Uknow InfoHub

Design Document

BlXLRSMB Team

Contents

1 Introduction

1.1 Purpose

For that there are many developers that got involved into this project, this document was wrote, in order to provide a mannual that developers can refer to when they are confused or forgot some protocol. Besides, this is also an introduction for new developers, who may join after this project was done. It will act as a overview of this project, to which new hands can refer.

1.2 Background

Uknow InfoHub is a collector that aimed to gather information. It's a project which was assigned as team homework of 20132014 Fall, Softerware Engineering class of Tsinghua University. And it was bootstraped by blxlrsmb group, which has blahgeek, jiakai66, ppwwyyxx, vera, vuryleo and zxytim as members. It would be run on a VPS located in U.S., provided by Linode Ltd. co.

1.3 Definition

1.3.1 Items

An item is piece of information retrieved from various sources, and shown in lists for further reading. It consists of following parts:

- Title
- Source
- Content
- Labels
- Comments

Item is the core entity in Uknow system.

1.3.2 Labels

A label is an attribute describing items. An item can have multiple labels. A label associated with an item may be automatically assigned by system, or tagged

by users. A label can be either a system-wide label, which is visible to all user, or a user specified label which indicates user preferences on an item.

As described aforementioned, functions like 'archive' is actually a process of tagging an item by the label 'archived'

1.3.3 Plugins

Plugin is an essential concept in Uknow system, which comprises the implementations of varied functionalities in the system. A plugin should process a bunch of items, returning processed items. The number of items before and after need not to be the same, that is, a plugin can either shrink items or enrich items (filtering job) or process on contents of items.

1.3.4 Tabs

Tab is a collection of filtered items, in which the filter is defined by plugins. A typical use of tab is to divide items into different categories, which can be exclusive or not.

2 Archtecture

2.1 Fetcher

As the core component, fetcher is designed to be extremely flexible. It will be inherited into various kinds of fetchers to deal with different fetch targets. All the fetchers will be executed in a cluster as time based works, just like other web spiders, keeps fetching all the time.

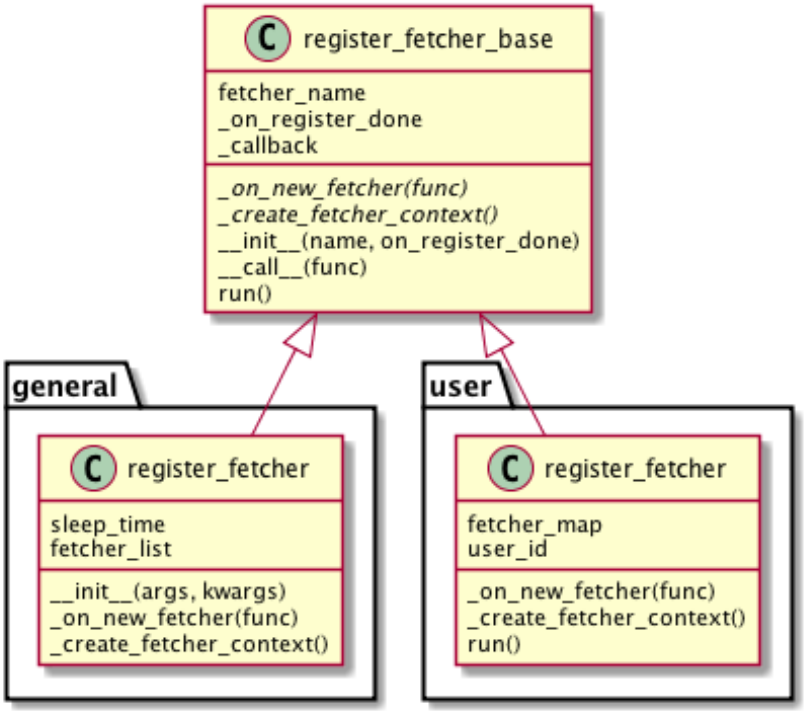

Figure 1: Fetcher

2.1.1 Function

The fetcher will be assigned to a target website or other kind of source like SNS or RSS feeds. Each time called, it will return a list of informations, almost in raw text but with some basic information, such as source, origin url, fetched time and so on.

2.1.2 Performance

Fetcher shall be called at least once per day. For hot resources such as SNS, it's fetcher must have high speed, so each time it shall return result in minute level.

2.1.3 Input

Fetcher don't need input after it's well configured. However, for augmentbility reason, a callback function is acceptable. The callback funtion will deal with the result by default.

2.1.4 Output

Fetcher will return a FetcherContext which wraps the raw information. For default situation, it will be dealed by a Prefilter.

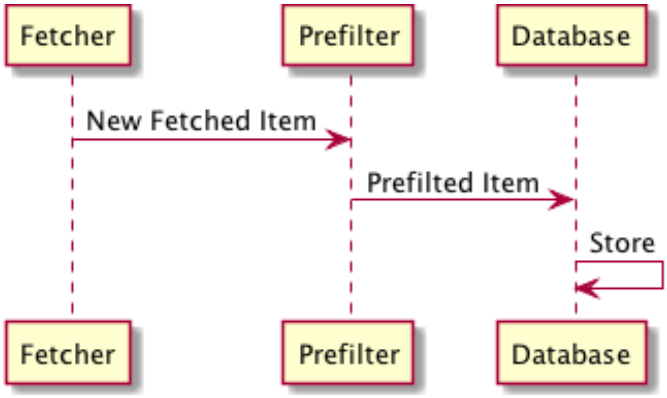

Figure 2: Workflow of Fetcher

Item fetchers simply retrieve items from various information sources, adding basic attributes such as ID, description, source URL (if available), and then forward items to prefilter for later processing, as illustrated in Fig. 2.

2.2 Prefilter

What users want definitely is not raw information. Thus a prefilter will be applyed on each piece of information after it's fetched. Prefilter will be called each time a fetcher returned a piece of information, then if it's not abandoned, prefilter will save it into database.

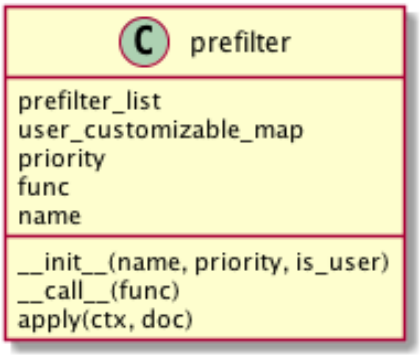

Figure 3: Prefilter

2.2.1 Function

Prefilter is the first classifiter applyed on the information. It can be system defined or user customized. Itself has priority levels, thus different prefilter will be executed on a information in order. It will put different tags on a piece of information, so in the next step it can be selected out easily.

2.2.2 Performance

It shall give the result almost immediately after it was called. So it only has several seconds of time.

2.2.3 Input

A FetcherContext item which was just fetched by a fetcher.

2.2.4 Output

A FetcherContext item which has multiple tags base on its content read by prefilter. Notice that a item may be abandoned by a prefilter for it may be duplicate, redundant or illegal.

2.3 Postfilter

Users can't read all the information that we fetched. Thus a postfilter is designed to search for information that users want to read. It will be called each time user make a action that need get new information.

2.3.1 Function

Postfilter is designed aimed to select the best information that users want. It will pick out information that have the same or similiar tags with what user requested. More, it will sort the information list in better order so user can get what they are most desired to get at first skim.

2.3.2 Performance

For that users must wait for it when asking for more information, it shall have extremely high performance, as fast as possible.

2.3.3 Input

A user defined query request, has various properties and limits.

2.3.4 Output

A list of information, which fits user's request well.

2.4 API Website

All things talked above is considered as the backend and core items of this project. What exposed to client software is the api website. Which will run in Linode server so that every client can find it easily.

2.4.1 Function

API website will check the requester's permission and validate the request format. Then it will call the components described above to get the response, then provide them to the requester.

2.4.2 Performance

The same, users must wait for it when posting request, so it shall have extremely high performance.

2.4.3 Input

Various kinds of requests.

2.4.4 Output

Responses to them.

3 Data Storage

3.1 Session

Session that used to determine which user is currently posting request is stored in redis which is a light key-value pair database. Though only one value can be stored in one key, its high performance makes it's the best to store sessions data.

The sessions is stored automaticly without any design, for a third party package will handle all of this.

3.2 Item

Item mainly is the only things that we need to store other than user's information which needed for all apps.

With considering of that each piece of information mainly doesn't have relationship with others but has multiple complex attributes in itself, NoSQL that doesn't need table struture but store data in documents is what we want.

An item will be mapped into a FetcherContext object in program, and it will be stored into database simply with all attributes in key-value pair format.

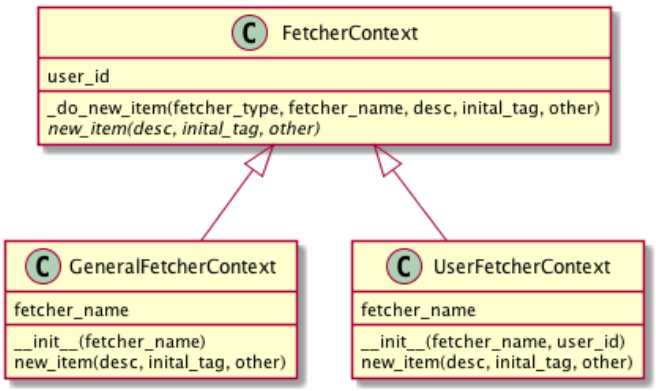

Figure 4: FetcherContext

3.3 Labels

Labels is a quite simple one in storage step. Each label is just a basic string describing itself. So it doesn't have even an individual table. Labels are stored as a part of Item, in a list as a subnode of it.

For performance reason, Labels are often used to filtering out Items, so index was build according to it.

4 Interface

4.1 Inner Interface

Interfaces that shouldn't be called or even seen by client are considered as inner interfaces. Mainly they are providing functions among components.

This section shall describe the usage of each interface, but we maintained a well constructed API document which describes each part of this project with explicit details. So, another duplicate description is not that useful, will causing more resources to maintain and may cause disagreements after changing some part of interfaces.

In a word, referring to API document is suggested when want to look up for one interface.

4.2 Client Interface

This part will discuss with caution dreadfully, for that interfaces may changed a lot when developing. Part that we can be sure of is that JSON is used when communicating between client and API server. Then the API server will call various methods provided by different classes that fit the requests.

For that this project is still in the bootstraping step, and it may change everyday, we decided to mantain the interface information with wiki and API document,

other than write it done on paper. Meanwhile, requests' format is still under discussion and it would change a lot. We don't think providing them now is a good idea.

More, the prototype system does not containing a client program and user involved part, so there's no need for stable client interfaces' design up to now.