**JDBC**

## JDBC概念

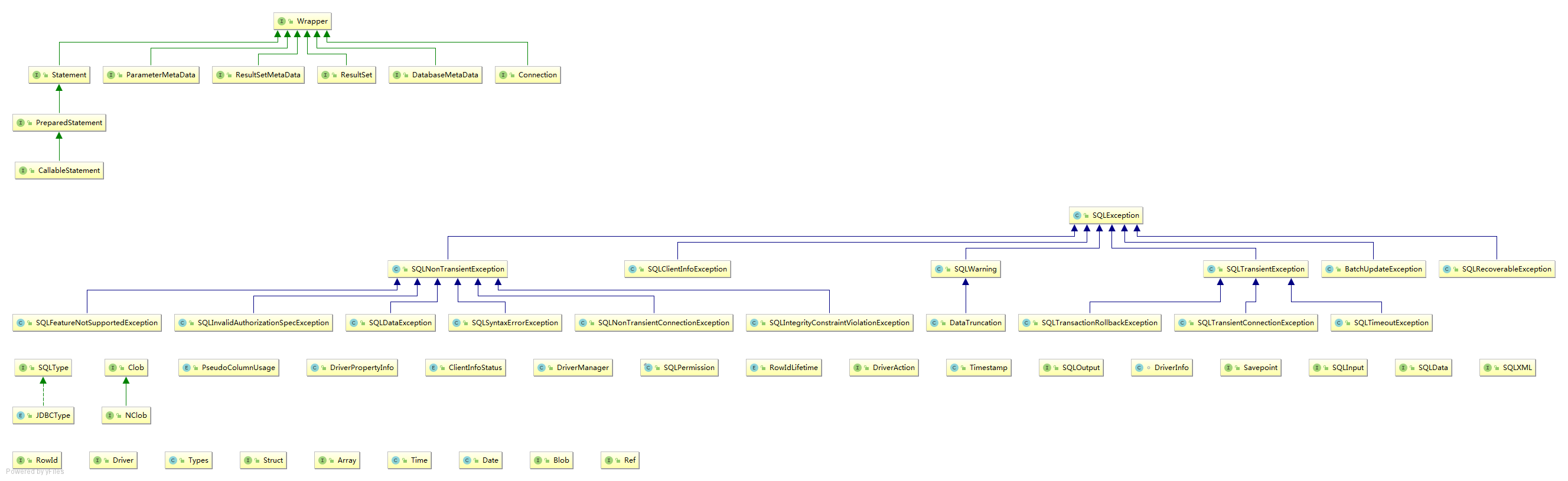
JDBC（Java DataBase Connectivity,java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，由一组用[Java语言](https://baike.baidu.com/item/Java%E8%AF%AD%E8%A8%80" \t "https://baike.baidu.com/item/_blank)编写的类和接口组成。JDBC提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序

有了JDBC，向各种关系数据发送SQL语句就是一件很容易的事。有了JDBC API，就不必为访问[Sybase数据库](https://baike.baidu.com/item/Sybase%E6%95%B0%E6%8D%AE%E5%BA%93" \t "https://baike.baidu.com/item/_blank)专门写一个程序，为访问[Oracle数据库](https://baike.baidu.com/item/Oracle%E6%95%B0%E6%8D%AE%E5%BA%93" \t "https://baike.baidu.com/item/_blank)又专门写一个程序，或为访问Informix数据库又编写另一个程序等等，程序员只需用JDBC API写一个程序就够了，它可向相应数据库发送SQL调用。同时，将Java语言和JDBC结合起来使程序员不必为不同的平台编写不同的应用程序，只须写一遍程序就可以让它在任何平台上运行，这也是Java语言“编写一次，处处运行”的优势。

[Java数据库连接](https://baike.baidu.com/item/Java%E6%95%B0%E6%8D%AE%E5%BA%93%E8%BF%9E%E6%8E%A5" \t "https://baike.baidu.com/item/_blank)体系结构是用于Java应用程序连接数据库的标准方法。JDBC对Java程序员而言是API，对实现与数据库连接的服务提供商而言是接口模型。作为API，JDBC为程序开发提供标准的接口，并为数据库厂商及第三方中间件厂商实现与数据库的连接提供了标准方法。JDBC使用已有的SQL标准并支持与其它数据库连接标准，如ODBC之间的[桥接](https://baike.baidu.com/item/%E6%A1%A5%E6%8E%A5" \t "https://baike.baidu.com/item/_blank)。JDBC实现了所有这些面向标准的目标并且具有简单、严格类型定义且高性能实现的接口。

## JDBC API

java.sql.\*包下所有的类



## 核心步骤

Class.forName("×××Driver");

Connection conn = DriverManager.getConnection(DB\_URL,USER,PASS);

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql);

## JDBC驱动程序

JDBC Driver API（第三方驱动包如mysql-connector-java）实现了 JDBC API 中定义的接口，该接口用于与数据库服务器进行交互，与数据库的连接，查询数据的网络传输，默认是基于tcp协议的，对接受到的数据的封装处理等这些都是驱动包完成的。使用 JDBC 驱动程序可以让你打开数据库连接，并通过发送 SQL 或数据库命令，然后通过 Java 接收结果。java.sql 包中附带的 JDK，包含了定义各种类与他们的行为和实际实现，这些类都在第三方驱动程序中完成。第三方供应商在他们的数据库驱动程序中都实现了 java.sql.Driver 接口

## 常见JDBC组件

DriverManager ：这个类管理一系列数据库驱动程序。匹配连接使用通信子协议从 JAVA 应用程序中请求合适的数据库驱动程序。识别 JDBC 下某个子协议的第一驱动程序将被用于建立数据库连接。

Driver : 这个接口处理与数据库服务器的通信。你将很少直接与驱动程序互动。相反，你使用 DriverManager 中的对象，它管理此类型的对象。它也抽象与驱动程序对象工作相关的详细信息。

Connection : 此接口具有接触数据库的所有方法。该连接对象表示通信上下文，即，所有与数据库的通信仅通过这个连接对象进行。

Statement : 使用创建于这个接口的对象将 SQL 语句提交到数据库。除了执行存储过程以外，一些派生的接口也接受参数。

PreparedStatement：表示预编译SQL语句的对象。预编译SQL语句并将其存储在PreparedStatement 对象中,语句在被DB的编译器编译后的执行代码被缓存下来,那么下次调用时只要是相同的预编译语句就不需要编译,只要将参数直接传入编译过的语句执行代码中(相当于一个涵数)就会得到执行,可以使用此对象多次有效地执行此语句，该语句为每个 IN 参数保留一个问号（“？”）作为[占位符](http://baike.baidu.com/view/873819.htm" \t "https://www.cnblogs.com/weiyi1314/p/_blank),当运行时动态地把参数传给PreprareStatement时，通过setInt、setString等方式设置，具体实现由不同的驱动实现，但都会进行必要的安全检查和转义，如对字符串中有’单引号、or等特殊字符的处理,起到了SQL注入的作用。继承Statement所以有其所有功能一般都用PreparedStatement效率高又安全。

Mybatis中的#{}、${}

#{}对应PreparedStatement，通过？设置，更具jdbcType类型设置，如string类型则检测和转义后会给传入的参数加一个双引号

${}对应Statement，直接将参数和sql语句拼接在一起当成整个字符串，传入的参数直接显示生成在sql中，不会添加引号，所以动态表明、列名、in（）则必须用${}

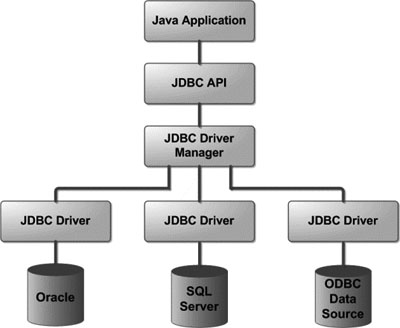
ResultSet : 在你使用语句对象执行 SQL 查询后，这些对象保存从数据获得的数据。它作为一个迭代器，让您可以通过它的数据来移动。

JDBCType:JDBC 驱动程序在将 Java 数据类型发送到数据库之前，会将其转换为相应的 JDBC 类型,由于 SQL 数据类型和 Java 数据类型是不同的，因此需要某种机制在使用 Java 类型的应用程序和使用 SQL 类型的数据库之间来读写数据,java.sql.Types 类中定义了一系列的常规 SQL 类型标识符。这些类型可用于表示那些最为常用的 SQL 类型。在用 JDBC API 编程时，通常可以使用这些 JDBC 类型来引用一般的 SQL 类型，而无须关心目标数据库所用的确切 SQL 类型的名称

SQLException : 这个类处理发生在数据库应用程序的任何错误。

常见的JDBC异常

* java.sql.SQLException——这是JDBC异常的基类。
* java.sql.BatchUpdateException——当批处理操作执行失败的时候可能会抛出这个异常。这取决于具体的JDBC驱动的实现，它也可能直接抛出基类异常java.sql.SQLException。



## 关键源码分析

加载驱动类（将驱动类实例化到虚拟机中）Class.forName("×××Driver")

|  |
| --- |
| Mysql-connector-java中驱动类Driver  public class Driver extends NonRegisteringDriver implements java.sql.Driver {  public Driver() throws SQLException {  }  static {  try {  //将new Driver()注册到DriverManager中给DriverManager管理  DriverManager.registerDriver(new Driver());  } catch (SQLException var1) {  throw new RuntimeException("Can't register driver!");  }  } } |
| DriverManager中的registerDriver方法  public static synchronized void registerDriver(java.sql.Driver driver,  DriverAction da)  throws SQLException {   /\* Register the driver if it has not already been added to our list \*/  if(driver != null) {  registeredDrivers.addIfAbsent(new DriverInfo(driver, da));  } else {  // This is for compatibility with the original DriverManager  throw new NullPointerException();  }  println("registerDriver: " + driver); } |

获取连接Connection conn = DriverManager.getConnection(DB\_URL,USER,PASS);

|  |
| --- |
| getConnection方法核心代码  for(DriverInfo aDriver : registeredDrivers) {  // If the caller does not have permission to load the driver then  // skip it.  if(isDriverAllowed(aDriver.driver, callerCL)) {  try {  println(" trying " + aDriver.driver.getClass().getName());  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());  } } |

Jdbc4.0版本后不手动注册驱动也能使用JDBC [ 去除class.forName ]

|  |
| --- |
| riverManager的源码中可以看到一个静态块 static {  loadInitialDrivers();  println("JDBC DriverManager initialized"); }  private static void loadInitialDrivers() {  String drivers;  try {  drivers = AccessController.doPrivileged(new PrivilegedAction<String>() {  public String run() {  return System.getProperty("jdbc.drivers");  }  });  } catch (Exception ex) {  drivers = null;  }  AccessController.doPrivileged(new PrivilegedAction<Void>() {  public Void run() {  //加载类路径下所有jar包中META-INF/services/java.sql.Driver文件中定义的类  ServiceLoader<Driver> loadedDrivers = ServiceLoader.load(Driver.class);  Iterator<Driver> driversIterator = loadedDrivers.iterator();  try{  while(driversIterator.hasNext()) {  driversIterator.next();  }  } catch(Throwable t) {  // Do nothing  }  return null;  }  }); |

## JDBC事务支持

Jdbc api 对数据库事务的操作都在Connection 中

|  |
| --- |
| public interface Connection extends Wrapper, AutoCloseable {  将此连接的自动提交模式设置为给定状态。如果连接处于自动提交模式，则其所有SQL语句将作为单独的事务执行并提交。否则，它的SQL语句被分组为事务，这些事务通过调用方法 commit 或方法 rollback 来终止。默认情况下，新连接处于自动提交状态  void setAutoCommit(boolean autoCommit) throws SQLException;  检索此Connection对象的当前自动提交模式。  boolean getAutoCommit() throws SQLException;  自上次commit / rollback永久更改以来所做的所有更改都会释放此Connection对象当前持有的所有数据库锁。此方法仅在禁用自动提交模式时使用。  void commit() throws SQLException;  撤消当前事务中所做的所有更改，并释放此Connection对象当前持有的任何数据库锁。只有在禁用自动提交模式时才应使用此方法。  void rollback() throws SQLException;  } |

## JDBC不足

尽管JDBC在JAVA语言层面实现了统一，但不同数据库仍旧有许多差异。为了更好地实现跨数据库操作，于是诞生了Hibernate、mybatis、spring-jdbc等项目，对JDBC的再封装，实现了对数据库操作更宽泛的统一和更好的可移植性。