三、dotnetCore.CAP框架实践

3.1 框架介绍

微软MVP--姓名：杨晓东

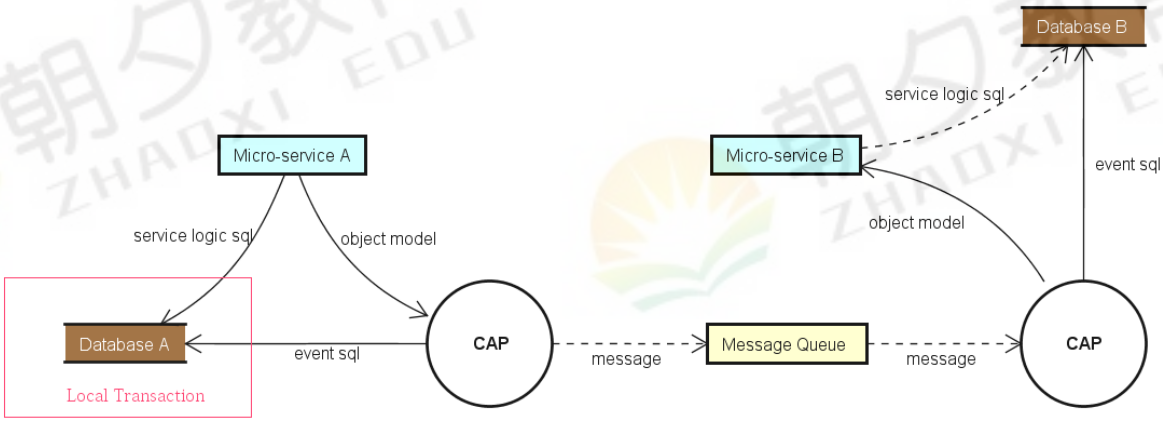
CAP 是一个用来解决微服务或者分布式系统中分布式事务问题的一个开源项目解决方案

同样可以用来作为 EventBus 使用，

github地址：https://github.com/dotnetcore/CAP

官网地址： https://cap.dotnetcore.xyz/

官网文档：https://cap.dotnetcore.xyz/user-guide/zh/cap/idempotence/



3.2 开始CAP

3.2.1 基础环境准备

1 多个空的Core WebApi

2 多个SqlServer数据库脚本+ 多个DBContext数据库访问

```
数据库初始化脚本:
1  Customers和Customers-copy1  是SQLServer的演示
2  LogisticsService、OrderService、PaymentService UserService StorageService
    都是SQLServer的
```

3 RabbitMQ

单体

```
1.拉取RabbitMQ镜像

   docker pull rabbitmq:management   //拉取包含web管理界面的RabbitMQ镜像

2.启动容器
```

```
   docker run -d --hostname my-rabbit --name rabbit -p 15672:15672 -p 5672:5672
rabbitmq:management       // 用户名密码都guest

   或者

   docker run -d --hostname my-rabbit --name rabbit -e
RABBITMQ_DEFAULT_USER=user -e RABBITMQ_DEFAULT_PASS=password -p 15672:15672 -p
5672:5672 rabbitmq:management   // 自定义用户名和密码

3.可视化
http://192.168.3.230:15672/
guest   guest
```

集群

步骤一：安装RabbitMQ

```
docker run -d --hostname rabbit1 --name myrabbit1 -p 15672:15672 -p 5672:5672 -e
RABBITMQ_ERLANG_COOKIE='rabbitcookie' rabbitmq:3.6.15-management

docker run -d --hostname rabbit2 --name myrabbit2 -p 5673:5672 --link
myrabbit1:rabbit1 -e RABBITMQ_ERLANG_COOKIE='rabbitcookie' rabbitmq:3.6.15-
management

docker run -d --hostname rabbit3 --name myrabbit3 -p 5674:5672 --link
myrabbit1:rabbit1 --link myrabbit2:rabbit2 -e
RABBITMQ_ERLANG_COOKIE='rabbitcookie' rabbitmq:3.6.15-management
```

具体的参数含义，参见上文“启动RabbitMQ”部分。

注意点：

多个容器之间使用“-link”连接，此属性不能少；
Erlang Cookie值必须相同，也就是RABBITMQ_ERLANG_COOKIE参数的值必须相同，原因见下文“配置相同Erlang Cookie”部分；
步骤二：加入RabbitMQ节点到集群

设置节点1：

```
docker exec -it myrabbit1 bash
rabbitmqctl stop_app
rabbitmqctl reset
rabbitmqctl start_app
exit
```

设置节点2，加入到集群：

```
docker exec -it myrabbit2 bash
rabbitmqctl stop_app
rabbitmqctl reset
rabbitmqctl join_cluster --ram rabbit@rabbit1
rabbitmqctl start_app
exit
```

参数“-ram”表示设置为内存节点，忽略次参数默认为磁盘节点。

设置节点3，加入到集群：

```
docker exec -it myrabbit3 bash
rabbitmqctl stop_app
rabbitmqctl reset
rabbitmqctl join_cluster --ram rabbit@rabbit1
rabbitmqctl start_app
exit
```

设置好之后，使用http://192.168.3.230:15672 进行访问了，默认账号密码是guest/guest，

4 MongoDB集群

```
1  拉取mongoDB镜像

docker pull mongo

2  创建本地挂在目录--建议先删除

mkdir -p /app/docker/mongo1/db   #创建挂载的db目录
mkdir -p /app/docker/mongo2/db   #创建挂载的db目录
mkdir -p /app/docker/mongo3/db   #创建挂载的db目录

3  启动多个MongoDB实例

#第一台：
docker run --name mongo-server1 -p 30001:27017 --restart=always -v
/app/docker/mongo1/db:/data/db -v /etc/localtime:/etc/localtime -d mongo  --
replSet "rs0"  --bind_ip_all
#第二台：
docker run --name mongo-server2 -p 30002:27018 --restart=always -v
/app/docker/mongo2/db:/data/db -v /etc/localtime:/etc/localtime -d mongo  --
replSet "rs0"  --bind_ip_all
#第三台：
docker run --name mongo-server3 -p 30003:27019 --restart=always -v
/app/docker/mongo3/db:/data/db -v /etc/localtime:/etc/localtime -d mongo  --
replSet "rs0"  --bind_ip_all

4  搭建集群

#进入容器 进入主的容器
docker exec -it mongo-server1 bash
#连接客户端
mongo

rs.initiate()
rs.add("localhost:30002")
rs.addArb("localhost:30003")
exit


5 查询使用和验证
docker exec -it mongo-server1 bash
mongo
use test
db.test.insert({msg: 'this is from primary', ts: new Date()})
```

```
docker exec -it mongo-server2 bash
mongo
use test
db.test.find()

使用工具 NoSQL Manager for MongoDB Freeware
连接：192.168.3.230:30001
```

3.2.2 项目初始化

https://github.com/dotnetcore/CAP/blob/master/README.zh-cn.md

1 Nuget引用

```
PM> Install-Package DotNetCore.CAP
PM> Install-Package DotNetCore.CAP.Dashboard   #consul
```

CAP 支持 Kafka、RabbitMQ消息队列，你可以按需选择下面的包进行安装：

```
PM> Install-Package DotNetCore.CAP.Kafka
PM> Install-Package DotNetCore.CAP.RabbitMQ
```

CAP 提供了 Sql Server, MySql, PostgreSQL，MongoDB 的扩展作为数据库存储：

```
// 按需选择安装你正在使用的数据库
PM> Install-Package DotNetCore.CAP.SqlServer
PM> Install-Package DotNetCore.CAP.MySql
PM> Install-Package DotNetCore.CAP.PostgreSql
PM> Install-Package DotNetCore.CAP.MongoDB
```

2 代码配置

生产者-ConfigureService：

```
string conn = this.Configuration.GetConnectionString("UserServiceConnection");
            string rabbitMQ =
this.Configuration.GetConnectionString("RabbitMQ");
            services.AddCap(x =>
            {
                x.UseSqlServer(conn);
                x.UseRabbitMQ(rabbitMQ);
                x.FailedRetryCount = 5;
                x.FailedThresholdCallback = failed =>
                {
                    var logger =
failed.ServiceProvider.GetService<ILogger<Startup>>();
                    logger.LogError($@"MessageType {failed.MessageType} 失败了，
重试了 {x.FailedRetryCount} 次，
                        消息名称: {failed.Message.GetName()}");//do anything
                };

                #region 注册Consul可视化
                x.UseDashboard();
                DiscoveryOptions discoveryOptions = new DiscoveryOptions();
```

```
                this.Configuration.Bind(discoveryOptions);
                x.UseDiscovery(d =>
                {
                    d.DiscoveryServerHostName =
discoveryOptions.DiscoveryServerHostName;
                    d.DiscoveryServerPort =
discoveryOptions.DiscoveryServerPort;
                    d.CurrentNodeHostName =
discoveryOptions.CurrentNodeHostName;
                    d.CurrentNodePort = discoveryOptions.CurrentNodePort;
                    d.NodeId = discoveryOptions.NodeId;
                    d.NodeName = discoveryOptions.NodeName;
                    d.MatchPath = discoveryOptions.MatchPath;
                });
                #endregion
            });
```

对应配置文件：

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "RabbitMQ": "39.96.34.52",
    "UserServiceConnection": "Server=ElevenPC;Database=UserService;User
Id=sa;Password=PasswOrd;"
  },
  "DiscoveryOptions": {
    "DiscoveryServerHostName": "localhost",
    "DiscoveryServerPort": 8500,
    "CurrentNodeHostName": "localhost",
    "CurrentNodePort": 9999,
    "NodeId": "Zhaoxi.DistributedTransaction.UserService",
    "NodeName": "Zhaoxi.DistributedTransaction.UserService",
    "MatchPath": "\"\"/cap\"\"" //默认路径
  }
}
```

3.2.3 生产者代码

任选控制器---注入ICapPublisher---直接Publish（同步/异步）---之前是body，3.0之后可以header了---

消息划分为 Header 和 Body 来进行传输。

Body 中的数据为用户发送的原始消息内容，也就是调用 Publish 方法发送的内容，我们不进行任何包装仅仅是序列化后传递到消息队列。

在 Header 中，我们需要传递一些额外信息以便于CAP在收到消息时能够提取到关键特征进行操作。

以下是在异构系统中，需要在发消息的时候向消息的Header 中写入的内容（好像已经默认添加了，再次写入id会报错）：

键	类型	说明
cap-msg-id	string	消息Id， 由雪花算法生成，也可以是 guid
cap-msg-name	string	消息名称，即 Topic 名字
cap-msg-type	string	消息的类型, 即 typeof(T).FullName (非必须)
cap-senttime	stringg	发送的时间 (非必须)

注意：

1 dotnet run --urls="http://*:9999" #端口号需要跟consul注册一致

2 x.FailedRetryCount = 5;
x.FailedRetryInterval = 60;

写入队列的重试次数，先写入数据库---关闭队列---还是能成功---交给重试机制--不行再发邮件人工介入

3 消息数据清理

数据库消息表中具有一个 ExpiresAt 字段表示消息的过期时间，当消息发送成功或者消费成功后，CAP 会将消息状态为 Successed 的 ExpiresAt 设置为 1天 后过期，会将消息状态为 Failed 的 ExpiresAt 设置为 15天 后过期。

CAP 默认情况下会每隔一个小时将消息表的数据进行清理删除，避免数据量过多导致性能的降低。清理规则为 ExpiresAt 不为空并且小于当前时间的数据。 也就是说状态为Failed的消息（正常情况他们已经被重试了 50 次），如果你15天没有人工介入处理，同样会被清理掉。

测试

```
Zhaoxi.DistributedTransaction.UserService

dotnet run --urls=http://*:11111
访问：

http://localhost:11111/without/transaction
只有发布任务---数据库看---队列无数据
http://localhost:11111/adotransaction/sync
http://localhost:11111/efcoretransaction/async
本地事务+队列消息
```

3.2.4 消费者代码

得先启动一次，队列才能写入数据，否则会丢失

1 配置文件信息一致，只修改下端口号

2 控制器里面的Action，注意NonAction 表明不是Action

CapSubscribe前面写入的名称

Group表示多几个观察者，对应RabbitMQ的队列，默认是“xxx”

```
 [NonAction]
        [CapSubscribe("RabbitMQ.SQLServer.UserService")]
        public void Subscriber(Person person)
        {
            Console.WriteLine($@"{DateTime.Now} Subscriber invoked, Info:
{person}");
            throw new Exception("Subscriber failed");
        }

        [NonAction]
        [CapSubscribe("RabbitMQ.SQLServer.UserService", Group = "group.test2")]
        public void Subscriber2(Person p, [FromCap] CapHeader header)
        {
```

3 进入处理动作之前，就已经把数据结构和任务从队列拿到数据库里了，这里的动作，仅仅是用来重试执行任务的

header为空，是因为有的地方写入的时候为空了

测试

```
Zhaoxi.DistributedTransaction.OrderService

dotnet run --urls=http://*:22222
检查RabbitMQ已经绑定多个队列---然后就会自动执行---数据库是ok的
```

3.2.5 生产者+消费者

只是在消费者的环境，Subscriber3里面通过 Group = "Group.Queue3")来区分，里面有写入数据

跟消费者一样，然后再来个写入流程就行

1 dotnet run --urls="http://*:7777"

一方面get，一方面set

靠名字+group区分

3.2.6 MySQL版生产者+消费者

这个没写代码，上一期架构班实战用的是这个，跟SQLServer基本一样

3.2.7 MongoDB版生产者+消费者

从OrderService发布过来

验证Zhaoxi.DistributedTransaction.PaymentService