微服务设计模式： 后台作业模式实现

==========

后台作业模式 用于无需用户等待或需要长时间完成的执行任务的场景，比如用户点击取消订单按钮会立即返回，但是退款和订单变为已取消状态会等待一会才能完成。对该模式更多了解可参考

* [后台作业 - Azure文档](http://cxyroad.com/”https://learn.microsoft.com/zh-cn/azure/architecture/best-practices/background-jobs”)

设计

--

现在模拟一个抽象的后台作业流程，用户提交异步任务后，需要等待任务执行完成，任务执行完成后可以查看任务执行结果。技术流程图如下：

```
sequenceDiagram
    autonumber
    participant user as 用户
    participant asyncWorkService as Job 服务
    participant workerService as Job 执行服务
    participant db as Job 数据库
    user ->> asyncWorkService : 创建 new Job
    asyncWorkService ->> db : 保存 jobDetail
    asyncWorkService ->> Kafka : 发送 newJobEvent (包含jobId、jobName等)
    par 异步任务执行
        workerService -->> Kafka : 接收 newJobEvent
        activate workerService
        workerService ->> db : 查询 jobDetail
        workerService ->> workerService : 执行任务
        workerService ->> db : 执行结束，更新 job 状态 和 job 执行结果
        deactivate workerService
    end
    user ->> asyncWorkService : 查询 job 当前状态/详情
    asyncWorkService ->> db : 查询 job 信息
    db -->> asyncWorkService : 返回 job 信息
    asyncWorkService -->> user : 返回 job 当前状态/详情
```

```
    user ->> asyncWorkService : 查询 job 执行结果
    asyncWorkService ->> db : 查询 job 执行结果
    db -->> asyncWorkService : 返回 job 执行结果
    asyncWorkService -->> user : 返回 job 执行结果
```

```

* **Job 服务**：提供了创建任务、查询任务当前状态、查询任务结果的接口
* **Job 执行服务**：实际执行异步任务的服务，可在不同的机器甚至使用不同的语言实现
* **Job 数据库**：包含了 Job 元信息和 Job 执行结果信息
* 异步任务服务 和 异步任务执行服务 都可以访问 Job 数据库，新的任务执行通过 kafka 事件驱动
* 步骤 5 和 12 执行的时候 步骤6 可能还未完成


实现
--


* 用户通信协议：RESTful API
* 数据库：便于本地开发的 [jsondb](http://cxyroad.com/”https://jsondb.io/”)
* 消息队列：kafka on Docker
* 开发框架：Spring Boot、Vert.X
* JDK Version：JDK 21


### Job 模型和数据库


**AsyncJob** 包含了 Job 的基本信息



```
@Data
@Document(collection = ”asyncJobs”, schemaVersion= ”1.0”)
public class AsyncJob {

    @Id
    private String id;

    private String name;

    private String value;

    private State state;
```
```

```
    // 执行用时，单位 ms
    private int executionTime;

    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss.SSS")
    private LocalDateTime createdTime;

    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss.SSS")
    private LocalDateTime updatedTime;

    /**
     * Job 状态
     */
    public enum State {
        NEW, // 新的
        RUNNING, // 执行中
        PAUSE,  // 已暂停
        FINISH, // 已完成
        CANNEL, // 已取消
        FAILURE // 失败
    }
}
```

数据存储基于本地 JSON 文件，所以数据访问层也很简单:

```
public class AsyncJobStore {

    public static final String DBFILES_LOCATION =
Objects.requireNonNullElse(System.getenv("JSONDBFILES_LOCATION"),
"c:\\jsondb");

    private static final JsonDBTemplate jsonDBTemplate = new
JsonDBTemplate(DBFILES_LOCATION,
"com.onemsg.commonservice.store");

    static {
        if (!jsonDBTemplate.collectionExists(AsyncJob.class)) {
            jsonDBTemplate.createCollection(AsyncJob.class);
        }
        if (!jsonDBTemplate.collectionExists(AsyncJobResult.class)) {
            jsonDBTemplate.createCollection(AsyncJobResult.class);
        }
    }
```

```
    public static JsonDBTemplate jsondb() {
        return jsonDBTemplate;
    }
}
```

AsyncJobResult 包含了任务执行成功生成的结果信息

```
@Data
@Document(collection = "asyncJobResults", schemaVersion= "1.0")
public class AsyncJobResult {

    @Id
    private String jobId;

    private String result;

    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss.SSS")
    private LocalDateTime finishedTime;
}
```

`AsyncWorkEvent` 是通过 Kafka 传递的事件

```
public record AsyncWorkEvent(
    String jobId,
    String name,
    String value) {
}
```

Job 服务

Job 服务提供关于异步任务管理的接口：

API	HTTP Method	参数	说明
`/api/asyncwork`	POST	jobName、jobValue	创建新的 Job，返回202（ACCEPTED）和新 Job 的 jobId 和状态查询url
`/api/asyncwork/state`	GET	jobId	查询指定 jobId 的状态，不存在则返回404；job执行成功顺带返回执行结果url
`/api/asyncwork`	GET	jobId	查询指定 jobId 的信息，不存在则返回404
`/api/asyncwork/result`	GET	jobId	查询指定 jobId 的执行结果，不存在则返回404

Spring MVC 实现代码如下：

```
@Slf4j
@RestController
@RequestMapping("/api/asyncwork")
public class AsyncWorkController {

    @Autowired
    private KafkaTemplate<String,String> kafkaTemplate;

    @Autowired
    private ObjectMapper objectMapper;

    private static final String TOPIC = Topics.ASYNC_JOB_TOPIC;

    public record AsyncWorkRequest(String name, String value) {}

    /**
     * 创建新 Job
     * @param work
     * @return
     */
    @PostMapping("")
    public ResponseEntity<Object> postAsyncWork(@RequestBody
AsyncWorkRequest work) {

        // 生成任务
        String jobId = UUID.randomUUID().toString();

        var jsonDBTemplate = AsyncJobStore.jsondb();
        AsyncJob job = new AsyncJob();
        job.setId(jobId);
```

```
        job.setName(work.name());
        job.setValue(work.value());
        job.setState(AsyncJob.State.NEW);
        job.setCreatedTime(LocalDateTime.now());
        job.setCreatedTime(job.getCreatedTime());
        jsonDBTemplate.insert(job);

        log.info("Job已创建 {}", jobId);

        // 发送任务处理事件到消息队列
        AsyncWorkEvent event = new AsyncWorkEvent(jobId, work.name(),
work.value());

        sendEvent(event);

        // 返回带状态检查路径响应
        String statucEndpoint = "/api/asyncwork/state?jobId=" + jobId;
        var data = Map.of("jobId", jobId, "stateEndpoint", statucEndpoint,
"retryAfter", 1000);
        return ResponseEntity.status(HttpStatus.ACCEPTED).body(data);
    }

    /**
     * 查询 Job 状态
     * @param jobId
     * @return
     */
    @GetMapping("/state")
    public ResponseEntity<Object> getState(@RequestParam String jobId)
{

        var jsonDBTemplate = AsyncJobStore.jsondb();
        jsonDBTemplate.reLoadDB();
        var job = jsonDBTemplate.findById(jobId, AsyncJob.class);
        if (job == null) {
            return ResponseEntity.notFound().build();
        }

        if (job.getState() == AsyncJob.State.FINISH) {
            var resultEndpoint = "/api/asyncwork/result?jobId=" + jobId;
            var data = Map.of("jobId", jobId, "state", job.getState(),
"resultEndpoint", resultEndpoint);
            return ResponseEntity.ok(data);
        }

        var data = Map.of("jobId", job.getId(), "state", job.getState());
        return ResponseEntity.ok(data);
    }
```

```
    /**
     * 查询 Job 详情
     * @param jobId
     * @return
     */
    @GetMapping()
    public ResponseEntity<Object> getDetail(@RequestParam String
jobId) {
        var jsonDBTemplate = AsyncJobStore.jsondb();
        jsonDBTemplate.reLoadDB();
        var job = jsonDBTemplate.findById(jobId, AsyncJob.class);

        if (job == null) {
            return ResponseEntity.notFound().build();
        }

        return ResponseEntity.status(200).body(job);
    }

    /**
     * 查询 Job 执行结果
     * @param jobId
     * @return
     */
    @GetMapping("/result")
    public ResponseEntity<Object> getResult(@RequestParam String
jobId) {

        var jsonDBTemplate = AsyncJobStore.jsondb();
        jsonDBTemplate.reLoadDB();
        var result = jsonDBTemplate.findById(jobId, AsyncJobResult.class);
        if (result == null) {
            return ResponseEntity.notFound().build();
        }

        return ResponseEntity.ok(result);
    }


    private String toJsonString(AsyncWorkEvent event) throws
ResponseStatusException {
        try {
            return objectMapper.writeValueAsString(event);
        } catch (JsonProcessingException e) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "请求数据无效", e);
        }
```

```
    }

    private void sendEvent(AsyncWorkEvent event) throws
ResponseStatusException  {
        var value = toJsonString(event);
        var future = kafkaTemplate.send(TOPIC, value);
        try {
            var result = future.get(5, TimeUnit.SECONDS);
            log.info("Kafka 事件已发送 {} {} {}",
result.getRecordMetadata().topic(),
                result.getRecordMetadata().offset(), event.jobId());
        } catch (ExecutionException e) {
            log.warn("发送 Kafka 失败 {} {} {}", TOPIC, value, e.getCause());
            throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "服务
器内部处理错误");
        } catch (Exception e) {
            log.warn("发送 Kafka 失败 {} {} {}", TOPIC, value, e);
            throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "服务
器内部处理错误");
        }
    }

}
```

异步任务执行服务

使用 Vert.X 编写，接收 kafka 消息并开始执行 Job，模拟了一个 job 的执行

```
@Slf4j
public class AsyncJobWorkerVerticle extends AbstractVerticle {

    @Override
    public void start(Promise<Void> startPromise) throws Exception {

        Map<String, String> config = new HashMap<>();

        String host =
Objects.requireNonNullElse(System.getenv("WSL_HOST"), "localhost");
        config.put("bootstrap.servers", host + ":9092");
```

```
        config.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
        config.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
        config.put("group.id", "vertx-service-group");
        // latest earliest
        config.put("auto.offset.reset", "latest");
        config.put("enable.auto.commit", "false");

        KafkaConsumer<String, String> consumer =
KafkaConsumer.create(vertx, config);

        // 接受 kafka 事件并处理
        consumer.handler(kafkaRecord -> {
            handleEvent(kafkaRecord.value());
            consumer.commit();
        });

        consumer.subscribe(Topics.ASYNC_JOB_TOPIC)
            .onSuccess(startPromise::complete)
            .onFailure(startPromise::fail);
    }


    private void handleEvent(String event) {
        AsyncWorkEvent asyncWorkEvent;
        try {
            asyncWorkEvent = Json.decodeValue(event,
AsyncWorkEvent.class);
        } catch (DecodeException  e) {
            log.warn("Kafka 反序列化失败 {} {}", event, e);
            return;
        }

        // 在工作线程池中执行任务
        vertx.executeBlocking(() -> {
            runJob(asyncWorkEvent.jobId());
            return null;
        });
    }

    private void runJob(String jobId) {

        // 模拟任务执行

        var jsonDBTemplate = AsyncJobStore.jsondb();
        jsonDBTemplate.reLoadDB();
        var job = jsonDBTemplate.findById(jobId, AsyncJob.class);
```

```
        if (job == null) {
            log.warn("不存在Job {}", jobId);
            return;
        }

        if (job.getState() != AsyncJob.State.NEW && job.getState() !=
AsyncJob.State.PAUSE) {
            log.warn("无法开始执行Job {}, 状态:{}", job.getId(), job.getState()
);
            return;
        }

        log.info("开始执行Job {}", jobId);

        // 更新 job state -> RUNNING
        long startTime = System.currentTimeMillis();
        job.setState(AsyncJob.State.RUNNING);
        job.setUpdatedTime(LocalDateTime.now());
        jsonDBTemplate.upsert(job);

        sleep(ThreadLocalRandom.current().nextLong(3000, 10000));
        // 1/5 概率失败
        boolean failed = ThreadLocalRandom.current().nextInt(5) == 1;
        if (!failed) {
            // 存储执行结果
            String content =
Arrays.toString(ThreadLocalRandom.current().ints(10).toArray());
            AsyncJobResult result = new AsyncJobResult();
            result.setJobId(jobId);
            result.setResult(content);
            result.setFinishedTime(LocalDateTime.now());
            jsonDBTemplate.insert(result);
        }
        long endTime = System.currentTimeMillis();

        // 更新 job state -> FAILURE | FINISH
        job.setState(failed ? AsyncJob.State.FAILURE :
AsyncJob.State.FINISH);
        job.setUpdatedTime(LocalDateTime.now());
        job.setExecutionTime((int) (endTime - startTime));

        jsonDBTemplate.upsert(job);

        log.info("执行结束Job {}, {}, {} ms", jobId, job.getState(),
job.getExecutionTime());
    }
```

```
    private static void sleep(long millis) {
        try {
            Thread.sleep(millis);
        } catch (Exception e) { }
    }


    public static void main(String[] args) {

        Vertx vertx = Vertx.vertx();

        vertx.deployVerticle(AsyncJobWorkerVerticle.class, new
DeploymentOptions())
            .onComplete(ar -> {
                if (ar.succeeded()) {
                    log.info("JobWorker 部署成功");
                } else {
                    log.error("JobWorker 部署失败", ar.cause());
                }
            });

    }
```

```
测试
--


这里写了一个 python 脚本来模拟用户行为，执行步骤


1. 创建新 Job
2. 循环检查 Job 状态，直到 Job 变为终止状态
3. 如果 Job 执行成功，查询执行结果



```
```
"""
测试 后台作业模式
"""

import requests
import random
import json
import time
```

```
JOB_SERVICE = "http://127.0.0.1:7701"

create_job_api = JOB_SERVICE + "/api/asyncwork"
job_detail_api = JOB_SERVICE + "/api/asyncwork"

HEADERS = {
    "Content-Type": "application/json"
}

def print_response(title, res):
    print(title + ":", res.status_code, res.json())

# 创建任务
def create_job():
    n = random.randint(1, 1000)
    request_data = {
        "name": "name-" + str(n),
        "value": "value-" + str(n)
    }
    res = requests.post(create_job_api, json.dumps(request_data),
headers=HEADERS)
    data = res.json()
    if res.status_code == 202:
        print("Create job:", data["jobId"])
        return data
    else:
        print("Create job failed:", res.status_code, data)
        raise Exception("Create job failed")

# 循环检查任务状态
def check_job_state(state_endpoint):
    url = JOB_SERVICE + state_endpoint
    i = 0
    while i < 20:
        i += 1
        res = requests.get(url)
        if res.ok:
            data = res.json()
            print("Job state:", data["state"])
            if data["state"] in ["RUNNING", "NEW"]:
                time.sleep(1.0)
            else:
                return data
        else:
            print("Job state failed:", res.status_code, res.json())
            raise Exception("Check job state failed")
    raise Exception("Retry timeout")
```

```
# 检查任务执行结果
def check_job_result(result_endpoint):
    url = JOB_SERVICE + result_endpoint
    res = requests.get(url)
    data = res.json()

    if res.ok:
        print("Job result:", data["result"])
    else:
        print("Job result failed:", res.status_code, data)
        raise Exception("Check job result failed")

if __name__ == "__main__":

    job = create_job()
    job = check_job_state(job["stateEndpoint"])

    if "resultEndpoint" in job:
        check_job_result(job["resultEndpoint"])


```

终端输出：

```
Create job: 67d3d8e3-2546-4fde-a4a8-d35495d94e5c
Job state: NEW
Job state: RUNNING
Job state: RUNNING
Job state: RUNNING
Job state: RUNNING
Job state: RUNNING
Job state: RUNNING
Job state: FINISH
Job result: [1976214741, 1266274731, 1287610214, 1703472092, -1315441378, -986112033, -1842836671, 225273176, 554521009, 1450321281]

```

查看 job 信息

```

```

```
GET /api/asyncwork?jobId=67d3d8e3-2546-4fde-a4a8-d35495d94e5c

HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Fri, 21 Jun 2024 16:37:29 GMT
Connection: close

{
  "id": "67d3d8e3-2546-4fde-a4a8-d35495d94e5c",
  "name": "name-150",
  "value": "value-150",
  "state": "FINISH",
  "executionTime": 6708,
  "createdTime": "2024-06-22 00:35:06.550",
  "updatedTime": "2024-06-22 00:35:13.731"
}

```

查看 job 状态

```
GET /api/asyncwork/state?jobId=67d3d8e3-2546-4fde-a4a8-
d35495d94e5c

HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Fri, 21 Jun 2024 16:39:34 GMT
Connection: close

{
  "jobId": "67d3d8e3-2546-4fde-a4a8-d35495d94e5c",
  "resultEndpoint": "/api/asyncwork/result?jobId=67d3d8e3-2546-
4fde-a4a8-d35495d94e5c",
  "state": "FINISH"
}

```

查看 job 结果

```
```

```
GET /api/asyncwork/result?jobId=67d3d8e3-2546-4fde-a4a8-d35495d94e5c

HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Fri, 21 Jun 2024 16:39:34 GMT
Connection: close

{
  "jobId": "67d3d8e3-2546-4fde-a4a8-d35495d94e5c",
  "resultEndpoint": "/api/asyncwork/result?jobId=67d3d8e3-2546-4fde-a4a8-d35495d94e5c",
  "state": "FINISH"
}

```