

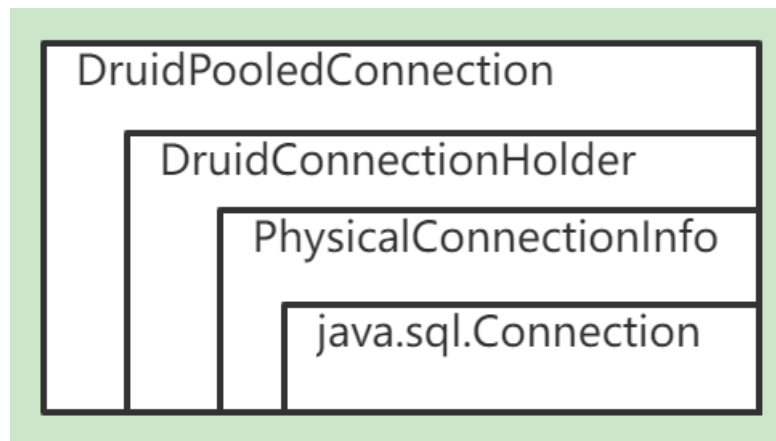
## 研究问题

为什么要有 `DruidConnectionHolder`？

完成对“池化技术”的部分了解后，选择了研究 `DruidConnectionHolder` 的业务意义，看能否从中学习到一些设计模式或者“如何扩展、增强”等知识。

因此在研究holder的前，我划分好要了解的内容：**1.**类的组成结构（指的是所在包、接口、继承等）。**2.**类本身有哪些变量？**3.**该类与哪些类相关联？

带着以上三个内容，我从前面章节中提到的“connection的组成”开始研究，即下图所示结构：

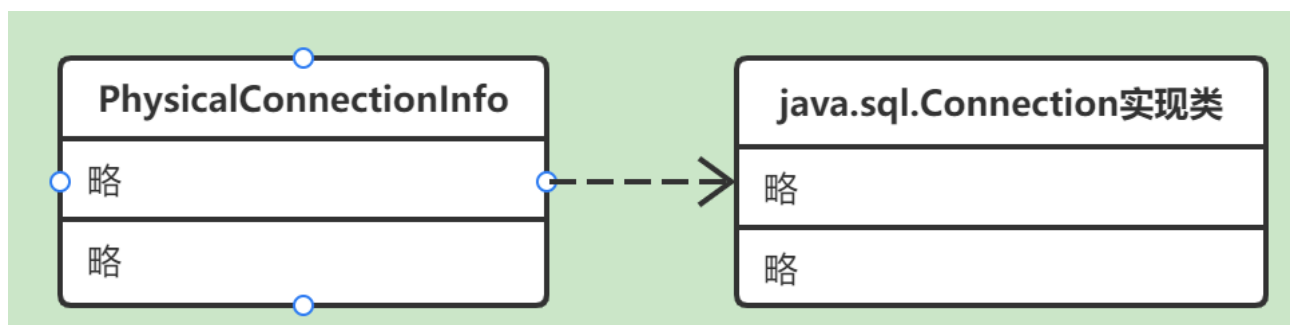


## java.sql.Connection

JDBC相关接口，具体内容不展开讨论

## PhysicalConnectionInfo

`PhysicalConnectionInfo` 中持有 `java.sql.Connection` 实现类的实例，关系如下图



# 成员变量

这里指出我所清楚的几个变量，`connectStartNanos`、`connectedNanos`、`initedNanos`、`validatedNanos`。这四个变量分别的含义是“建立连接前的系统当前时间”、“建立连接后的系统当前时间”、“初始化连接后的系统当前时间”、“检验连接后的系统当前时间”。形如用 `connectedNanos` 减去 `connectStartNanos`，便可以知道与数据库建立连接时所消耗的时间。其他变量同理。具体代码在 `DruidAbstractDataSource.createPhysicalConnection()` 中。

# 方法

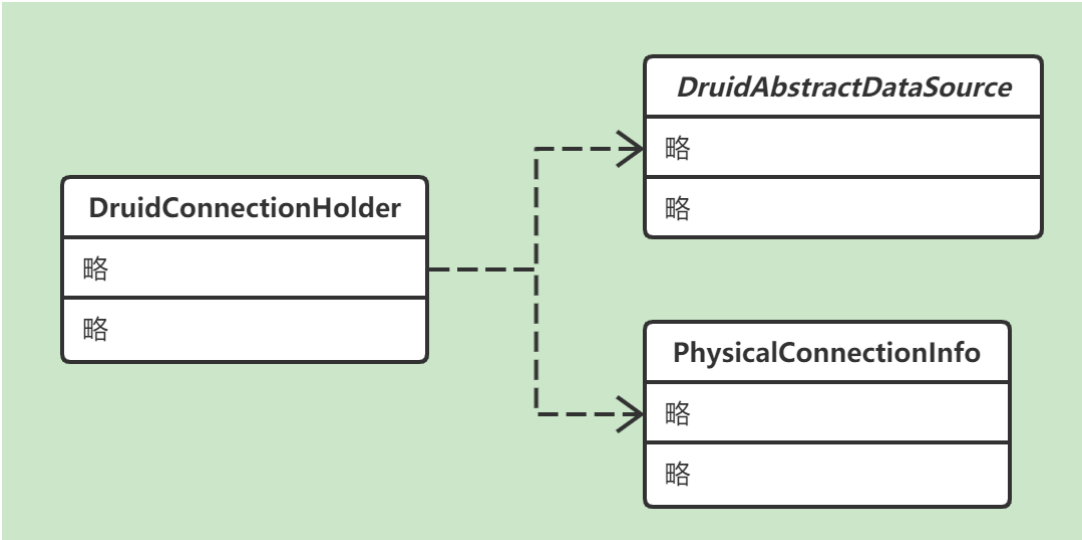
`DruidAbstractDataSource.initPhysicalConnection(Connection, Map<String, Object>, Map<String, Object>)` 和 `DruidAbstractDataSource.validateConnection(Connection)`。若在创建连接池时配置了创建物理连接的初始化方法和检验业务，这两个方法分别会执行对用的业务。

# 意义

不难看出 `PhysicalConnectionInfo` 存在的目的是扩展JDBC的 `Connection` 的功能。通过关联 `java.sql.Connection` 实现类，增加了“检验物理连接”、“初始化物理连接”的业务方法。因此我对 `PhysicalConnectionInfo` 的理解是：“存放与物理连接相关的内容”。

# DruidConnectionHolder

holder持有者 `PhysicalConnectionInfo` 和 `DruidAbstractDataSource` 的实例引用，关系如下图



## 成员变量

目前了解到的有 `lastActiveTimeMillis`、`lastKeepTimeMillis`、`statementPool`。这三个变量分别的作用是“最后活跃时间”、“最后保活时间（存疑）”、“当前连接缓存的执行语句”

## 方法

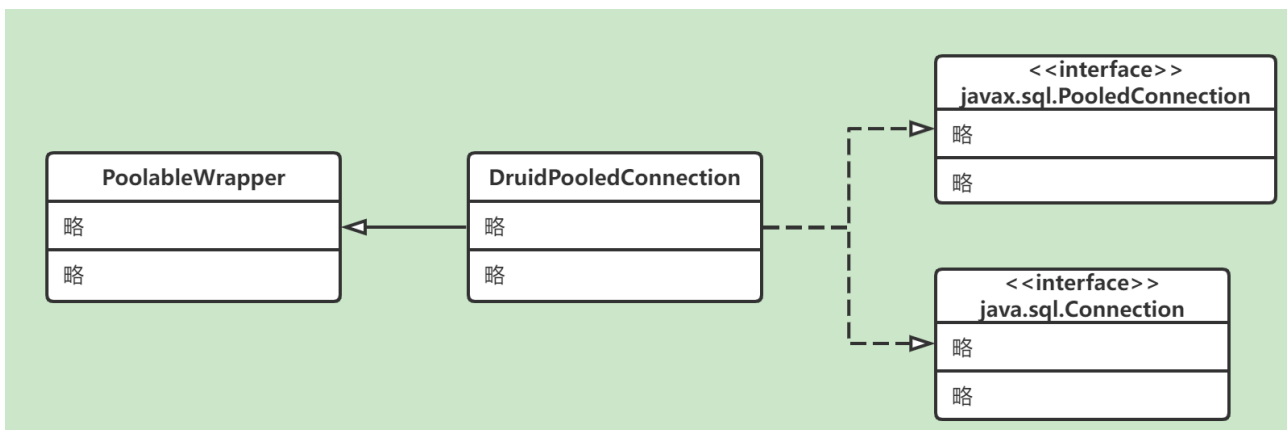
通过了解`toString()`展现的消息来初步了解上面提到的成员变量。

## 意义

其实在涉及到的成员变量中，就能够看出`holder`主要是负责保存跟执行SQL相关的信息。换句话说来说，`holder`专门用于跟踪应用程序业务信息。

## DruidPooledConnection

先给出这个类的关系图



根据UML图的关系，`DruidPooledConnection`既然实现了`javax.sql.PooledConnection`这个接口，说明`DruidPooledConnection`拥有了`javax.sql.PooledConnection`的特性。因此我选择分析`javax.sql.PooledConnection`这个接口。

## javax.sql.PooledConnection

引用openjdk11中该接口的描述：

An object that provides hooks for connection pool management. A `PooledConnection` object represents a physical connection to a data source. The connection can be recycled rather than being closed when an application is finished with it, thus reducing the number of connections that need to be made.

翻译的大体意思是：`javax.sql.PooledConnection`为连接池的管理提供了“钩子（hook）”。该接口让数据库连接在调用`close()`方法时会被连接池回收而不是真的关闭与数据库的物理连接。

其中涉及到的“hook”，在网上定于为“钩子函数”。其目的是在目标函数被调用前，先调用该钩子函数。其实在Java中常见的代理模式与这个hook有着相似的功能，例如触发某个时间后，代理类先处理非业务逻辑（例如写日志），再调用目标业务方法。

这个接口涉及到的方法如下：

```
public interface PooledConnection {
    void close() throws SQLException; // App调用该方法并不是关闭物理连接。
                                     // 关闭物理连接的操作应该由连接池来完成。
    Connection getConnection() throws SQLException;
    void addConnectionEventListener(ConnectionEventListener listener);
    void removeConnectionEventListener(ConnectionEventListener listener);
    public void addStatementEventListener(StatementEventListener listener);
    public void removeStatementEventListener(StatementEventListener listener);
}
```

看`close()`和`getConnection()`这两个方法，可以知道要代理`java.sql.Connection`中对应的方法。这里就通过方法覆盖和动态链接实现了与hook类似的效果。以下是`DruidPooledConnection`重写后的`close()`。

```
@Override
public void close() throws SQLException {
    if (this.disable) {
        return;
    }
}
```

```

        DruidConnectionHolder holder = this.holder;
        if (holder == null) {
            if (dupCloseLogEnable) {
                LOG.error("dup close");
            }
            return;
        }

        DruidAbstractDataSource dataSource =
holder.getDataSource();
        boolean isSameThread = this.getOwnerThread() ==
Thread.currentThread();

        if (!isSameThread) {
            dataSource.setAsyncCloseConnectionEnable(true);
        }

        if (dataSource.isAsyncCloseConnectionEnable()) {
            syncClose(); // 根据条件选择关闭连接
            return;
        }

        if (!CLOSING_UPDATER.compareAndSet(this, 0, 1)) {
            return;
        }

        try {
            for (ConnectionEventListener listener :
holder.getConnectionEventListeners()) {
                listener.connectionClosed(new
ConnectionEvent(this));
            }

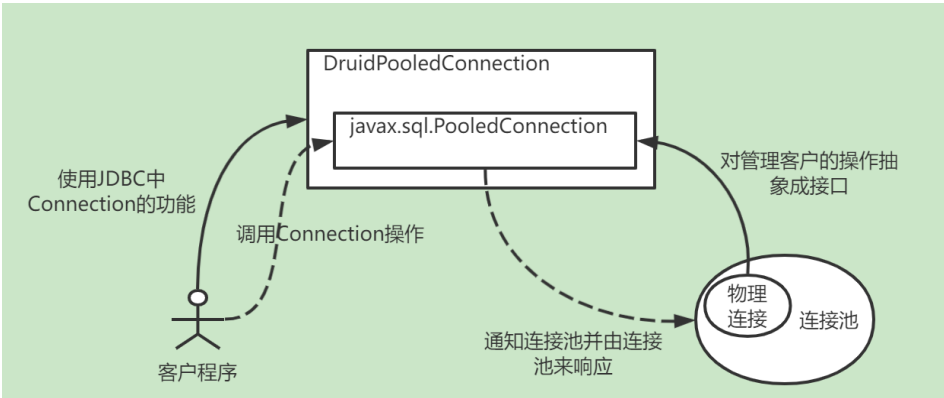
            List<Filter> filters = dataSource.getProxyFilters();
            if (filters.size() > 0) {
                FilterChainImpl filterChain = new
FilterChainImpl(dataSource);
                filterChain.dataSource_recycle(this);
            } else {
                recycle(); // 回收该连接
            }
        } finally {
            CLOSING_UPDATER.set(this, 0);
        }
    }

```

```
    }

    this.disable = true;
}
```

至此，实现了 `javax.sql.PooledConnection` 接口的类就代表着该类的数据库连接具备着池化属性。对于客户程序来讲是透明的。而对连接池而言，该接口则是一层抽象框架，目的是抽象出来连接池中的连接应该有的行为以及告诉连接池如何响应客户在连接上做出的一些行为（该接口的监听方法）。 `javax.sql.PooledConnection` 像是"客户 <---> 连接池"之间的一个桥梁，用下图来表示：



意义

目前了解到在 `DruidPooledConnection` 大部分的成员变量与holder相对应，其行为也类似。比较新颖的是在 `DruidPooledConnection` 还会记录着拥有该连接的线程。`DruidPooledConnection`最主要的功能还是为客户程序提供了与 `java.sql.Connection` 相同的功能，同时又为了连接池的管理重写了部分功能。

信息划分图

