JDBC 学习笔记

接口的作用

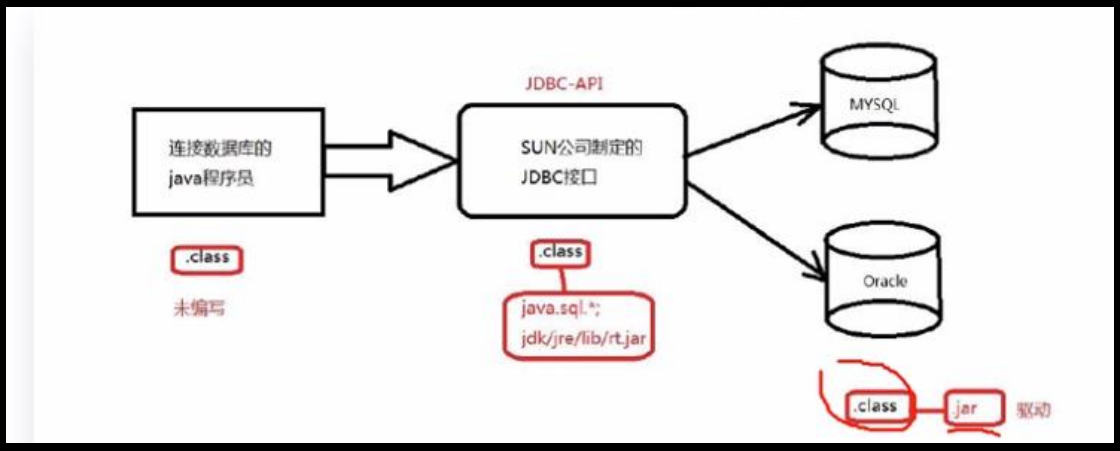
- 易于扩展
- 一套规范，需要面向接口编程。

（菜单的例子）

JDBC

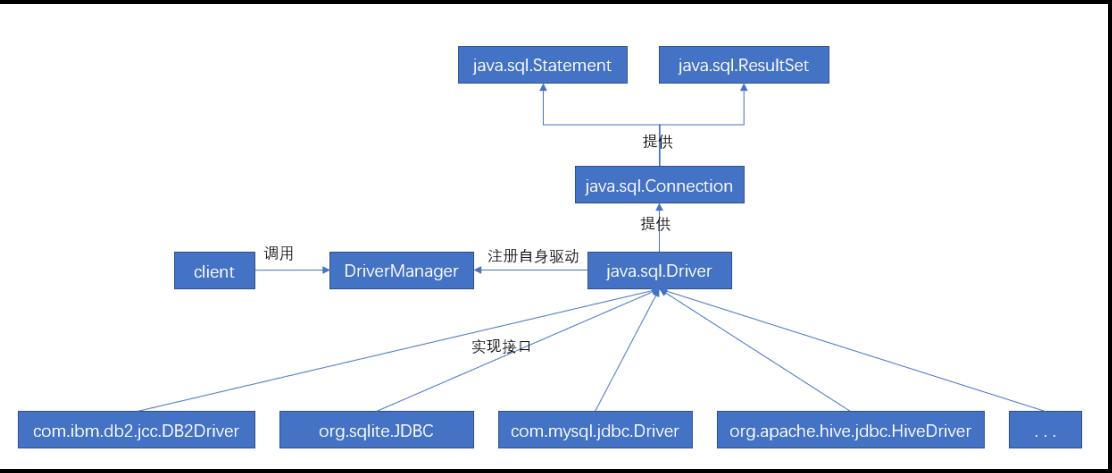
- Java 连接数据库的
- Java data base Connectivity　Java 连接数据库系统
- Jdbc 是一套 class 文件
- SUN 公司制定了 JDBC 规范
- **JDBC 一套规范，各大数据库都转交给它，因此一套 JDBC 就可以实现多种数据库。**
- 连接数据库驱动：数据库公司实现了的 JDBC 的接口，然后生成.class，打包即为数据库驱动。

原理



JDBC 是 java 中的一个数据连接技术，它提供了统一的 API 允许用户访问任何形式的表格数据，尤其是存储在关系数据库中的数据。

JDBC 之所以能提供统一的 API，是基于对所有数据库的抽象及合理的定义。但是每个数据库厂家毕竟是不一样的，JDBC 自然要屏蔽这种不一样，它是如何做到的呢？这就是本文讨论的 **DriverManager**，它是一个**桥接模式**的完美应用。



JDBC 的编程模式是固定的，也就说操作步骤基本是一定的，如下：

```
public void testJdbcRaw() throws Exception {

        //1.加载驱动程序
       Class.forName("com.mysql.jdbc.Driver");
       //2. 获得数据库连接
        Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
        //3.操作数据库，实现增删改查, 连接模式有 2 种: createStatement / prepareStatement
        Statement stmt = conn.createStatement();
        // PreparedStatement ptmt = conn.prepareStatement(sql); //预编译 SQL，减少 sql 执行  //预编译
        ResultSet rs = stmt.executeQuery("SELECT username, age FROM user");

        //如果有数据，rs.next()返回 true
        while(rs.next()){
            System.out.println(rs.getString("username")+"  年龄："+rs.getInt("age"));
        }

        // 4. 关闭连接
        conn.close();

    }
```

- **加载驱动**
- **获取数据库连接**
- **操作数据库：CURD。连接模式有 2 种: createStatement / prepareStatement;**
- **关闭连接**

驱动是如何加载的？

如果我们不考虑统一各数据库的统一性，比如需要创建一个 mysql 的连接，那么我们只需要将 mysql 的连接工具类，new 一个对象出来就可以了。然而，jdbc 却是不可以这么干的，因为它要成为一种标准。实现很简单，直接通过一个反射方法，就可以加载驱动了，那么具体是如何加载的呢？

以 mysql 为例，使用反射方法去找到 驱动类 Class.forName("com.mysql.jdbc.Driver"); 所以，如何驱动起来，也是这个驱动类应该做的事了

```
// mysql 的驱动类如下
// 重点 1: 该驱动类必须实现 java.sql.Driver 接口
public class Driver extends NonRegisteringDriver implements java.sql.Driver {
    //
    // Register ourselves with the DriverManager
    //
```

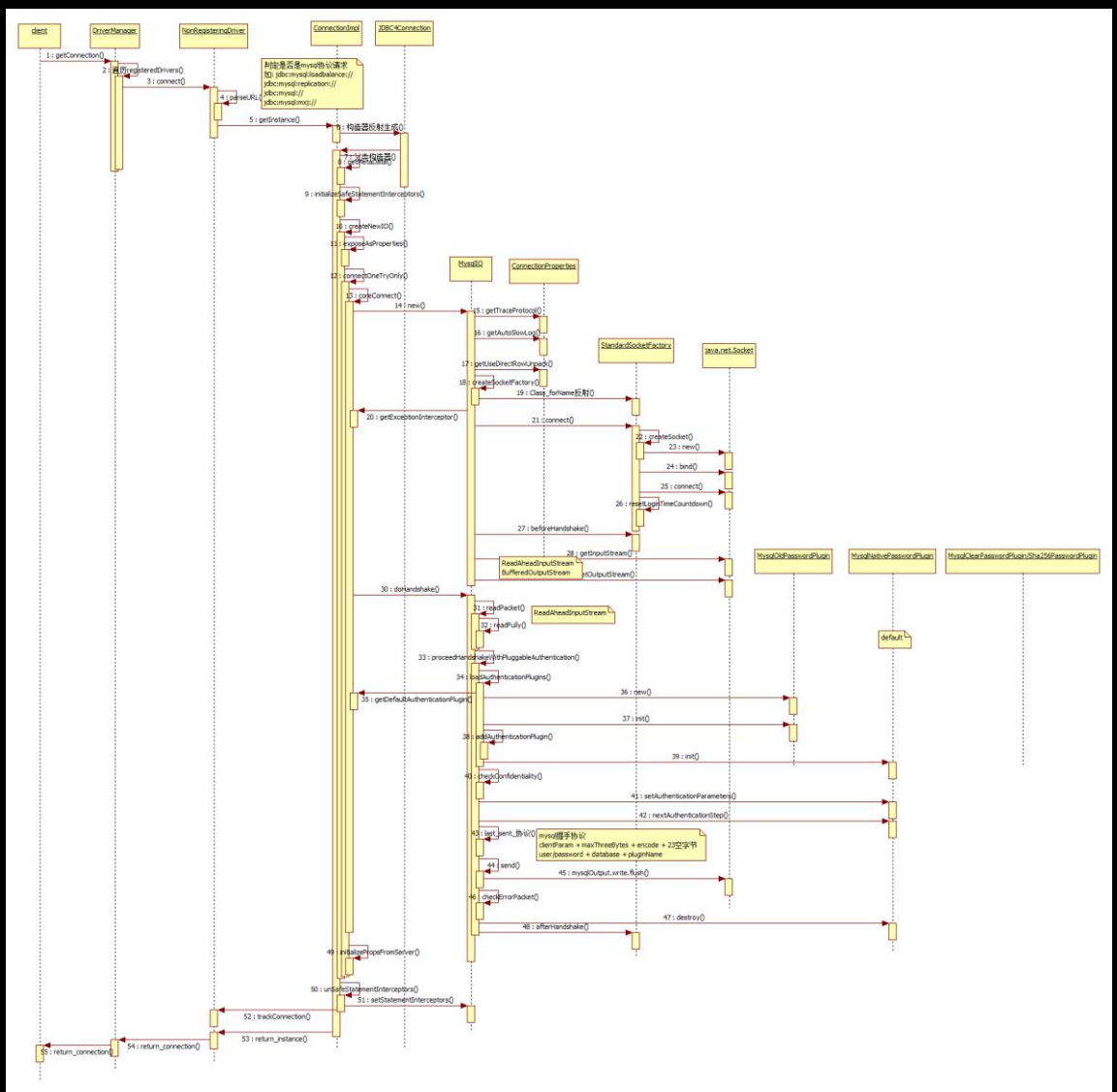
```
    static {
        try {
            // 重点 2: 必须在加载时，就将自身注册到 DriverManager 中
            java.sql.DriverManager.registerDriver(new Driver());
        } catch (SQLException E) {
            throw new RuntimeException("Can't register driver!");
        }
    }
    }
```

这样，mysql 的驱动就注册到 DriverManager 中了，也就是可以接受 DriverManager 的管理了，需要注意的是，这里的类加载是特别的，它是违背“双亲委托加载模型”的一个案例，使用的是 **contextClassLoader** 进行加载驱动的。接下来我们要讲的统一的 API 获取数据库连接。

如何获取数据库连接？

通过注册的方式，我已经将数据库的实例，交给了 DriverManager，此时再要获取数据库连接，也就只需要问 DriverManager 要就行了。

以一个时序图总览全局：




```
@CallerSensitive
    public static Connection getConnection(String url,
        String user, String password) throws SQLException {
```

```
        java.util.Properties info = new java.util.Properties();

        if (user != null) {
            info.put("user", user);
        }
        if (password != null) {
            info.put("password", password);
        }
        // 统一将必要信息封装到 Properties 中，方便各自的驱动按需获取
        return (getConnection(url, info, Reflection.getCallerClass()));
    }
    //   Worker method called by the public getConnection() methods.
    private static Connection getConnection(
        String url, java.util.Properties info, Class<?> caller) throws SQLException {
        /*
         * When callerCl is null, we should check the application's
         * (which is invoking this class indirectly)
         * classloader, so that the JDBC driver class outside rt.jar
         * can be loaded from here.
         */
        // callerCL 可能为空，因为加载不到外部调用的类，此处违反了 双亲委派模型
        ClassLoader callerCL = caller != null ? caller.getClassLoader() : null;
        synchronized(DriverManager.class) {
            // synchronize loading of the correct classloader.
            if (callerCL == null) {
                // 通过 ContextClassLoader 进行加载
                callerCL = Thread.currentThread().getContextClassLoader();
            }
        }

        if(url == null) {
            throw new SQLException("The url cannot be null", "08001");
        }

        println("DriverManager.getConnection(\"" + url + "\")");

        // Walk through the loaded registeredDrivers attempting to make a connection.
        // Remember the first exception that gets raised so we can reraise it.
        SQLException reason = null;

        for(DriverInfo aDriver : registeredDrivers) {
            // If the caller does not have permission to load the driver then
            // skip it.
            // 检查 classloader 是否相同，从而确认是否可以进行加载
            if(isDriverAllowed(aDriver.driver, callerCL)) {
                try {
                    println("        trying " + aDriver.driver.getClass().getName());
                    // 其实是一个个驱动地尝试连接，直到找到第 1 个可用的连接
                    // 其实一般是通过 连接协议来自行判定的，稍后我们以 mysql 的连接示例看一下
                    Connection con = aDriver.driver.connect(url, info);
                    if (con != null) {
                        // Success!
                        println("getConnection returning " + aDriver.driver.getClass().getName());
                        return (con);
                    }
                } catch (SQLException ex) {
                    if (reason == null) {
                        reason = ex;
                    }
                }

            } else {
                println("        skipping: " + aDriver.getClass().getName());
            }
```

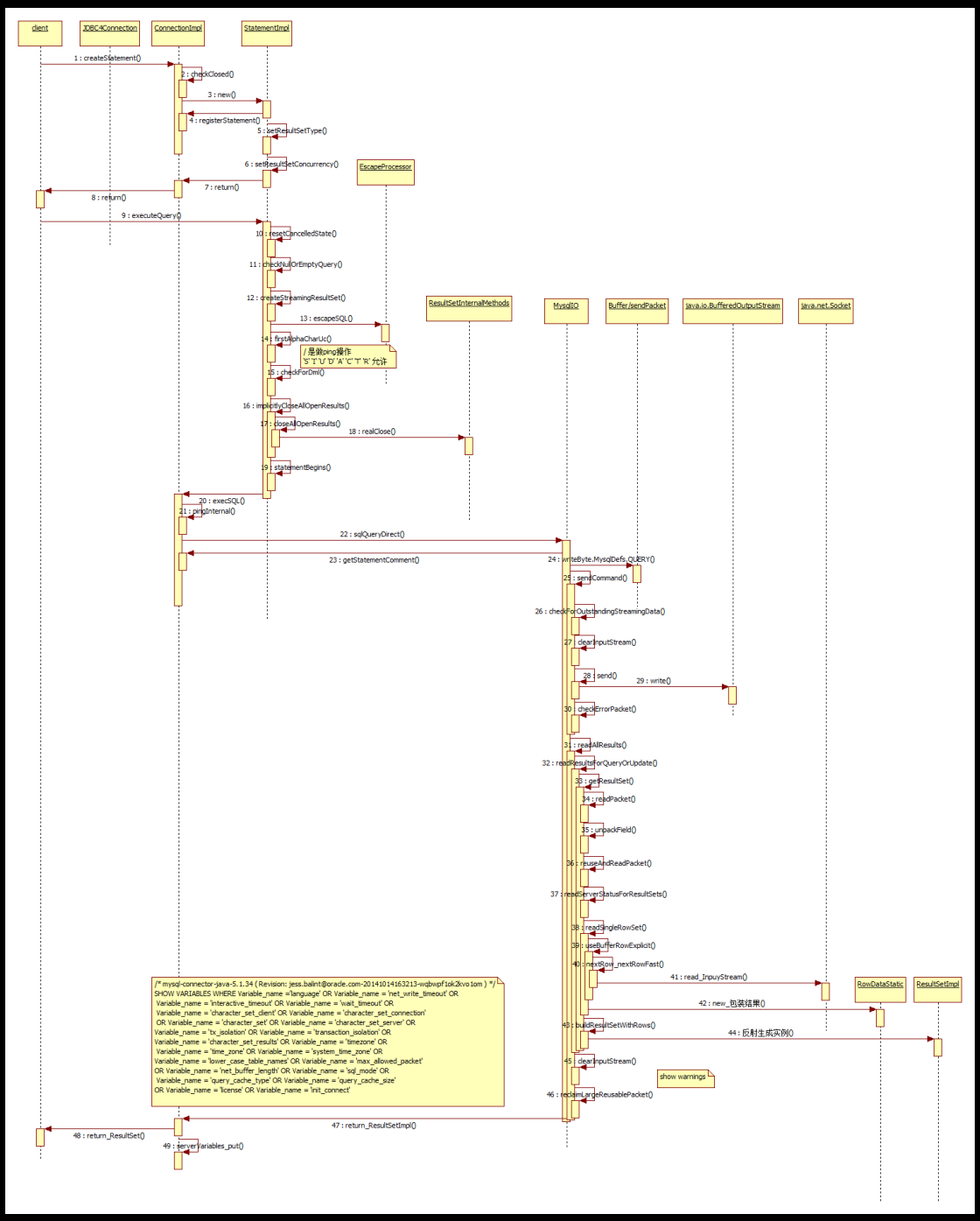
```
            }

            // if we got here nobody could connect.
            if (reason != null)       {
                println("getConnection failed: " + reason);
                throw reason;
            }

            println("getConnection: no suitable driver found for "+ url);
            throw new SQLException("No suitable driver found for "+ url, "08001");
        }
        // 检查 driver 属于 classLoader 的管理范围
        private static boolean isDriverAllowed(Driver driver, ClassLoader classLoader) {
            boolean result = false;
            if(driver != null) {
                Class<?> aClass = null;
                try {
                    aClass =    Class.forName(driver.getClass().getName(), true, classLoader);
                } catch (Exception ex) {
                    result = false;
                }

                 result = ( aClass == driver.getClass() ) ? true : false;
            }

            return result;
        }
```

DriverManager 通过遍历所有驱动列表的形式，查找是否是某种类型的数据库操作。虽然看起来好像有点费事，但是毕竟是做通用的框架，这样做可以保证正确性，况且几次调用对性能影响也不大。虽然各驱动可以自行处理或拒绝某协议请求，但是一般都是以 url 前缀作为判断接受与否的。我们来看下 mysql 如何处理？

如何执行 sql 操作

主要有两种方式, **statement 和 prepareStatement.**

PreparedStatement 继承自 Statement，两者都是接口。区别是：PreparedStatement 是预编译的(mysql 提供的能力)，比 Statement 效率高，可以使用占位符，可防止 SQL 注入。




```
//获取预处理对象
        statement = getConnection().prepareStatement(sql);
        statement.executeUpdate(sql);
```

如何获取查询结果？

ResultSet 的处理。

```
public java.sql.ResultSet executeQuery(String sql) throws SQLException {
        synchronized (checkClosed().getConnectionMutex()) {
                MySQLConnection locallyScopedConn = this.connection;

                this.retrieveGeneratedKeys = false;

                resetCancelledState();
```

```
                checkNullOrEmptyQuery(sql);

                boolean doStreaming = createStreamingResultSet();

                // Adjust net_write_timeout to a higher value if we're streaming result sets. More often than not, someone
runs into an issue where they blow
                // net_write_timeout when using this feature, and if they're willing to hold a result set open for 30 seconds or
more, one more round-trip isn't
                // going to hurt
                //
                // This is reset by RowDataDynamic.close().

                if (doStreaming && this.connection.getNetTimeoutForStreamingResults() > 0) {
                        executeSimpleNonQuery(locallyScopedConn, "SET net_write_timeout=" +
this.connection.getNetTimeoutForStreamingResults());
                }

                if (this.doEscapeProcessing) {
                        //  避免 sql 注入
                        Object escapedSqlResult = EscapeProcessor.escapeSQL(sql,
locallyScopedConn.serverSupportsConvertFn(), this.connection);

                        if (escapedSqlResult instanceof String) {
                                sql = (String) escapedSqlResult;
                        } else {
                                sql = ((EscapeProcessorResult) escapedSqlResult).escapedSql;
                        }
                }

                char firstStatementChar = StringUtils.firstAlphaCharUc(sql, findStartOfStatement(sql));

                if (sql.charAt(0) == '/') {
                        if (sql.startsWith(PING_MARKER)) {
                                doPingInstead();

                                return this.results;
                        }
                }

                checkForDml(sql, firstStatementChar);

                implicitlyCloseAllOpenResults();

                CachedResultSetMetaData cachedMetaData = null;

                if (useServerFetch()) {
                        this.results = createResultSetUsingServerFetch(sql);

                        return this.results;
                }

                CancelTask timeoutTask = null;

                String oldCatalog = null;

                try {
                        if (locallyScopedConn.getEnableQueryTimeouts() && this.timeoutInMillis != 0 &&
locallyScopedConn.versionMeetsMinimum(5, 0, 0)) {
                                timeoutTask = new CancelTask(this);
                                locallyScopedConn.getCancelTimer().schedule(timeoutTask, this.timeoutInMillis);
                        }

                        if (!locallyScopedConn.getCatalog().equals(this.currentCatalog)) {
                                oldCatalog = locallyScopedConn.getCatalog();
```

```
                locallyScopedConn.setCatalog(this.currentCatalog);
            }

            //
            // Check if we have cached metadata for this query...
            //

            Field[] cachedFields = null;

            if (locallyScopedConn.getCacheResultSetMetadata()) {
                cachedMetaData = locallyScopedConn.getCachedMetaData(sql);

                if (cachedMetaData != null) {
                    cachedFields = cachedMetaData.fields;
                }
            }

            locallyScopedConn.setSessionMaxRows(this.maxRows);

            statementBegins();

            this.results = locallyScopedConn.execSQL(this, sql, this.maxRows, null, this.resultSetType,
this.resultSetConcurrency, doStreaming,
                    this.currentCatalog, cachedFields);

            if (timeoutTask != null) {
                if (timeoutTask.caughtWhileCancelling != null) {
                    throw timeoutTask.caughtWhileCancelling;
                }

                timeoutTask.cancel();

                locallyScopedConn.getCancelTimer().purge();

                timeoutTask = null;
            }

            synchronized (this.cancelTimeoutMutex) {
                if (this.wasCancelled) {
                    SQLException cause = null;

                    if (this.wasCancelledByTimeout) {
                        cause = new MySQLTimeoutException();
                    } else {
                        cause = new MySQLStatementCancelledException();
                    }

                    resetCancelledState();

                    throw cause;
                }
            }
        } finally {
            this.statementExecuting.set(false);

            if (timeoutTask != null) {
                timeoutTask.cancel();

                locallyScopedConn.getCancelTimer().purge();
            }

            if (oldCatalog != null) {
                locallyScopedConn.setCatalog(oldCatalog);
            }
```

```
                }
                // lastInsertId
                this.lastInsertId = this.results.getUpdateID();

                if (cachedMetaData != null) {
                        locallyScopedConn.initializeResultsMetadataFromCache(sql, cachedMetaData, this.results);
                } else {
                        if (this.connection.getCacheResultSetMetadata()) {
                                locallyScopedConn.initializeResultsMetadataFromCache(sql, null /* will be created */,
this.results);
                        }
                }

                return this.results;
        }
    }
```

如何关闭数据库连接？

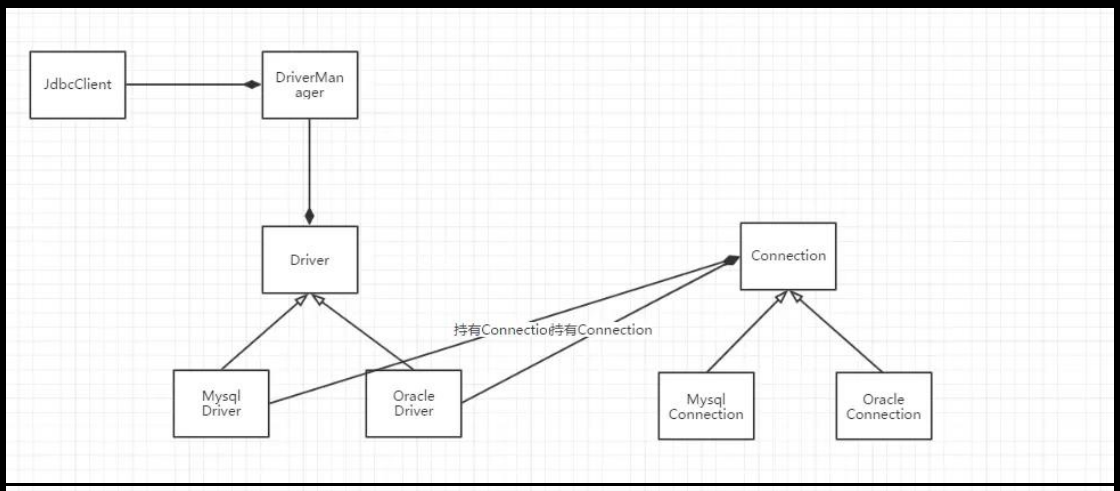
```
public void close() throws SQLException {
        synchronized (getConnectionMutex()) {
                // 关闭前如果有拦截器，先调用拦截器处理
                if (this.connectionLifecycleInterceptors != null) {
                        new IterateBlock<Extension>(this.connectionLifecycleInterceptors.iterator()) {
                                @Override
                                void forEach(Extension each) throws SQLException {
                                        ((ConnectionLifecycleInterceptor) each).close();
                                }
                        }.doForAll();
                }

                realClose(true, true, false, null);
        }
    }
```

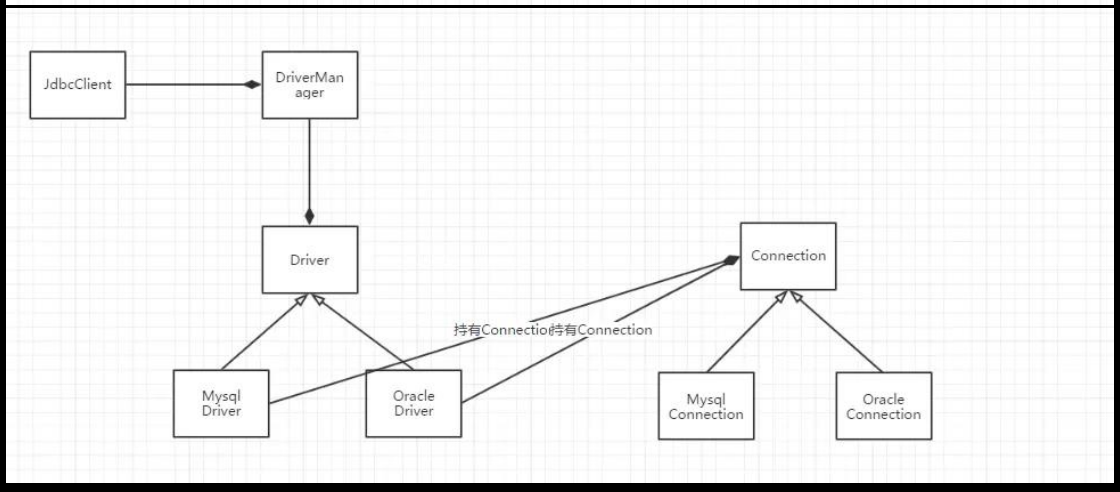
然而，jdbc 这样的操作毕竟太过于模板化，如果在每个项目里反复写这些模板代码，那就是太伤了。所以，涌现出大量的 orm 框架，如: hibernates, mybatis. 将我们从模板代码中解放出来。底层受益出 jdbc 的设计，高层高效服务于开发人员。

JDBC 中利用了那些设计模式

单例模式

桥接模式





Mysql 为例，通过 **Class.forName("com.mysql.jdbc.Driver")类加载的时候执行静态代码块将 Driver 注册到 DriverManager,DriverManager** 是个 Driver 容器，管理不同的 Driver,这样具体的数据 Driver 实现就统一交给容器管理，客户端通过 DriverManager 执行验证连接，获取连接的操作。

```
static {
        try {
            DriverManager.registerDriver(new Driver());
        } catch (SQLException var1) {
            throw new RuntimeException("Can't register driver!");
        }
    }
```

API

Application program interface

包括：

- API **字节码**
- API **源码**
- API **帮助文档**

版本需要保持一致

这里关于 JDBC 的所有接口都可以在 JavaAPI 中查找

配置环境

1.添加驱动：下载驱动包：**mysql-connector-java.jar**

在环境变量中，系统变量添加 classpath，加入：.:+该包绝对路径。

（.:表示先在该代码路径找，找不到再到后面的绝对路径找。）

JDBC 编程六部曲

- 第一步：注册驱动
  1. 获取驱动对象
  2. 注册驱动
- 第二步：获取数据库连接
- 第三步：获取数据库操作对象
- 第四步：执行 SQL 语句
- 第五步：处理查询结果集
- 第六步：关闭资源

```java
package SixSteps;
import java.sql.Connection;
import java.sql.Statement;
import com.mysql.cj.protocol.Resultset;
import java.sql.Driver;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
//import com.mysql.cj.xdevapi.Statement; 不是这里的Statement，java.sql.Statement
//实际上就是具体的数据库来实现jdbc的接口： MySQL <= JDBC
public class JDBCTest01 {
    private static Connection connection = null;
    private static Resultset resultset = null;
    public static void main(String[] args) {
        try {
//第一步：注册驱动
//      1.1 获取驱动对象
            //除了在环境变量配置好驱动外，这里也需要导入mysql-connector-java.jar

//      java.sql.Driver
        Driver driver = new com.mysql.cj.jdbc.Driver();//mysql对接口的实现，以前是com.mysql.jdbc.Driver()，现在更新：com.mysql.cj.jdbc.Driver()
        //jdbc 的接口
        //由于类名一样，必须使用全类名

//      1.2 注册驱动
        DriverManager.registerDriver(driver);
```

```java
    2.获取数据库连接
    //localhost:3366???
    String url = "jdbc:mysql://localhost/edwin?Encoding=utf-8&autoReconnect=true&useSSL=false&testOnBorrow=true&validationQuery=select";
    // url: 数据库地址 jdbc:mysql://连接主机IP:端口号//数据库名字（主机IP使用，localhost）
    String user = "root";
    String password = "xt222483";
    connection = DriverManager.getConnection(url, user, password);
    System.out.println(connection);

    //3.获取数据库操作对象。
    Statement statement = connection.createStatement();
    System.out.println(statement);

    //4.执行SQL语句
        //4.1 DQL语句， --> 5
    String sql = "select pet.name,pet.species,pet.gender,pet.birth from pet";
    resultset =  (Resultset) statement.executeQuery(sql);
    System.out.println(resultset);
        //4.2 DML语句: --> over

    //5.处理查询结果集
    while (((ResultSet) resultset).next()) {
        String name = ((ResultSet) resultset).getString("name");
        String species = ((ResultSet) resultset).getString("species");
        String gender = ((ResultSet) resultset).getString("gender");
        String birth = ((ResultSet) resultset).getString("birth");
```

```
            //不建议：
//          String name = ((ResultSet) resultset).getString(1);
//          String species = ((ResultSet) resultset).getString(2);
//          String gender = ((ResultSet) resultset).getString(3);
//          String birth = ((ResultSet) resultset).getString(4);
            System.out.println(name+"  "+species +"  "+gender+"  "+birth);
        }
        }catch (Exception e) {
            e.printStackTrace();
        }finally {
            //6.关闭资源
            if (resultset!=null) {
                try {
                    ((ResultSet) resultset).close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

自己写的：

```
public static void main(String[] args) throws Exception {
    Class.forName("com.mysql.jdbc.Driver"); //加载驱动、注册驱动
    Connection connection = DriverManager //获取连接
            .getConnection( url: "jdbc:mysql://localhost:3306/utool?serverTimezone=UTC", user: "root", password: "xt222483");
    String sql = "select * from types";
    Statement statement = connection.createStatement(); //创建数据库操作对象 Statement
    ResultSet set = statement.executeQuery(sql); //执行SQL，返回结果
    while (set.next()){//打印结果
        System.out.println(set.getString( columnLabel: "id") +" " + set.getString( columnLabel: "enName"));
    }
}
```

JDBC 编程模板

```
package SixSteps;

//import java.sql.Connection;

//import java.sql.Driver;

//import java.sql.DriverManager;

//import java.sql.ResultSet;

//import java.sql.SQLException;

//import java.sql.Statement;

import java.sql.*;

/**
 * JDBC 编程六部曲：
 *    第一步：注册驱动。
 *         1.1 获取驱动对象
 *         1.2 注册驱动
 *    第二步：获取数据库连接
 *    第三步：获取数据库操作对象
 *   第四步：执行 SQL 语句
 *   第五步：处理查询结果
 *   第六步：关闭资源
 */
public class JDBCTest02 {

        private static Connection connection = null;

        private static ResultSet resultSet = null;

        private static Statement statement = null;

        public static void main(String[] args) {

                try {

                        //1.注册驱动
```

```
            // 1.1 获取驱动对象
            Driver driver = new com.mysql.cj.jdbc.Driver();
            // 1.2 注册
            DriverManager.registerDriver(driver);
            //2.获取数据库连接
            String url =
"jdbc:mysql://localhost:3306/world?Encoding=utf-8&autoReconnect=true&useSSL=false&testOnBorrow=true&validationQuery=select";
            String user    = "root";
            String password = "xt222483";
            connection =DriverManager.getConnection(url, user, password);
            //3.获取数据库操作对象。
            statement = connection.createStatement();
            //4：执行 SQL 语句
            String mysql = "select Language, isOfficial from countrylanguage";
            resultSet = statement.executeQuery(mysql);
            //5.处理查询结果
            System.out.println("Language                    IsOfficial");
            System.out.println("----------------------------");
            while (resultSet.next()) {
                String language =resultSet.getString("Language");
                System.out.println(language+JDBCTest02.getSpace(25-language.length())+resultSet.getString("isOfficial"));
            }
        }catch (Exception e) {
            e.printStackTrace();
        }finally {
            //6.关闭资源
            if (resultSet!=null) {
                try {
                    resultSet.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (connection!=null) {
                try {
                    connection.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (statement!=null) {
                try {
                    statement.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
```

```
        }
    }
}
```

DQL、DML、DDL、DCL

SQL : Structure Query Language 数据库核心语言（Query：问号、询问、质疑）

分类：

1. **DQL：数据查询语言**
   基本结构是由 select、form、 where 语句构成的查询块。

   JDBC：**statement.executeQuery(sql)**
2. **DML：数据操纵语言**
   a) 插入 INSERT
   b) 更新 UPDATE
   c) 删除 DELETE

   JDBC：**statement.executeUpdate(sql)——返回 int:返回执行的行计数(若 sql 本身不返回值，返回 0)**
3. **DDL：数据定义语言**
   用于创建数据库的各种对象：表、视图、索引、同义词、聚簇等。
   CREATE TABLE / VIEW / INDEX/ SYN / CLUSTER
4. **DCL：数据库控制语言**
   授予或回收访问数据库的某种特权，并控制数据库操纵事物发生的时间及效果。

注册驱动方式 2——静态代码块

- 建立新类
- 在类中添加静态代码块

```
public class Register {
    static {
        try {
            Driver driver = new    com.mysql.cj.jdbc.Driver();
            DriverManager.registerDriver(driver);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

- 使用 java 反射机制调用：

```
Class.forName("SixSteps.Register");
```

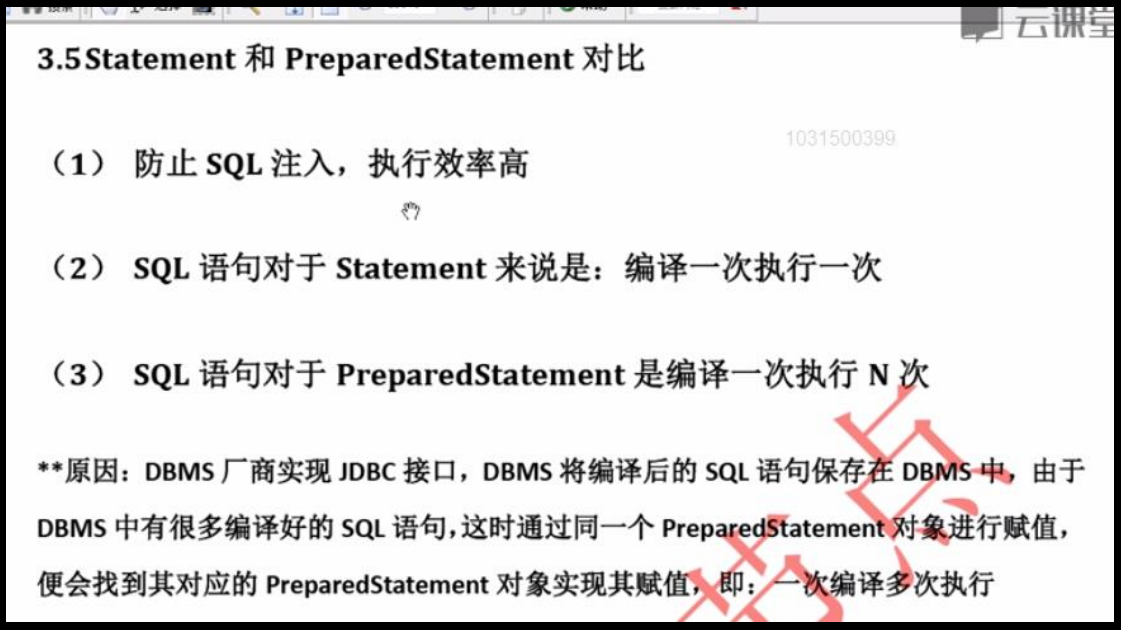
Class.forName(Total Name)

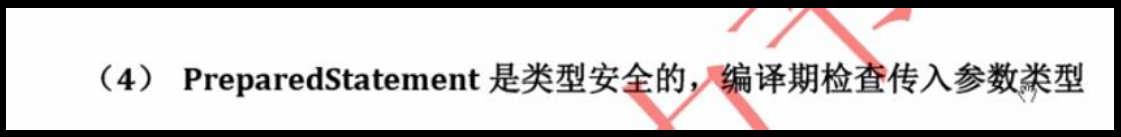
注册驱动方式 3——反射机制

Com.forName(“com.mmysql.cj.jdbc.Driver”)

SQL 注入

注册登录时输入的字符串包含关键字，拼接成 sql 后 SQL 的语句意思变化了。
（通过 select 选择后的 statement.next()判断是否在数据库中存在）
解决方案：
PrepareStatement
String sql = “select usename from table1 where usename = ? and password = ?”
PrepareStatement ps = connection.prepareStatement(sql)
Ps.setString(1,usename)
Ps.setString(2,password)





错误日志

- Sun Mar 24 21:57:58 CST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServerCertificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide
  String
  - 解决方案：
    url="jdbc:mysql://localhost:3306/demo?autoReconnect=true&useSSL=false";
    add :autoReconnect=true&useSSL=false
- cmd:Install/Remove of the Service Denied　　需要使用管理员
- The server time zone value 'ÖÐ¹ú±ê×¼Ê±¼ä' is unrecognized or represents more than one time zone. You
  时区错误

```
mysql> set global time_zone='+8:00';
Query OK, 0 rows affected (0.03 sec)
```

- Operation not allowed after ResultSet closed
  一个 statement 不能打开多个 ResultSet
  比如你不能使用一个 statement 分别调用 executeQuery() 和 executeUpdate()
  所以可以多创建几个 statement
  但是 DML、DQL 可以分别使用一个 Statement

ResultSet is from UPDATE. No Data.

在 executeQuery 之后用 while(rs.next)要注意 select from 的表里面要有数据，如果没有数据就会出现 java.sql.SQLException: ResultSet is from UPDATE. No Data.