腾讯微服务平台

分布式事务

产品文档

腾讯云

文档目录

分布式事务

概述

最近更新时间：2018-12-19 15:30:46

TSF 提供金融级别高可用分布式事务能力，保证大规模的分布式场景下的业务一致性。TSF 框架下的分布式事务基于 TCC（Try、Confirm 和 Cancel 的简称）模式，支持跨数据库、跨服务的使用场景。为金融、制造业、互联网等行业客户保驾护航。

TCC 模式

TCC 模式是一种补偿性分布式事务，包含 Try、Confirm、Cancel 三种操作。其中 Try 阶段预留业务资源，Confirm 阶段确认执行业务操作，Cancel 阶段取消执行业务操作。
TCC 模式解决了跨服务操作的原子性问题，对数据库的操作为一阶段提交，性能较好。因此，TCC 模式是现今被广泛应用的一种分布式事务模式。

相关文档

- 事务管理词汇表
- 事务管理常见问题

开发文档

最近更新时间：2019-04-12 17:17:38

配置 TCC 事务

1. 添加依赖

通过 Maven 引入依赖，在项目的 pom 文件添加下述配置项：

```
<dependency>
<groupId>com.tencent.tsf</groupId>
<artifactId>tcc-transaction-core</artifactId>
<version>1.10.1.TSF-RELEASE</version>
</dependency>
```

2. 启用 TCC 事务

在 Spring Cloud 的启动类加入注解 EnableTcc：

```
@SpringBootApplication
@EnableTcc
public class App {
public static void main(String[] args) {
SpringApplication.run(App.class);
}
}
```

3. 添加注解

您可以在事务的入口函数上添加 @TsfTcc 注解。建议将注解加在接口定义的方法上。
TsfTcc注解有如下属性：

- serviceName：事务所属的服务名，必选。
- type：事务类型，TransactionType.ROOT表示主事务，TransactionType.BRANCH表示子事务，默认值为Root，可选。
- timeout_ms：事务的超时时间，属性可选，默认为60秒，可选。
- confirmMethodName：confirm方法名，主事务可选，子事务必选。
- cancelMethodName：cancel方法名，主事务可选，子事务必选。
- autoRetry：事务超时后，是否由服务器托管继续自动重试。

定义主事务

主事务函数只需要定义一个事务入口函数即可。type 选择 Root类型；函数抛出 Throwable 异常，返回值没有特殊要求；建议业务将所需的参数都封装成一个对象，只使用这一个对象作为入参即可（所有的参数必须实现 Serializable 接口），一个主事务函数定义的样例如下：

```
public class MainTransaction {

@TsfTcc(serviceName = "myTcc", type = TransactionType.ROOT, timeout_ms = 60000)
public String beginTcc(MyParams params) throws Throwable {
//这里写业务逻辑，调用子事务函数
}
}
```

定义子事务

一个完整的子事务需要定义3个方法：**Try**、**Confirm**、**Cancel**。
其中**只有 Try 方法需要加 TsfTcc 注解**，在 TsfTcc 标签中指定对应 Confirm 和 Cancel 方法名即可。

子事务函数定义约束如下：

- Try、Confirm、Cancel 的前两个参数必须为 String txId 和 long branchId(在主事务调用子事务的时候，这两个参数分别为 null 和0即可)，其中 txId 为主事务 ID，全局唯一；branchId为子事务Id，用于区分子事务的父子关系和子事务之间的调用顺序
- Try 函数返回 Throwable 异常；Try 函数的返回值为 void，TCC 通过异常返回失败结果。
- Confirm 和 Cancel 函数返回值为 Boolean 类型，操作成功返回 true，操作失败返回 false。

示例：子事务函数定义

```
public interface SubService {

@TsfTcc(serviceName = "subService", type = TransactionType.BRANCH, confirmMethodName = "subConfirm", cancelMethodName = "subCancel")
public void subTry(String txId,long branchId,MyParams params) throws Throwable;

public boolean subConfirm(String txId,long branchId,MyParams params) throws Throwable;

public boolean subCancel(String txId,long branchId,MyParams params) throws Throwable;

}
```

示例：主事务与子事务结合

```
public class MainTransaction {

SubService subService;

@TsfTcc(serviceName = "myTcc", type = TransactionType.ROOT, timeout_ms = 60000)
public String beginTcc(MyParams params) throws Throwable {
subService.subTry(null,0,params);
}
}
```

检查事务调用

启动服务，从外部调用主事务函数，调用完成之后查看业务日志，一次成功的事务调用日志如下：

```
14:21:30.470 INFO [main] executeLog - begin transaction: 775025cf-d3e6-3fe8-8c8f-31d44fcde46d
14:21:30.649 INFO [main] executeLog - transaction: 775025cf-d3e6-3fe8-8c8f-31d44fcde46d branch index 1 succ,retry times: 0
14:21:30.652 INFO [main] executeLog - confirm 775025cf-d3e6-3fe8-8c8f-31d44fcde46d branch index: 1, times: 1
14:21:30.653 INFO [main] executeLog - commit 775025cf-d3e6-3fe8-8c8f-31d44fcde46d branch index: 1 succ
14:21:30.653 INFO [main] executeLog - transaction:775025cf-d3e6-3fe8-8c8f-31d44fcde46d confirm succ
```

子事务访问外部服务

在 SpringCloud 中，您可以通过集成 FeignClient 访问外部服务，再次以上述的子事务为例，假如子事务需要调用一个代金券服务的相关接口，则配置如下：

```
@FeignClient(value = "couponService")
@RequestMapping(value = "/api/v6/data/couponService")
public interface SubService {

@TsfTcc(serviceName = "subService", type = TransactionType.BRANCH, confirmMethodName = "subConfirm", cancelMethodName = "subCancel")
@RequestMapping(value = "/try", method = RequestMethod.POST)
public void subTry(String txId,long branchId,MyParams params) throws Throwable;

@RequestMapping(value = "/confirm", method = RequestMethod.POST)
```

```
public boolean subConfirm(String txId,long branchId,MyParams params) throws Throwable;

@RequestMapping(value = "/cancel", method = RequestMethod.POST)
public boolean subCancel(String txId,long branchId,MyParams params) throws Throwable;
}
```

在接口上增加了如下注解：

- `@FeignClient(value = "couponService")` ，value 的值为外部服务在服务注册中心注册的服务名。
- `@RequestMapping(value = "/api/v6/data/couponService")` ，value 的值为外部服务的根 URL。

在方法上增加了如下注解：

- `@RequestMapping(value = "/try", method = RequestMethod.POST)` ，value 的值为外部服务的 API 的 URL 和请求方式。

由于分布式事务框架本身会针对异常的调用做重试，业务方在调用事务相关接口（包含主事务和子事务）的时候请关闭请求重试策略。这里以 Spring Cloud 标准的 FeignClient 和 Ribbon 为例，指导如何关闭请求重试策略。

- FeignClient 默认是不会进行重试的，如果业务自定义了 FeignClient 上面的 retryer，取消自定义配置即可。
- Ribbon 默认的配置是，当请求失败时，会选择另外一个可用的服务实例进行重试，因此需要手动配置 Ribbon 重试策略。
  i. 为项目创建一个 Spring boot 的标准配置文件 application.properties，如果项目已经有 Spring boot 的 properties 文件，可以不额外创建，直接添加内容即可。
  ii. 指定关闭某个服务的自动重试策略。在配置文件中添加配置项：

  ```
  couponService.ribbon.MaxAutoRetriesNextServer=0
  couponService.ribbon.MaxAutoRetries=0
  ```

  上述配置表示当调用服务名为 couponService 的服务时，关闭 Ribbon 的自动重试，服务名根据业务实际情况进行填写。

分布式事务的超时和重试机制

用户可以在主事务的注释中定义超时时间，默认超时时间为60秒，此时的超时时间为整个事务的超时时间。当事务超时后，事务管理器对事务进行接管，以每5秒一次的重试频率自动触发重试，重试频率逐渐增长，直到600秒。用户可以在控制台上控制中断这一重试过程，也可以手动触发重试。7天后，重试自动终止。

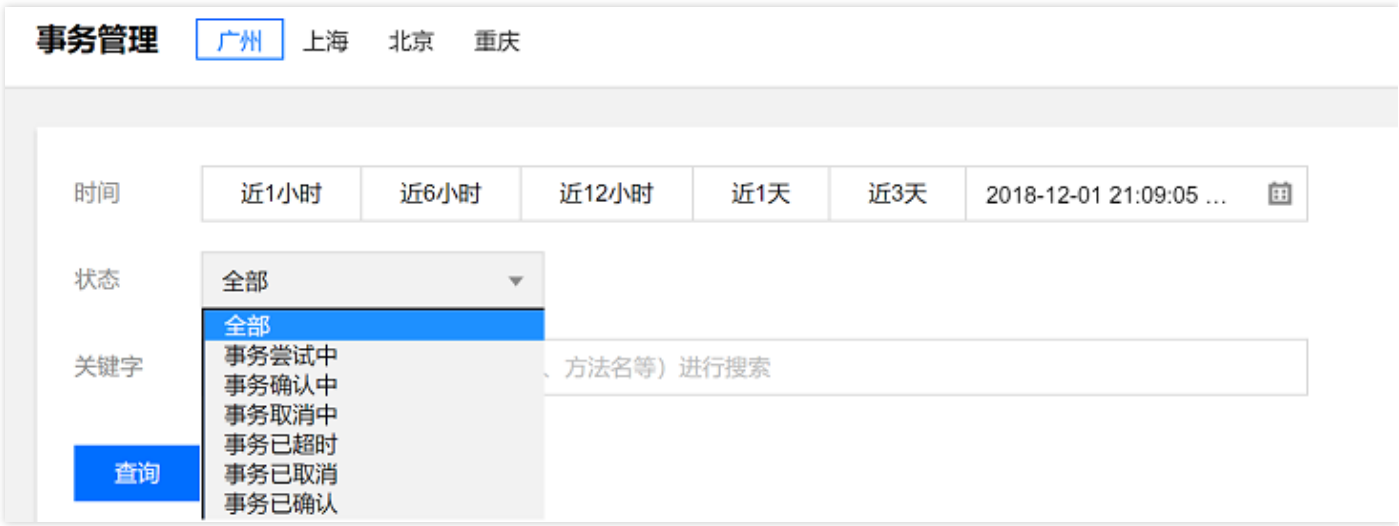
对于未超时的事务，Try 阶段自动重试3次，3次重试不成功自动触发 Cancel 进行回滚。对于 Confirm 或者 Cancel 阶段始终不能执行成功的情况，会重试直到超时。超时后由事务管理器接管。

控制台基本操作

最近更新时间：2019-04-12 18:08:44

查询事务信息

1. 登录 TSF 控制台。
2. 在左侧导航栏中，单击【事务管理】。
3. 在事务管理界面，选择需要查看的事务时间段、事务状态，并填写关键词。关键词可以选择事务相关的服务名、方法名称。
4. 单击【查询】按钮，可查看事务 ID、起始节点、超时时长等相关信息。



5. 在查询结果列表中，单击主事务 ID，可以查看主事务下的子事务运行状态，包括子事务的方法名、服务名、节点、状态、起止时间等信息。
   每一次子事务发起的请求都会记录在子事务列表中，包含子事务的确认、取消、重试等。当子事务发生重试时，您可以查看子事务重试原因（由主事务触发、由事务管理器启动发起或者由用户在控制台上手动触发）。
6. 在子事务列表中，单击操作列的【查看事务参数】，可以查看每条子事务的参数内容。

处理已超时事务

当主事务的处理时间超过了 SDK 中设置的超时时限后，系统会认为事务已处于超时状态。针对超时事务，TSF 提供了两种重试渠道：自动重试和手动重试。

自动重试

当事务超时时，框架自动重试，重试时间间隔从5秒/次倍速递增直到600秒/次，当超时时间超过7天后，会自动停止重试。

手动重试

用户可以在控制台上，通过鼠标点击的方式触发重试。

1. 登录 TSF控制台。
2. 在左侧导航栏中，单击【事务管理】。
3. 批量选择需要进行手动重试或需要停止/触发自动重试的事务，单击【手动触发重试】。

样例部署

最近更新时间：2019-07-08 11:02:58

业务场景

本样例是常见的一个线上的代金券/现金协同购物场景。在进行购物的时候，消费用户可以通过使用代金券来抵消一部分的现金费用：用户在消费的时候出示一张2元的代金券，在购买价值20元的物品的时候，只需要从微信钱包中支付18元即可。
在整个购物事务场景中，假设涉及三个的不同子服务：代金券服务、微信钱包服务以及商家账务服务。各个子服务部署在不同的节点上，使用不同的数据库。本样例展示了如何使用 TCC 来完成一次跨服务/跨数据库的分布式事务。

样例模块说明

下载样例 >>
下载 Demo 样例工程之后，进行解压，项目结构如下：

```
tcc-transaction-sample
|__sample-tcc-consumer
|__sample-tcc-couponService
|__sample-tcc-mainService
|__sample-tcc-transferService
|__sample-tcc-walletService
|__tcc-test-simple
|__init_database.sql
```

sample-tcc-consumer：消费用户 Client，发起购物事务。
sample-tcc-couponService：代金券子服务，处理代金券使用流程。
sample-tcc-walletService：微信钱包子服务，处理消费用户钱包消费流程。
sample-tcc-transferService：商家账务子服务，转账给对应商家。
sample-tcc-mainService：购物服务，购物事务入口，协调执行3个子服务。
tcc-test-simple：完整的简易 Demo，只需要一台服务器就能运行，无需安装 MySQL。
init_database.sql：初始化样例数据库。

公有云构建方法

1. 下载 Demo 压缩包并解压，在 Demo 的根目录下执行以下命令。

```
mvn clean package
```

等待 maven 打包成功之后，在以下目录找到 jar 包，用于公有云部署。

- sample-tcc-couponService/target
- sample-tcc-mainService/target
- sample-tcc-transferService/target
- sample-tcc-walletService/target
- tcc-test-simple/target

2. 在公有云上申请机器，并按照 couponService、transferService、walletService、mainService 的顺序部署应用。
   如果您只需简单试用，可以只部署 tcc-test-simple，跳过以下3、4、5步骤，执行步骤6。

3. 在 mainService 上安装 MySQL 和 ubuntu 服务器。

```
sudo apt-get install mysql-server
sudo vim /etc/mysql/mysql.conf.d/mysqld.cnf //修改bind-address为0.0.0.0
service mysql restart
```

4. 配置 MySQL 远程登录权限，并初始化数据库。

```
mysql -u root -p
grant all privileges on *.* to root@'%' identified by "root";
create database demo;
use demo;
source init_database.sql //需要拷贝到机器上去
```

5. 配置数据库地址。
   由于完整版本 Demo 需要连接 MySQL 数据库，因此您需要对每个服务配置数据库的地址。您可以使用 TSF 的 应用配置 功能进行配置，每个服务的配置文件是各个服务的 resources 中的 application.yml，例如 couponService 的配置文件如下。只需要按照实际修改 MySQL 的 url 即可。

```
spring:
application:
name: "couponService"
datasource:
url: jdbc:mysql://127.0.0.1:3306/demo
username: root
```

```
password: root
server:
tomcat:
basedir: servlet
port: 8082
address: 0.0.0.0
```

6. 在安装了 mainService 的机器上，通过 HTTP 请求 mainService。如果 mainService 所在的实例有公网 IP，也可以直接通过公网 IP 访问。

```
curl -X POST \
http://localhost:8083/buy \
-H 'Cache-Control: no-cache' \
-H 'Content-Type: application/json' \
-d '{
"couponId" : "9",
"userId" : "1",
"merchantId" : "2",
"money" : "20",
"couponValue" : "5"}'
//tcc-test-simple命令
curl -X POST \
http://localhost:8083/buy \
-H 'Cache-Control: no-cache' \
-H 'Content-Type: application/json' \
-d '{
"from":"a",
"to":"b",
"money":1
}'
```