目录
- 系统设计题
  - 如何设计一个即时通讯系统？
  - 如何设计一个推荐系统？
  - 如何设计一个搜索系统？
  - 如何设计一个短信系统？
  - 如何设计微博 Feed 流/信息流系统？
  - 如何设计一个短链系统？
  - 如何设计一个实时协同编辑系统？
  - 如何搭建大文件上传系统？
  - 如何设计分布式锁？
- Java
- 数据库
- 常见面试
- 分布式
- 高并发
- 编程题

如何设计微博 Feed 流/信息流系统？

“如何设计微博 Feed 流/信息流系统？”是一道比较常见的系统设计题，重点考察如何处理大规模数据分发、个人最新内容、有新粉丝关注时如何动态生成时间线等核心问题。

这篇文章将从设计目标、核心数据模型、两种经典方案（推拉模式）、大V用户的特殊处理、缓存与存储、以及高可用设计等角度，为你系统的梳理这个经典设计题。

阅读时间： 约 25 分钟

关键词： 推拉模式 · 时间线 · 缓存 · 消息队列 · 分布式存储 · 高可用

一、设计目标与核心挑战

1. 核心功能

- 用户发布一条新内容（文字、图片、视频）
- 关注的人发布内容，我的时间线（Feed）实时更新
- 支持下拉刷新、上滑加载历史内容
- 支持点赞、评论、转发等互动

2. 非功能性需求

- 时间线生成延迟 P99 < 500ms
- 单日活跃用户 1 亿，峰值 QPS 数万级别
- 大V（粉丝 > 100 万）发帖不能拖垮系统
- 高可用：单机故障不影响整体服务

3. 核心挑战

- 一个大V发一条内容，要推给 1000 万粉丝，瞬间写入 1000 万条时间线，写放大极大
- 一个人关注 500 个人，每次刷新要从 500 个人的内容中合并出最新 20 条，读放大极大
- 内容实时性 vs 系统吞吐：粉丝看到内容的延迟，和系统能承受的写入压力，是一对天然的权衡
- 冷启动：新用户关注了很多人，第一次打开时时间线是空的，如何快速填充？
- 删除/屏蔽：内容被删除、用户拉黑对方、内容被举报，如何保证时间线上这条内容消失？

二、核心数据模型

```
User(user_id, name, avatar, ...)
Post(post_id, author_id, content, media_ids, created_at, ...)
Follow(follower_id, followee_id, created_at)
Timeline(user_id, post_id, ranked_score, created_at)
Interaction(user_id, post_id, type, created_at)
```

- `Follow` 表是关注关系，决定谁的内容要推给谁
- `Timeline` 表是每个用户的时间线，存的是 `post_id` 和排序分数
- `Interaction` 表记录点赞、评论、转发，用于排序和统计

三、推拉模式（Push / Pull）

1. 推模式（Push / Fan-out on Write）

发帖时，主动把内容推送到所有粉丝的时间线上。

```
A 发帖
→ 查 Follow 表：A 的粉丝 B、C、D
→ 写入 Timeline(B, post_1)
→ 写入 Timeline(C, post_1)
→ 写入 Timeline(D, post_1)
```

- 优点：读时极快，直接从自己的 Timeline 表查最新 20 条即可
- 缺点：大V发一条内容，要写 1000 万条记录，写放大极大，消息队列积压、延迟飙升

2. 拉模式（Pull / Fan-out on Read）

发帖时，只存自己的内容，不主动推送。

```
B 刷新时间线
→ 查 Follow 表：B 关注的人 A、E、F
→ 从 A 的内容中拉最新 20 条
→ 从 E 的内容中拉最新 20 条
→ 从 F 的内容中拉最新 20 条
→ 合并、排序、取 Top 20
```

- 优点：写入极轻，大V发一条内容，只写一条记录
- 缺点：读时要实时计算，关注人数多、内容量大时，合并排序成本高，延迟不可控

3. 推拉结合（Hybrid）

- 普通用户：推模式，发帖时主动推送
- 大V用户：拉模式，粉丝刷时实时拉取大V的内容
- 合并时：时间线 = 推送过来的内容 + 实时拉取大V内容，按 `ranked_score` 排序

四、大V用户的特殊处理

1. 为什么大V是特殊情况？

- 大V粉丝数 > 100 万，发一条内容写放大 1000 万倍
- 大V日发帖 10 条，写入量 = 1 亿条/天，单表写入已经很吃力
- 大V内容被大量拉取，热点数据集中在少数内容上，缓存命中率高但单点压力大

2. 大V内容的拉取优化

- 大V内容单独存储，不写入粉丝的 Timeline 表
- 拉取时：从大V的内容表中按时间倒序取最新 20 条
- 缓存热点大V内容，减少重复查询
- 粉丝数阈值可配置，运营可调整大V的判定标准

五、缓存与存储

1. 时间线缓存

- 用 Redis 存储每个用户的时间线，key 格式：`timeline:{user_id}`
- 数据结构：Sorted Set，score 为 `ranked_score`，member 为 `post_id`
- 保留最近 500 条，超出部分淘汰
- 设置合理 TTL，长期不活跃用户的时间线自动过期

2. 内容缓存

- 热门内容的详情（文字、图片URL、点赞数）缓存在 Redis
- 使用 Hash 结构，key 格式：`post:{post_id}`
- 设置 TTL，内容更新或删除时主动失效

3. 数据持久化

- 主存储：MySQL 分库分表，按 `post_id` 或 `author_id` 分片
- 时间线可从 Follow + Post 重新计算，Redis 挂掉不丢数据
- 定期全量重建时间线，作为兜底方案

六、高可用设计

- 消息队列（Kafka）解耦发帖与推送，发帖服务只写 MySQL 和 Kafka，推送服务异步消费
- 推送服务水平扩展，按 `author_id` 分片保证同一大V的内容有序处理
- 熔断：推送服务压力过大时，暂停推送，只保证大V内容可拉取
- 降级：时间线生成失败时，回退到纯拉模式，保证可用性
- 监控：关注 P99 延迟、Kafka 消费积压、缓存命中率、MySQL 慢查询

七、删除/屏蔽的处理

- 内容删除：发布删除消息，异步从粉丝时间线中清理
- 拉黑：对方内容实时过滤，不再出现在时间线
- 举报：内容进入审核队列，审核通过后删除
- 取关：从对方内容中实时过滤，不主动清理已推送的内容

八、扩展思考

- 多维度排序：时间 + 互动热度 + 关系亲密度，加权计算 `ranked_score`
- 个性化推荐：在时间线中插入推荐内容，提升用户粘性
- 跨地域部署：时间线就近生成，内容跨地域同步
- 实时推送：通过 WebSocket 主动推送新内容，减少轮询

参考

- Feed 流系统设计 - 拉模式（可参考此文章）
- Feed 流设计：推拉混合的时间线 APP（可以以此APP高级别的 Feed 流的一部分）

相关问题

微博相关问题：Feed 流是如何实现的？ 知乎的相关提问

7 人点赞

写笔记 · ★ 3 · 小字

SnailDove · 05-29 14:11 · 阅 4601 · 已 1 · 修改

上一篇 如何设计一个即时通讯？

下一篇 如何设计一个短链系统？

加入课程，参与知识分享与交流

注册 或 登录 查看更多评论

立即加入

推荐评论 (1)

平凡君 · 06-13 17:04 · 作者广告

作者回复 · 微信搜索公众号

怎么把新粉丝吸引来？

注册 或 登录 查看更多评论

关于课程 · 使用帮助 · 隐私协议 · 服务协议 · English · 简体