用户体系总体设计

目录

用户体系总体设计 1

1 名词解释 4

2 设计目标 4

2.1 功能目标 4

2.1.1 需求汇总 4

2.2 性能需求 4

2.3 其他 5

3 系统环境 5

3.1 假设及与其他系统联系 5

3.2 相关软件及硬件 6

3.3 系统限制 6

4 设计思路及折中 6

4.1 核心数据设计 6

4.1.1.1 一般空间用户数据模型 6

4.2 空间用户数据操作模型设计 8

4.3 性能问题的考虑 8

5 系统设计，对外接口 8

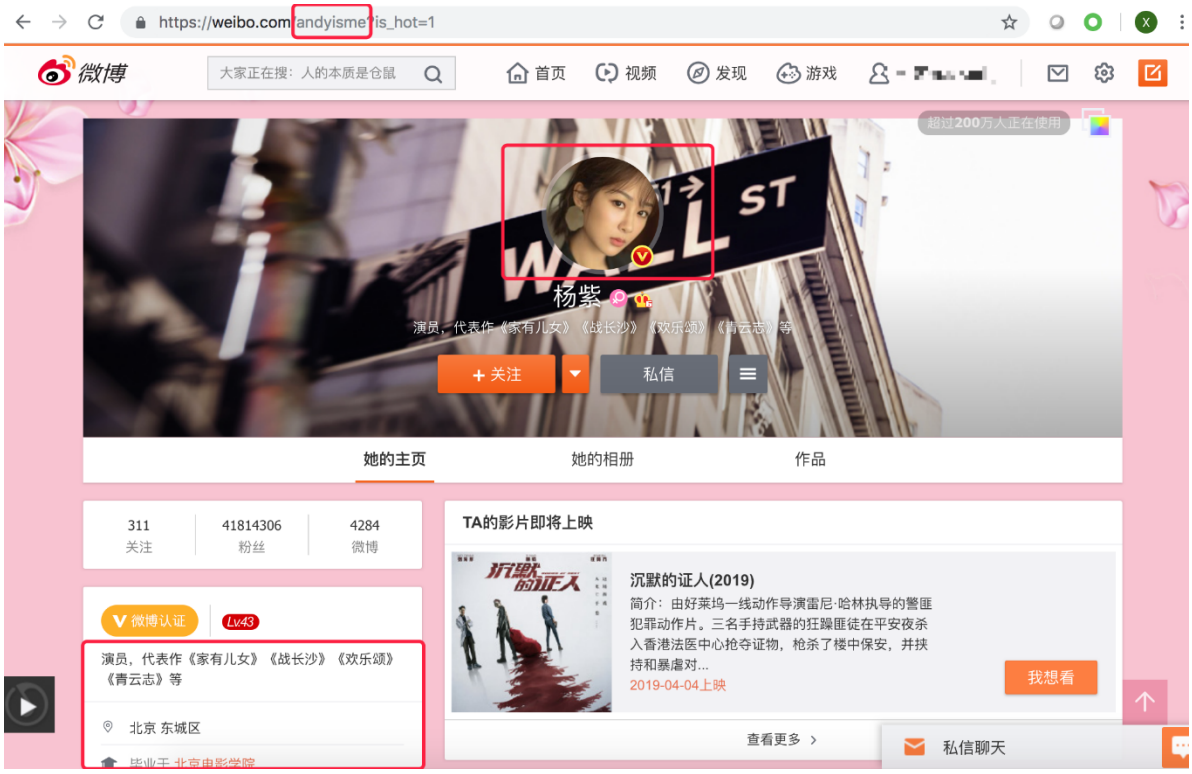
6 数据库，cache 设计 11

6.1 数据库 11

6.2 表定义 11

背景

随着空间（个人主页等）用户数量的日益增多，亟需对用户空间体系进行设计。



1 名词解释

空间用户体系

用户：uid

2 设计目标

2.1 功能目标

2.1.1 需求汇总

空间用户体系维护了空间用户的基本数据，需求汇总如下：

1. 用户注册空间；
2. 回收用户空间；通过 spurl 和 spid 等进行回收；
3. 查询空间基本信息；根据 spurl、spid 和 uid 等进行查询相应空间的基本信息；
4. 批量查询空间信息；根据 spurl、spid、uid 等批量查询空间信息，比如头像是否认证、空间是否封禁等等。

2.2 性能需求

空间用户体系是一个高速稳定的系统，它主要定位于用户空间信息的注册和修改以及空间信息的浏览相关操作。因此在整个性能方面，我们涉及包括如下的几个部分。

- 提交：

针对“空间用户注册”和“空间用户信息的修改和回收”这两类操作，要求提交吞吐量可以

达到 20000s^{-1}。

- 查询：

空间用户体系对查询的性能要求极高，可以通过全内存加冗余的方式解决其高性能，其全局的访问量要求可以达到 50000s^{-1}。

2.3 其他

1. 除了上面提到的需求，空间用户体系，还需要有其他的需求：将用户空间的注册解耦出来，提供单独的 spid 分配机制；
2. 空间用户体系将面向国际化的跨 IDC 部署，需要提供多 IDC 之间的数据互通机制，以及多 IDC 根据 spurl 排它分配 spid 的冲突解决机制；
3. 把用户和空间的一对一关系，扩展为多对多关系，一个用户可以拥有多个类型的空间，同样一个空间也可以由多个用户共同来管理(比如：情侣空间等)；
4. 空间用户体系需要提供及其高性能的查询，通过全内存的冗余机制解决之。

3 系统环境

3.1 假设及与其他系统联系

对于提交部分功能如下：

1. 空间用户信息的注册；
2. 用户空间信息的修改；

更新接口(定长 head+变长 body)，PB 格式。

对于查询部分：

1. 根据 spurl、spid、uid、username 等查询空间信息；

3.2 相关软件及硬件

新的空间用户体系运行于传统的 64 位 Linux 环境中 ,要求 16 核机器 ,Linux 内核 2.6，32GByte 内存。

3.3 系统限制

4 设计思路及折中

4.1 核心数据设计

4.1.1.1 一般空间用户数据模型

空间用户数据包括用户的基本信息(spid、spurl、uid)和用户空间一些属性信息(空间是否封禁、头像是否认证、文章的显示方式等)，我们对这些空间的用户数据进行抽象，对其建模如下：

字段	类型，说明
spid	uint64_t //空间的唯一标识 id
sptype	uint32_t //空间所属类型，个人空间，多人空间等
spurl	varchar(33) //空间的 url
spurltype	uint32_t //spurl 的来源类型：空间 URL、邮箱名、手机号等
uid	uint64_t //空间创建主人的 uid
flag	uint64_t //位标志，空间封禁、头像认证等等
other	varchar(65535) //扩展字段

other 字段是扩展字段，以 JSON(text)的形式进行存储，将来的需求字段都可以填入此

other 字段中。字段以 K/V 的形式放入 other 字段中，这样将使得整个表结构变得简洁。

在设计中，对于空间注册操作，需要判断此空间 URL 是否已经注册，同时，对于空间回收操作，会首先判断拥有此空间的用户数量，如果多个用户所有，只会解除此用户对此类空间的所有权，等多个用户都要求删除空间时，才会真正的回收此空间资源。

对于通用根据字符串排他性分配全局唯一 id 的服务，需要维护下面两种类型的表(产品线名_str_id 和产品线名_id_str)

- 产品线名_str_id:

字段	类型，说明
str	varchar(33) //字符串
strtype	uint32_t　//字符串的来源类型：邮箱名、手机号等
id	uint64_t // id
timestamp	uint64_t // 产生此条记录的时间

其中，{str，strtype}为主键

- 产品线名_id_str:

字段	类型，说明
id	uint64_t // id
str	varchar(33) //字符串
strtype	uint32_t　//字符串的来源类型：邮箱名、手机号等
timestamp	uint64_t // 产生此条记录的时间

其中，{id}为主键

4.2 空间用户数据操作模型设计

对于 logic，分别采用 uid、spid 取模进行存储，对于 di，采用对 spid 取除的方式进行存储。

在访问模式上，存在如下几种操作：

1. logic 和 di 的事务性插入新记录，需要根据 spurl 排他性的分配 spid(用户注册空间)；
2. 修改空间基本属性信息(spname，spdesc，spmtime 等等)；
3. 根据{uid，[sptype]，spid}或者{uid，[sptype]，spurl，spurltype}增加用户对空间的拥有权；
4. 根据{uid，[sptype]，spid}或者{uid，[sptype]，spurl，spurltype}对空间资源进行回收；
5. 根据{uid，sptype}[]或者{spurl，spurtype}[]或者{spid}[]查询空间用户数据；

4.3 性能问题的考虑

空间用户体系的更新量以及其特点，在性能上，大量的查询会威胁到整个空间用户体系的性能(每天的查询请求千亿亿次)。为了提供高速稳定的服务，采用空间用户数据全内存的方式解决海量的查询请求，并提供 Cache 的冗余机制解决查询服务的稳定和高可用性。

5 系统设计，对外接口

对于空间用户体系系统设计，具体架构如下：

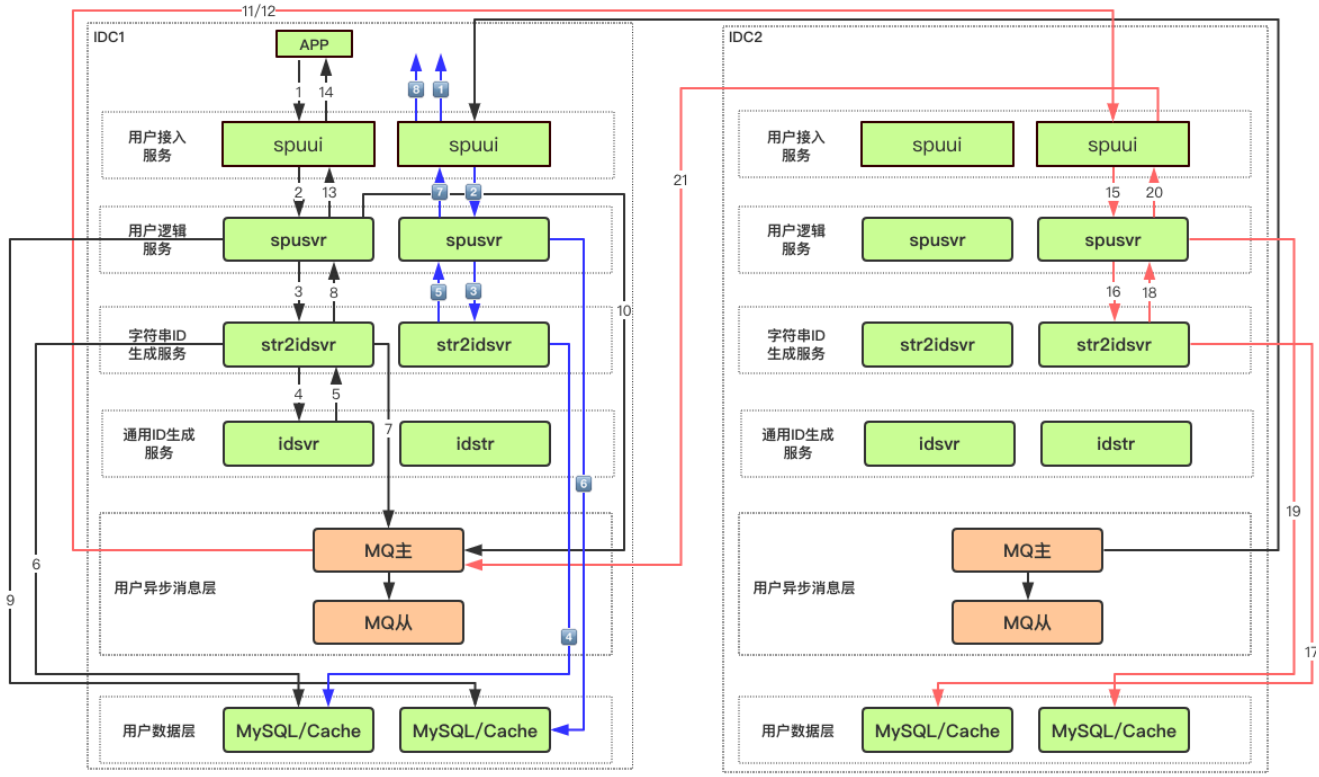


图 6-1 空间用户体系整体架构图

整个空间用户体系系统的设计，采用了用户接入服务、用户逻辑服务、字符串 ID 生成服务、通用 ID 生成服务、用户异步消息层、用户数据层这样的层次进行架构。

整个空间用户体系的分层，分为如下几个组成部分:

1. spuui 模块

   这个模块是空间用户体系的接入模块，实现了所有的接入操作。

2. spusvr 模块

   这个模块是空间用户体系的核心模块，实现了所有的数据操作。

   这些操作包括：

   a) 用户空间注册操作，这种操作，会提供一个”空间”的所有信息，其中 spid 由 str2idsvr 模块根据注册用户提供的 spurl 排他性分配。

   b) 一个空间将会被多个用户共同管理，因此根据{uid，[sptype]，spid}或者{uid，[sptype]，spurl，spurltype}增加用户对空间的拥有权

c) 修改空间基本属性信息(spname，spdesc，spmtime 等等)；

d) 根据{uid，[sptype]，spid}或者{uid，[sptype]，spurl，spurltype}对空间资源进行回收；

e) 根据{uid，sptype}[]或者{spurl，spurltype}[]或者{spid}[]批量查询空间用户数据；

f) 国际化跨 IDC 部署，需要接受其他 IDC 的空间更新数据，并更新主库；

g) 国际化跨 IDC 部署，为了保证 spurl 的全局唯一，需要定义偏序关系，解决多 IDC 同时分配 spurl 的冲突；

h) 统一序列化本 IDC 更新操作，提交到其他的 IDC。

spusvr 是空间用户体系的核心数据处理模块。

3. str2idsvr 模块

str2idsvr 模块根据 str 排他性的分配全局唯一的 id。此模块的操作包括：

a) 根据用户提供的{str，strtype}分配全局唯一的 id；

b) 序列化用户提交命令，并提交到其他的 IDC；

c) 接受查询服务(str<->id)；

d) 删除 str<->id。

4. idsvr 模块

直接使用多 IDC 下通用 ID 分配服务 idsvr。

str2idsvr 向 idsvr 请求获取全局唯一的 id。

5. 用户异步消息模块

直接使用 RocketMQ 消息队列。

spusvr 和 str2idsvr 把更新命令提交到 RocketMQ，转发到其他的 IDC。

在整个系统的设计上，对外形成如下接口层：

spuui 对外形成 HTTP/socket TCP 数据接口层

6 数据库，cache 设计

6.1 数据库

数据库使用 MySQL，存储引擎: InnoDB， 编码:UTF8MB4

- spusvr

数据库名为:spuser

分表策略

logic 按照 uid，spid 取模进行分表策略

di 按照 spid 去除进行分表策略

在部署上，采用主从的部署结构，多 IDC 之间需要数据的同步。

- str2idsvr

数据库名为:str2id

分表策略

按照 hash(str)，id 取模进行分表策略

在部署上，采用主从的部署结构，多 IDC 之间需要数据的同步。

6.2 表定义

cache 设计

所有的 cache ,准备直接让 spusvr 采用 Redis 作为缓存 ,并考虑到服务的高性能和稳定性，

需要提供用户数据的全内存和 cache 的冗余机制。

缓存包括如下的一些部分:

spid->di

{uid}->{sptype, spid}[]

{spid}->{uid}[]

{spid}->{spurl}

{hash(spurl)}->{spurltype，spid}[]

缓存代表全量的空间用户数据。