Flink 分布式JVM微服务

一、所以说，如果我们将 Flink 集群整体视为一个冯诺伊曼结构计算机的话:
计算节点就对应着 CPU
计算节点上的状态后端就对应着内存
计算完成后,经过输入、输出保存到kafka或者redis、mysql,对应于物理机器的磁盘 I/O 设备

在 Flink 系统架构中，它明确地将状态管理纳入到了它的系统架构中。在各个 Flink 节点进行计算时，它将状态保存到本地，并通过 checkpoint 机制和诸如 HDFS 这样的分布式文件系统，实现了状态的分布式管理。

二、计算节点是可以水平扩展的，计算节点上的状态后端（比如内存、文件和 RocksDB）也是实现了分布式存储和管理的。
Flink 就成了一个CPU 和内存都可以近乎无限扩展的冯诺依曼机器，。

三、原理

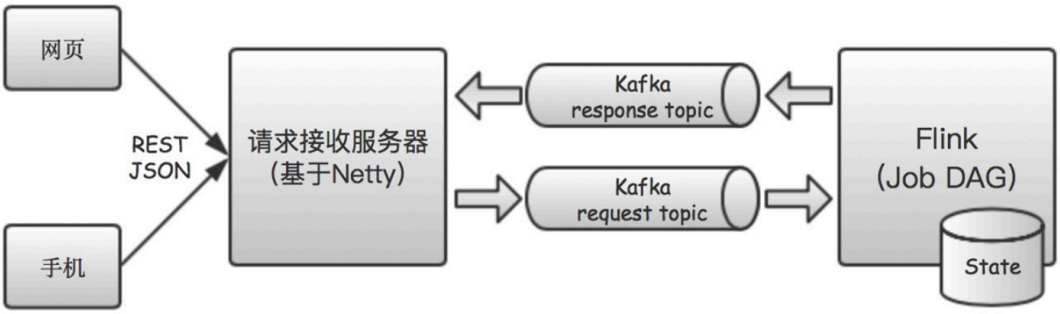

图 3 使用Flink实现微服务系统

我们的微服务系统整体上采用了事件驱动（也就是异步执行）的方式。
首先，REST 请求接收服务器采用基于 Netty 的 NIO 和异步编程框架，将接收到的 REST 请求发送到 Kafka。
然后，Flink 从 Kafka 中将请求读取出来进行处理。
之后，Flink 再将请求处理结果发送回 Kafka。
最后，请求接收服务器从 Kafka 中取出请求处理结果，并将请求处理结果返回给客户端。

四、基于 Flink 的微服务方案，有以下四点优势。
1.整个系统完全是异步的，可以极大提升系统的整体性能，包括请求处理的吞吐量和平均响应时延。
2.在 Flink 上我们是通过 DAG 来描述业务流程的。由于 DAG 可以描绘得非常复杂，这就意味着我们在 Flink 上可以实现各种复杂的业务逻辑。并且可以通过 UI 界面直接观察到，所以我们可以一目了然地看清整个业务的执行流程。这样即使是非常复杂的业务流程，开发和管理起来都非常方便，完胜调用关系复杂的传统微服务架构。
3.我们可以灵活地实现资源水平扩展或收缩，只需要设置不同的 Flink 算子并行度即

可。这样就可以轻轻松松地提升或降低系统的处理性能。
4.我们可以通过反向压力，轻松方便地实现自适应的流量控制，充分发挥出资源的使用效率。

五、实现
1.服务端接受请求,发送到kafka。
其中由于异步原因,所以把请求id先放到内存Map中。注意防止内存溢出，要限制map的size数量，即相当于控制容量。

```
void channelRead0(ChannelHandlerContext ctx, HttpRequest req)
        throws Exception {
    CompletableFuture
            .supplyAsync(() -> this.httpDecode(ctx, req), httpDecoderExecutor)
            .thenAcceptAsync(e -> this.sendRequestToKafka(ctx, req, e, refContro
}
JSONObject httpDecode(ChannelHandlerContext ctx, HttpRequest req) {
    byte[] body = readRequestBodyAsString((HttpContent) req);
    String jsonString = new String(body, Charsets.UTF_8);
    return JSON.parseObject(jsonString);
}
void sendRequestToKafka(ChannelHandlerContext ctx, HttpRequest req,
                                JSONObject event, RefController ref) {
    // 这里简单地用 UUID 来生成唯一ID。
    // 更严格的唯一ID，应该考虑"主机名 + IP地址 + 进程PID + timestamp + 随机数"等因素，可
    String eventId = UUID.randomUUID().toString();
    // EVENT_ID 用于后续接收到响应后，从 blockingMap 中找回之前的请求
    event.put(EVENT_ID_TAG, eventId);
    // 由于请求只是发送到kafka，并不知道什么时候响应能够返回，
    // 所以将请求上下文和相关信息先保存到blockingMap里，
    // 等到之后从kafka里读出响应后，再到blockingMap里找到之前的请求上下文和相关信息，从而返
    RequestItem requestItem = new RequestItem(ctx, req, event, ref);
    blockingMap.put(eventId, requestItem);
    // 将请求发送到kafka的request topic，
    // 之后Flink会从kafka中将该请求读取出来并进行处理，并将处理的结果再发送到kafka的respon
    kafkaWriter.send(requestTopic, event.toJSONString().getBytes(Charsets.UTF_8)
    // 当请求已经发送到kafka后，就启动一个超时任务，
    // 如果到时候超时设置的时间到了，但是请求对应的响应还没有来，就当超时返回。
    CompletableFuture<Void> timeoutFuture = TimeoutHandler.timeoutAfter(10000, T
    timeoutFuture.thenAcceptAsync(v -> this.timeout(eventId), this.timeoutExecut
```

2.flink从kafka读取数据，处理数据，输出到另外一个kafka

```
public static void main(String[] args) throws Exception {
    final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecuti
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
    FlinkKafkaConsumer010<String> myConsumer = createKafkaConsumer();
    DataStream<String> stream = env.addSource(myConsumer);
    DataStream<String> counts = stream
            .map(new MapFunction<String, JSONObject>() {
                @Override
                public JSONObject map(String s) throws Exception {
                    if (StringUtils.isEmpty(s)) {
                        return new JSONObject();
                    }
                    return JSONObject.parseObject(s);
                }
            })
            .map(new MapFunction<JSONObject, String>() {
                @Override
                public String map(JSONObject value) throws Exception {
                    value.put("result", "ok");
                    return JSONObject.toJSONString(value);
                }
            });

    counts.addSink(createKafkaProducer()).name("flink-connectors-kafka").setPara
    env.execute("FlinkService");
}
```

3.服务端从kafka读取response数据，然后在内存中找到请求信息，返回给客户端

```
KafkaResponseHandler kafkaReader = new KafkaResponseHandler(
        zookeeperConnect, responseTopic, groupId, 2, responseExecutor) {
    @Override
    public Void process(byte[] body) {
        String jsonString = new String(body, Charsets.UTF_8);
        JSONObject e = JSON.parseObject(jsonString);
        String eventId = e.getString(EVENT_ID_TAG);
        // 从 blockingMap 中取出请求上下文信息
        RequestItem requestItem = blockingMap.remove(eventId);
        try {
            // 将请求处理结果返回给客户端
            sendResponse(requestItem.ctx, OK, RestHelper.genResponse(OK.code(),
        } finally {
            requestItem.ref.release();
        }
        return null;
    }
};
```

六、可能存在的问题

1.微服务如何扩容。

需要找两个集群，做互备。扩容的时候，开启备份集群，代码重新启动，流量切换到备份集群，老集群等待任务执行完成后退出即可。