# Servicecomb-saga-demo

一:从GitHub上将saga的代码拉到本地

<https://github.com/apache/incubator-servicecomb-saga.git>进入到对应目录中,将saga打入本地maven仓库

1.构建可执行文件：

mvn clean install -DskipTests

2.构建可执行文件和docker镜像：

mvn clean install -DskipTests -Pdocker

3.构建可执行文件以及Saga发行包

mvn clean install -DskipTests -Prelease

在执行以上任一指令后, 可在 alpha/alpha-server/target/saga/alpha-server-${version}-exec.jar中找到alpha-server的可执行文件

二:运行alpha-server

使用mysql|PostgreSQL作为alpha的数据库:

1.添加mysql-connector-java依赖到alpha/alpha-server/pom.xml中(PostgreSQL跳过)

<dependency>

   <groupId>mysql</groupId>

   <artifactId>mysql-connector-java</artifactId>

</dependency>

2.完成一步骤,找到可执行文件alpha/alpha-server/target/saga/alpha-server-

${version}-exec.jar

3.运行DB

a.运行mysql(docker环境)

docker run -d -e "MYSQL\_ROOT\_PASSWORD=password" -e "MYSQL\_DATABASE=saga" -e "MYSQL\_USER=saga" -e "MYSQL\_PASSWORD=password" -p 3306:3306 mysql/mysql-server:5.7

4. 创建一个名为saga的数据库

5. 运行alpha

 1> 通过docker

docker run -d -p 8080:8080 -p 8090:8090 -e "JAVA\_OPTS=-Dspring.profiles.active=mysql -Dspring.datasource.url=jdbc:mysql://${host\_address}:3306/saga?useSSL=false" alpha-server:${saga\_version}

 2> 通过可执行文件 更改数据库配置

### spring.profiles.active >> saga使用的数据库类型

### spring.datasource.url >> mysql链接

### spring.datasource.username >> mysql用户

### spring.datasource.password >> 密码

mysql：

java -Dspring.profiles.active=mysql

-Dspring.datasource.url=jdbc:mysql://localhost:3306/saga?useSSL=false -

Dspring.datasource.username=root -Dspring.datasource.password=xbf12138 -jar

alpha-server-0.3.0-SNAPSHOT-exec.jar

postgresql：

java -Dspring.profiles.active=prd -Dspring.datasource.url=jdbc:postgresql://localhost:5432/saga?useSSL=false -Dspring.datasource.username=postgres -Dspring.datasource.password=x5 -jar alpha-server-0.3.0-SNAPSHOT-exec.jar

注意:默认情况下，端口8080用于通过RPC提供omega的请求，而端口8090用于查询存储在alpha中的事件（8080、8090端口不可被占用）。

三:使用saga

1.在项目中引入maven依赖

<saga.version>0.3.0-SNAPSHOT</saga.version>

<dependency>

   <groupId>org.apache.servicecomb.saga</groupId>

   <artifactId>omega-spring-starter</artifactId>

   <version>${saga.version}</version>

</dependency>

<dependency>

   <groupId>org.apache.servicecomb.saga</groupId>

   <artifactId>omega-transport-servicecomb</artifactId>

   <version>${saga.version}</version>

</dependency>

2.添加saga注解以及相应的补偿方法

a. 在应用的入口添加 @EnableOmega 注解来初始化omega的配置并与alpha建立连接

@SpringBootApplication

@EnableOmega

public class Application {

 public static void main(String[] args) {

   SpringApplication.run(Application.class, args);

}

}

b. 在全局事务的起点添加@SagaStart注解

@SagaStart(timeout=10)

public boolean transferMoney(String from, String to, int amount) {

 transferOut(from, amount);

 transferIn(to, amount);

}

c. 在子事务处添加@Compensable注解, 并指明对其的补偿方法，参数保持一致

@Compensable(timeout=5, compensationMethod="cancel")

public boolean transferOut(String from, int amount) {

repo.reduceBalanceByUsername(from, amount);

}

// 补偿方法

public boolean cancel(String from, int amount) {

repo.addBalanceByUsername(from, amount);

}

d. 对转入的服务重复第三步即可

3. 从saga-0.3.0, 可以在服务函数或者取消函数中通过访问[OmegaContext](https://github.com/apache/incubator-servicecomb-saga/blob/master/omega/omega-context/src/main/java/org/apache/servicecomb/saga/omega/context/OmegaContext.java)来获取gloableTxld以及localTxld信息（私有成员注入的方式）

注意:

 1> 实现的服务和补偿必须满足幂等的条件

 2> 默认情况下, 超时设置需要显式声明才生效

 3> 若全局事务起点与子事务起点重合, 需要同时声明@SagaStart注解和@Compensable注解

五:配置omega,在application.yaml中添加一下配置

1.添加alpha集群地址

a.application.properties

alpha.cluster.address=127.0.0.1:8080

b.microservice.yaml

spring:

application:

name: {application.name}

alpha:

cluster:

address: {alpha.cluster.addresses}

2.添加消费者、提供者

#Add Saga Handler

handler:

chain:

Consumer:

default: loadbalance,saga-consumer

servicecomb:

#Add Saga Handler

handler:

chain:

Provider:

default: saga-provider

六:通过访问http://${alpha-server:port}/events来获取所有的saga事件信息。

七:过程中的bug

1.@compensable 得不到GlobalTxId NPE

原因不明

SAGA依赖 版本并无关系

<dependency>

<groupId>com.google.guava</groupId>

<artifactId>guava</artifactId>

<version>20.0</version>

</dependency>

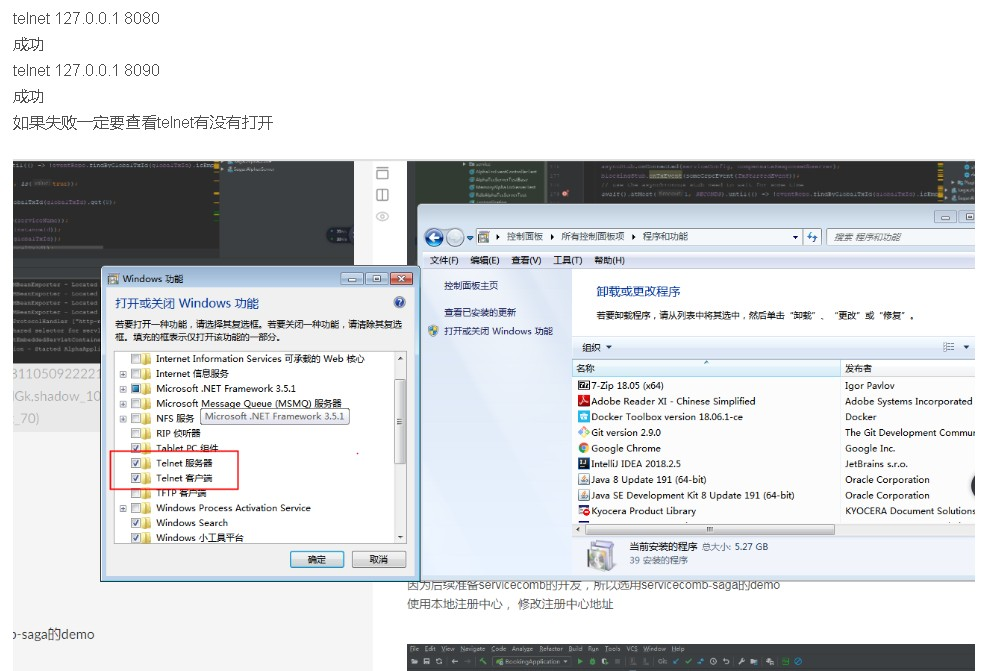
2.内存溢出

项目不要直接导入saga包

3.有关参数序列化的问题

参数是需要能序列化的，因为目前参数是需要从Omega传递到Alpha

4.编译 alpha 失败



5. io.grpc.StatusRuntimeException: UNKNOWN

业务方法 与补偿接口 不匹配

6.补偿方法不生效

a.调用补偿时，线程状态为cancelled，断开与alpha通信

b.本地业务异常，调用之前微服务的补偿方法，alpha在操作command 显示eventId重复

原因：在alpha调用补偿时，调用方法及参数都是alpha相对路径。

在hoolink实例中 rpc 调用base device ：

alpha要依赖sdk，保证补偿事务的正确进行

7.insert业务的补偿事务拿不到id

补偿事务只能获得传入的参数，在业务中生成的id没有记录，无法取得。

1.简单业务：

将业务拆解为两个部分A\B，A负责操作insert，用本地事务注解，一旦失败，直接回滚；B用于执行其他操作，参数为insert的主键id，用子事务@Compensable注解，用于补偿insert操作

2.复杂业务：

由于saga模式支持在controller调用Compensable注解的业务方法，不支持在service中调用Compensable注解的业务方法。

此时，用加表的方式记录id，用于insert的补偿事务

a.用注入OmegaContext的方式获得全局事务ID globalTxId 本地事务ID localTxId

b.加上主键id，类别字段，永久化数据，待补偿时取用

八: demo代码: https://github.com/Atobe-wm/transaction-saga-demo

九:链路介绍

1.链路介绍

a.rpc—base device 本地业务

b.rpc-base-device

2.解决的问题

Q1：a中base异常

A1：不触发补偿事务

Q2：a中device异常

A2：base补偿事务触发

Q3：a中本地业务异常

A3：base device补偿事务触发

Q4：b中base异常

A4：不触发补偿事务

Q4：b中device异常

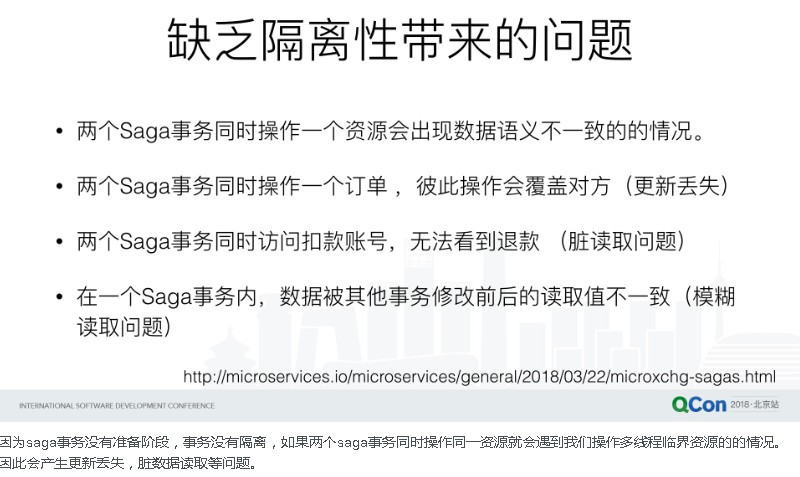
A4：base补偿事务触发

结论：在saga机制中，微服务本身异常，是不会补偿的，只会补偿该微服务之前调用的业务代码

解决：saga配合本地事务或是级联事务使用@Transactional

备注：一旦对业务代码trycatch，alpha不会记录异常事务，也不会有代码补偿

3.遗留问题



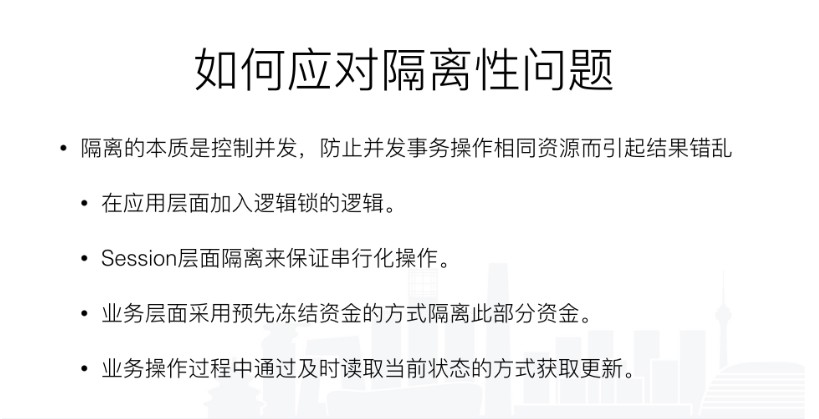
4.解决办法：

在应用层面加入逻辑锁的逻辑

Session层面隔离来保证串行化操作。

业务层面采用行锁的方式隔离覆盖

业务层面通过及时读取当前状态方式进行更新



对于遗留问题2

在数据库加行锁，或者，加个字段isuser ，当第一个事务触发的时候修改状态。

简单的说saga的缺陷是事务的隔离，隔离的本质是控制并发，那么防止并发事务操作相同的资源而引起的错乱，就可以弥补掉saga事务的缺陷。