Задание 4. REST-сервис с асинхронной обработкой запросов

Цель: на языке высокого уровня (Java, C#, Python, Go и др. – на выбор обучающегося) реализовать REST веб-сервис, принимающий на обработку CSV-файлы и *реализующий их асинхронную (отложенную) обработку*. Отложенное выполнение задач необходимо для того, чтобы обеспечить быстрый отклик в веб-приложениях и должно применяться во всех потенциально длительных операциях (посылка писем, обработка файлов и т.д.)

Веб-сервис должен предоставлять веб-API, обеспечивающий загрузку в сервис таблицы в виде CSV-файла для ее обработки и получения результата в виде N элементов из данного файла с максимальным значением значения данных по определенному полю данного CSV-файла (TOP-N). Первая строка CSV-файла содержит заголовки соответствующих столбцов таблицы. В качестве примера можно использовать данные со страницы https://support.spatialkey.com/spatialkey-sample-csv-data/ (например, http://samplecsvs.s3.amazonaws.com/Sacramentorealestatetransactions.csv).

1.1 Критерии оценивания

№	Задача	**Баллы**
1.	Реализовать и разместить локально простейший REST-сервис «Эхо» по адресу `example.edu/echo` принимающий строку текста по PUT-запросу и возвращающий эту же строку текста по GET-запросу	5
2.	Реализовать и разместить локально веб-сервис, предоставляющий по адресу `example.edu/top` веб-API для загрузки файлов с параметрами: – `field` - поле, по которому нужно сортировать записи – `count` - количество верхних записей соответственно. В ответ на запрос должна возвращаться ссылка на результирующий файл, в который будут добавлены результаты обработки.	5

3.	Реализовать метод `example.edu/upload`, обеспечивающий загрузку, асинхронную обработку записей из CSV-файла и добавление их в БД. По запросу `GET example.edu /top?field=FIELD&count=N` должен возвращаться JSON-ответ, содержащий верхние N записей по полю FIELD.	5
4.	Разместить разработанный REST-сервис в облаке и продемонстрировать его работу.	5
5*	**Обеспечить распределенную обработку CSV-файла из задания 2. Файл должен разбиваться на части, и отдаваться на независимую обработку набору исполнителей. Каждый из исполнителей осуществляет независимый разбор файла и находит N верхних записей для своего сегмента, затем результаты обработки объединяются и берутся верхние N записей от результата (реализовать концепцию "MAP-REDUCE" в ручном режиме).**	**8**

1.2 Методические указания

Для реализации данного задания необходимо воспользоваться очередью, обеспечивающей асинхронный обмен сообщениями между компонентами системы. Рассмотрим ключевые подходы к реализации очередей сообщений на платформах Redis, RabbitMQ и Apache Kafka.

1.3 Примеры реализации

1.3.1 Использование системы Redis на базе Java

Как одно из возможных решений, можно воспользоваться системой Redis: http://redis.io. Для выполнения лабораторной работы можно как самостоятельно развернуть данную систему, так и воспользоваться готовым облачным решением, например http://redistogo.com/.

Для реализации веб-сервиса, легче всего воспользоваться одним из готовых фреймворков для вашего языка программирования. Для языков Python, Ruby и Go можно рекомендовать такие платформы как Flask, Sinatra и Martini соответственно (https://realpython.com/blog/python/python-ruby-and-golang-a-web-Service-application-comparison/).

Рассмотрим пример реализации простейшего веб-сервиса, реализующего асинхронный процесс обработки загруженных файлов, на базе фреймворка Spark для языка Java.

1. Для реализации асинхронного обмена и отложенного задания необходимо завести аккаунт на http://redistogo.com/ (внизу есть маленькая кнопочка free plan), после чего вам выдадут connection string вида redis://redistogo:34534987987@angelfish.redistogo.com:10649/
2. Создадим новый Java-проект, который будет состоять из двух независимых компонентов: веб-сервиса и обработчика.
3. Для работы с проектом, нам потребуется внедрить в проект системы для работы с платформой Redis (jedis) и веб-сервис Spark. Это можно реализовать посредством платформы Maven, указав соответствующие зависимости в конфигурационном файле pom.xml. Рассмотрим блок dependencies которые необходимо добавить в pom.xml (полный код pom.xml доступен вот тут: https://gist.github.com/skayred/0c0a9dc57ad8f9fa9744eea49f1fb50a):

```
<dependencies>
    <dependency>
      <groupId>redis.clients</groupId>
      <artifactId>jedis</artifactId>
<version>2.8.1</version>
    </dependency>
```

```
    <dependency>
      <groupId>com.sparkjava</groupId>
      <artifactId>spark-core</artifactId>
      <version>2.3</version>
    </dependency>
</dependencies>
```

Добавление данных строк в pom.xml и пересборка Maven-проекта загрузит необходимые для реализации проекта зависимости.

4. Сервер обеспечивает обработку POST запросов по адресу “\upload”, принимающих один параметр “file”. Полученный в запросе файл сохраняется во временном хранилище, после чего его имя передается в очередь для дальнейшей обработки.

Рассмотрим исходный код веб-сервиса:

```
package ru.susu;
```

```
import redis.clients.jedis.Jedis;
import static spark.Spark.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.nio.*;
import java.nio.file.*;
import java.util.*;


public class AppServer {
    public static void main( String[] args ) {
        // Соединяемся с Редисом
        Jedis jedis = new Jedis("АДРЕС_РЕДИСА");


        // Тут мы используем фремворк Spark, следующий вызов вешает
слушателя на метод POST
        post("/upload", "multipart/form-data", (request, response) ->
{
                // Получаем файл из ПОСТа и сохраняем на диск
                String location = "/oldhome/sk_/projects/tmp/";
// the directory location where files will be stored
                long maxFileSize = 100000000;       // the maximum
size allowed for uploaded files
                long maxRequestSize = 100000000;    // the maximum
size allowed for multipart/form-data requests
                int fileSizeThreshold = 1024;       // the size
threshold after which files will be written to disk


                MultipartConfigElement multipartConfigElement = new
MultipartConfigElement(

location, maxFileSize, maxRequestSize, fileSizeThreshold);

request.raw().setAttribute("org.eclipse.jetty.multipartConfig",
                                           multipartConfigElement);


                String fName =
request.raw().getPart("file").getSubmittedFileName();


                Part uploadedFile = request.raw().getPart("file");
                Path out = Paths.get(location + fName);
                try (final InputStream in =
uploadedFile.getInputStream()) {
                    Files.copy(in, out);
                    uploadedFile.delete();
                }
                // cleanup
                multipartConfigElement = null;
                uploadedFile = null;
```

```
                // После сохранения файла отправляем путь к нему
обработчику
                jedis.rpush("queue", out.toString());


                return "OK";
            });
    }
}
```

5. Реализуем исполнителя

Исполнитель получает адрес файла, который необходимо обработать, после чего выводит содержимое данного файла.

```
package ru.susu;

    import
redis.clients.jedis.Jedis;  import
java.util.*;  import java.io.*;
 import java.nio.file.*;


    public class App
{
     public static void main( String[] args ) {
         // Соединяемся с Редисом
         Jedis jedis = new Jedis("АДРЕС_РЕДИСА");
         List<String> messages = null;
// Ждем сообщения из очереди
         while(true){
           System.out.println("Waiting for a message in the queue");
           messages = jedis.blpop(0,"queue");
           String payload = messages.get(1);
           // Выполняем задачу
processFile(payload);
         }
     }

     // Выводим содержимое файла в консоль
     private static void processFile(String filename) {
try {
             System.out.println(readFile(filename));
         } catch (IOException e) {
         }
     }
        static String readFile(String path) throws IOException {
byte[] encoded = Files.readAllBytes(Paths.get(path));
return new String(encoded);
     }
}
```

1.3.2 Использование системы RabbitMQ на базе Go

RabbitMQ – это брокер сообщений. Его основная цель – принимать и отдавать сообщения. RabbitMQ позволяет взаимодействовать различным программам при помощи протокола AMQP. RabbitMQ является отличным решением для построения SOA (сервис-ориентированной архитектуры) и распределением отложенных ресурсоемких задач. RabbitMQ написан на языке Erlang и базируеться на базе СУБД Mnesia которая также написана на Erlang. Mnesia – это распределённая СУБД реального времени, по своей сути используется для встраиваемых решений и этим похожа на Berkeley DB

Для использования RabbitMQ в программах на языке Go необходимо подключить библиотеку (пакет), использующую протокол AMQP. AMQP — это протокол обмена сообщениями на уровне соединения, который можно использовать для создания кроссплатформенных приложений для обмена сообщениями. Цель этого протокола проста — определить механизм безопасной, надежной и эффективной передачи сообщений между двумя сторонами. Для написания программы в ходе работы будет использован пакет amqp для языка Go.

Рассмотрим реализацию простейшего приложения, использующего очередь RabbitMQ для обмена данными.

1. Для разработки и компиляции программы на языке Go необходимо установить стандартный компилятор языка – gc, а также редактор исходного кода, например Visual Studio Code.
2. Чтобы загрузить и установить Go необходимо скачать компилятор с официального сайта http://golang.org/doc/install.html
3. Для установки Visual Studio Code необходимо скачать установщик с официального сайта https://www.visualstudio.com/ru-ru/products/code-vs.aspx Затем нужно запустить исполняемый файл VSCodeSetup-stable.exe. После этого в меню View-Command Palette в VS Code необходимо найти и скачать расширение для языка Go. Для этого во всплывающей строке нужно ввести ext inst и выбрать пункт Extensions: Install Extension. Из полученного списка нужно выбрать расширение для Go. Далее надо установить пакеты и отладчик. Для установки пакетов нужно открыть любой go-файл и нажать на сообщение "Analysis Tools Missing" в правом нижнем углу. Пакеты установятся автоматически. Для установки отладчика необходимо прописать в командную строку Windows команду

```
go get -u github.com/derekparker/delve/cmd/dlv
```

После этого отладчик установится автоматически.

4. Далее следует установить сервер RabbitMQ. Для этого нужно:

1. Скачать и установить сервер RabbitMQ с официального сайта https://www.rabbitmq.com/install-windows.html. На данной странице для просмотра также доступна инструкция по установке данного сервера.
2. Скачать и установить все пакеты Erlang с официального сайта http://www.erlang.org/download.html
3. После завершения обеих установок необходимо перезагрузить компьютер.

5. После выполнения перезагрузки компьютера необходимо запустить командную строку RabbitMQ. В командной строке RabbitMQ необходимо ввести команду

```
rabbitmq-server -detached
```

Данная команда осуществит запуск сервера. В том случае, если сервер уже запущен, в командной строке будет отображена соответствующая запись. Далее надо ввести команду

```
rabbitmq-plugins enable rabbitmq_management
```

Данная команда осуществит включение плагинов. Далее необходимо перейти по ссылке: http://localhost:15672/ и авторизоваться на открывшейся странице (login: guest, password: guest).

6. Также потребуется установить пакет amqp для языка Go. Для этого в командной строке Windows нужно ввести команду

```
go get github.com/streadway/amqp
```

После установки необходимого ПО можно приступать к разработке и реализации программы.

7. Для демонстрации принципа работы RabbitMQ потребуется написать 2 программы: одна будет отправлять сообщения на сервер, а вторая – загружать их оттуда и выводить в консоль. Первая программа будет называться send.go. В ней нужно подключить необходимые для написания программы пакеты

```go
package main

import (
    "fmt"
    "log"
    "os"

    "github.com/streadway/amqp"
)
```

8. Также необходимо написать вспомогательную функцию для проверки возвращаемого значения

```go
func failOnError(err error, msg string) {
    if err != nil {
        log.Fatalf("%s: %s", msg, err)
        panic(fmt.Sprintf("%s: %s", msg, err))
```

```
    }
}
```

9. Затем подключиться к брокеру сообщений, находящемуся на локальном сервере. Для подключения к брокеру, находящемуся на другой машине, необходимо заменить localhost на IP-адрес этой машины.

```
    conn, err := amqp.Dial("amqp://guest:guest@localhost:5672/%2f")
    failOnError(err, "Failed to connect to RabbitMQ")
    defer conn.Close()
```

10. Далее нужно создать канал для передачи сообщений на сервер

```
    ch, err := conn.Channel()
    failOnError(err, "Failed to open a channel")
    defer ch.Close()
```

11. Для отправки сообщения необходимо создать очередь, в которую будет отправляться данное сообщение

```
    q, err := ch.QueueDeclare(
        "hello", // name
        false,   // durable
        false,   // delete when unused
        false,   // exclusive
        false,   // no-wait
        nil,     // arguments
    )
    failOnError(err, "Failed to declare a queue")
```

12. Далее следует код, отвечающий за отправку сообщения

```
    if len(os.Args) > 1 {
        for tmp := 1; tmp < len(os.Args); tmp++ {
            body += os.Args[tmp] + " "
        }
    } else {
        body = "message"
    }

    err = ch.Publish(
        "",     // exchange
        q.Name, // routing key
        false,  // mandatory
        false,  // immediate
        amqp.Publishing{
            ContentType: "text/plain",
```

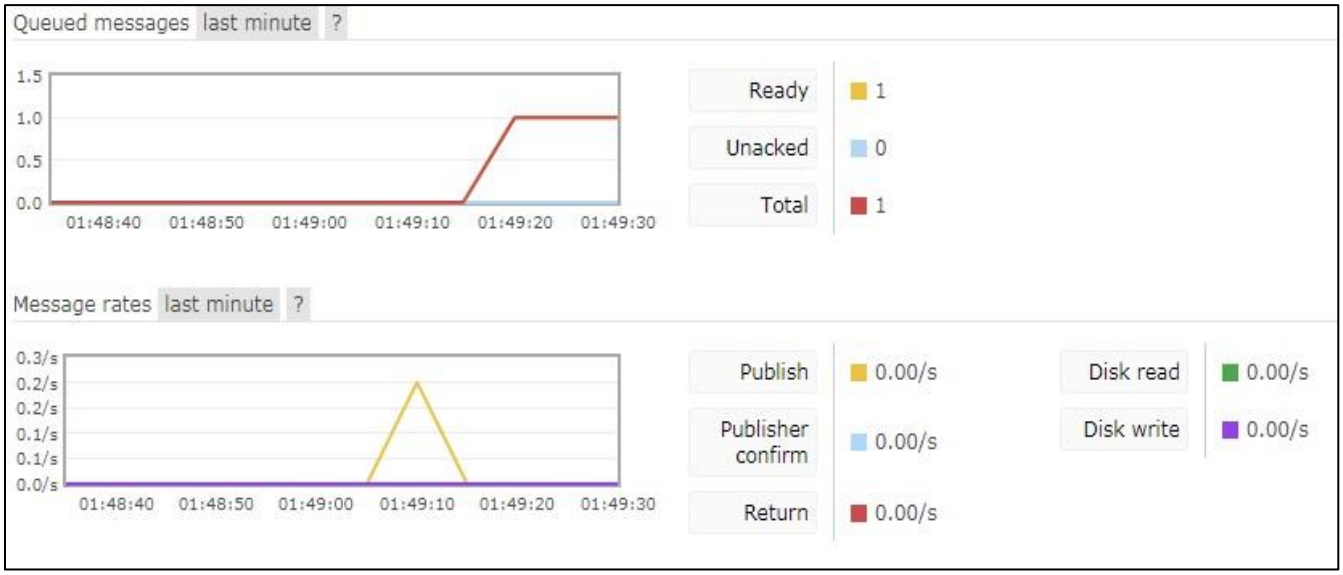
```
            Body:        []byte(body),
        })
    failOnError(err, "Failed to publish a message")
```

13. Теперь необходимо запустить данную программу. Для этого в командной строке Windows нужно перейти в каталог, в котором размещен файл send.go. Затем необходимо написать команду

```
go build
```

которая создаст в данном каталоге исполняемый файл send.exe. Далее в командной строке можно ввести send.exe и сообщение, которое нужно отправить в очередь. Если не указано сообщение, то программа отправит на сервер сообщение "message".

14. После отправки сообщения можно проверить, дошло ли оно, и сколько сообщений находится в очереди в данный момент. Для этого необходимо перейти по ссылке http://localhost:15672/.



15. Далее нужно реализовать программу для получения сообщений с сервера. Исходный файл будет называться receive.go. Нам также потребуется подключить необходимые библиотеки, написать вспомогательную функцию, подключиться к брокеру, создать канал, а также создать очередь (на случай, если получатель будет запущен раньше отправителя). После этого можно выгружать сообщения с сервера из очереди. Так как сервер будет поставлять нам сообщения асинхронно, то мы будем считывать сообщения в горутину.

```
    msgs, err := ch.Consume(
        q.Name, // queue
        "",     // consumer
```

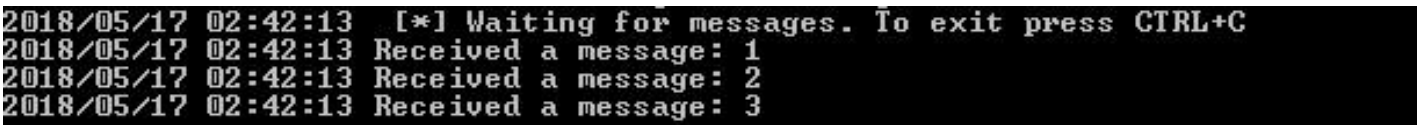
```
        true,   // auto-ack
        false,  // exclusive
        false,  // no-local
        false,  // no-wait
        nil,    // args
    )
    failOnError(err, "Failed to register a consumer")

    forever := make(chan bool)

    go func() {
        for d := range msgs {
            log.Printf("Received a message: %s", d.Body)
        }
    }()

    log.Printf(" [*] Waiting for messages. To exit press CTRL+C")
    <-forever
```

16. Чтобы запустить получателя, нужно так же в командной строке Windows создать и запустить файл receive.exe. После этого в командной строке будут выведены сообщения, которые в данный момент находились в очереди, а получатель продолжит ждать новые сообщения.

17. Для проверки корректности работы попробуем отправить на сервер 3 сообщения: "1", "2", "3".

```
C:\Users\EGOR\Documents\VS Code Projects\Messenger\Send>send.exe 1
C:\Users\EGOR\Documents\VS Code Projects\Messenger\Send>send.exe 2
C:\Users\EGOR\Documents\VS Code Projects\Messenger\Send>send.exe 3
```

18. Далее запускаем приложение-получатель, который должен вывести в консоль соответствующие сообщения.



Получатель вывел сообщения в консоль в нужном порядке, после чего продолжает ждать новые сообщения.

1.4 Задание на самостоятельную работу

На основе представленных примеров вам предлагается самостоятельно реализовать веб-сервис, обеспечивающий решение задач 1-4.

1.5 Ссылки

Для дальнейшего самостоятельного изучения данной темы можно воспользоваться следующими ресурсами:

1. Руководство для начинающих в платформе Amazon Web Services доступны по ссылке: https://aws.amazon.com/ru/getting-started/?nc1=h_ls.
2. Использование асинхронного обмена сообщениями для улучшения доступности https://habr.com/ru/company/piter/blog/458344/
3. Asynchronous Task Execution Using Redis and Spring Boot https://dzone.com/articles/asynchronous-task-executor-using-redis-and-spring
4. Asynchronous Microservices with RabbitMQ and Node.js https://www.manifold.co/blog/asynchronous-microservices-with-rabbitmq-and-node-js