دانشگاه صنعتی امیرکبیر

(پلی‌تکنیک تهران)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

گزارش کارآموزی (هفته دوم)

محل کارآموزی: شرکت سامانه گستر سحاب‌پرداز

نام استاد کارآموزی

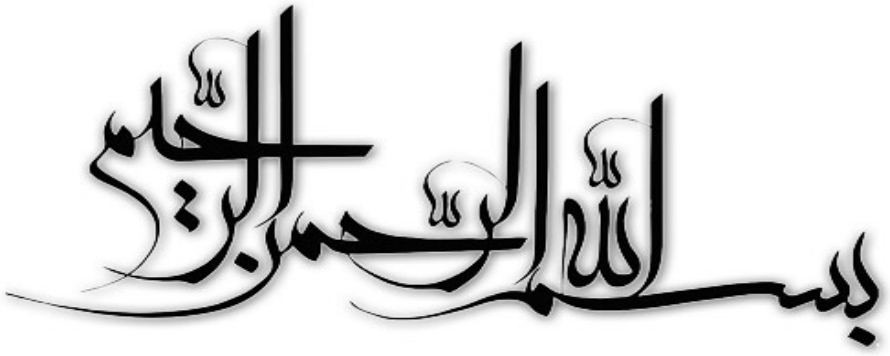
دکتر مسعود صبائی

نام دانشجو

امیرمحمد پیرحسینلو

۹۵۳۱۰۱۴

تابستان ۱۳۹۸


بسم الله الرحمن الرحيم

فهرست مطالب

فهرست اشکال

فصل اول
پروژه اول

۱-۱ مقدمه

در هفته‌ی دوم، یک پروژه تعریف شد که در باید در قالب تیم‌های دو نفره انجام می‌شد. تعریف پروژه در ادامه آمده است. هدف این پروژه آشنایی با رابط‌های برنامه نویسی و تمرین کار گروهی است. در انتها کدهای تیم‌ها مورد ارزیابی قرار گرفت و نواقص رفع شد.

۱-۲ تعریف پروژه اول

در این پروژه قصد داریم با استفاده از رابط برنامه نویسی سایت `satori.com` به کانال **github**[۱] دسترسی پیدا کرده و اطلاعاتی را از آن استخراج کنیم. این اطلاعات شامل نام کاربری افراد با بیشترین تعداد **commit** و **push** در سایت `github.com` در یک ساعت اخیر است که به طور تناوبی هر یک دقیقه یک‌بار این اطلاعات استخراج می‌شود.

۱-۳ ابزارهای پیاده‌سازی پروژه

برای پیاده‌سازی پروژه از ابزارهای زیر استفاده شده‌است:

۱-۳-۱ زبان برنامه نویسی جاوا

زبان برنامه نویسی جاوا یک زبان سطح بالا است و یکی از قدرتمندترین زبان‌های برنامه نویسی به شمار می‌رود که در اکثر زمینه‌ها کاربرد دارد. از ویژگی‌های آن می‌توان به موارد زیر اشاره کرد:

- **platform independency**[۲]
- **garbage collector**[۳]
- فراهم بودن تعداد بسیار زیادی از **library** ها و **API (Application Programming Interface)**[۴] ها
- مستندات عالی برای زبان
- تعداد بسیار زیادی ابزار مدیریت **package** و...

۱-۳-۲ git

git یک ابزار مدیریت نسخه است که در توسعه نرم‌افزارها از آن استفاده می‌شود. از کاربردهای آن می‌توان به نگه‌داری کدها، آسان‌سازی برنامه نویسی برای تیم‌ها و موارد بسیار زیاد دیگر اشاره کرد.

۱-۳-۳ maven

maven یک ابزار مدیریت **package** است که برای ساخت نرم‌افزارهای جاوایی از آن استفاده می‌شود. هنگامی که تعداد کتابخانه‌های استفاده شده در پروژه زیاد می‌شود، به شدت روند ایجاد فایل **jar** را آسان می‌کند.

---

[۱] `https://www.github.com`

[۲] مستقل از سکو است، یعنی در هر سیستم عاملی می‌توان از آن استفاده کرد.

[۳] `https://www.geeksforgeeks.org/garbage-collection-java/`

[۴] رابط برنامه نویسی

۱-۴ مفاهیم و اصطلاحات

پیش از شرح معماری پروژه بهتر است بعضی از اصطلاحات و مفاهیم توضیح داده شود تا ابهامات رفع شود و ابعاد پروژه روشن گردد.

۱-۴-۱ channel

channel[۵] یک **queue**[۶] است که همزمان چندین **agent**[۷] می‌توانند در آن داده **push** کنند و از طرف دیگر **agent** های دیگر می‌توانند داده‌ها را دریافت کنند. به دلیل اینکه چندین **agent** در حال کار با این صف هستند، این صف باید **thread-safe**[۸] باشد تا مشکل **race condition**[۹] پیش نیاید.

یک نکته مهم: عامل‌هایی که از صف داده‌ها را می‌خوانند می‌توانند به همه‌ی داده‌های قرار داده شده در کانال دسترسی داشته باشند.

۲-۴-۱ publisher

عامل‌هایی هستند که در صف (کانال) داده قرار می‌دهند.

برای مثال یک برنامه می‌تواند **log** های خروجی خود را داخل یک کانال قرار دهد. در اینجا **log** همان داده است که در کانال قرار می‌گیرد و برنامه هم همان عامل است

۳-۴-۱ subscriber

عامل‌هایی که داده‌ها را از صف (کانال) برداشته و مورد استفاده قرار می‌دهند. برای مثال یک برنامه ناظر که خروجی برنامه‌های دیگر را از کانال دریافت کرده و عمل نظارت را انجام می‌دهد. در صورتی که خطایی در یک برنامه رخ داده باشد یا میزان **load** در یک برنامه بالا رفته باشد، هشدار می‌دهد. در اینجا عامل همان برنامه‌ی ناظر است و داده‌های کانال خروجی برنامه‌ها هستند.

۴-۴-۱ real time messaging (rtm)

امروزه با پیشرفت تکنولوژی‌ها تعداد پیام‌رسان‌ها و شبکه‌های اجتماعی زیاد شده است. برای پیاده‌سازی این نرم‌افزارها معماری‌های زیادی وجود دارد. یکی از این معماری‌ها، معماری **publisher-subscriber**[۱۰] است.

در این معماری تعدادی از کاربرها (عامل‌ها) عضو یک کانال هستند پیام‌های آن کانال را دریافت می‌کنند و مورد استفاده قرار می‌دهند. به این کاربران **subscriber** گفته می‌شود. تعدادی از کاربران (عامل‌ها) هم در کانال داده (پیام) قرار می دهند که به آن‌ها **publisher** گفته می‌شود.

برای مثال در پیام‌رسان تلگرام کانال سایت varzesh3.com وجود دارد که تعداد زیادی کاربر دارد که هر روزه از اخبار جدید ورزشی بهره‌مند می‌شوند. این کاربران نقش **subscriber** را دارند. ادمین‌های این کانال نقش **publisher** را ایفا می‌کنند چون آن‌ها مطالب را در کانال قرار می‌دهند.

فرایند ذکر شده به این صورت است که بلافاصله بعد از اینکه یک ادمین مطلبی را در کانال قرار داد، این مطلب در اختیار کاربران قرار می‌گیرد. به همین دلیل این فرایند یک پیام‌رسانی برخط **(rtm)** است.

[۵] کانال
[۶] صف
[۷] عامل

[۸] https://www.baeldung.com/java-thread-safety
[۹] https://stackoverflow.com/questions/34510/what-is-a-race-condition
[۱۰] https://en.wikipedia.org/wiki/Publish-subscribe_pattern

satori rtm ۵-۴-۱

وبسایت www.satori.com این امکان را فراهم می‌کند که با استفاده از یک رابط کاربری خوب بدون درگیر شدن با جزئیات ایجاد کانال به صورت نرم‌افزاری، با چند کلیک یک کانال ایجاد کنیم (برای استفاده‌های غیرتجاری ایجاد کانال رایگان است). این کانال بر روی سرورهای این شرکت ایجاد شده و می‌توانیم با استفاده از رابط برنامه نویسی از آن استفاده کنیم.
در ابتدا باید در وبسایت ثبت نام کنیم. بعد از ساخت کانال یک **endpoint** و یک **appkey** در اختیار ما قرار می‌گیرد که برای وصل شدن به کانال به آن‌ها نیاز داریم. پیکربندی‌های مختلفی برای یک کانال وجود دارد که به دو مورد آن در این‌جا می‌پردازیم.

- یک کانال می‌تواند **public**[۱۱] باشد یعنی هر کسی با دانستن نام کانال و **appkey** و **endpoint** می‌تواند به داده‌های آن دسترسی پیدا کند.
- یک کانال می‌تواند **private**[۱۲] باشد که در این صورت برای دسترسی به داده‌های آن باید احراز هویت به وسیله نام کاربری و رمز عبور صورت گیرد.

برای کانال‌ها پیکربندی‌های زیادی وجود دارد. برای مثال می‌توان تعیین کرد که چه داده‌هایی وارد کانال شود و جلوی ورود چه نوع داده‌هایی گرفته شود.
اعمال زیادی نیز می‌توان بر روی داده‌های کانال‌ها انجام داد. برای مثال:

- مشاهده **history**[۱۳] کانال که شامل داده‌های پیشین کانال است.
- فیلترکردن پیام‌های دریافتی بر اساس پارامترها
- دریافت پیام‌ها در یک بازه زمانی خاص به صورت **stream**[۱۴]

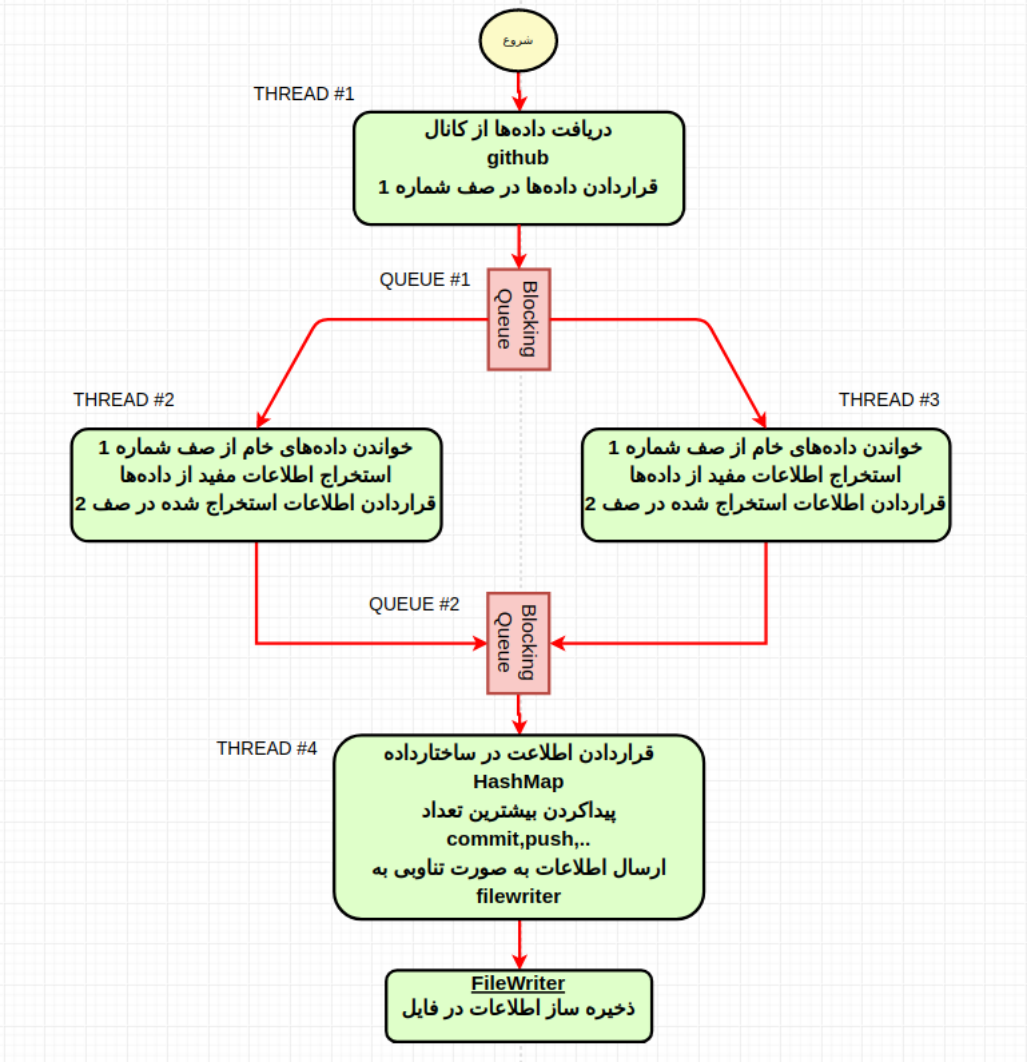
github channel ۵-۱

کانال **github** یک کانال عمومی است که رویدادهای اخیر سایت github.com در آن ثبت می‌شود. هر کاربری با داشتن **appkey** و **endpoint** آن می‌تواند به داده‌های این کانال دسترسی پیدا کند.
داده‌های این کانال شامل رویدادهای رخداده مانند ایجاد یک **commit**، عملیات **push** ، **tag** و ... است. از این کانال برای پیاده‌سازی پروژه استفاده می‌کنیم.

satori API ۶-۱

سایت satori.com یک رابط برنامه نویسی در اختیار ما قرار می‌دهد که با استفاده از کلاس‌ها و فراخوانی توابع آن و در دست داشتن نام کانال، **appkey** و **endpoint** می‌توانیم به یک کانال وصل شده و از آن داده دریافت کنیم. برای ارسال داده هم باید علاوه بر موارد ذکر شده احراز هویت نیز انجام دهیم. در این پروژه به کانال **github** متصل شده و از آن داده دریافت می‌کنیم و از رویدادها مطلع می‌شویم.

[۱۱] عمومی
[۱۲] خصوصی
[۱۳] تاریخچه
[۱۴] جریان



شکل ۱-۱: معماری نرم‌افزار

۱-۷ معماری نرم‌افزار

شمای کلی نرم‌افزار در شکل ۱-۱ رسم شده است.

به صورت صریح چهار نخ در برنامه حضور دارند و از دو صف استفاده شده است.

ابتدا توسط نخ شماره یک پیام‌ها از کانال **github** توسط رابط برنامه نویسی سایت **satori** دریافت شده و داخل یک **blocking queue**[۱۵] قرار داده می‌شود. حال سوالی که مطرح می‌شود این است که **blocking queue** چه فرقی با صف معمولی دارد؟

blocking queue صفی است که **thread-safe** است یعنی همزمان می‌توان در آن داده قرار داد و همزمان داده خواند. چون در این‌جا یک نخ در حال قرار دادن داده‌های کانال در صف و دو نخ در حال خواندن داده‌ها هستند از این ساختار داده استفاده کردیم. این ساختار داده در زبان جاوا پیاده‌سازی شده است.

از آن‌جا که سرور با سرعت زیادی داده در کانال قرار می‌دهد (تعداد کاربران **github** خیلی زیاد است به همین دلیل تعداد رخدادها زیاد است.)، با یک نخ داده‌ها را در صف یک قرار داده و با دو نخ داده‌ها را از آن صف می‌خوانیم و پردازش می کنیم تا صف اشباع نشود. این

[۱۵] https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html

تعداد نخ به صورت تجربی به دست آمده است.

پردازش به این شکل است که تنها داده‌هایی که شامل **commit** و **push** است را فیلتر کرده و سایر داده‌ها را دور می‌ریزیم. حال داده‌های به دست آمده را در صف شماره دو قرار می‌دهیم تا یک نخ دیگر عملیات نهایی را انجام دهد.

نخ شماره چهار داده‌های فیلتر شده را دریافت کرده و آن‌ها را در یک **HashMap**[۱۶] ذخیره می‌کند. در این ساختار داده، **Key** برابر نام کاربری و **value** شامل تعداد **commit** ها و **push** های شخص در یک ساعت اخیر است. هر یک دقیقه یک بار نام کاربری شخصی که بیشترین تعداد **commit** و **push** را دارد از **HashMap** استخراج شده و به ماژول **FileWriter** فرستاده می‌شود تا در یک فایل ذخیره شود.

در انتها این فایل خروجی برنامه است که هر یک دقیقه یک‌بار نام شخصی که بیشترین فعالیت را در یک ساعت اخیر در **github** داشته است در انتهای آن ثبت می‌شود.

۱-۸ نتیجه‌گیری

برنامه‌ی مورد نظر در یک هفته آماده شد و نحوه‌ی **build**[۱۷] و اجرای آن در ادامه آمده است.

۱-۸-۱ code

کد در آدرس زیر موجود است:

https://github.com/Su6lime/GitHubTrends

۱-۸-۲ ساخت فایل **jar** و اجرای آن

mvn clean install

java -jar GitHubTrends-1.0-SNAPSHOT-jar-with-dependencies.jar

java -jar GitHubTrendsGUI.jar

[۱۶] https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

[۱۷] فرایند ساختن فایل **jar** را **build** می‌گویند.

واژه‌نامه‌ی انگلیسی به فارسی

A

agent . . . . . عامل نرم‌افزاری یا انسانی

B

blocking queue . . . . . نوع خاصی از صف که همزمان چندین نفر می‌توانند در آن داده بگذارند و چندین نفر داده‌ها را دریافت کنند بدون این که مشکلی پیش آید.

build . . . . . فرایند ایجاد فایل **jar**

C

channel . . . . . کانال

G

garbage collector . . . . . موجودیتی در جاوا است که حافظه اشیایی که دسترسی به آن‌ها از بین رفته است را آزاد می‌کند.

H

history . . . . . تاریخچه

L

library . . . . . مجموعه‌ای از کلاس‌ها و توابع (کدها) که کاربرد خاصی دارند و در کنار هم جمع شده‌اند.

load . . . . . بارگیری

P

platform . . . سکو، در اینجا منظور سیستم عامل است.

private . . . . . خصوصی

public . . . . . عمومی

publisher . . . . . انتشاردهنده داده (پیام) در کانال

Q

queue . . . . . صف

R

rtm(real time messaging) . پیام رسانی لحظه ای

S

stream . . . . . جریانی از داده‌ها که با سرعت بالا ارسال می‌شوند.

subscriber . . . . . در اینجا به معنای مشتری یک کانال که به تمام پیام‌های ارسال شده به کانال دسترسی دارد