**《数据库实践》课程报告**



**GitHub数据库设计**

**姓 名 林堃 纪宗尧 郭永旭**

**学 院 智能与计算学部**

**专 业 计算机科学与技术**

**年 级 2019级**

**任课教师 陈世展**

**2021年 10 月 2 日**

# 需求描述

需求概括：利用网络爬虫技术抓取 github.org数据，设计相应的本地数据库

需求细节：

1. 需要实现仓库信息的存储，包括仓库地址，所有者/组织，语言，参与贡献的成员，贡献（commit），issue，pull\_request;
2. 需要实现用户信息的存储，包括用户名称，邮箱，创建的仓库，参与的组织，收藏的仓库(star\_at),监视的仓库(watch),地区，注册时间等；
3. 需要设计关系型数据库的schema处理仓库和用户，仓库和仓库，用户和用户之间具有的关系，例如仓库与仓库之间存在fork\_from关系，用户和仓库之间存在star，watch，pull\_request,own，commit等关系，用户和用户之间存在follow，organize等关系；
4. 需要设计index提高数据库的查询效率；
5. 需要设计视图方便对常用的数据进行查询，例如
6. 需要实现存储过程供后台查询数据时调用，包括：

1. 查询一天、一星期、一个月内新增的仓库数；

2. 查询用户仓库，按获得的star降序排序；

3. 查询特定语言的仓库，按获得的star降序排序；

4. 查询本日/周/月新增仓库与上一日/周/月新增仓库的变化值；

5. 查询用户，按照获得star的数量降序排序

6. 查询最近三个月最活跃的组织；

7. 插入用户数据，并捕捉异常和处理异常（目前已知的异常包括插入重复异常）

8. 插入仓库数据，并捕捉异常和处理异常（目前已知的异常包括fork from外键不存在

异常，重复插入异常）

9. 获取一天内 国家为country 且提交了 language语言 项目 commit 的用户

10. 获取一个 issue 或 pull request 的所有参与者

11. 查询一个 pull request 中的所有action

12. 列出一个仓库所有的commits

13. 查询 Language语言（可多种，传入多个参数时结果取并集）使用的最新计数

14. 查询最近三个月最活跃的组织；

15. 查询今天/星期/月最多的 commit 来自哪个地区

1. 需要实现前端用网页展示出3中的数据
2. 需要实现后端，进行数据查询，供前端调用接口获取数据
3. 需要实现爬虫，爬取github上的数据。

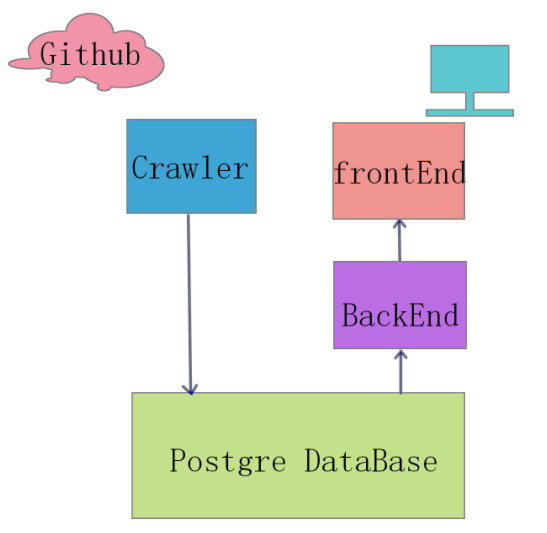
项目结构：

图1 项目结构

# E-R图

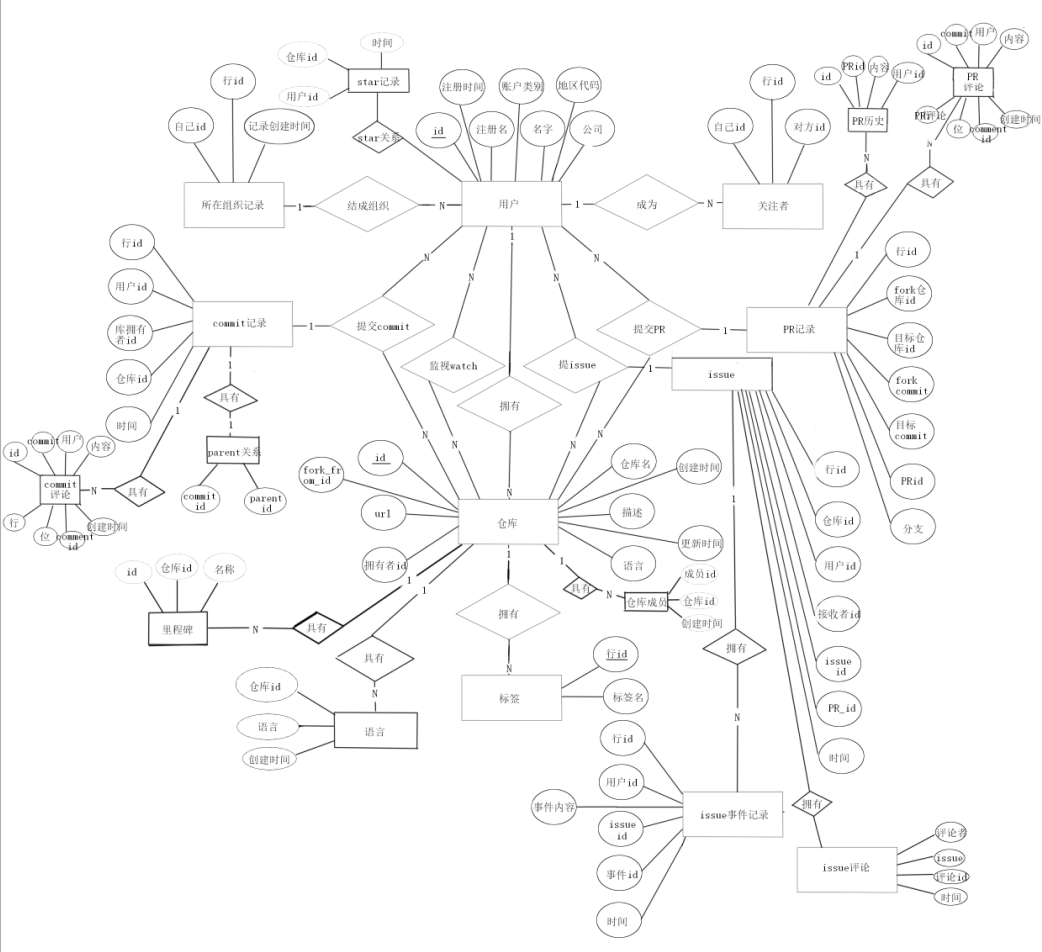


图2 E-R图

# Schema

## 总体设计

主要考虑了一以下组成：

实体的基本表，包含了实体的基本简单属性；

属于实体的附属表，记录了实体的多值属性；

实体与实体之间的关系表，记录了实体与实体之间的关系。

用户

* 用户基本表
* Follower是用户的多值属性，用table表示

仓库

* 仓库基本表
* 仓库里程碑表是仓库的一个多值的属性，用关系存储
* 仓库语言是仓库的一个多值属性，用关系存储
* 仓库标签是一个多值属性，用关系存储
* Project\_Commits是仓库的多值属性，用关系表示

仓库与仓库之间的关系

* Commit之间存在父子属性，用关系表示

用户与仓库之间的关系

* Pull request
* Issue
* Issue，pull\_request都存在多值属性comment，label，用关系表示

用户与用户的关系：

* organization 维护用户与用户之间结成组织的关系：
* organization\_member是organizattion的多值属性，用关系存储

存储过程

查询一天内/一周内/一星期内新增的仓库

查询用户仓库，按获得的star降序排序；

查询特定语言的仓库，按获得的star降序排序；

查询本日/周/月新增仓库与上一日/周/月新增仓库的变化值；

查询用户，按照获得star的数量降序排序

查询最近三个月最活跃的组织；

插入用户数据，并捕捉异常和处理异常（目前已知的异常包括插入重复异常）

插入仓库数据，并捕捉异常和处理异常（目前已知的异常包括fork from外键不存在）

触发器

## 详细设计

#### 用户

**用户table：**

create table "user"(

id int primary key,

login varchar(255),

name varchar(255),

company varchar(255),

email varchar(255),

created\_at timestamp,

type varchar(255),

fake int check (fake >= 0 and fake <= 255),

deleted int check (deleted >= 0 and deleted <= 255),

long decimal(11,8),

lat decimal(10,8),

country\_code char(3),

state varchar(255),

city varchar(255)

)

#### 用户与用户的关系

**follower表维护用户与用户之间的关注关系：**

create table followers(

user\_id int references "user"(id),

follower\_id int references "user"(id),

created\_at timestamp

)

**组织表 维护用户与用户之间结成组织的关系：**

create table organization\_members(

org\_id int,

user\_id int references "user"(id),

created\_at timestamp

)

#### 仓库

**仓库table：**

create table project(

id int primary key,

url varchar(255),

owner\_id int references "user"(id),

name varchar(255),

descriptior varchar(255),

language varchar(255),

created\_at varchar(255),

forked\_from int references project(id),

deleted int,

updated\_at timestamp

)

**仓库里程碑表是仓库的一个多值的属性，用关系存储**

create table repo\_milestones(

id int,

repo\_id int references project(id),

name varchar(24)

)

**仓库语言是仓库的一个多值属性，用关系存储**

create table project\_language(

project\_id int references project(id),

language varchar(255),

bytes int,

created\_at timestamp

)

**仓库标签是一个多值属性，用关系存储**

create table repo\_labels(

id int primary key,

repo\_id int references project(id),

name varchar(24)

)

**Project\_Commits是仓库的多值属性，用关系表示**

create table project\_commits(

project\_id int references project(id),

commit\_id int references commits(id)

)

**仓库的Commit之间的父子属性**

create table commit\_parents(

commit\_id int references commits(id),

parent\_id int references commits(id)

)

**Commit的多值属性commit\_comments**

create table commit\_comments(

id int primary key,

commit\_id int references commits(id),

user\_id int references "user"(id),

body varchar(256),

line int,

position int,

comment\_id int,

created\_at timestamp

)

#### 仓库与用户的关系

**Watchers是仓库与用户的关系之一**

create table watchers(

repo\_id int references project(id),

user\_id int references "user"(id),

created\_at timestamp

)

**Star\_at是仓库与用户的关系之一**

create table star\_at(

repo\_id int references project(id),

user\_id int references "user"(id),

created\_at timestamp

)

**Project\_member记录了用户对于仓库N对1的参与关系**

create table project\_members(

repo\_id int references project(id),

user\_id int references "user"(id),

created\_at timestamp,

ext\_ref\_id varchar(24)

)

**Commit是用户对于仓库N对N的关系**

create table commits(

id int primary key,

sha varchar(40),

author\_id int references "user"(id),

commiter\_id int references "user"(id),

project\_id int references project(id),

created\_at timestamp

)

**用户对于仓库提出pull request的关系**

create table pull\_request(

id int primary key,

head\_repo\_id int references project(id),

base\_repo\_id int references project(id),

head\_commit\_id int references commits(id),

base\_commit\_id int references commits(id),

pullreq\_id int,

intra\_branch int

)

**用户对于仓库提出issue的关系**

create table issues(

id int primary key,

repo\_id int references project(id),

reporter\_id int references "user"(id),

assignee\_id int references "user"(id),

issue\_id text,

pull\_request int,

pull\_request\_id int references pull\_request(id),

created\_at timestamp

)

**用户对于仓库的PR关系**

create table pull\_request\_commits(

pull\_request\_id int references pull\_request(id),

commit\_id int references commits(id)

)

**PR的多值属性comment**

create table pull\_request\_comments(

pull\_request\_id int references pull\_request(id),

user\_id int references "user"(id),

comment\_id text,

position int,

body varchar(256),

commit\_id int references commits(id),

created\_at timestamp

)

**Pr的多值属性history**

create table pull\_request\_history(

id int,

pull\_request\_id int references pull\_request(id),

created\_at timestamp,

action varchar(255),

actor\_id int references "user"(id)

)

**Issue的多值属性comments**

create table issue\_comments(

issue\_id int references issues(id),

user\_id int references "user"(id),

comment\_id text,

created\_at timestamp

)

**Issue的多值属性events**

create table issue\_events(

event\_id text,

issue\_id int references issues(id),

actor\_id int references "user"(id),

action varchar(255),

action\_specified varchar(255),

created\_at timestamp

)

**Issue的多值属性labels**

create table issue\_labels(

label\_id int references repo\_labels(id),

issue\_id int references issues(id)

)

-- indexs

**对于用户表，按照登录名的首字母次序进行排序后分段生成index，其他一些表按照id进行排序并分段生成index**

CREATE UNIQUE INDEX login ON "user" (login ASC);

CREATE UNIQUE INDEX sha ON commits (sha ASC);

CREATE UNIQUE INDEX comment\_id ON commit\_comments (comment\_id ASC);

CREATE INDEX follower\_id ON followers (follower\_id ASC);

CREATE UNIQUE INDEX pullreq\_id ON pull\_request (pullreq\_id ASC, base\_repo\_id ASC);

CREATE INDEX name ON project (name ASC);

CREATE INDEX commit\_id ON project\_commits (commit\_id ASC);

CREATE INDEX project\_id ON project\_language (project\_id);

#### 存储过程

-- 由于github api 爬不到star，所以暂时没有按star数量排序

##### -- 查询一天内/一周内/一星期内新增的仓库

-- 1 : 一天内， 2 : 一周内，3 : 一个月内

drop function list\_by\_star(time\_type int);

create or replace function list\_by\_star(time\_type int)

returns TABLE (

id int,

name varchar(255),

descriptior varchar(255)

)

as $$

declare

time\_gap varchar;

begin

if time\_type = 1 then time\_gap = '1 day'; end if;

if time\_type = 2 then time\_gap = '1 week'; end if;

if time\_type = 3 then time\_gap = '1 month'; end if;

return query

select project.id,project.name,project.descriptior from project

where created\_at::date > ((select current\_timestamp)::date - time\_gap::interval);

end;

$$ language plpgsql;

##### -- 查询这一天/周/月比上一天/周/月新增的仓库数量

-- 由于github api 爬不到star，所以暂时没有按star数量排序

create or replace function

increase\_count(time\_type int)

returns int

as $$

declare

time\_gap varchar;

cnt1 int;

cnt2 int;

begin

if time\_type = 1 then

time\_gap = '1 day'; end if;

if time\_type = 2 then

time\_gap = '1 week'; end if;

if time\_type = 3 then

time\_gap = '1 month'; end if;

select count(project.id) from project

where created\_at::date > ((select current\_timestamp)::date - time\_gap::interval)

into cnt1;

select count(project.id) from project

where

created\_at::date

<

((select

current\_timestamp)::date - time\_gap::interval)

and

created\_at::date

>

((select

current\_timestamp)::date - time\_gap::interval\*2)

into cnt2;

return cnt1 - cnt2;

end;

$$ language plpgsql;

##### -- 查询特定语言的仓库，按获得的star降序排序；

-- 由于github api 爬不到star，所以暂时没有按star数量排序

select \* from project

create or replace function repo\_of\_language(target\_language varchar )

returns TABLE (

id int,

name varchar(255),

descriptior varchar(255)

)

as $$

declare

time\_gap varchar;

begin

return query

select project.id,project.name,project.descriptior from project

where language = target\_language;

end;

$$ language plpgsql;

##### 查询今天最多的commit来自哪个地区

select \* from project

create or replace function most\_commit\_area()

returns TABLE (

country\_code int,

count int

)

as $$

declare

time\_gap varchar;

begin

select u.country\_code, count(\*)

from commits c, "user" u

where c.author\_id = u.id

and date(c.created\_at) = date(now())

group by u.country\_code;

end;

$$ language plpgsql;

##### 查询一个月内最活跃的组织

create or replace function active\_org()

returns TABLE (

login varchar(255)

)

as $$

declare

time\_gap varchar;

begin

return query

select distinct(u.login) as login

from commits c, "user" u, project\_commits pc, "user" u1, project p

where u.id = c.committer\_id

and u.fake is false

and pc.commit\_id = c.id

and pc.project\_id = p.id

and p.owner\_id = u1.id

and p.name = 'rails'

and u1.login = 'rails'

and c.created\_at > DATE\_SUB(NOW(), INTERVAL 3 MONTH)

union

select distinct(u.login) as login

from pull\_requests pr, projects p, "user" u, "user" u1, pull\_request\_history prh

where u.id = prh.actor\_id

and prh.action = 'merged'

and u1.id = p.owner\_id

and prh.pull\_request\_id = pr.id

and pr.base\_repo\_id = p.id

and prh.created\_at > DATE\_SUB(NOW(), INTERVAL 3 MONTH)

and p.name = 'rails'

and u1.login = 'rails';

end;

$$ language plpgsql;

**-- 列出一个仓库所有的commits**

create or replace function active\_org()

returns TABLE (

login varchar(255)

)

as $$

declare

time\_gap varchar;

begin

return query

select c.\*

from commits c, project\_commits pc, projects p, users u

where u.login = 'rails'

and p.name = 'rails'

and p.id = pc.project\_id

and c.id = pc.commit\_id

order by c.created\_at desc;

end;

$$ language plpgsql;

**-- 查询一个 pull request 中的所有action**

create or replace function active\_org()

returns TABLE (

action varchar(255),

created\_at varchar(255),

login varcher(255)

)

as $$

declare

time\_gap varchar;

begin

return query

select user, action, created\_at from

(

select prh.action as action, prh.created\_at as created\_at, u.login as user

from pull\_request\_history prh, users u

where prh.pull\_request\_id = ?

and prh.actor\_id = u.id

union

select ie.action as action, ie.created\_at as created\_at, u.login as user

from issues i, issue\_events ie, users u

where ie.issue\_id = i.id

and i.pull\_request\_id = ?

and ie.actor\_id = u.id

union

select 'discussed' as action, ic.created\_at as created\_at, u.login as user

from issues i, issue\_comments ic, users u

where ic.issue\_id = i.id

and u.id = ic.user\_id

and i.pull\_request\_id = ?

union

select 'reviewed' as action, prc.created\_at as created\_at, u.login as user

from pull\_request\_comments prc, users u

where prc.user\_id = u.id

and prc.pull\_request\_id = ?

) as actions

order by created\_at;

end;

$$ language plpgsql;

-- 获取x天前的仓库数量，x可以指定

create or replace function num\_days\_ago(days int)

returns int

as $$

declare

time\_gap varchar;

cnt1 int;

cnt2 int;

begin

select count(project.id) from project

where created\_at::date > ((select current\_timestamp)::date - '1 day'::interval\*(days-1))

into cnt1;

select count(project.id) from project

where created\_at::date > ((select current\_timestamp)::date - '1 day'::interval\*(days))

into cnt2;

return cnt1 - cnt2;

end;

$$ language plpgsql;

#### 触发器

每次star产生时对star\_count 进行记录

#### 异常捕捉

1. fork from不存在异常捕捉和插入重复异常捕捉，用途：在爬取数据时防止外键错误（引用的行没有被爬下来），插入数据时启用触发器，临时插入被引用行的主键属性

这个异常发生时 postgre server 返回的是 23505， 并且这一 sql 是在python中数据爬取下来后执行的，所以采用下面的方式：

sql = '

BEGIN

INSERT INTO project VALUES ({},'{}', {},'{}','{}','{}','{}', {},  {},  '{}'  )

EXCEPTION

    -- 违反外键存在约束

    WHEN 23503 THEN BEGIN

        INSERT INTO project VALUES({},null,null,null,null,null,null,null,null,null);

        INSERT INTO project VALUES ({},'{}', {},'{}','{}','{}','{}', {},  {},  '{}'  );

        END;

    -- 违反主键 id 唯一 约束

    WHEN 23505 THEN BEGIN

        ALTER TABLE project

        set url = '{}', owner = {}, id = {}, name = '{}',des = '{}',language = '{}', created\_at = '{}',fork\_from = {}，deleted= {}， updated\_at= {}

        where id = {};

    END;

END;

'

.format(repo["id"],repo["url"],repo["owner"]["id"],repo["name"],des,repo["language"],repo["created\_at"],repo["fork\_from"],'null',repo["updated\_at"],

repo["fork\_from"],

repo["id"],repo["url"],repo["owner"]["id"],repo["name"],des,repo["language"],repo["created\_at"],repo["fork\_from"],'null',repo["updated\_at"],

repo["url"],repo["owner"]["id"],repo["name"],des,repo["language"],repo["created\_at"],repo["fork\_from"],'null',repo["updated\_at"],

repo["id"]

);

#### 经验总结

在本次实践中我们总结出了一些经验：

* 要在python等高级语言中执行sql语句操作数据库时，以往我们习惯于把sql写成字符串来执行，其实更为方便的做法是写成存储过程，在高级语言中直接调用call function(参数)；
* 在本次实践中我们熟练掌握了多值属性的存储方式，就是用关系存储属性值，并且该关系以原来的实体为外键；
* 我们在思考数据库的结构时容易按照表中嵌套表的模型去建模，这样在一定程度上更贴近现实，降低了建模难度，尽管实现时不能直接实现，但是经过这种思维建模出来的模型比较简单，用数据库的关系将这些“嵌套的表（或者说是具有子属性的属性）”表示成多值属性，可以建立出易于理解的数据库；
* 在进行数据库设计时，我们初期选取了一些显然具有唯一性的属性作为主键（如用户的邮箱），但是在爬取github数据时，我们发现github的每一个表都设置了一个额外的id作为主键，而不使用那些显然具有唯一性的属性，这给了我们启示，我们从中学到简化和格式统一化的规范；

#### 细节讨论

**仓库文件版本迭代与分支管理的解决方案**

这是一个困难的问题，我们在进行了一些思考后，为了防止我们自身的思考过于局限，我们在网上搜索了非常切合github本身实现办法的方法

我们的想法

- 仓库的每一个版本都独立存储为一个实体

- 只存储仓库的第一个版本，随后每次有更新就只存储更新的内容

在Quora中得到的优化解

- 每一个commit的版本有一个哈希值

- commit记录了这个版本中所有object，

- 仓库commit后没有被修改的文件哈希值不变，被修改的哈希值改变

- 每个commit有一个哈希能够映射commit对应的所有文件

- commit之间使用parent关系表示版本之间的关系

实现：

commit 以仓库的 id 为外键， commit 设置 parent 属性指向它的上一个版本。

**仓库fork from信息爬取**

在爬取仓库时，会发现有的仓库fork from其他仓库，按照正常的建库逻辑，fork from（在表中是int属性，仓库的id）的源头必须存在；但是在爬取仓库信息时，可能会遇到fork from的仓库还没有被爬取到的情况，会因此而违反外键约束，对此有以下解决办法可参考：

- 捕获 Exception，跳过当前行的插入

- 捕获 Exception，插入当前行并把 fork from 置为 null

- 捕获 Exception，插入当前行，并使用触发器在 project 表中插入 fork from，后续爬取数据时如果爬到这个 fork from 的仓库则不执行 insert 操作而执行 alter 操作

我们逐步地想出了这三种方案，但在尝试了第三种方案后，我们退回来选择了第二种方案，原因是第三种方案在异常处理中仍然可能遇到for from的仓库没有爬取下来的问题，这样就会嵌套/递归地进行异常处理，非常复杂，因此我们“回滚”到了第二种方法。

**主键的选取**

注意到用户是通过邮箱登录的，因此考虑把邮箱作为主键（能节省使用其他额外字段作为属性的空间），但是爬取github数据时发现仓库，用户，commit都有id作为主键，意识到应该给用户分配一个 int 作为id；

**计数缓存的设计**

**一些属性需要计算数值，例如star的数量，watch的数量等，我们考虑设置缓存和不设置缓存两种方案，如果设置计数缓存，需要设置触发器对缓存进行更新；如果不设置计数缓存，则在查询时需要耗费更多时间；我们设置了计数缓存，以提高后台的查询效率。**

**评论的设计**

**评论的设计：**

**- 在issue，commit，pull\_request都用到了评论，对此我们讨论出两种可选的方法：**

**- 1. 三个场景都是用相同的评论schema，好处是不用建更多的表，坏处是查询代价会提高（一开始的设计）**

**2. 三个场景分别建表，好处是查询代价可以降低（经过小组成员审核后提出的方案）**

**- 我们选择了第二种方案，只需要再建表时多做几步，就可以持久地提高数据库的效率。**

## 爬虫

一部分数据用 scrapy 爬取，一部分数据用api爬取，使用 yaml 保存爬虫运行时将用到的信息，如密码，需要执行的 sql 等。

步骤：

* 从yaml获取参数
* 请求数据
* 如果不用url，解析数据生成json
* 解析json中的值
* 插入到数据库

###### 前端设计

Index页面:仓库趋势页面，展示一天内，一周内，一个月内的popular repo 及其数量，展示的火热仓库尝试着实现可以按照地区，语言，标签等进行分区

* 7日内新增仓库数量变化
* 本月新增仓库的数量与上一月的变化
* 本季度新增仓库数量与上一季度相比的变化
* 每日收获star数量用户排行榜
* 每日收获star数量仓库排行榜
* 每日star数量所用语言排行榜
* 本月收获star最多的仓库

Table页面：显示更多的表：

* 列出一天内 国家为country 且提交了 language语言 项目 commit 的用户
* 列出一个 issue 或 pull request 的所有参与者
* 列出一个 pull request 中的所有action
* 列出一个仓库所有的commits
* 列出不同 Language语言使用的最新计数
* 列出最近三个月活跃的组织排行；
* 列出今天/星期/月最多的 commit 来自哪个地区

爬取github仓库到数据库后展示到前端，使用sql筛选要展示的数据，使用前面实现的存储过程进行查询

效果：

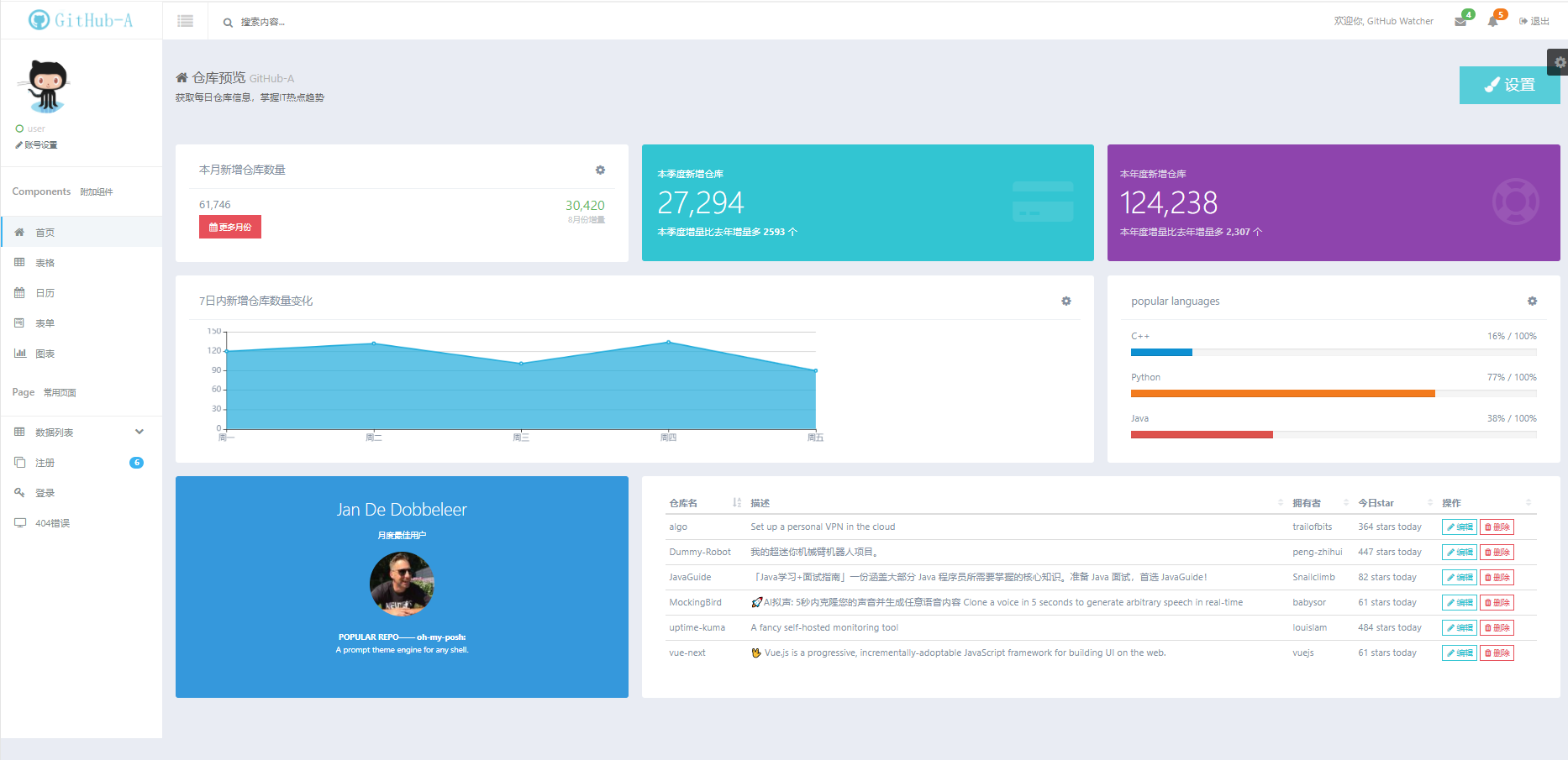


图3 index页面效果

#### 后端设计

使用flask部署python app作为后台运行的server，提供以下接口：

* 查询一天/周/月内的仓库，
  + 传入参数1：天
  + 传入参数2：周
  + 传入参数3：月
  + 传入参数4：季度
  + 传入参数5：年
  + 返回json，内容为查找到的仓库列表，
* 查询一天/周/月内的新增仓库数量，返回int，
  + 传入参数1：天
  + 传入参数2：周
  + 传入参数3：月
  + 传入参数4：季度
  + 传入参数5：年
  + 返回int
* 查询任意指定一天的当天新增仓库数量
  + 传入参数：days天前，int
  + 返回days天前新增仓库的数量，int