微博系统开发计划见：《微博系统开发计划》

开发分工简单描述：

1、前台的框架由孙奥完成（目前已经完成）

2、后台的框架由王辉完成（吴剑协助）

3、爬虫穆德宝完成

4、后台管理系统放到目前的OA系统中完成

5、其它各模块每个模块的开发周期大约是2周完成，平均每人2个模块

开发过程和方法：

1、首先由王贺提供模块的模板（html和css）

2、开发人员根据前台开发规范开始写自己的模块。

3、当需要和后台交互的时候则实现一个Action，每个交互是一个Action。

4、当需要Action里面需要涉及到数据存储和访问的则有王辉统一处理。

5、基于Action的日志系统和权限系统是自动的，不用实现代码，只需注册和使用（授权）。

6、和后台的所有交互都是通过前台提供的AJ完成。

7、所有的页面是基于模板输出，由后台TemplateServlet提供，每个模块需要根据自己的权限控制来实现具体的IModuleTemplate的html方法返回合适的HTML内容。

数据库设计方案

1、数据库结构已经建好，具体不写了，每个字段都有说明。

服务器地址：192.168.2.13

数据库名称：xk\_weibo

2、数据库表命名按照（模块）\_（业务表）

3、目前所有的数据都放在RDBM里面，表后缀为$的为零时设计，将来可能存放在其它地方。具体参考下面的《微博关系存储方案demo》

4、采样数据表标记为\*，跟踪表是业务采样都以\_Track结尾。

5、所有的日志都记录192.168.2.13的xk\_weibo\_logs的xk\_logs表。

6、数据库表设计中有任何遗漏都可补充，但必须按照上面的约束修改和添加。

# 数据库表设计

## Account\_\*：和微博帐号相关的表

Account\_BaseInfo$：微博帐号基本信息表，采样数据。\*

Account\_BaseInfo\_Track：微博帐号跟踪表，记录某些帐号的数据变化（粉丝数、微博数、关注数）。

Account\_Followers\_Track：微博帐号粉丝跟踪表，记录某个帐号的粉丝的粉丝数量、微博数量、关注数量。\*

Account\_Posts$：微博列表，用于传播分析，记录具体的微博信息。\*

Account\_Posts\_Track：微博跟踪表，用来记录某个微博的变化情况，方便趋势统计。

## Base\_\*：基础数据表

Base\_Actions：业务行为（Action）描述表。该表用来注册开发过程中使用的Action，每个开发人员和后台交互都通过Action完成，所以开发人员只需要写自己的Action类并且将其注册到系统里（就是写到这张表里面），微博系统运行的时候工厂方法会根据这张表来创建具体的Action。Action是有具有权限的，可以加入权限系统中。

Base\_Approved：业务审批表，未完成。

Base\_Customs：客户信息表，记录客户基本信息。

Base\_Logs：业务日志描述表，默认每个Action都会按照ActionName产生日志描述信息，其它日志描述信息需要自己创建系统才会自动记录。

Base\_Modules：系统模块基本信息。

Base\_Organizations：机构信息。

Base\_Permissions：基本权限信息表。

Base\_Profiles：个人配置信息表。K-V结构。

Base\_Roles：角色信息表。

Base\_Source：来源信息表，比如来自Iphone或者Note3或者网页，这是一个枚举。

Base\_User\_Approved：用户审批表。

Base\_User\_Roles：用户和角色关联表。

Base\_Users：用户基本信息表。

Base\_Weibo\_Accounts：官微或者要绑定的微博信息表。

## Disseminate\_\*：传播

Disseminate\_Posts：要进行传播分析的微博信息，具体的某个微博。

Disseminate\_Posts\_Track：微博跟踪表，跟踪指定微博的信息变化采样数据。

Disseminate\_Reposts$：微博回复列表\*。这个表可能很大。

## Event\_\*：事件

Event\_Events：事件基本信息表。

Event\_Weibo\_Track：事件对应的微博的跟踪表。

## Home\_\*：首页

Home\_Hots：首页热点信息表

Home\_Tasks\_Track：首页的舆情统计跟踪表，统计某个客户下所有的任务产生的舆情数量。

Home\_WeiboInfos：首页的微博基本信息表，显示当前客户下绑定的微博的基本信息，这个数据是个定时刷新的数据。

策略如下：根据服务器的配置（主要是API的限制，来计算本次调用的是API还是爬虫，按照企业特点和服务器资源计算每天每企业有几次调用API，当这个次数耗尽则走API），如果调用的是API则立刻返回新的数据并回写到基本信息表里面。如果是调用爬虫则显示上次的数据并产生一个爬虫任务爬虫回调更新，在用爬虫的时候这个值可能是一个快照值，会有延迟。

## My\_\*：我的微博

My\_Comments：我的回复表。保存当前用户的回复信息，记录当前用户用哪个微博帐号回复了什么信息。

My\_Follows\_Track：我的粉丝跟踪表。保存指定微博帐号的粉丝，以及这个粉丝的粉丝数量\*。

My\_Posts：我的微博表。通过微博系统发布的微博。

## Office\_\*：办公自动化

Office\_MessageReceivers：消息接收者信息

Office\_Messages：消息信息

Office\_Navies$：水军信心（网评员）

## Task\_\*：舆情

Task\_Collections：收集信息表

Task\_Groups：任务分组信息表

Task\_Tasks：任务定义信息表

微博关系存储方案(demo)

# 微博/博主存储

微博发布以后内容不会变化，可变化的内容由跟踪表跟踪变化。

## 微博内容存储（Store）

内容保存到讯库文件系统，支持定位（URL索引）。

支持关键字/表达式查询（由讯库支持）。

业务存储模型来实现具体的业务需求（微博系统实现）。

## 爬虫任务（Crawler）

定义（CralwerJob）：爬虫任务就是一组关键词，爬虫根据关键词去微博系统中爬取数据并建立讯库索引，并且调用微博系统的服务将爬取内容写入微博系统。

参数定义：

{

jobID:"作业编号，该编号由调度程序产生",

type:"keywords/url"

value:"关键字列表/URL",

depth:"深度",

callbackUrl:"回调的URL，执行成功后，把结果POST到callbackUrl"

result:"json"

}

返回结果：(JSON/XML)

{

err:0,//0无错,非0为错误码（自定义）

result:{...}

}

如果是同步调用则直接返回【返回结果】，如果是异步则调用callbackUrl（采用异步的方式）。

回写管道：爬虫回调的异步方法提供一个筛选管道的机制，可根据业务逻辑设置筛选条件分流回写数据。

interface IFilter{

void filter(string result, IDataStore store);

}

表达式过滤器：

class TaskFilter{

var task = gettask();

void filter(result,store){

var query = task.getQuery();

if(query.test(result)){

store.set(result);

}

}

}

回调可以考虑NodeJS可以解决大吞吐量的问题。

## 跟踪器（Tracker）

负责产生爬虫任务（一组关键词），并且提供关键词的排重（过滤重复的关键字）。

Task\_Tracker：任务跟踪器

Account\_Tracker：帐号跟踪器

Event\_Tracker：事件跟踪器

Weibo\_Tracker：微博跟踪器

...

Track\_Manager：跟踪器管理器

Job.Run(Track\_manager)

# 关系存储

## 业务关系分析

操作员：对应微博系统的帐号。该帐号是系统的内部帐号。

帐号：微博平台帐号，例如：腾讯微博帐号，新浪微博帐号。简称帐号。

### 微博帐号

1、授权帐号

通过系统注册的帐号受系统管理的帐号为授权帐号。

2、未授权帐号

除了授权帐号以外的帐号都是非授权帐号。

3、粉丝关系

关注了指定帐号的帐号。

4、好友（关注）关系

#### 存储模式

Radis存储粉丝列表和关注列表。

URL->(粉丝URL列表)

URL->(关注 URL列表)

微博系统不保存微博帐号的基本信息，需要保存业务数据，保存如下结构（**payload**）：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| URL | Follows | Friends | Weibos | FoAddress | FrAddress | WbAddress | ... |

URL：帐号的URL地址

Follows：粉丝数

Friends：关注数

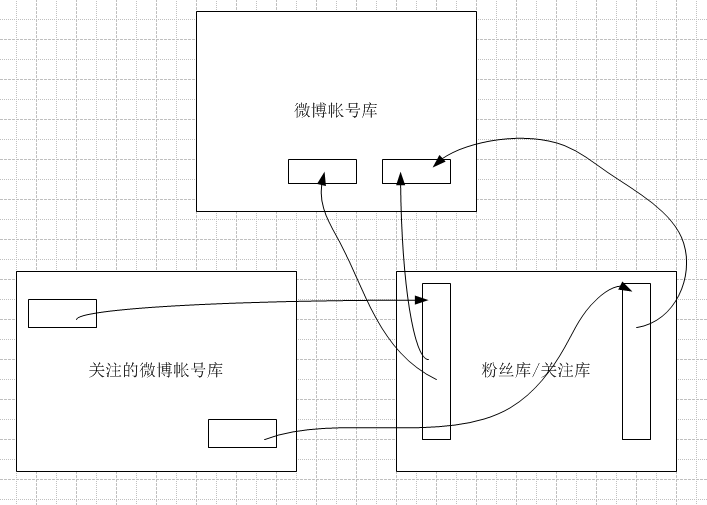
Weibos：发布的微博数

FoAddress：粉丝列表的地址，下图中的箭头

FrAddress：粉丝列表的地址，下图中的箭头

WbAddress：发送微博列表的地址，下图中的箭头

...：其他信息（情感信息）



微博帐号库：该库保存了所有系统打过交道的帐号，这个帐号库是各平台帐号库的一个采样集合。保存的是轻量级的帐号信息，参考payload。该库中的微博帐号几乎不更新或者更新频率很低。

关注的微博帐号库：微博系统需要监测的微博帐号库，包括授权帐号和非授权帐号的payload信息。

粉丝/关注库：专门用来存储关注微博帐号和微博库中的帐号的粉丝关系，Key就是关注帐号的ID，Value对应的微博帐号库里面的帐号。这里面保存的Value是已经按照粉丝数排好序的。

该方案可通过关注帐号迅速获得粉丝列表

### 微博

1、通过系统发布的我的微博

2、通过其他方式发布的我的微博（API获得）

3、跟踪微博转发、微博评论

URL->(转发URL列表)

URL->(评论URL列表)

URL->(情感URL列表)//赞

# 快速索引

原则： 关系数据库中保存元数据描述以及数字跟踪信息。

跟踪表都保存了完整的数字信息，采样数据则由爬虫和API完成。

URL是关联关系型和非关系型的KEY。

# 具体业务分析

## 微博舆情

CreateTask：创建舆情

TaskTracker.AddTask(Task)：添加到跟踪器开始跟踪任务

TaskTracker.GenCrawler()：产生爬虫任务

TaskTracker.Callback(json)：爬虫回调

基于任务的舆情系统，首先需要定义任务，根据任务产生爬虫任务，并且交给调度程序来调用爬虫，并且由爬虫回调爬取结果。

所以该部分信息是非实时（no-realtime）的（如果需要实时的可以调用商业的API接口来实现）。

admin create task -> task build Job -> job center -> Crawler run Job

operator view task -> invoke xunku webservice -> get fake realtime data from xunku-> show viewer

### 业务要求

筛选要求：

1、原创：讯库无法直接实现，需单独实现。

2、转发：同上。

3、平台（新浪、腾讯、人民），讯库的SourceID可直接获得，讯库支持。

排序要求：

1、按时间排序，讯库在时间字段上建立索引，讯库支持。

2、按转发排序，讯库无法完成，需单独实现。

3、按相似排序，同上。

4、按评论排序，同上。

响应时间：

1、调用商业API可立即获得数据。

2、调用爬虫需等待若干时间获得数据。（设计上要尽量缩短若干时间）

### 设计实现

在不考虑商业API的情况下，每个任务需要一个TaskDB用来保存这个Task相关的信息。

TaskDB是一个离线数据库，数据来源如下：

1、由爬虫回调函数完成数据的填充。

2、由JOB调度从讯库搜索来填充数据。

3、其它填充方法，API（如果有商业API则优先由API获得数据，商业API由用户自己购买）

~~TaskDB的特征是具有一类关键字的微博集合，该数据库由关键字来管理，所以该数据库是一个基于关键字的数据库~~。

列式存储（列可扩展）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| URL | Reposts | Comments | Similarities | PublishDate |

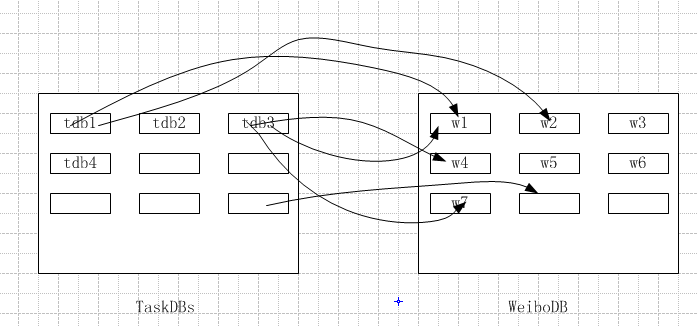
关键字会发生变化如何处理：

~~任务设置为有限多个关键字，关键字之间有逻辑关系，逻辑关系为AND OR NOT三种集合关系。例如：关键字组W1 AND W2 AND W3 OR W4 NOT W5。~~

1、重建任务库，抛弃一个DB，立刻重建一个并从讯库里面搜索数据填充库，重建库搜索讯库最近3个月的数据（已最快为目的）

2、追加任务库，历史数据保留

每个TaskDB都是任务到WeiboDB的索引管理器，每个TaskDB管理一组微博。



WeiboDB保存的是微博的基本信息集合，由讯库提供。

任务创建流程：

1、管理员创建任务

2、任务提交到任务管理器

3、管理器创建任务存储器

4、管理器创建任务作业（JOB）

5、提交任务作业到爬虫

6、等待爬虫回调任务管理器

7、回调后将数据写入任务存储器

任务使用流程：

1、打开任务视图

2、根据排序需求查询任务存储获得URL列表

3、Build查询结果集

4、返回结果集

## 事件监控

业务索引槽：根据每个业务需求，建立对应的索引槽满足业务排序的需求，索引槽都是排好续的。

索引槽存储：内存存储/硬盘存储？DUMP？