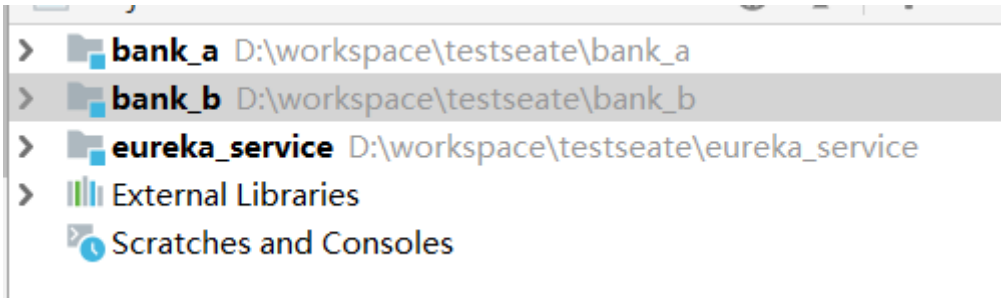
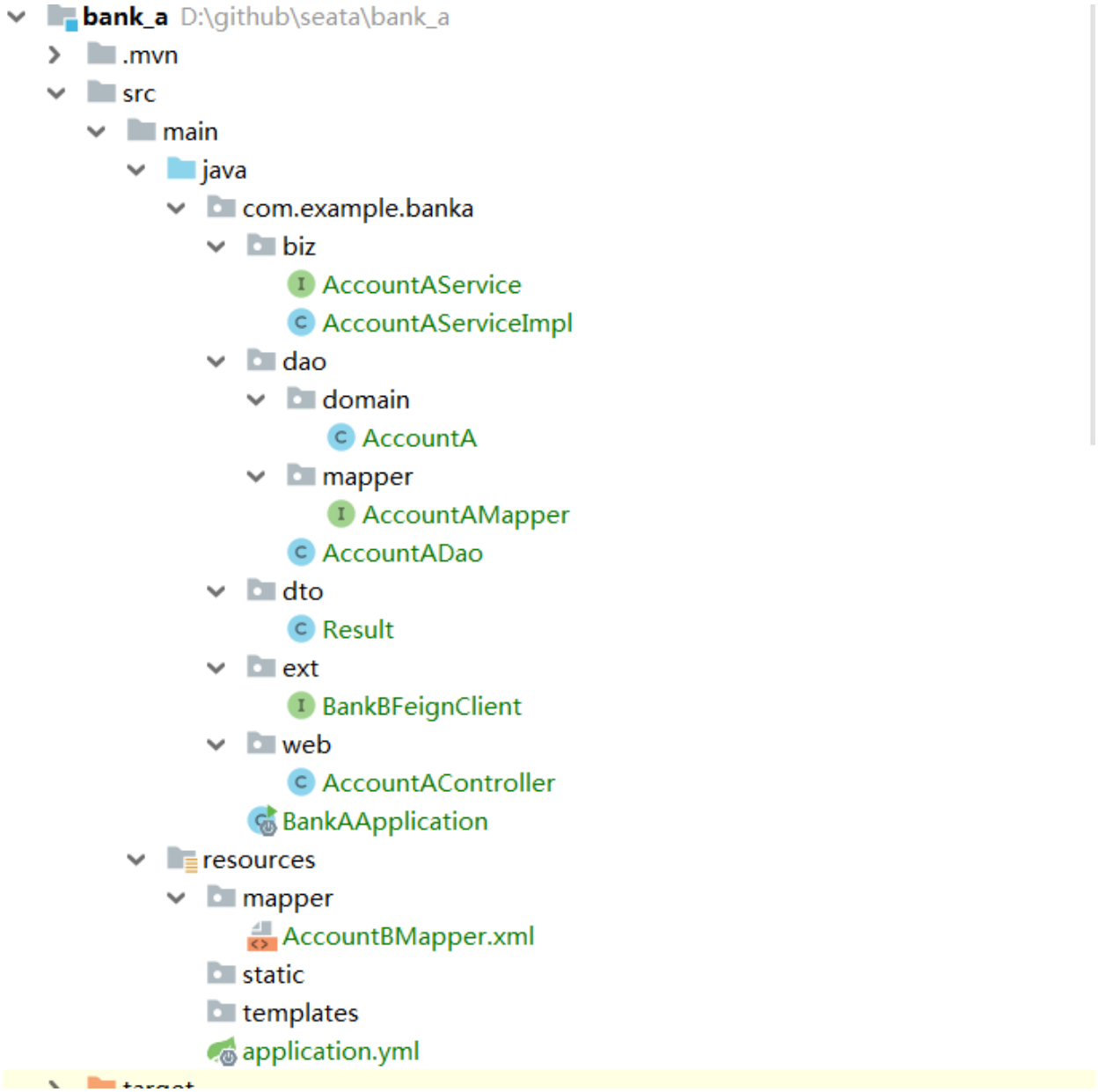
demo逻辑：搭建两个微服务，bank_a,bank_b，从银行A转账到B，A账号减钱，B账号加钱。

1、先搭建一个eureka-feign-mybatis这样的一组服务：

项目结构：

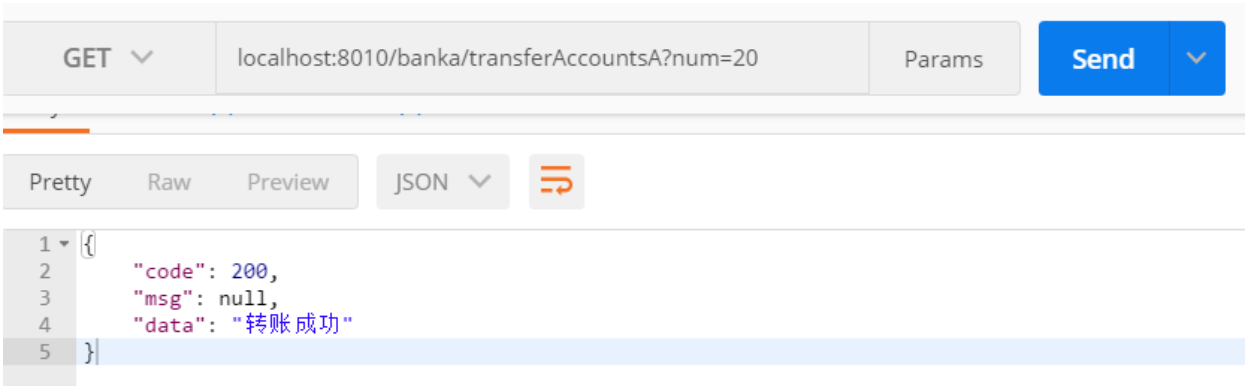


bank_a结构

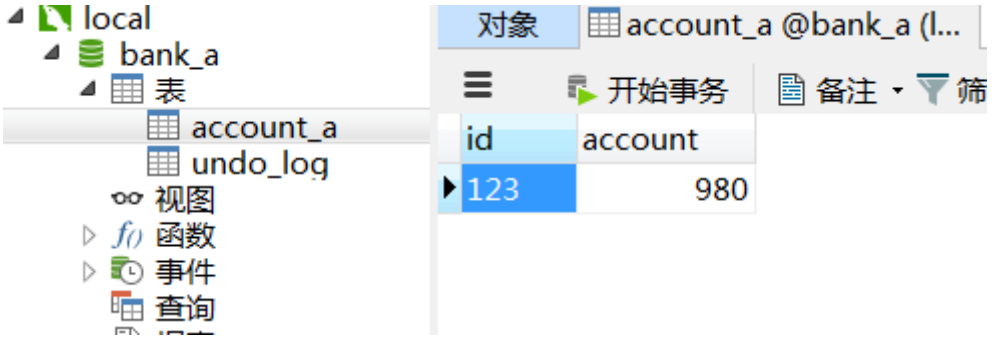


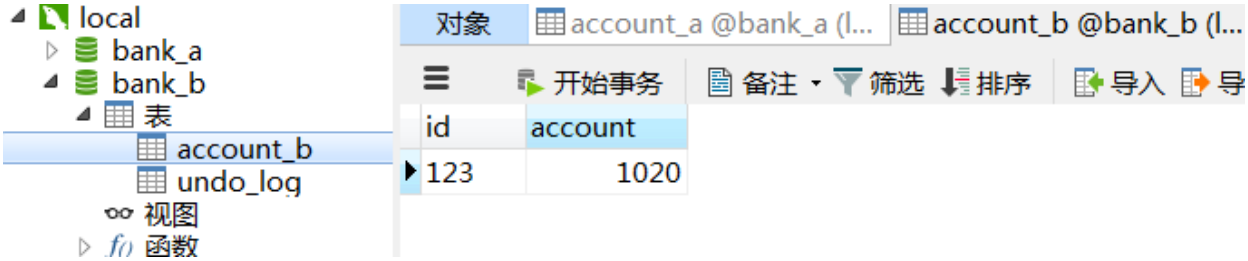
bank_b和bank_a结构类似；

依次启动eureka,bank_a,bank_b,请求接口



两个数据库初始值都是1000，请求后变化：



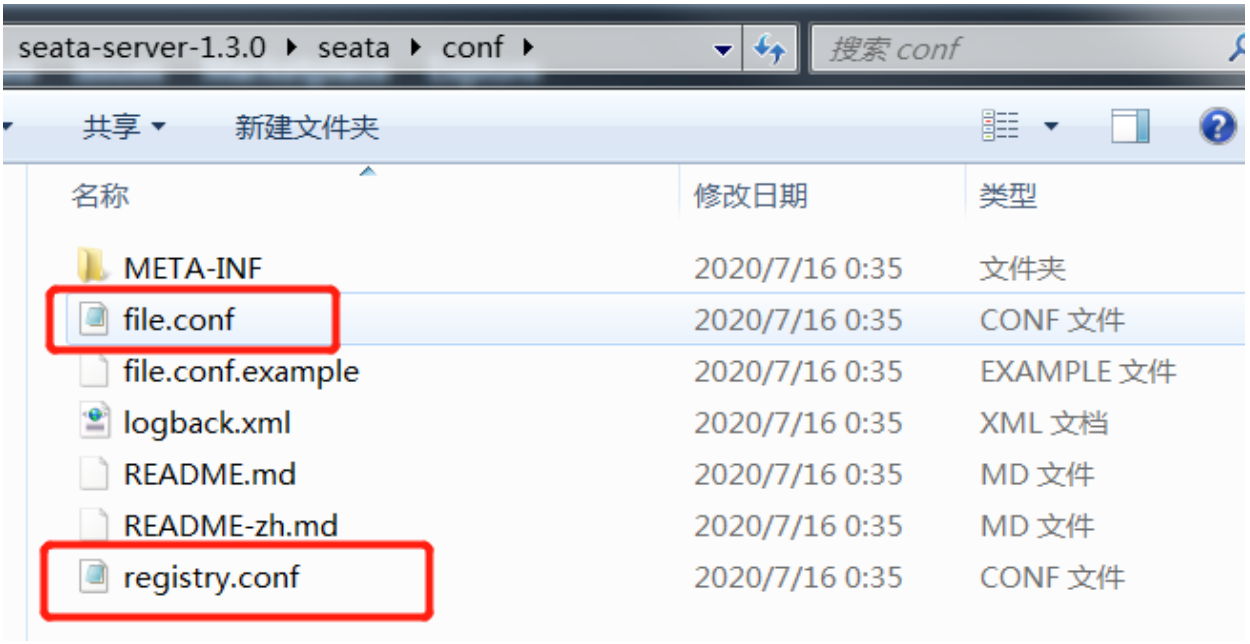


2、集成seata分布式事物：

1、下载seata的服务端：

https://github.com/seata/seata/releases，可以选择下载源码或者jar。

修改服务端参数：



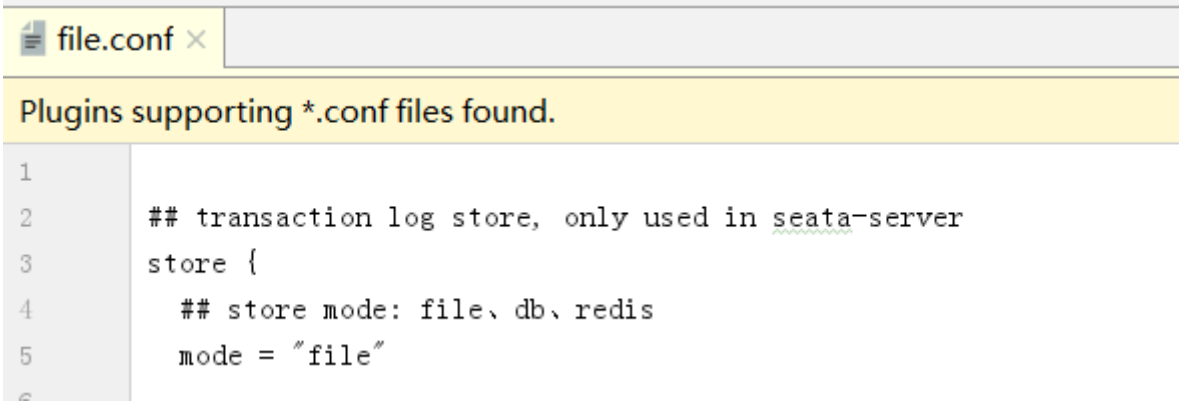
registry.conf

我们选择eureka作为注册中心，配置选择以文件方式

```
registry {
  # file 、nacos 、eureka、redis、zk、consul、etcd3、sofa  选择一种注册中心才方式
  type = "euraka"
  eureka {
    serviceUrl = "http://localhost:8000/eureka"
    application = "default"
    weight = "1"
  }
}
config {
  # file、nacos 、apollo、zk、consul、etcd3  选择一种获取配置数据的方式，file
  type = "file"
  file {
    name = "file.conf"
  }
}
```

file.conf

在registry.conf 中选择了文件方式配置，这里就需要修改file.conf



这里是针对seata-server日志配置，默认选择file，暂时不改了。

2、客户端集成seata

bank_a和bank_b配置是一样的。

2.1、引入依赖：这里以0.6.1版本为例

```
<!-- seata -->
<dependency>
    <groupId>io.seata</groupId>
    <artifactId>seata-all</artifactId>
    <version>0.6.1</version>
</dependency>
```

2.2、客户端配置

registry.conf 和服务端一致即可：

registry.conf

```
registry {
  # file 、nacos 、eureka、redis、zk、consul、etcd3、sofa 选择一种注册中心才方式
  type = "eureka"
  eureka {
    serviceUrl = "http://10.32.3.56:8000/eureka"
    application = "default"
    weight = "1"
  }

}

config {
  # file、nacos 、apollo、zk、consul、etcd3  选择一种获取配置数据的方式，file
  type = "file"
  file {
    name = "file.conf"
  }
}
```

file.conf

```
transport {
  # tcp udt unix-domain-socket
  type = "TCP"
  #NIO NATIVE
  server = "NIO"
  #enable heartbeat
  heartbeat = true
  #thread factory for netty
```

```
  thread-factory {
    boss-thread-prefix = "NettyBoss"
    worker-thread-prefix = "NettyServerNIOWorker"
    server-executor-thread-prefix = "NettyServerBizHandler"
    share-boss-worker = false
    client-selector-thread-prefix = "NettyClientSelector"
    client-selector-thread-size = 1
    client-worker-thread-prefix = "NettyClientWorkerThread"
    # netty boss thread size,will not be used for UDT
    boss-thread-size = 1
    #auto default pin or 8
    worker-thread-size = 8
  }
}
service {
  #vgroup->rgroup
  vgroup_mapping.my_test_tx_group = "default"
  #only support single node
  default.grouplist = "localhost:8091"
  #degrade current not support
  enableDegrade = false
  #disable
  disable = false
}
client {
  async.commit.buffer.limit = 10000
  lock {
    retry.internal = 10
    retry.times = 30
  }
}
```

2.3 seata代码配置

RequestHeaderInterceptor Feign拦截器，在fengin调用别的服务时把RootContext中的XID查到请求头里面

SeataXidFilter 获取请求头中的XID

SeataConstant 请求头的标示

SeataAutoConfig seata代码配置

```
@Component
public class RequestHeaderInterceptor implements RequestInterceptor {
   @Override
   public void apply(RequestTemplate template) {
      String xid = RootContext.getXID();
      if (StringUtils.isNotBlank(xid)) {
         template.header("Xid_Header", xid);
      }
   }
}
@Component
public class SeataXidFilter extends OncePerRequestFilter {
   protected Logger logger = LoggerFactory.getLogger(SeataXidFilter.class);
   @Override
   protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain filterChain)
         throws ServletException, IOException {
      String xid = RootContext.getXID();
      String restXid = request.getHeader(SeataConstant.XID_HEADER);
      boolean bind = false;
      if (StringUtils.isBlank(xid) && StringUtils.isNotBlank(restXid)) {
         RootContext.bind(restXid);
         bind = true;
         if (logger.isDebugEnabled()) {
            logger.debug("bind[" + restXid + "] to RootContext");
         }
      }
      try {
         filterChain.doFilter(request, response);
      } finally {
         if (bind) {
            String unbindXid = RootContext.unbind();
            if (logger.isDebugEnabled()) {
               logger.debug("unbind[" + unbindXid + "] from RootContext");
            }
            if (!restXid.equalsIgnoreCase(unbindXid)) {
```

```
                logger.warn("xid in change during http rest from " + restXid + " to " +
unbindXid);
                if (unbindXid != null) {
                    RootContext.bind(unbindXid);
                    logger.warn("bind [" + unbindXid + "] back to RootContext");
                }
            }
        }
    }
}

public class SeataConstant {
    public static final String XID_HEADER = "Xid_Header";
    public SeataConstant() {
    }
}
@Configuration
public class SeataAutoConfig {
    @Autowired
    private DataSourceProperties dataSourceProperties;

    /**
     * druid数据源
     *
     * @return
     * @author sly
     * @time 2019年6月11日
     */
    @Bean
    @Primary
    public DruidDataSource druidDataSource() {
        DruidDataSource druidDataSource = new DruidDataSource();
        druidDataSource.setUrl(dataSourceProperties.getUrl());
        druidDataSource.setUsername(dataSourceProperties.getUsername());
        druidDataSource.setPassword(dataSourceProperties.getPassword());
        druidDataSource.setDriverClassName(dataSourceProperties.getDriverClassName());
        druidDataSource.setInitialSize(0);
        druidDataSource.setMaxActive(180);
        druidDataSource.setMaxWait(60000);
```

```
        druidDataSource.setMinIdle(0);
        // druidDataSource.setValidationQuery("Select 1 from DUAL");
        druidDataSource.setTestOnBorrow(false);
        druidDataSource.setTestOnReturn(false);
        druidDataSource.setTestWhileIdle(true);
        druidDataSource.setTimeBetweenEvictionRunsMillis(60000);
        druidDataSource.setMinEvictableIdleTimeMillis(25200000);
        druidDataSource.setRemoveAbandoned(true);
        druidDataSource.setRemoveAbandonedTimeout(1800);
        druidDataSource.setLogAbandoned(true);
        return druidDataSource;
    }

    /**
     * 代理数据源
     *
     * @param druidDataSource
     * @return
     * @author sly
     * @time 2019年6月11日
     */
    @Bean
    public DataSourceProxy dataSourceProxy(DruidDataSource druidDataSource) {
        return new DataSourceProxy(druidDataSource);
    }

    /**
     * 初始化mybatis sqlSessionFactory
     *
     * @param dataSourceProxy
     * @return
     * @throws Exception
     * @author sly
     * @time 2019年6月11日
     */
    @Bean
    public SqlSessionFactory sqlSessionFactory(DataSourceProxy dataSourceProxy) throws
Exception {
        //☆ 需要使用MybatisSqlSessionFactoryBean 的sqlsession，不然mybatis一些基础方法会用不了
        MybatisSqlSessionFactoryBean factoryBean = new MybatisSqlSessionFactoryBean();
```

```
        factoryBean.setDataSource(dataSourceProxy);
        factoryBean.setMapperLocations(new
PathMatchingResourcePatternResolver().getResources("classpath:mapper/*.xml"));
        factoryBean.setTypeAliasesPackage("com.example.bankb.dao.domain");
        factoryBean.setTransactionFactory(new JdbcTransactionFactory());
        return factoryBean.getObject();
    }
    /**
     * 初始化全局事务扫描
     *
     * @return
     * @author sly
     * @time 2019年6月11日
     */
    @Bean
    public GlobalTransactionScanner globalTransactionScanner() {
        return new GlobalTransactionScanner("bank-b", "my_test_tx_group");
    }
}
```

2、4运行代码

依次运行eureka,seata_server,bank_a,bank_b

客户端以下显示及注册成功。

```
[imeoutChecker_1] i.s.core.rpc.netty.NettyPoolableFactory  : register success, cost 4 ms, version:0.6.1,
```

服务端显示

```
2020-07-31 14:24:14.621 INFO [ServerHandlerThread_5_500]io.seata.core.rpc.Defaul
tServerMessageListenerImpl.onRegRmMessage:112 -rm register success message:Regis
terRMRequest{resourceIds='jdbc:mysql://localhost:3306/bank_b?useSSL=false&useUni
code=true&characterEncoding=utf-8&allowMultiQueries=true&serverTimezone=UTC', ap
plicationId='bank-b', transactionServiceGroup='my_test_tx_group'},channel:[id: 0
x53b1f890, L:/10.32.3.56:8091 - R:/10.32.3.56:60025]
2020-07-31 14:24:17.475 INFO [NettyServerNIOWorker_2_8]io.seata.core.rpc.Default
ServerMessageListenerImpl.onRegTmMessage:128 -checkAuth for client:10.32.3.56:60
032 vgroup:my_test_tx_group ok
```

2.5测试代码

在bank_a 中编写如下代码：

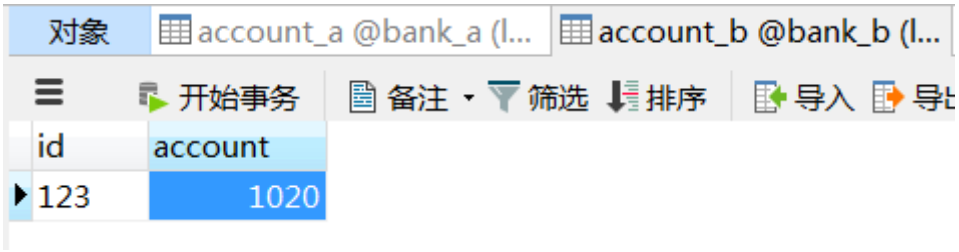
```
    //@GlobalTransactional()
@Override
public int updateAccount(int num) {
    Result result = bankB.transferAccounts(num);
    if (result.getCode()!=200){
        throw new IllegalStateException("bankB异常");
    }
```

```
    AccountA account = accountADao.query("123");
    account.setAccount(account.getAccount() - num);
    if (num >= 20){
        throw new RuntimeException("转账金额过大");
    }
    return accountADao.update(account);
}
```

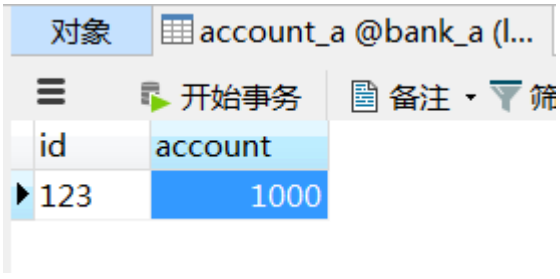
在不加@GlobalTransactional()注解的情况下，请求接口

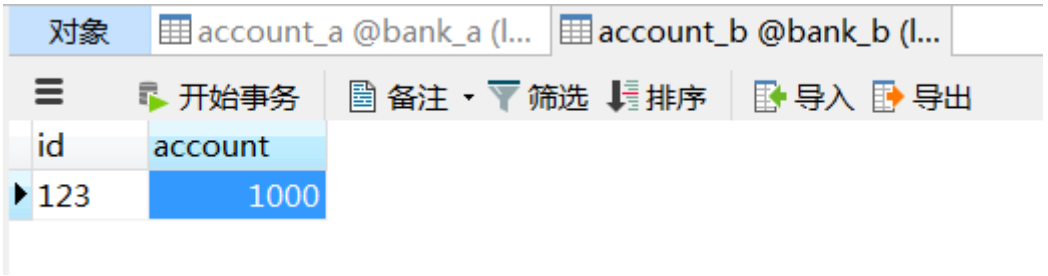
localhost:8010/banka/transferAccountsA?num=20

bank_b账号加了20，bank_a没有减少。



加上注解之后，重置数据库数据，效果如下：





在bank_b中会看到如下log,显示事务回滚成功。

```
i.s.core.rpc.netty.RmMessageListener     : onMessage:xid=10.32.3.56:8091:2049892225,branchId=2049892226,branchType=AT,resourceId=jdbc:mysql://localhost:3306/bank_b?useSSL=false&useUnicode=true&characterEncoding=utf-8&allowM
io.seata.rm.AbstractRMHandler            : Branch Rollbacking: 10.32.3.56:8091:2049892225 2049892226 jdbc:mysql://localhost:3306/bank_b?useSSL=false&useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&serverTimez
i.s.rm.datasource.undo.UndoLogManager    : xid 10.32.3.56:8091:2049892225 branch 2049892226, undo_log deleted with GlobalFinished
io.seata.rm.AbstractRMHandler            : Branch Rollbacked result: PhaseTwo_Rollbacked
io.seata.core.rpc.netty.RmRpcClient      : RmRpcClient sendResponse xid=10.32.3.56:8091:2049892225,branchId=2049892226,branchStatus=PhaseTwo_Rollbacked,result code =Success,getMsg =null
```

3、代码流程分析：

在bank_a方法里执行feign调用别的服务时，会生成一个XID，这个XID会存到请求头里面，在bank_b中会取出来，执行更新操作时，会在log表里建一条数据记录，后面bank_a中执行的回滚时，根据这条记录回滚当前数据库数据。没有错误时，执行提交操作。

没有报错时的bank_b:

```
i.s.core.rpc.netty.RmMessageListener    : onMessage:xid=10.32.3.56:8091:2049892236,branchId=2049892237,branchType=AT,resourceId=jdbc:mysql://localhost:3306/bank_b?useSSL=false&useUnicode=
io.seata.rm.AbstractRMHandler           : Branch committing: 10.32.3.56:8091:2049892236 2049892237 jdbc:mysql://localhost:3306/bank_b?useSSL=false&useUnicode=true&characterEncoding=utf-8&
io.seata.rm.AbstractRMHandler           : Branch commit result: PhaseTwo_Committed
io.seata.core.rpc.netty.RmRpcClient     : RmRpcClient sendResponse xid=10.32.3.56:8091:2049892236,branchId=2049892237,branchStatus=PhaseTwo_Committed,result code =Success,getMsg =null
```

报错时的bank_b:

```
i.s.core.rpc.netty.RmMessageListener    : onMessage:xid=10.32.3.56:8091:2049892225,branchId=2049892226,branchType=AT,resourceId=jdbc:mysql://localhost:3306/bank_b?useSSL=false&useUnicode=true&characterEncoding=utf-8&allowM
io.seata.rm.AbstractRMHandler           : Branch Rollbacking: 10.32.3.56:8091:2049892225 2049892226 jdbc:mysql://localhost:3306/bank_b?useSSL=false&useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&serverTimez
i.s.rm.datasource.undo.UndoLogManager   : xid 10.32.3.56:8091:2049892225 branch 2049892226, undo_log deleted with GlobalFinished
io.seata.rm.AbstractRMHandler           : Branch Rollbacked result: PhaseTwo_Rollbacked
io.seata.core.rpc.netty.RmRpcClient     : RmRpcClient sendResponse xid=10.32.3.56:8091:2049892225,branchId=2049892226,branchStatus=PhaseTwo_Rollbacked,result code =Success,getMsg =null
```

bank_b中的seata日志表：

对象 undo_log @bank_b (l...

开始事务 备注 筛选 排序 导入 导出

id	branch_id	xid	rollback_info	log_status	log_created	log_modified	ext
10	2049892232	10.32.3.56:8091:2049892231	(BLOB)	0	2020-07-31 14:58:07	2020-07-31 14:58:07	(Null)