SphereEx | ShardingSphere

数据库协议入门与 Apache ShardingSphere Proxy 源码解析

吴伟杰

吴伟杰

- SphereEx 高级研发工程师
- Apache ShardingSphere Committer
- 参与 ShardingSphere Proxy 与 ElasticJob 的研发

目录

1. Apache ShardingSphere 简介

2. 如何进行数据库协议相关的开发

3. ShardingSphere Proxy 前端源码解析

4. 开源小白如何参与 ShardingSphere

Apache ShardingSphere 简介

Database Plus

构建多模数据库上层的标准和生态

连接

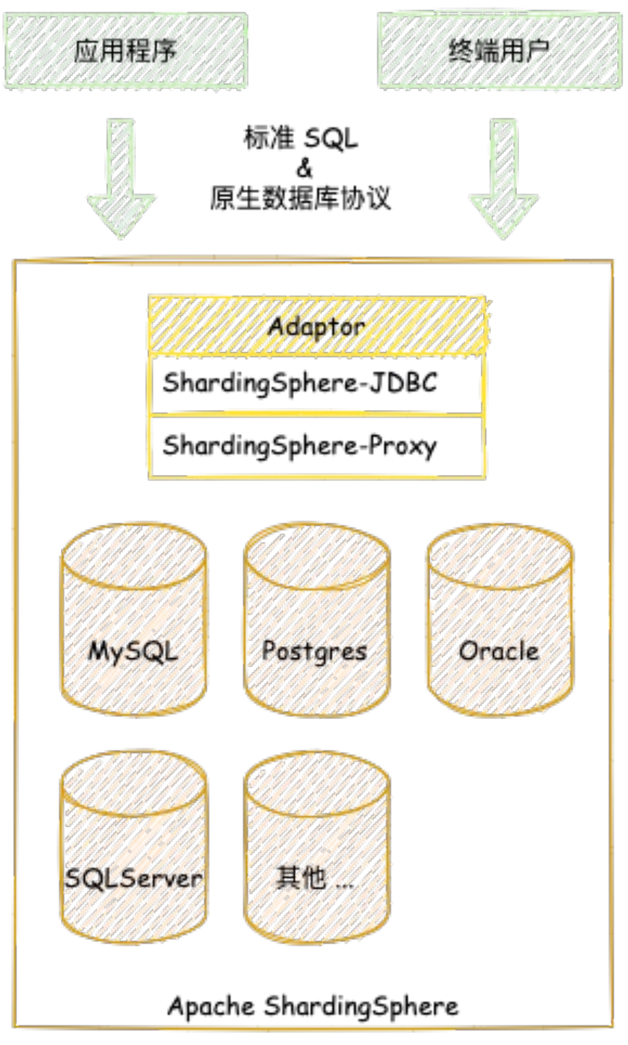
连接数据和应用，关注多模数据库之间的合作

增量

通过数据库入口流量的抓取提供透明化的增量功能

可插拔

微内核完全面向可插拔的三层架构体系设计



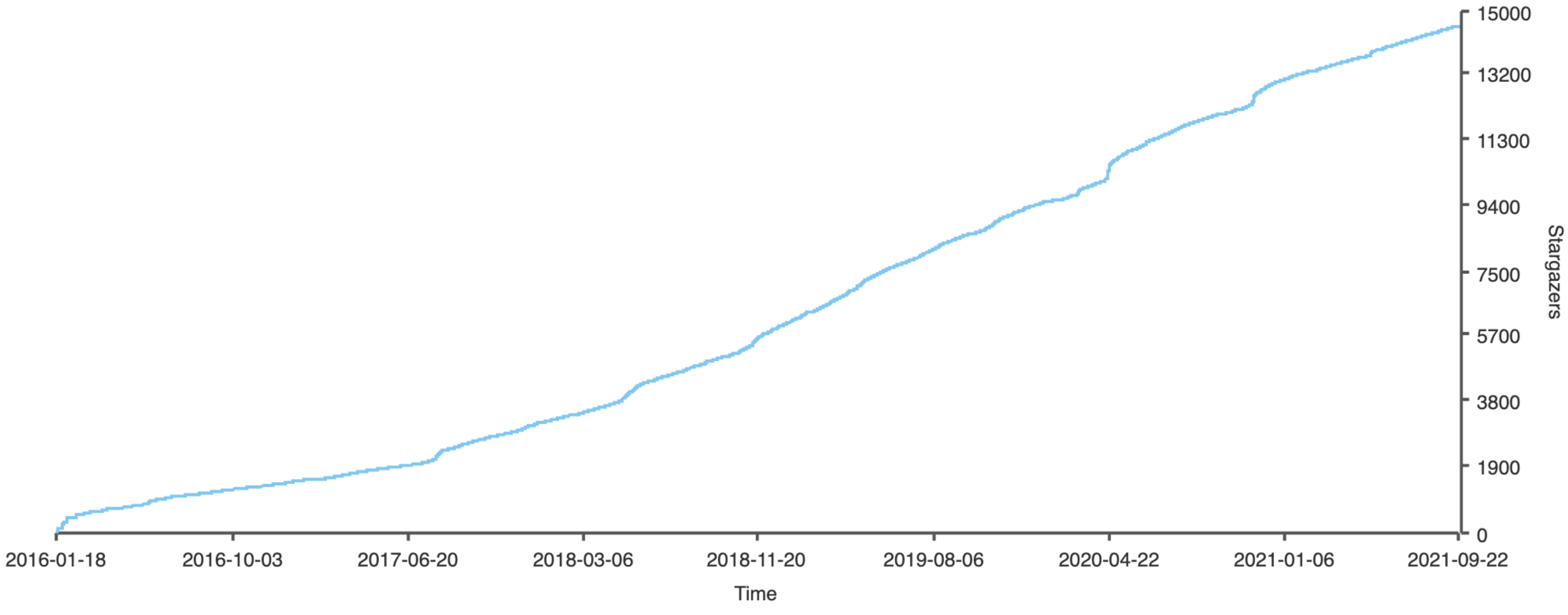
Apache ShardingSphere 简介

Star : 14.6k

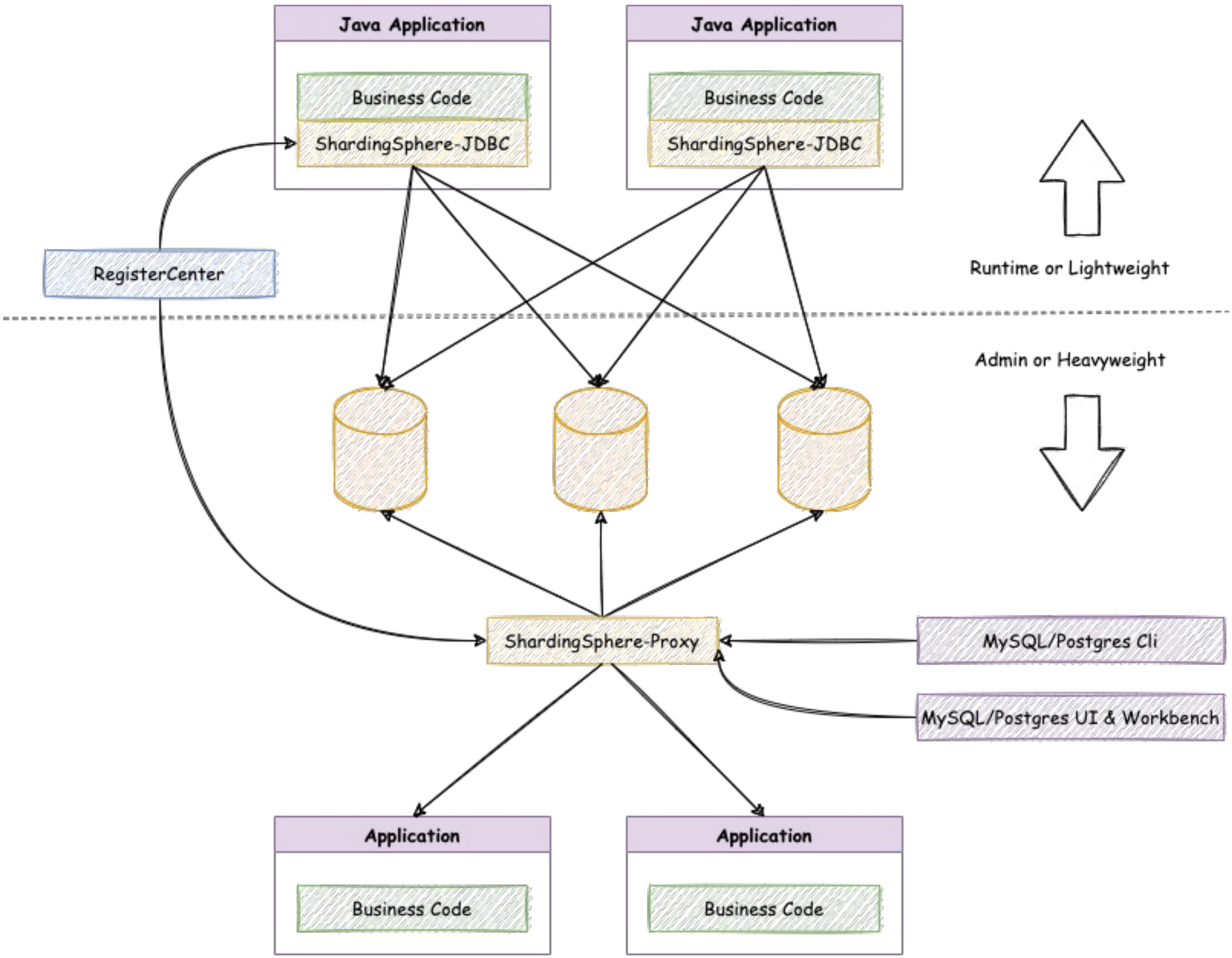
Fork : 5k

Contributors : 287

Awesome! apache/shardingsphere was created 5 years ago and now has **14561** stars.



Apache ShardingSphere 架构



SphereEx | ShardingSphere

Apache ShardingSphere 登记用户

登记使用 ShardingSphere 的公司： **170+**

Apache ShardingSphere 近期荣誉

SphereEx | ShardingSphere

如何进行数据库协议相关的开发

善用 Wireshark、tcpdump 等工具

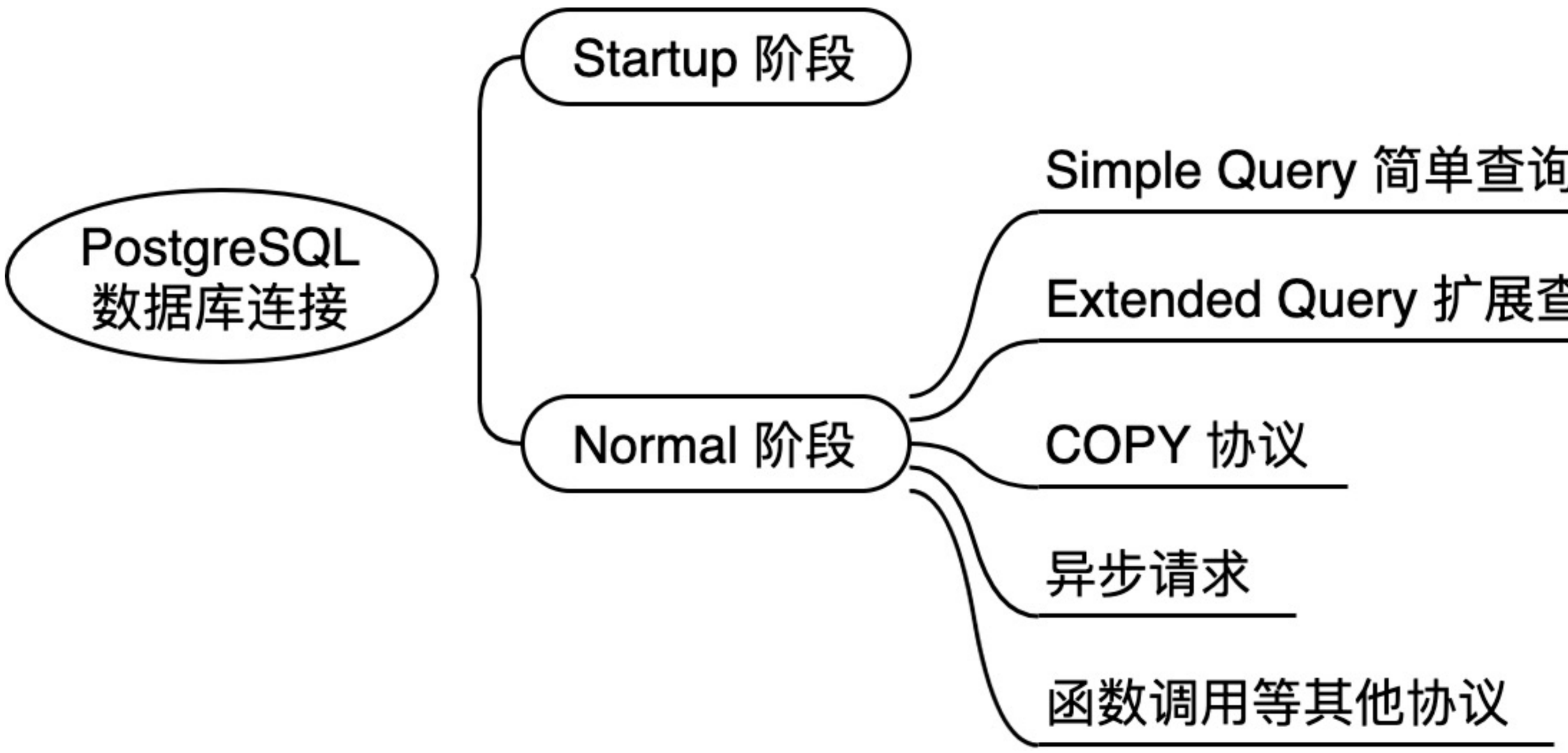
pgsql and tcp.port == 3307

No.	Time	Source	Destination	Src Port	Dst Port	Protocol	Length	Info
100	1.903669	127.0.0.1	127.0.0.1	57306	3307	PGSQL	64	>
106	1.920004	127.0.0.1	127.0.0.1	3307	57306	PGSQL	69	<R
108	1.930333	127.0.0.1	127.0.0.1	57306	3307	PGSQL	97	>p
110	1.930945	127.0.0.1	127.0.0.1	3307	57306	PGSQL	180	<R/S/S/S/Z
112	1.957339	127.0.0.1	127.0.0.1	57306	3307	PGSQL	119	>P/B/E/S
124	1.960070	127.0.0.1	127.0.0.1	3307	57306	PGSQL	81	<1/2/C/Z
126	1.960331	127.0.0.1	127.0.0.1	57306	3307	PGSQL	140	>P/B/E/S
132	1.961169	127.0.0.1	127.0.0.1	3307	57306	PGSQL	81	<1/2/C/Z

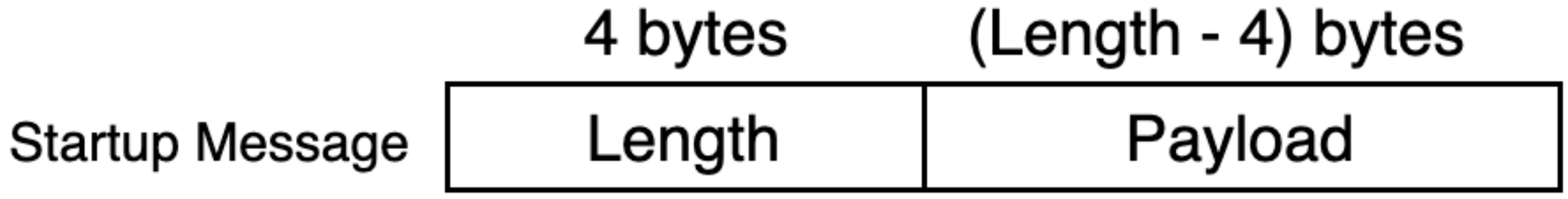
```
PostgreSQL
    Type: Parse
    Length: 58
    Statement:
    Query: select * from t_b where a = $1 and b = $2
    Parameters: 2
        Type OID: 0
        Type OID: 0
PostgreSQL
    Type: Bind
    Length: 33
```

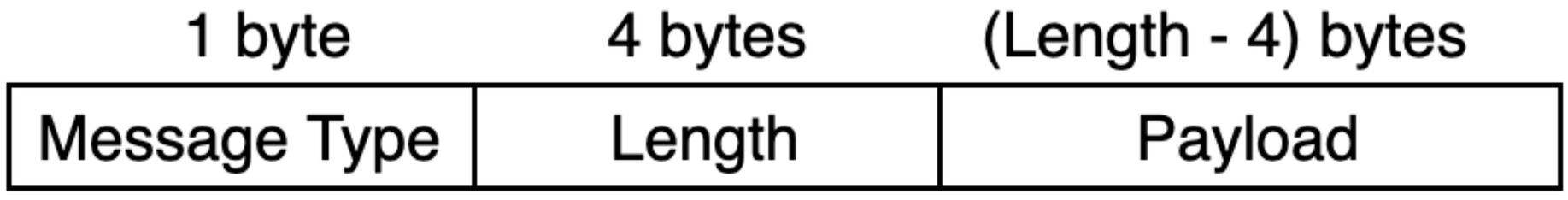
```
0000  02 00 00 00 45 00 00 a7  00 00 40 00 40 06 00 00   ····E··· ··@·@···
0010  7f 00 00 01 7f 00 00 01  df da 0c eb 00 d9 8d cd   ···· ···· ···· ····
0020  56 5d 7a f1 80 18 18 e6  fe 9b 00 00 01 01 08 0a   V]z· ···· ···· ····
0030  56 b5 4a c2 ca 6c 45 0b  50 00 00 00 3a 00 73 65   V·J· ·lE· P···:·se
0040  6c 65 63 74 20 2a 20 66  72 6f 6d 20 74 5f 62 20   lect * f rom t_b
0050  77 68 65 72 65 20 61 20  3d 20 24 31 20 61 6e 64   where a  = $1 and
0060  20 62 20 3d 20 24 32 20  00 00 02 00 00 00 00 00    b = $2  ·····
0070  00 00 00 42 00 00 00 21  00 00 00 02 00 00 00 00   ···B···! ····
0080  00 02 00 00 00 04 74 72  75 65 00 00 00 05 66 61   ····tr ue····fa
0090  6c 73 65 00 00 44 00 00  00 06 50 00 45 00 00 00   lse··D·· ··P·E···
00a0  09 00 00 00 00 00 53 00  00 00 04                 ····S· ···
```

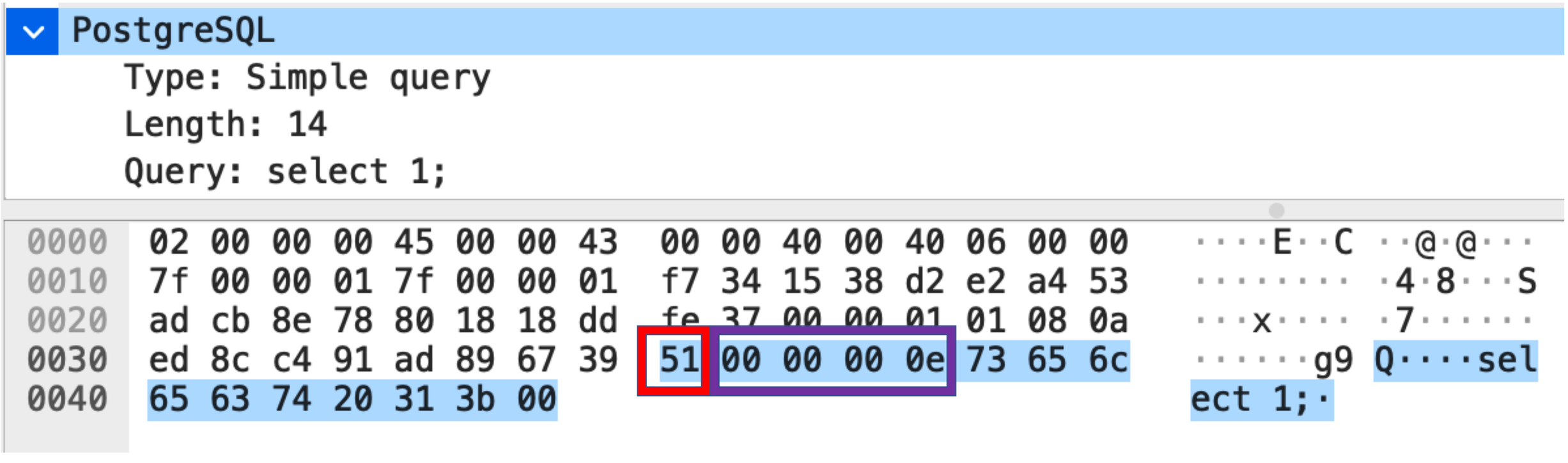
PostgreSQL 协议介绍



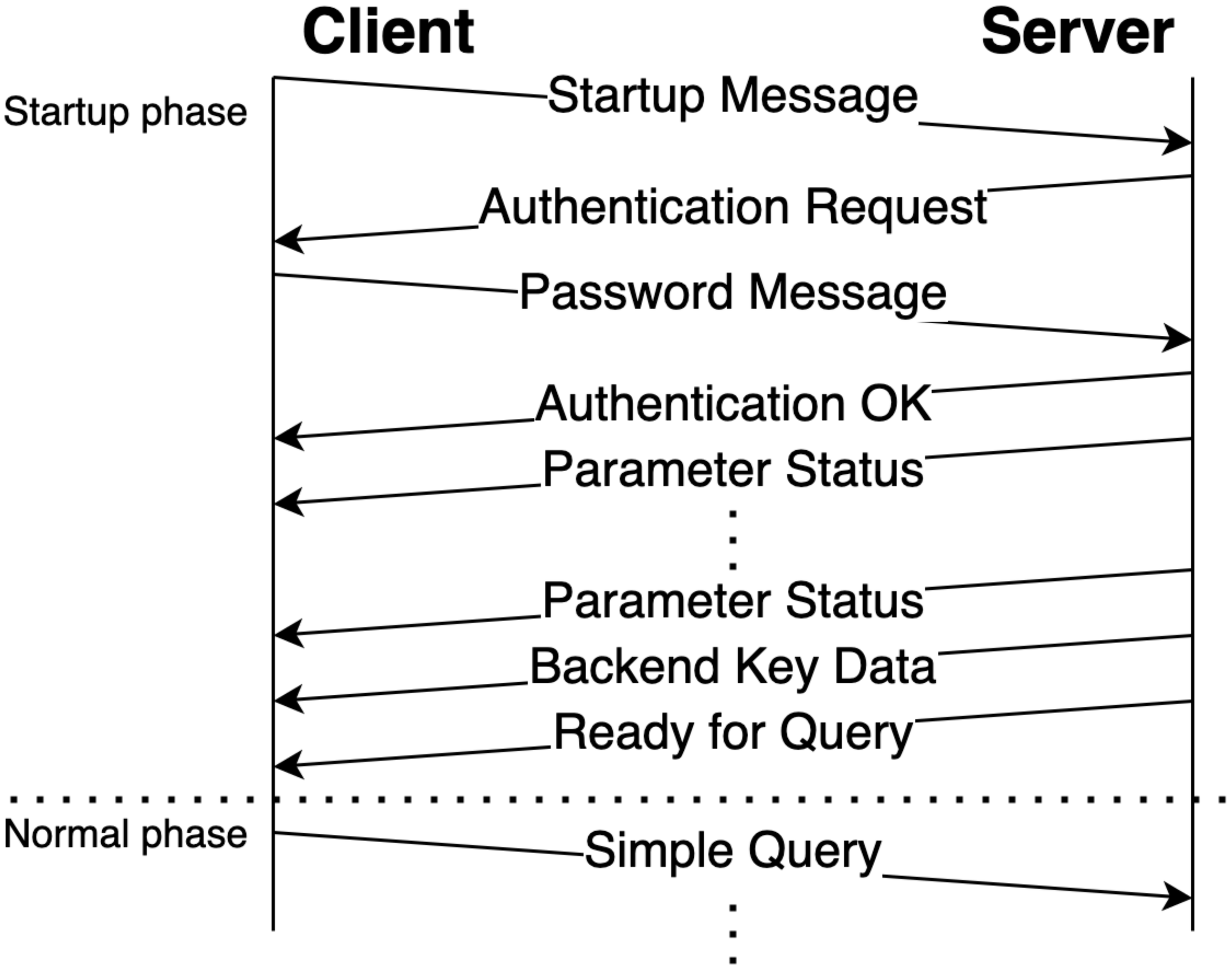
PostgreSQL 数据包格式





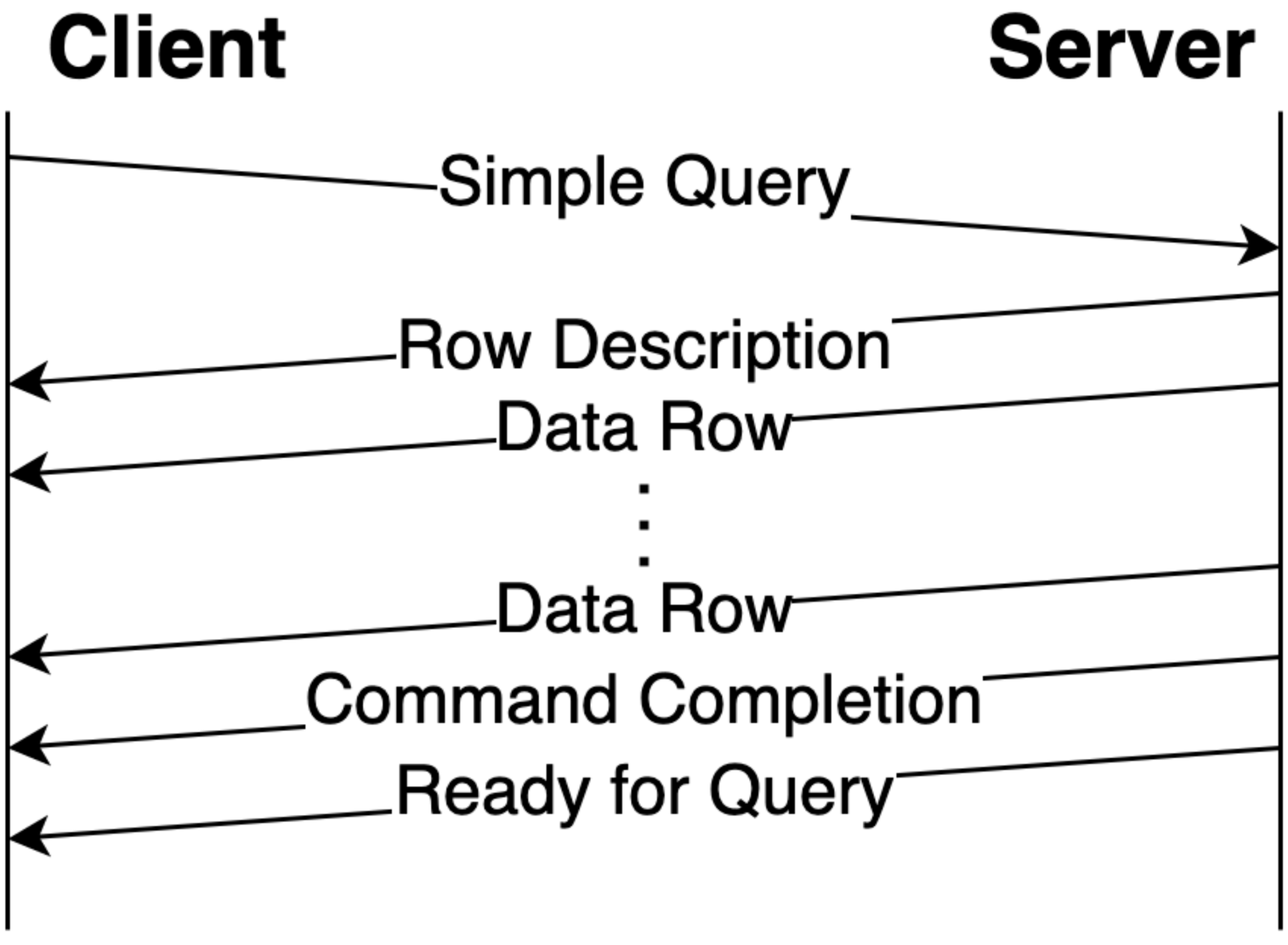


PostgreSQL 连接 Startup 阶段



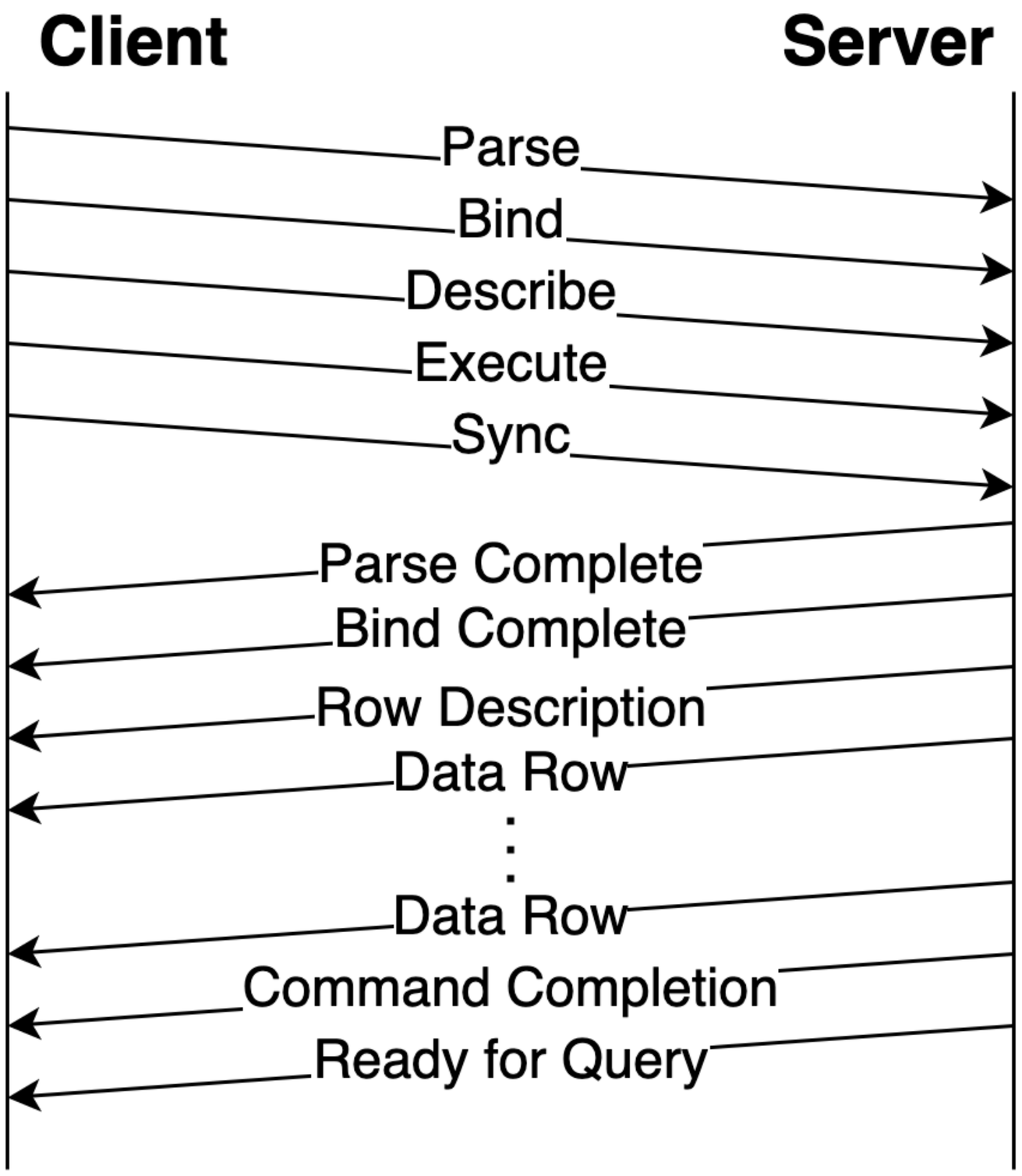
PostgreSQL 简单查询

```
biz_order=# select order_id, create_time from t_order limit 3;
       order_id       |        create_time
----------------------+----------------------------
 614158771245527040 | 2021-09-01 10:32:44.893
 614127204259311616 | 2021-09-01 07:55:18.471
 608625285797490688 | 2021-08-31 18:17:42.173
(3 rows)
```



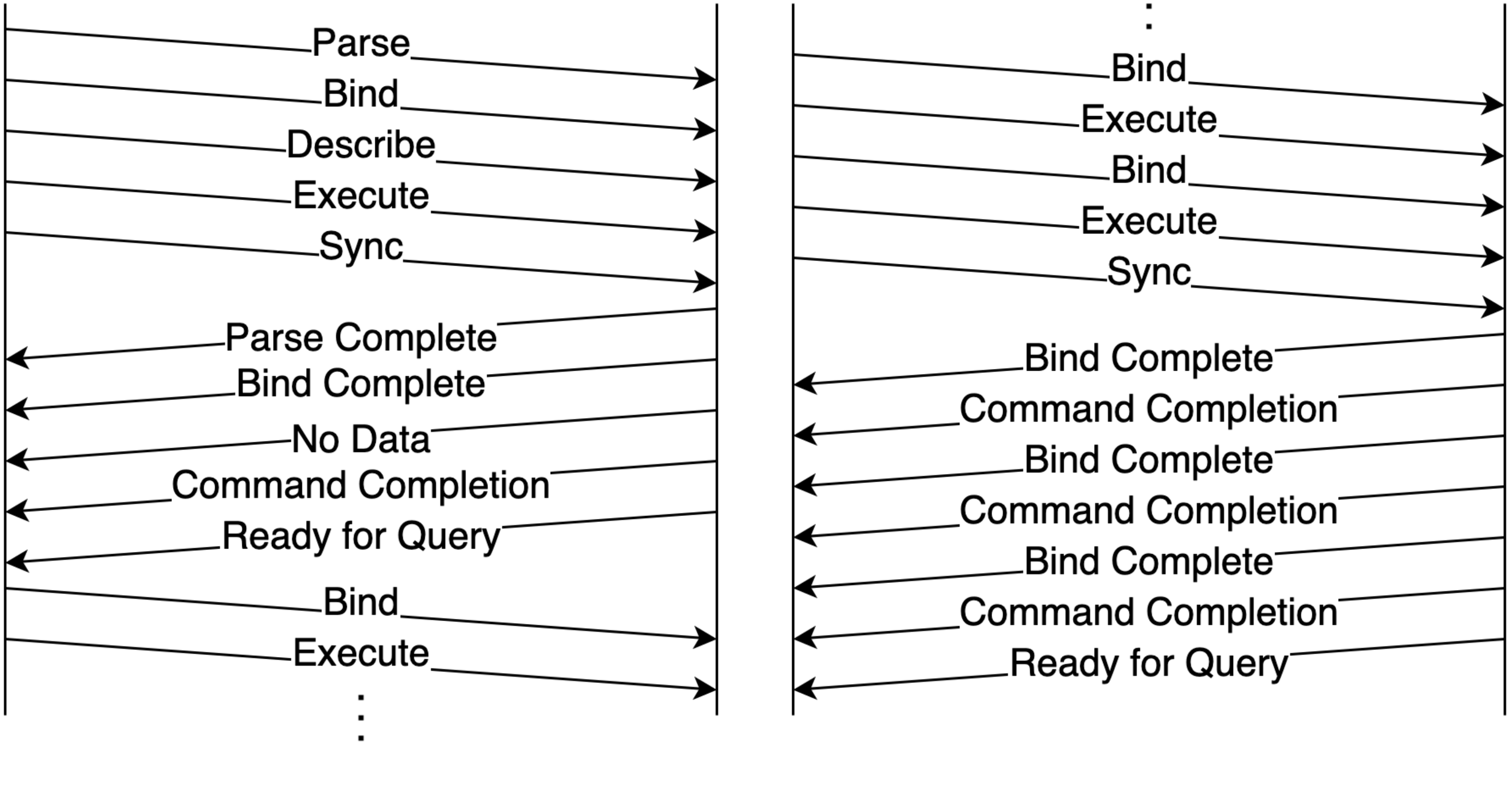
PostgreSQL 扩展查询

```
String sql = "select order_id, create_time from t_order where order_id = ?";
try (PreparedStatement ps = connection.prepareStatement(sql)) {
    ps.setLong( parameterIndex: 1,  x: 614127204259311616L);
    ps.executeQuery();
}
```



PostgreSQL 扩展查询

```java
try (PreparedStatement preparedStatement = connection.prepareStatement( sql: "insert into t_b values (?, ?, ?)")) {
    for (int i = 0; i < 5; i++) {
        preparedStatement.setLong( parameterIndex: 1, System.nanoTime());
        preparedStatement.setBoolean( parameterIndex: 2,  x: true);
        preparedStatement.setBoolean( parameterIndex: 3,  x: false);
        preparedStatement.addBatch();
    }
    preparedStatement.executeBatch();
}
```



示例：解析 Startup Message 信息

```java
public static void main(String[] args) throws Exception {
    ServerSocket serverSocket = new ServerSocket( port: 55432);
    Socket clientSocket = serverSocket.accept();
    DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream());
    DataInputStream in = new DataInputStream(new BufferedInputStream(clientSocket.getInputStream()));
    for (; ; ) {
        in.mark( readlimit: 4); // 记录数据流当前读取的位置
        int messageLength = in.readInt(); // 读取数据包声明长度
        System.out.printf("数据包声明长度: %d\n", messageLength);
        if (in.available() < messageLength - 4) { // TCP 接收到的数据少于数据包声明的长度
            in.reset(); // 重置数据流读取的位置
            continue;
        }
        if (messageLength == 8 && in.readInt() == 0x4D2162F) { // 处理 SSL request 请求
            out.writeByte( v: 'N'); // 不使用 SSL
            continue;
        }
        // 根据 PostgreSQL 协议文档 Startup Message 的构成读取数据
        int protocolMajorVersion = in.readUnsignedShort();
        int protocolMinorVersion = in.readUnsignedShort();
        System.out.printf("数据库协议版本: %d.%d\n", protocolMajorVersion, protocolMinorVersion);
        byte[] parameters = new byte[messageLength - 4 - 2 - 2]; // 读取数据包剩余部分
        in.read(parameters);
        String[] parameterPairs = new String(parameters).split( regex: "\0"); // pg 采用 C 风格的字符串
        for (int i = 0; i < parameterPairs.length - 1; i += 2) {
            System.out.printf("参数 %s: %s\n", parameterPairs[i], parameterPairs[i + 1]);
        }
        break;
    }
}
```

根据客户端源码编写 Packet 解析逻辑

```
1811        pgStream.sendInteger4((int) encodedSize); // Message size
   1        pgStream.sendInteger4(batchmum); // batchCount
   2        if (encodedPortalName != null) {
   3            pgStream.send(encodedPortalName); // Destination portal name.
   4        }
   5        pgStream.sendChar( val: 0); // End of portal name.
   6        if (encodedStatementName != null) {
   7            pgStream.send(encodedStatementName); // Source statement name.
   8        }
   9        pgStream.sendChar( val: 0); // End of statement name.
  10
  11        pgStream.sendInteger2(params.getParameterCount()); // # of parameter format codes
  12        for (int i = 1; i <= params.getParameterCount(); ++i) {
  13            pgStream.sendInteger2(params.isBinary(i) ? 1 : 0); // Parameter format code
```

```
3  @    public OpenGaussComBatchBindPacket(final PostgreSQLPacketPayload payload,
4           this.payload = payload;
5           payload.readInt4();
6           payload.readInt4();
7           payload.readStringNul();
8           statementId = payload.readStringNul();
9           int parameterFormatCount = payload.readInt2();
10          parameterFormats = new ArrayList<>(parameterFormatCount);
11          for (int i = 0; i < parameterFormatCount; i++) {
12              parameterFormats.add(payload.readInt2());
13          }
```

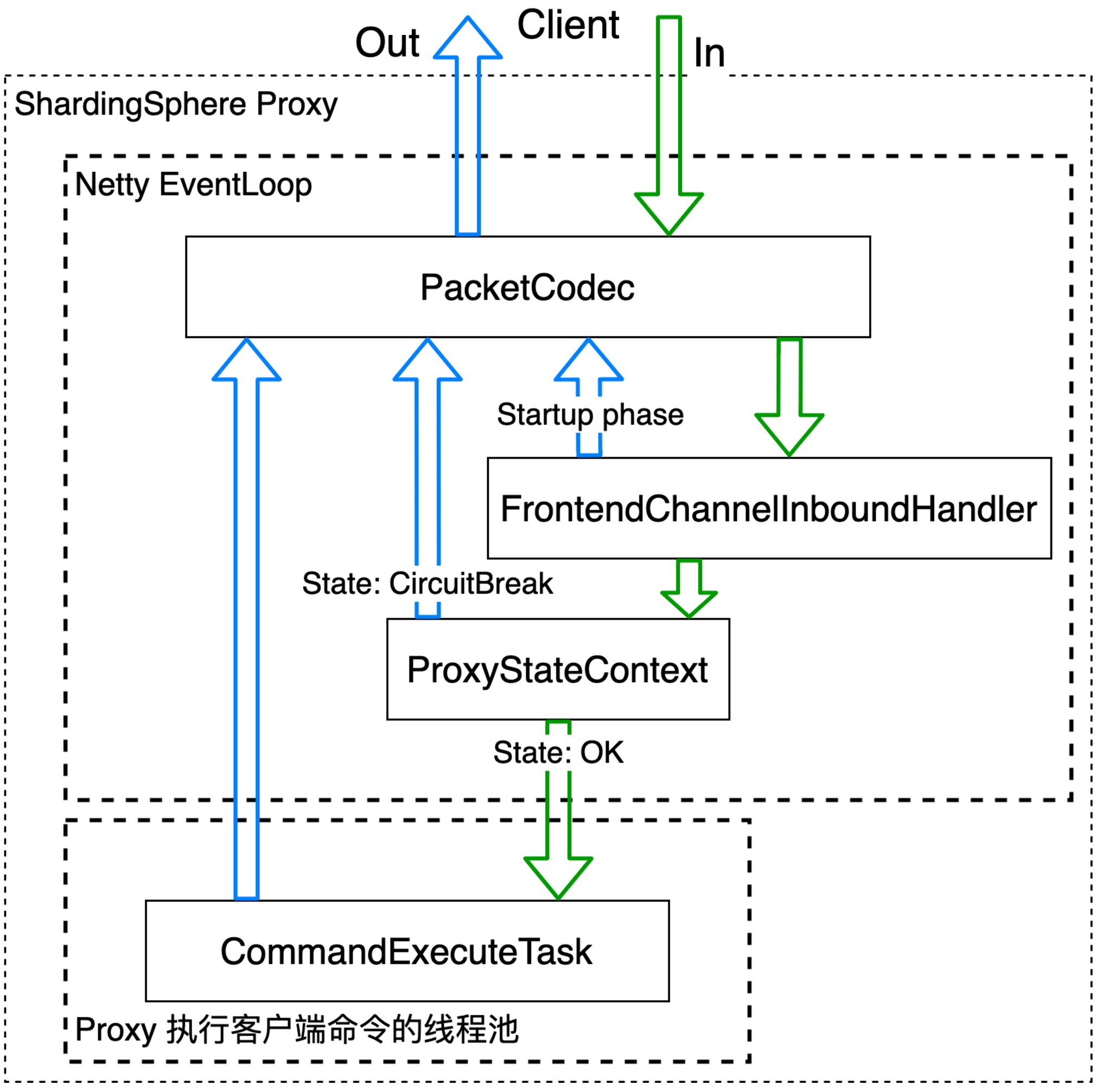
SphereEx | ShardingSphere

ShardingSphere Proxy 前端源码解析

ShardingSphere Proxy 项目结构

```
shardingsphere
├── shardingsphere-db-protocol
│   ├── shardingsphere-db-protocol-core
│   ├── shardingsphere-db-protocol-mysql
│   ├── shardingsphere-db-protocol-opengauss
│   ├── shardingsphere-db-protocol-postgresql
├── shardingsphere-proxy
│   ├── shardingsphere-proxy-backend
│   ├── shardingsphere-proxy-bootstrap
│   ├── shardingsphere-proxy-common
│   ├── shardingsphere-proxy-frontend
│   │   ├── shardingsphere-proxy-frontend-core
│   │   ├── shardingsphere-proxy-frontend-mysql
│   │   ├── shardingsphere-proxy-frontend-opengauss
│   │   ├── shardingsphere-proxy-frontend-postgresql
│   │   ├── shardingsphere-proxy-frontend-spi
```

ShardingSphere Proxy 前端架构

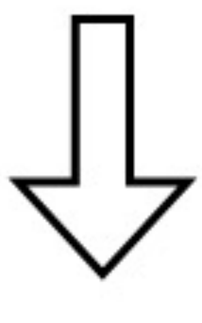


PacketCodec — 拆包/装包

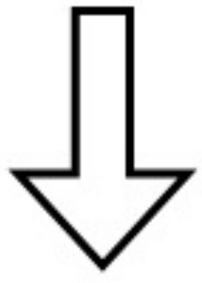
```
public final class PacketCodec extends ByteToMessageCodec<DatabasePacket<?>> {

    private final DatabasePacketCodecEngine databasePacketCodecEngine;

    @Override
    protected void decode(final ChannelHandlerContext context, final ByteBuf in,
                          final List<Object> out) {
        int readableBytes = in.readableBytes();
        if (!databasePacketCodecEngine.isValidHeader(readableBytes)) {
            return;
        }
        if (log.isDebugEnabled()) {...}
        databasePacketCodecEngine.decode(context, in, out);
    }

    @SuppressWarnings("unchecked")
    @Override
    protected void encode(final ChannelHandlerContext context,
                          final DatabasePacket<?> message, final ByteBuf out) {
        databasePacketCodecEngine.encode(context, message, out);
        if (log.isDebugEnabled()) {...}
    }
}
```

PacketCodec — PostgreSQL 拆包

```
0000  02 00 00 00 45 00 00 77  00 00 40 00 40 06 00 00   ····E··w ··@·@···
0010  7f 00 00 01 7f 00 00 01  df bb 15 38 85 a3 91 1b   ········ ···8····
0020  7f ca 9b ba 80 18 13 4a  fe 6b 00 00 01 01 08 0a   ·······J ·k······
0030  45 09 a2 ae 3d 15 7b 0e  50 00 00 00 1f 00 73 65   E···=·{· P·····se
0040  6c 65 63 74 20 63 75 72  72 65 6e 74 5f 73 63 68   lect cur rent_sch
0050  65 6d 61 28 29 00 00 00  42 00 00 00 0c 00 00 00   ema()··· B·······
0060  00 00 00 00 00 44 00 00  00 06 50 00 45 00 00 00   ·····D·· ··P·E···
0070  09 00 00 00 00 00 53 00  00 00 04                  ·····S·· ···
```



PacketCodec



Parse | Bind | Describe | Execute | Sync

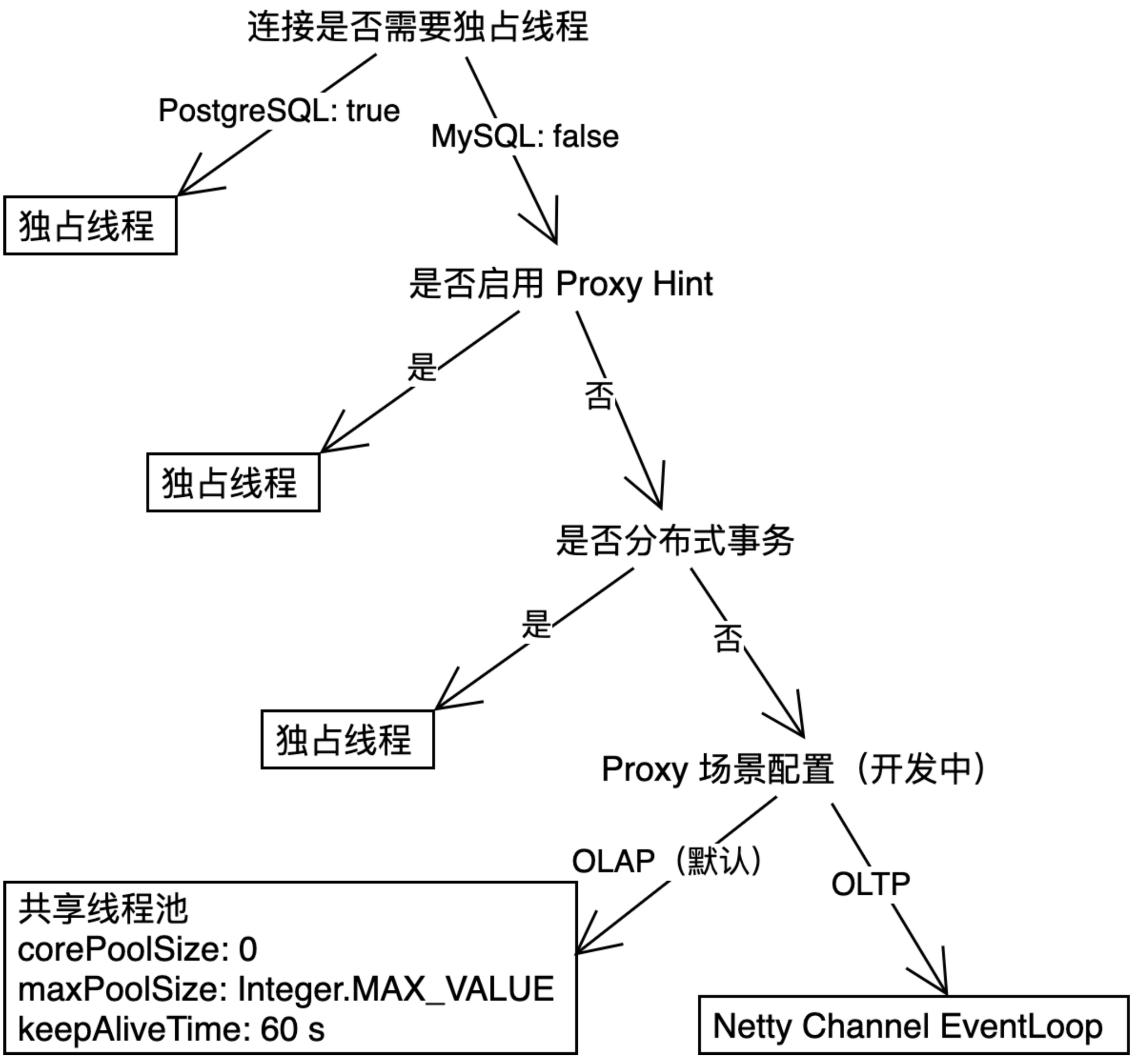
PostgreSQL 拆包实现逻辑

```java
public final class PostgreSQLPacketCodecEngine implements DatabasePacketCodecEngine<PostgreSQLPacket> {

    private static final int MESSAGE_TYPE_LENGTH = 1;

    private static final int PAYLOAD_LENGTH = 4;

    @Override
    public boolean isValidHeader(final int readableBytes) {
        return readableBytes >= MESSAGE_TYPE_LENGTH + PAYLOAD_LENGTH;
    }

    @Override
    public void decode(final ChannelHandlerContext context, final ByteBuf in, final List<Object> out) {
        while (isValidHeader(in.readableBytes())) {
            int messageTypeLength = 0;
            if ('\0' == in.markReaderIndex().readByte()) {
                in.resetReaderIndex();
            } else {
                messageTypeLength = MESSAGE_TYPE_LENGTH;
            }
            int remainPayloadLength = in.readInt() - PAYLOAD_LENGTH;
            if (in.readableBytes() < remainPayloadLength) {
                in.resetReaderIndex();
                return;
            }
            in.resetReaderIndex();
            out.add(in.readRetainedSlice(messageTypeLength + PAYLOAD_LENGTH + remainPayloadLength));
        }
    }
```

FrontendChannelInboundHandler — 身份验证、资源创建/释放

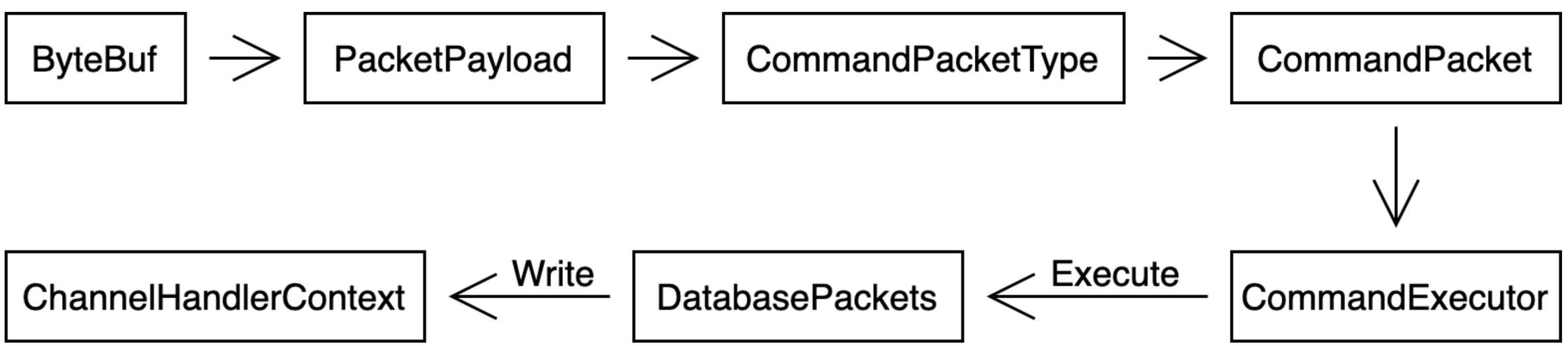
```
public final class FrontendChannelInboundHandler extends ChannelInboundHandlerAdapter {
...
    private volatile boolean authenticated;
...
    @Override
    public void channelActive(final ChannelHandlerContext context) {...}

    @Override
    public void channelRead(final ChannelHandlerContext context, final Object message) {
        if (!authenticated) {
            authenticated = authenticate(context, (ByteBuf) message);
            return;
        }
        ProxyStateContext.execute(context, message, databaseProtocolFrontendEngine, backendConnection);
    }

    private boolean authenticate(final ChannelHandlerContext context, final ByteBuf message) {...}

    @Override
    public void channelInactive(final ChannelHandlerContext context) {...}

    private void closeAllResources() {...}

    @Override
    public void channelWritabilityChanged(final ChannelHandlerContext context) {...}
```

OKProxyState — 命令执行线程选择

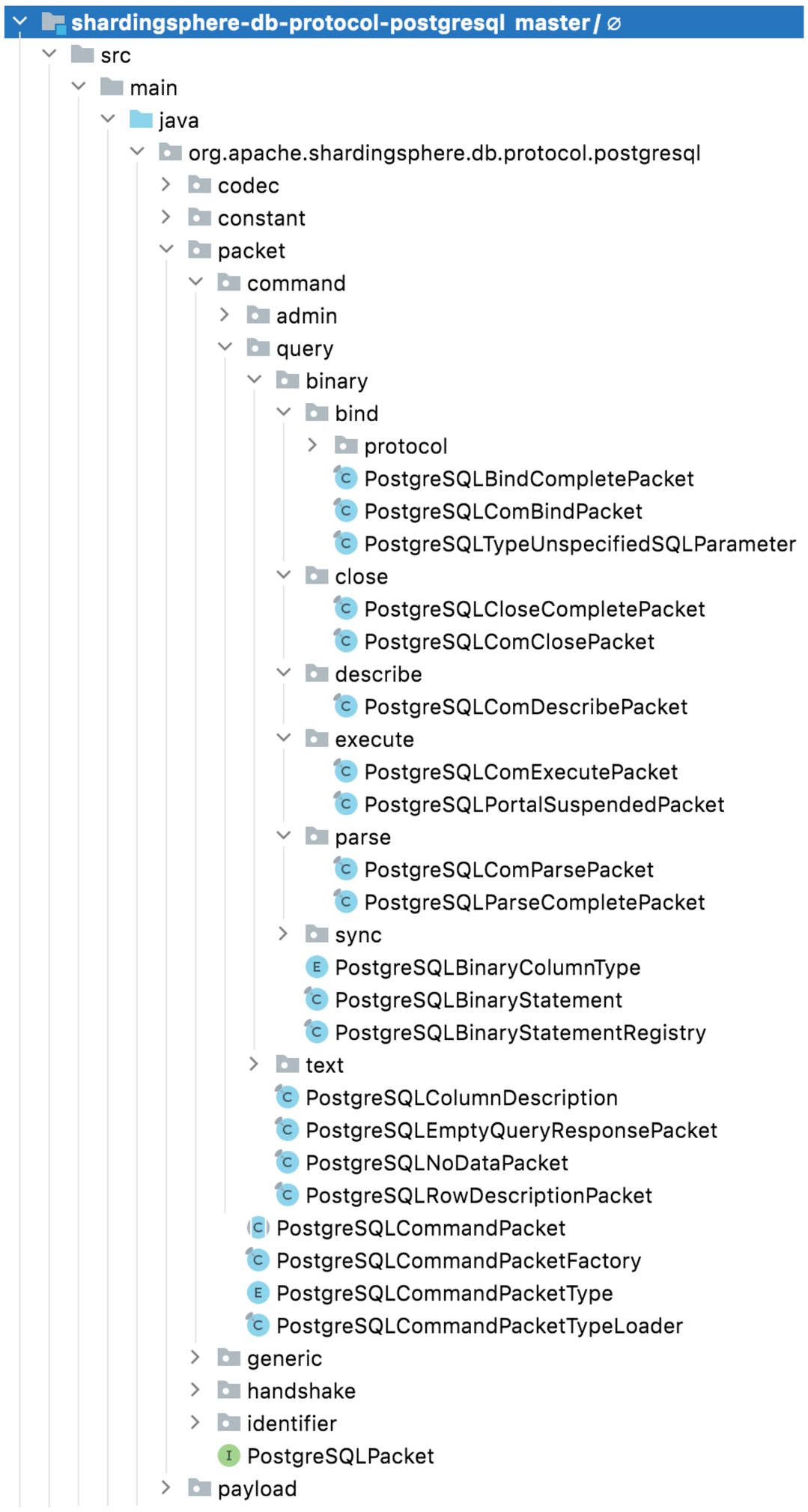


CommandExecuteTask 流程

```
public final class CommandExecutorTask implements Runnable {
...
    @Override
    public void run() {...}

    private boolean executeCommand(final ChannelHandlerContext context, final PacketPayload payload,
                                   final BackendConnection backendConnection) throws SQLException {
        CommandExecuteEngine commandExecuteEngine = databaseProtocolFrontendEngine.getCommandExecuteEngine();
        CommandPacketType type = commandExecuteEngine.getCommandPacketType(payload);
        CommandPacket commandPacket = commandExecuteEngine.getCommandPacket(payload, type, backendConnection);
        CommandExecutor commandExecutor = commandExecuteEngine.getCommandExecutor(type, commandPacket, backendConnection);
        try {
            Collection<DatabasePacket<?>> responsePackets = commandExecutor.execute();
...
            responsePackets.forEach(context::write);
            if (commandExecutor instanceof QueryCommandExecutor) {...}
        } finally {...}
        return databaseProtocolFrontendEngine.getFrontendContext().isFlushForPerCommandPacket();
    }
```



PostgreSQL 协议实现代码结构



SphereEx | ShardingSphere

开源小白如何参与 ShardingSphere

为什么要参与开源社区

为了 ~~免费的 JetBrains 全家桶、apache 邮箱~~

提升个人能力、扩大技术影响力

开源小白如何参与 ShardingSphere

- 文档勘误
- Examples 优化
- 补充单元测试用例
- 代码小重构/优化
- 其他 Good first issues
- 提出新的 issue

ShardingSphere 各模块相关知识

- **Infra:** 可插拔架构设计、SPI 机制、ShardingSphere 核心流程
- **Optimize:** Apache Calcite
- **Proxy:** Netty、数据库协议（MySQL、PostgreSQL）
- **JDBC:** JDBC 接口、Spring namespace、Spring Boot Starter
- **SQL Parser:** ANTLR
- **Transaction:** XA、BASE
- **Mode:** 分布式治理设计、ZooKeeper、etcd
- **Scaling:** Binlog 解析、分布式作业调度
- **Agent:** Java 探针设计、Byte Buddy、可观察性
- **Features:** 数据分片、读写分离、数据库高可用、加密、影子库等
- **Test:** 集成测试

SphereEx | ShardingSphere

欢迎加入 SphereEx

我们是这样的一家公司

- 浓厚的工程师文化
- 原生开源基因
- 国际化战略
- 为兴趣和梦想工作
- 团队快速扩张
- 顶级资本投入

我们的待招职位

- java 开发工程师
- 自动化测试专家
- 数据库中间件交付工程师
- 数据库中间件开发工程师
- 等多个职位在招聘

简历请发至: hr@sphere-ex.com

Thanks

商务合作

技术交流