**1NgSql模块设计模板**

## 项目组

NgSql项目组

## 功能模块

报文收发、用户验证

## 负责人

晓栋、lvbomr

## 功能说明

**用户验证**

1. 验证用户请求数据合法性。

**报文收发**

1. 启动处理业务请求线程。
2. 处理前端请求。
3. 处理后端请求。

## 架构

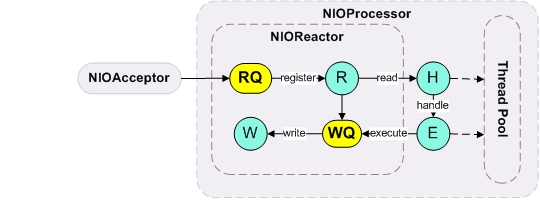


图1 前台通信

前台通信模式，主要应用于用户验证，接受和执行前端命令，返回查询最终结果集等。其中NIOAcceptor , NIOReactor中的R和W是在cobar启动的时候一起被初始化的线程。当客户端发起一个连接时，NIOAcceptor被唤醒，并生成一个FrontendConnection。

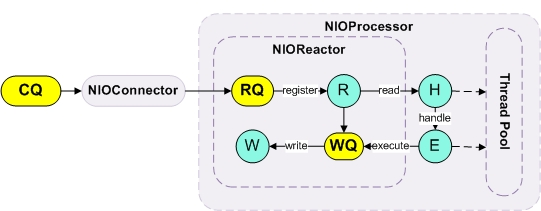


图2 后台通信

后台通信模式，主要应用于 MySql验证，各结点连接，返回各结点数据结果集等

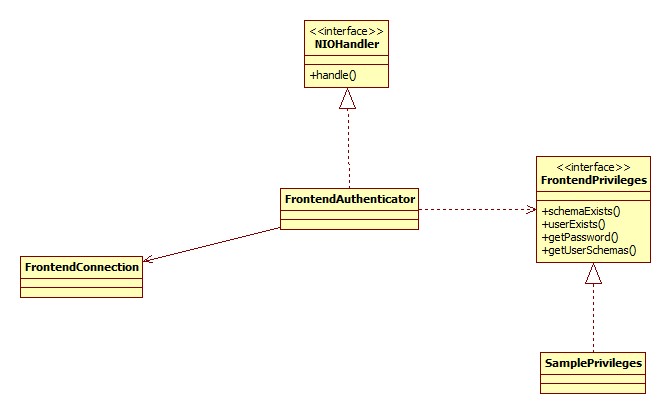
其中NIOConnector , NIOReactor中的R和W是在cobar启动的时候一起被初始化的线程。如果连接队列里有连接请求的话，NIOConnector被唤醒，并生成一个BackendConnection。

FrontendConnection和BackendConnection继承了AbstractConnection并实现了NIOConnection接口，是最基本的两个通信类。他们都有各自的主方法处理类，来自NIOHandler这个接口。FrontendConnection中的handler被初始化为FrontendAuthenticator，而BackendConnection里的handler是动态指定的。

但是NIOProcessor中的handler不同于FrontendConnection和BackendConnection中的handler。前者的功能范围更大。典型的，在FrontendConnection里面，handler这个对象方法的调用是通过NIOProcessor中的handler来完成的。图1和图2中的H和E就是NIOProcessor里面的handler和executor，他们都是NameableExecutor类型的。

**用户验证**

1. 验证请求信息，用户名、密码、schema合法性



**报文收发**

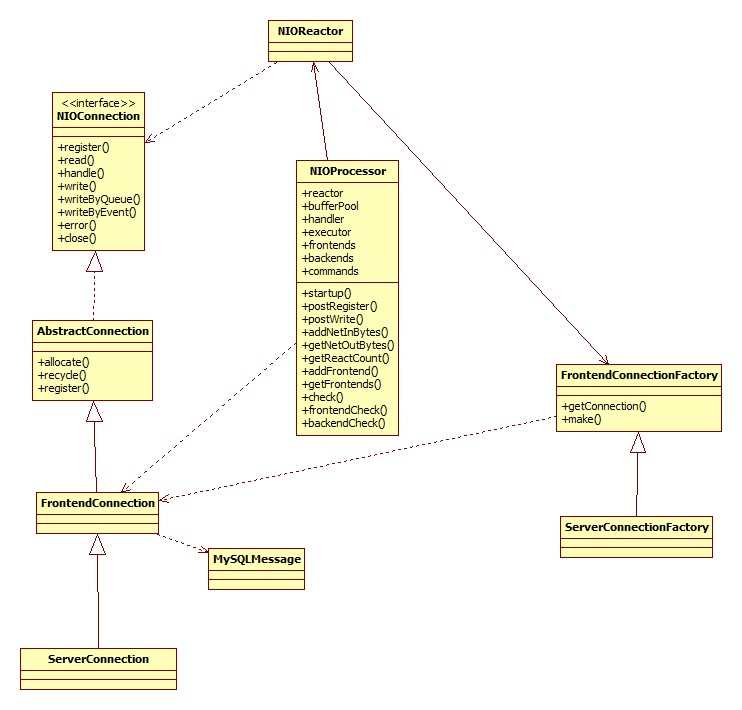
1. 层次说明

(1)NIOAcceptor处理业务请求线程。

(2)NIOProcessor业务请求处理。

(3)NIOReactor处理具体的业务请求，查询和插入分为读请求和写请求。

1. 类图



## 涉及的函数(选填)

用户验证

1. 前段权限验证

接口：“Frontend Privileges”

方法：**boolean** schemaExists(String schema);// 检查schema是否存在

**boolean** userExists(String user, String host);// 检查用户是否存在，并且可以使用host实行隔离策略

String getPassword(String user);// 提供用户的服务器端密码

Set<String> getUserSchemas(String user);// 提供有效的用户schema集合

2.报文收发

抽象类：前端链接工厂“FrontendConnectionFactory”

属性：**protected** **int** socketRecvBuffer = 8 \* 1024;//接收数据缓存大小

**protected** **int** socketSendBuffer = 16 \* 1024;//发送数据缓存大小

**protected** **int** packetHeaderSize = 4;//包头信息大小

**protected** **int** maxPacketSize = 16 \* 1024 \* 1024;//包大小

**protected** **int** writeQueueCapcity = 16;//写队列大小

**protected** **long** idleTimeout = 8 \* 3600 \* 1000L;//超时时间

**protected** String charset = "utf8";//设置字符编码

方法：**protected** **abstract** FrontendConnection getConnection(SocketChannel channel);//获取前端链接

**public** FrontendConnection make(SocketChannel channel) **throws** IOException;//创建前端链接

类：SampleConnectionFactory 继承类 FrontendConnectionFactory

方法：**protected** FrontendConnection getConnection(SocketChannel channel)

接口：NIOConnection

方法：**void** register(Selector selector) **throws** IOException;// 注册网络事件

**void** read() **throws** IOException;// 读取数据

**void** handle(**byte**[] data);// 处理数据

**void** write(ByteBuffer buffer);// 写出一块缓存数据

**void** writeByQueue() **throws** IOException;// 基于处理器队列的方式写数据

**void** writeByEvent() **throws** IOException;// 基于监听事件的方式写数据

**void** error(**int** errCode, Throwable t);// 发生错误

**boolean** close();//关闭连接

抽象类：AbstractConnection 实现接口NIOConnection

方法：**public** ByteBuffer allocate()//分配缓存

**public** **void** recycle(ByteBuffer buffer)//回收缓存

**public** **void** enableRead()//打开读事件

**public** **void** disableRead()//关闭读事件

**public** ByteBuffer checkWriteBuffer(ByteBuffer buffer, **int** capacity)// 检查WriteBuffer容量，不够则写出当前缓存块并申请新的缓存块

**public** ByteBuffer writeToBuffer(**byte**[] src, ByteBuffer buffer)// 把数据写到给定的缓存中，如果满了则提交当前缓存并申请新的缓存

**private** **void** enableWrite()//打开写事件

**private** **void** disableWrite()//关闭写事件

抽象类：FrontendConnection

方法：**public** **void** initDB(**byte**[] data)//初始化数据库信息

**public** **void** query(**byte**[] data)//查询信息

**public** **void** register(Selector selector) **throws** IOException//注册认证

**public** **void** ping()//ping端口

接口：FrontendQueryHandler

方法：**void** query(String sql);//执行查询语句

## 当前进度(选填)

1. 内容

## 下阶段开发计划(选填)

1. 内容