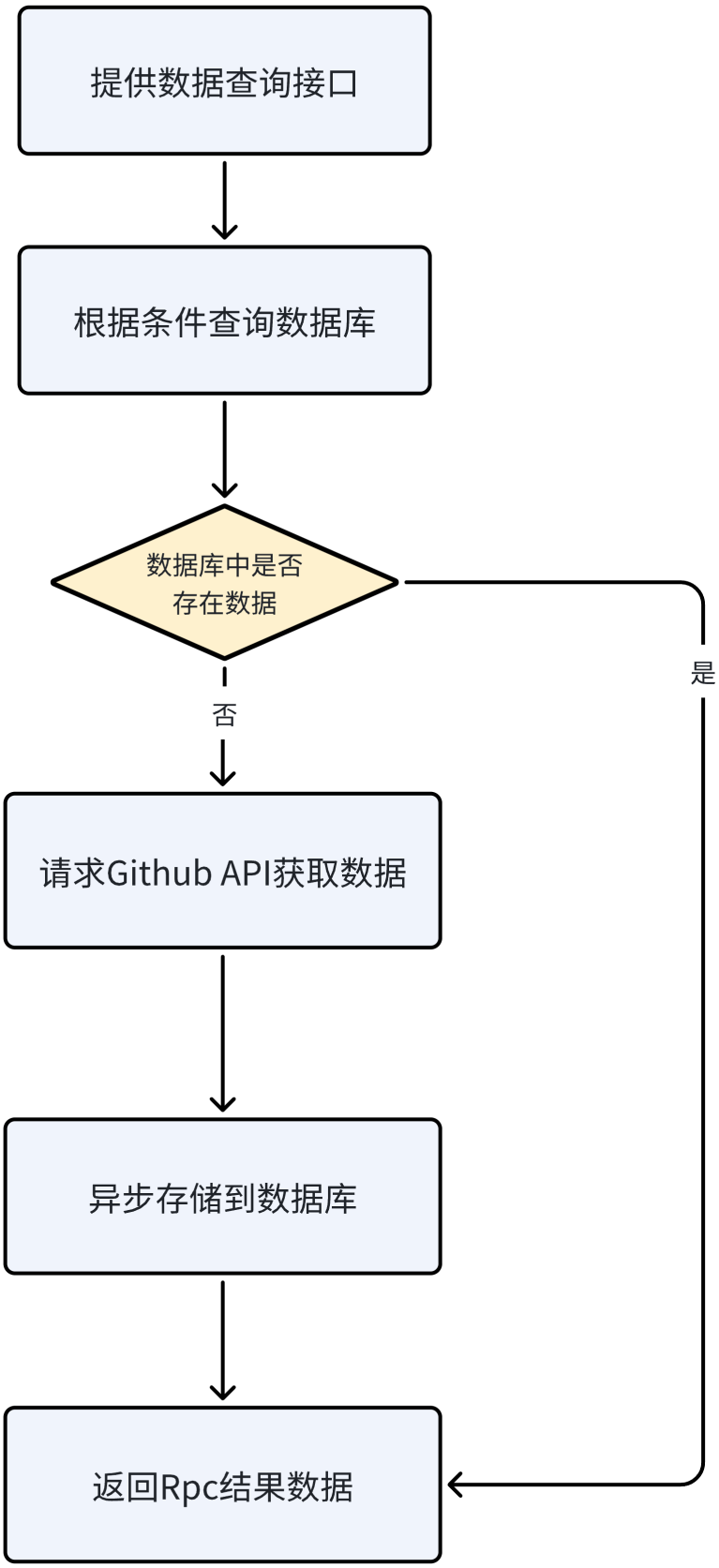
api-service是负责统一提供Rest API与前端交互的服务，专门服务于前端，并进行数据聚合与参数校验。由于后端微服务之间统一使用Dubbo的Rpc接口进行交互，为了让后端服务只负责后端业务逻辑，我们设计了api服务，承载前端的请求并进行参数校验，调用Rpc接口进行数据聚合，让后端服务专注于业务逻辑开发，无需关心与前端接口交互问题。

5.2.3 data-service

首先给出我们数据获取的策略：主动获取与被动获取相结合、定时任务主动获取加接口被动获取。

此模块最重要的是通过Restful API从Github中获取数据，然而Github API有调用速率限制，每小时几千的请求数，并且每次请求结果返回的数据最多为100条，所以我们无法短时间内通过Github API来获取全量的数据，甚至只能获取一小部分数据，因此，我们需要判断数据的优先级，对于系统来说价值高的数据因当具有更高的请求优先级。

以下是我们提供给系统的接口获取数据的流程图：

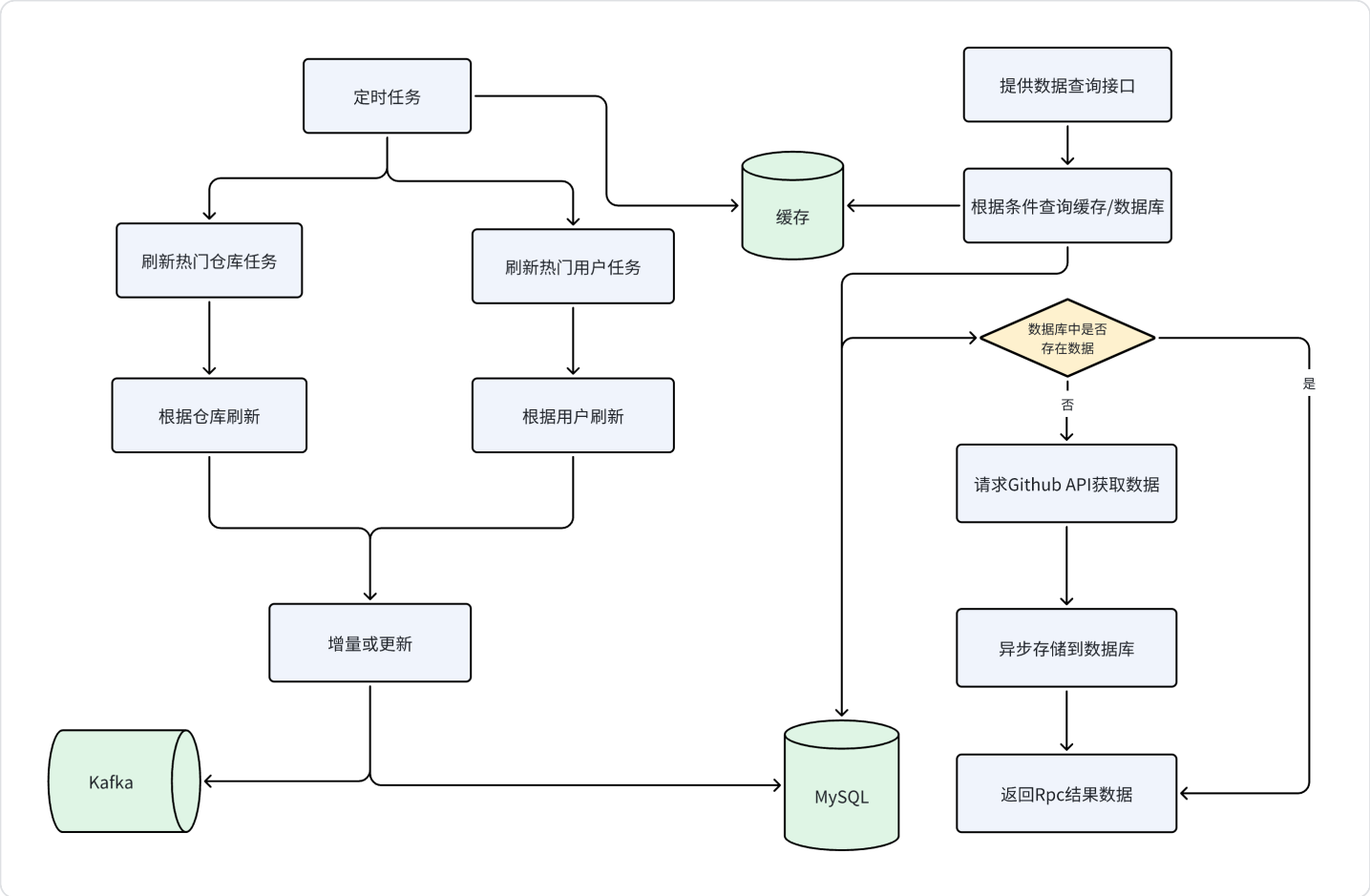


可以看出，第一次请求数据时没有数据就会请求Github API，然后异步将请求结果入库，下次相同条件的请求将不会请求Github API，然而这样会引发两个问题：

1. 第一次请求时依然可能会触发Github API速率限制，并且比较获取数据速率慢
2. 第一次获取数据后Github数据发生变化，系统数据库中的数据会变为过期数据

针对这两个问题，我们使用定时任务的方式去主动获取和刷新数据，而Github全量数据依然非常多，我们通过定时任务传参的方式指定数据获取范围以及根据热门数据去获取和刷新数据。获取新的数据后需要重新计算开发者信息，通过Kafka发送给业务系统进行计算。

从而我们的系统整体架构就是以下所示：



5.2.4 talentrank-service

talentrank-service主要负责计算开发者的技术能力。

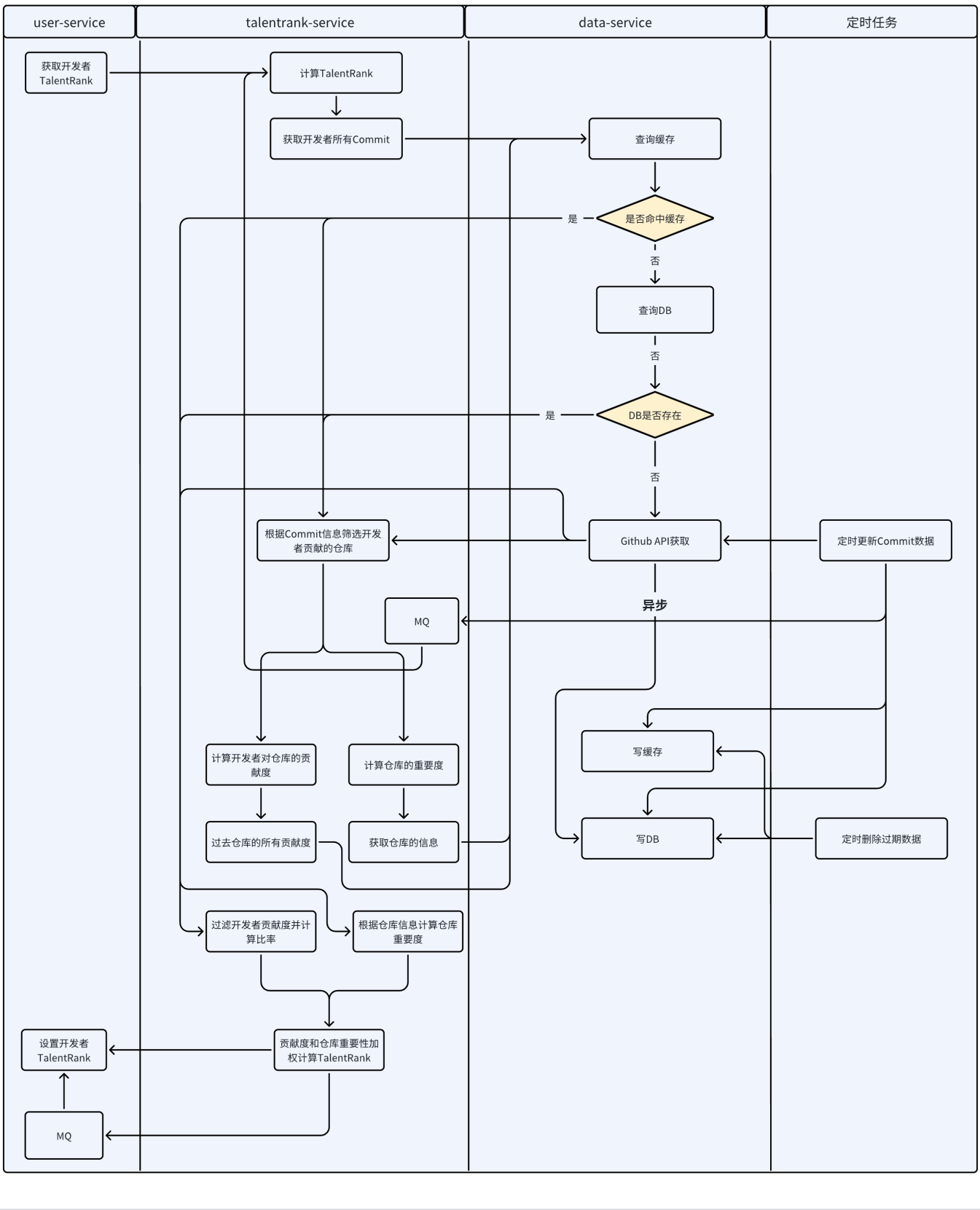
这里我们的开发能力定义为两个维度：

- 开发者参与贡献的仓库的重要度
- 开发对仓库的贡献度

其中，仓库重要度我们通过仓库的star数,fork数进行加权求和计算，开发者的贡献度我们使用开发者在仓库的贡献值与仓库的总贡献值相比。

我们计算TR的公式就是：仓库重要度*开发者贡献度。

以下是计算TR的整体流程图：



首先我们明确什么情况下会触发TR计算：

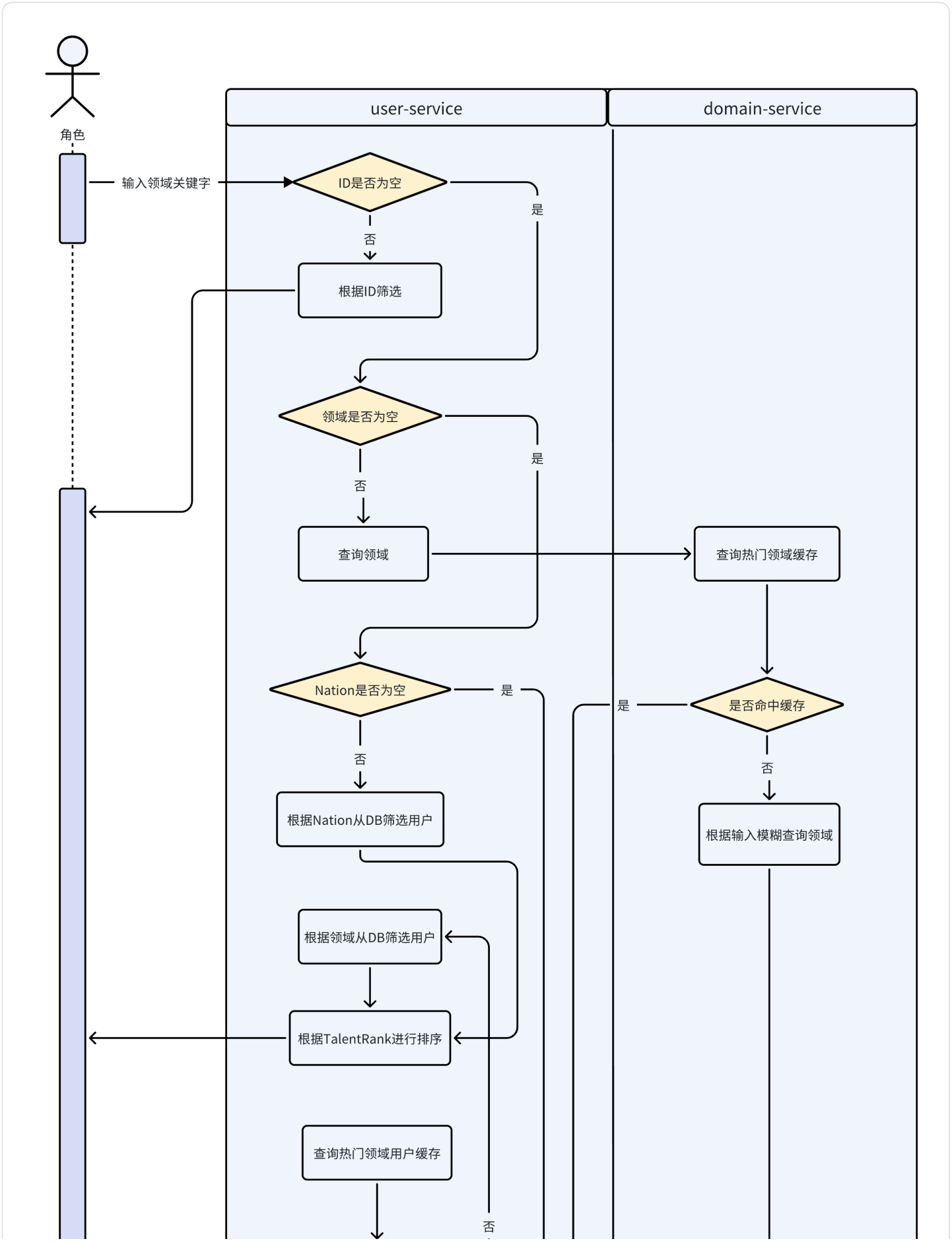
- 用户登陆时会触发一次用户自己的TR计算
- 查看开发者详细信息时会触发该开发者的TR计算
- 数据刷新后会通过MQ的方式触发热门仓库的贡献者的TR计算

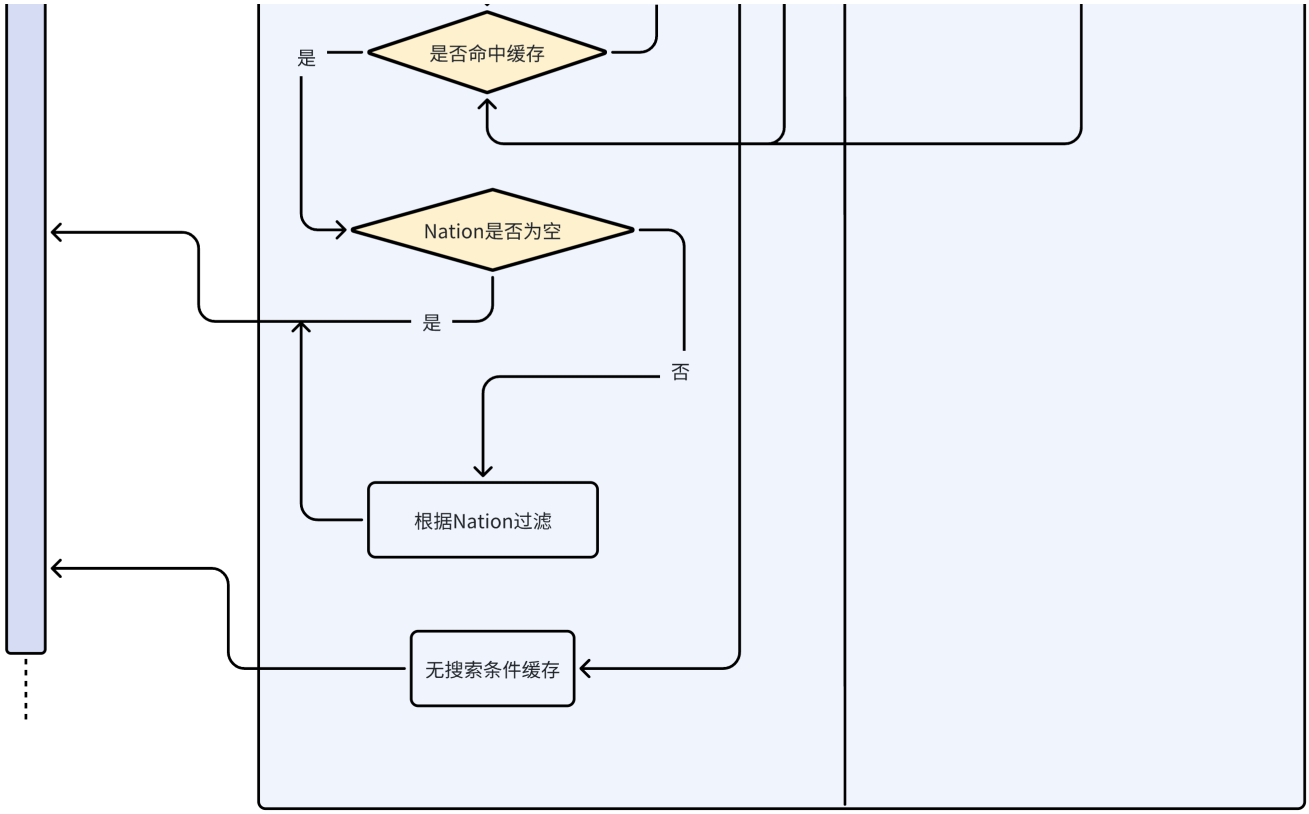
根据以上触发频率，我们可以在有限资源下最大化的对开发者的TR进行计算并保证一定的实时性。

5.2.5 user-service

用户服务最核心的功能是根据领域、Nation进行搜索。

以下是用户搜索的流程图：



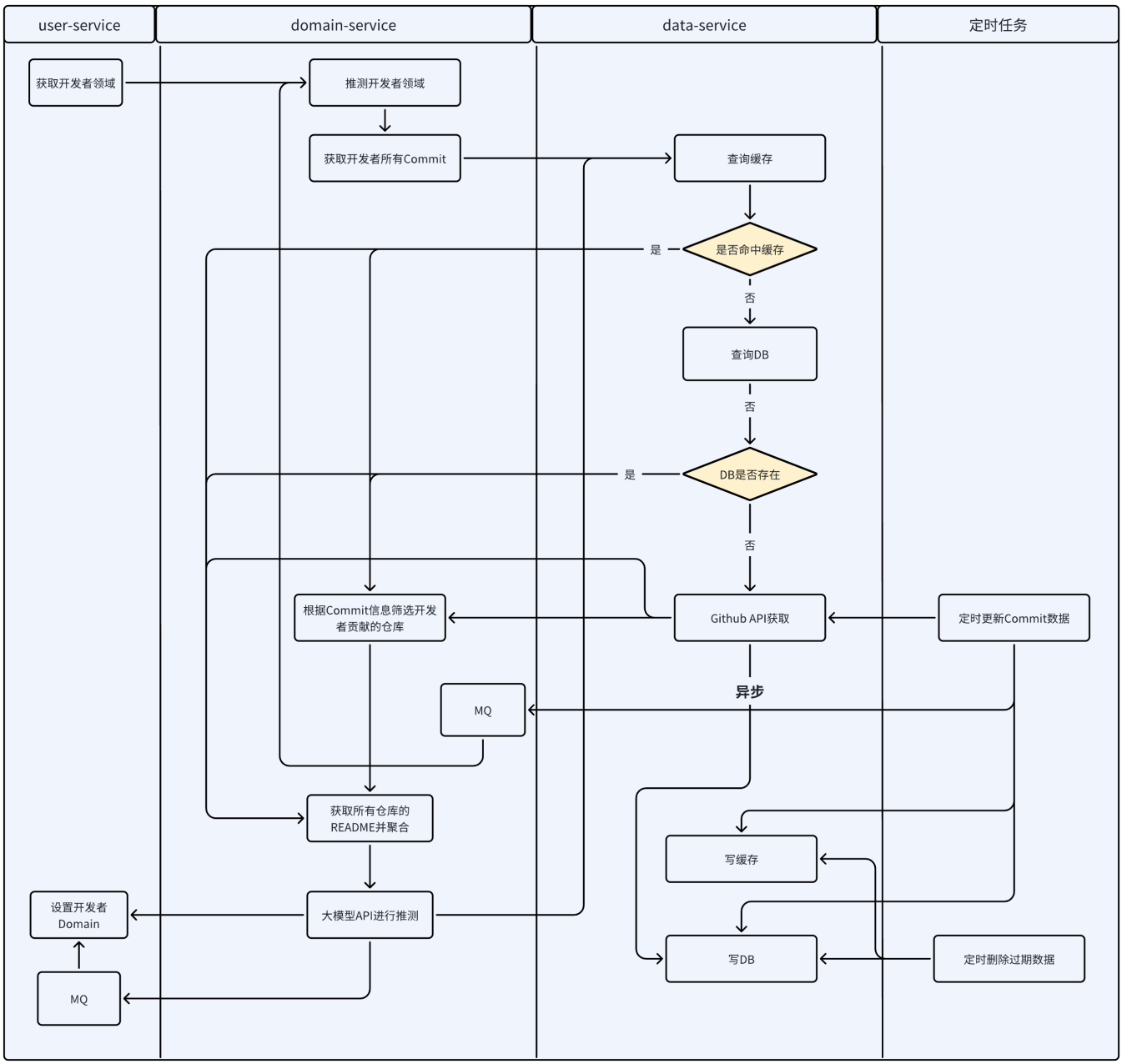


5.2.6 domain-service

domain-service的职责是推测开发者的领域以及维护热门领域等领域信息。

我们根据开发参与提交的仓库对开发者的专业领域进行推测。我们会根据开发者的Commit信息筛选到他所参与的仓库，然后提取其中的README信息，同时我们截取前一部分README为了保证推测的速率以及成本，最后将聚合后的README信息推送到大模型进行领域的推测。

以下是推测领域的流程图：



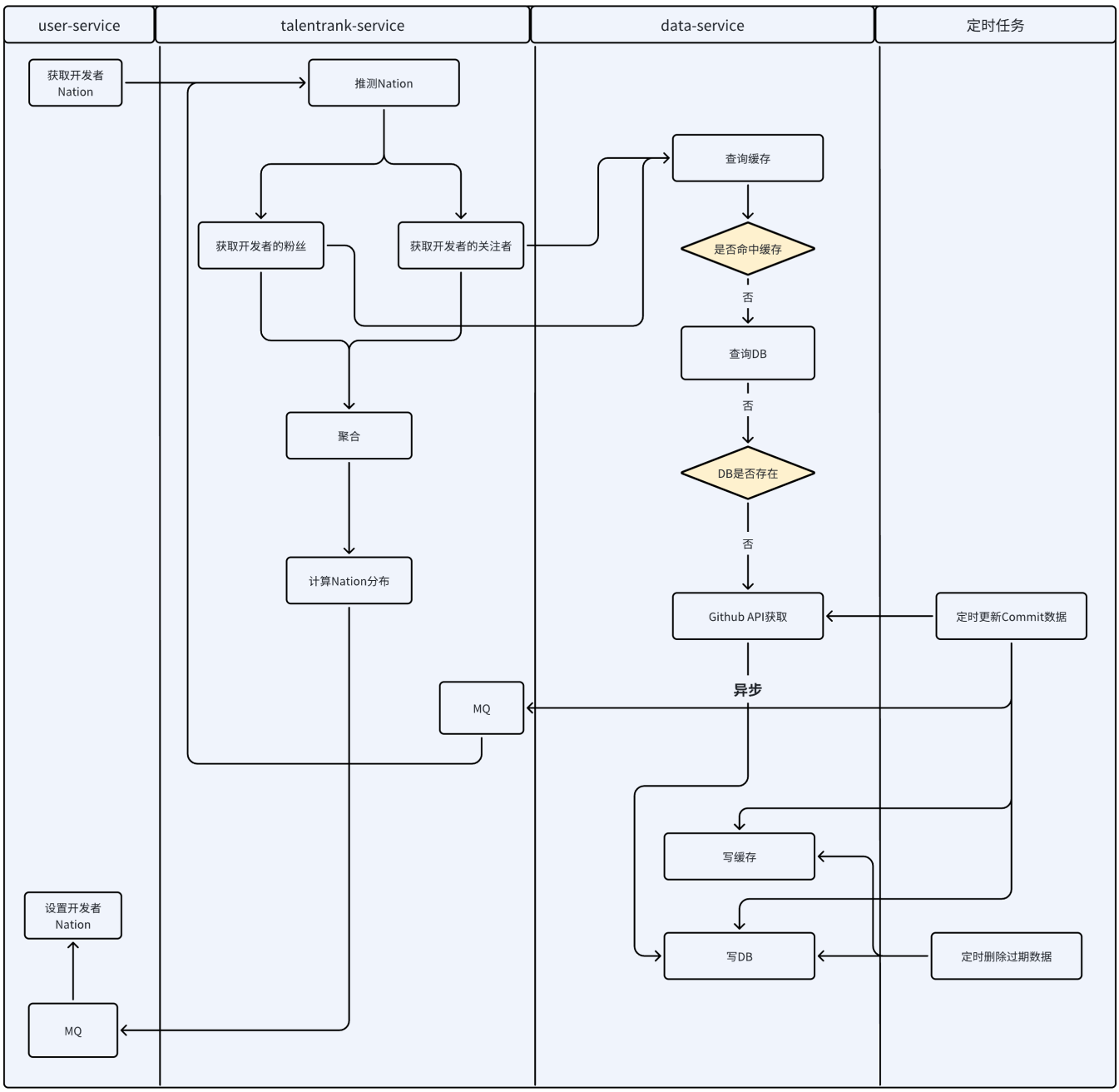
以下是触发推测的条件：

- 用户登陆时触发用户的推测
- 查看开发者详细信息时如果该开发者的domain为空，则触发该开发者的推测
- 数据刷新时通过MQ消费触发开发者的推测，并将结果通过MQ推送到用户服务

这里我们还使用定时任务对热门领域进行统计，并将其缓存到Redis中以承载大部分请求。

5.2.7 nation-service

以下是计算Nation的流程图：



以下是触发ation推测的条件：

- 用户登陆时触发用户的nation推测
- 查看开发者详细信息时如果该开发者的nation为空，则触发该开发者的nation推测
- 数据刷新时通过MQ消费触发开发者的nation推测，并将结果通过MQ推送到用户服务

5.3 设计总结

Gitgle 系统整体为Redis缓存加查MySQL数据库来为用户提供接口访问能力，热门信息将命中缓存大大提高查询速率，查库操作也都覆盖索引，保证了数据量大的情况下给用户提供快速友好的接口访问体验。将耗时的计算和推测操作完全由后台执行，避免阻塞用户操作。同时我们大量使用了异步和线程池技术提高I/O效率以及计算效率。我们还实现了对Github API的限流操作的应对措施，使用Gitgle系统侧存储和刷新机制结合被动触发数据获取机制，大大减少了Github API限制对系统的瓶颈。