杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

JDBC Course

Brighten Your Way And Raise You Up.

Goals

- Create a multitier database application using the Java programming language and the JDBC API
- Map an object-oriented design to a relational database

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Module 1

JDBC Overview

Brighten Your Way And Raise You Up.

The Ways connect to Databases

- **ODBC—Open Database Connectivity**
  - **A C-based interface to SQL-based database engines, provides a consistent interface for communicating with a database and for accessing database metadata**
- **JDBC**
  - **A Java version ODBC**

JDBC API?

- The JDBC API is:
  - A standard data access interface to a wide range of relational databases
  - A set of classes and interfaces that are part of the Java programming language
- JDBC enables:
  - Connecting to a database
  - Sending a string SQL query to the database
  - Processing the results

JDBC API?

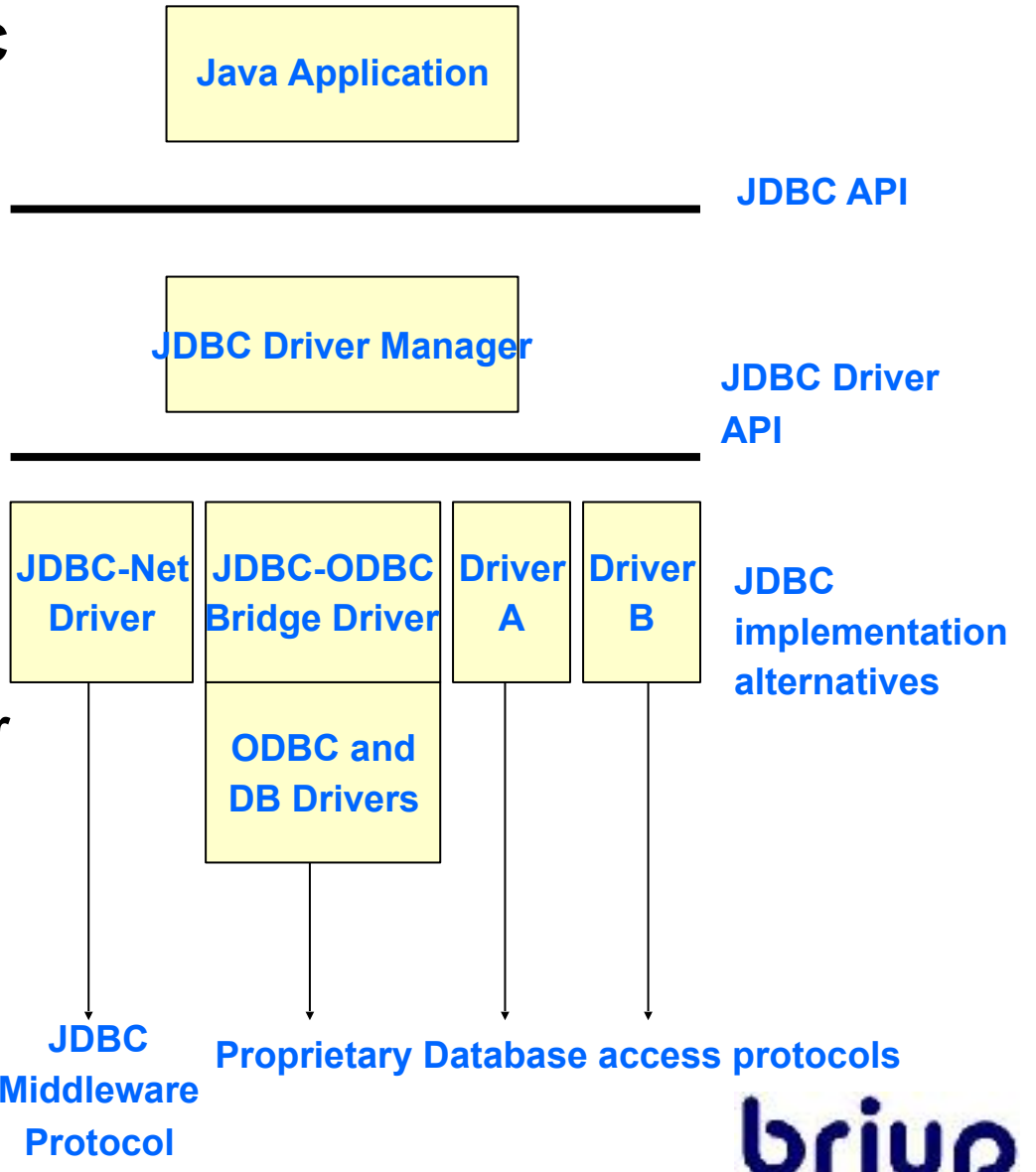
- The JDBC API has two main parts:
  - Java application developers' interface for Java programming language developers
  - JDBC driver developers' implementation interface
- Concentrating on the developer interface

JDBC Driver

- A collection of classes implementing JDBC classes and interfaces
- Providing a class that has implemented the java.sql.Driver interface

Four Types of JDBC Driver

- JDBC-ODBC bridge plus ODBC driver
  - Provides JDBC access via ODBC drivers.
- Native-API partly-Java driver
  - Converts JDBC calls into calls on the native
  - client API of a specific RDBMS
- JDBC-net pure Java driver
  - Translates JDBC calls into a DBMS independent net protocol, which is then translated to a DBMS protocol by a server
- Native protocol pure Java driver
  - Converts JDBC calls directly into the network protocol used by DBMS



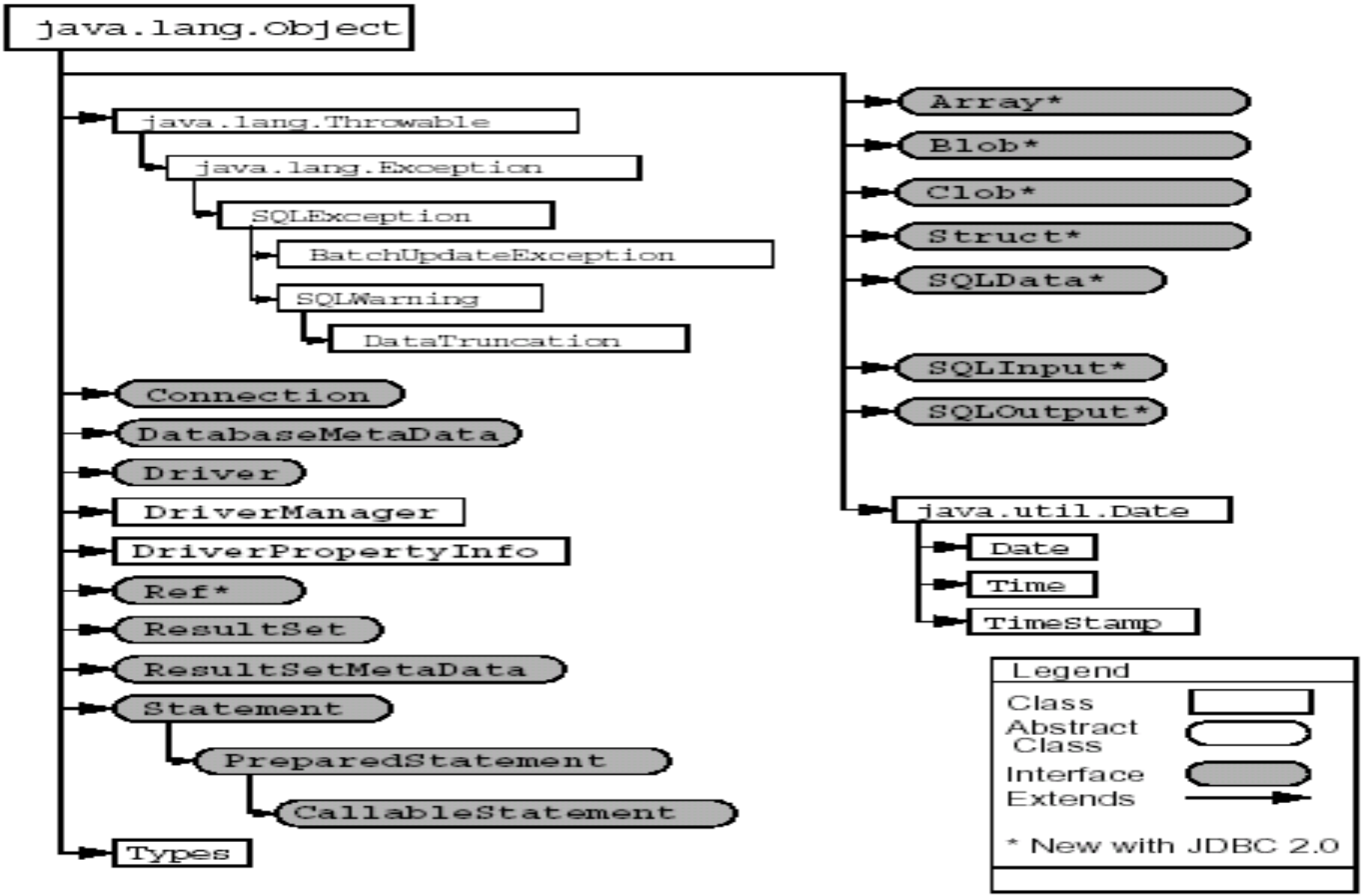
JDBC Developer Interface

- java.sql—Primary features of JDBC in Java 2 platform, standard edition(J2SE)
- javax.sql—Extended functionality in Java 2 platform, enterprise edition(J2EE)

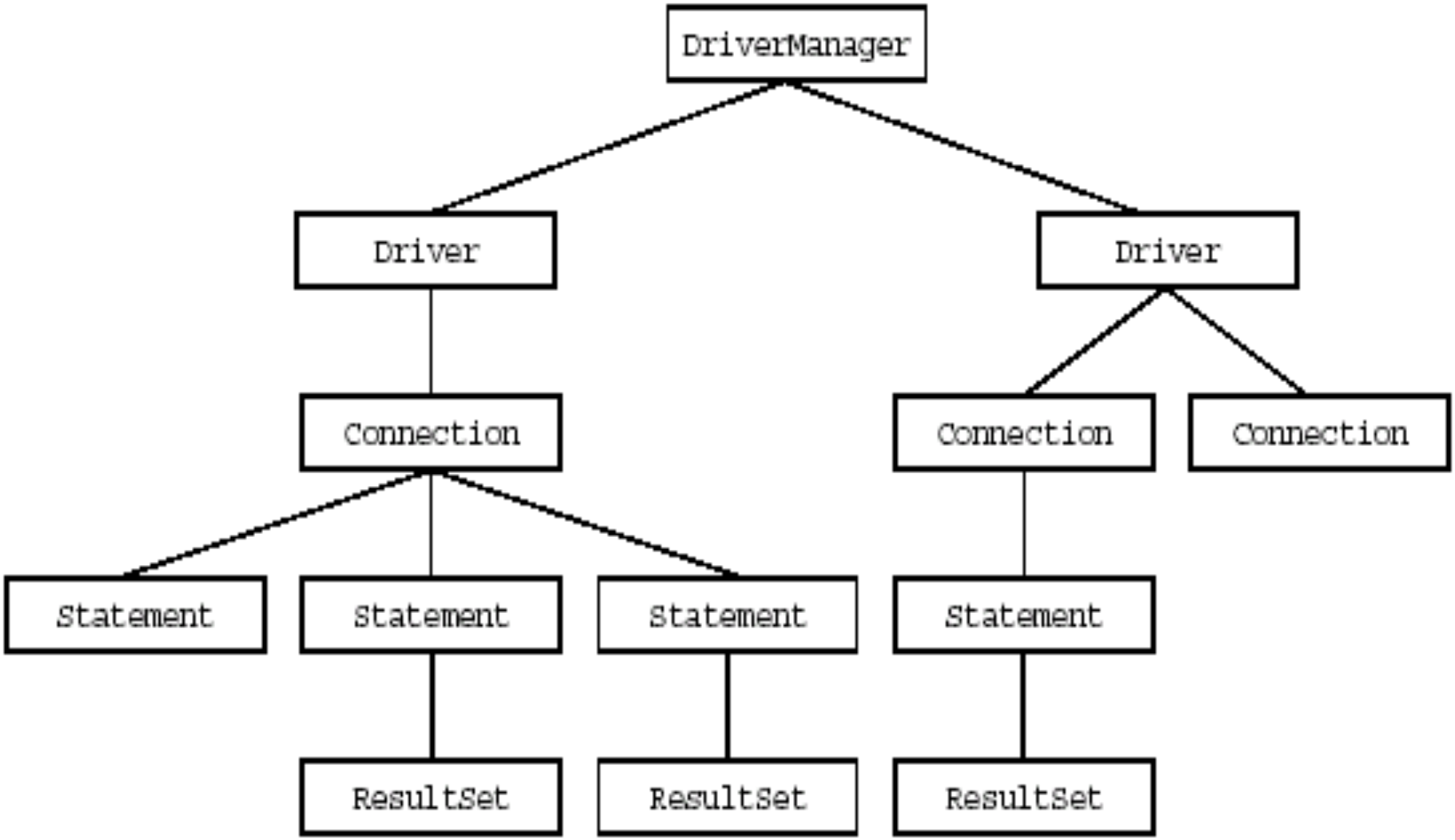
Interfaces and Classes

- Driver
- *DriverManager*
- Connection
- Statement
- PreparedStatement
- CallableStatement
- ResultSet
- DatabaseMetadata
- ResultSetMetadata
- *Types*

Interfaces and Classes



JDBC Flow


DriverManager
Driver
Driver
Connection
Connection
Connection
Statement
Statement
Statement
Statement
ResultSet
ResultSet
ResultSet

Identifying a Database Using a URL

- Identifies a database so that the correct driver can recognize and establish a connection with it
- Is determined by vendor
- Contains encoded information
- Allows for a level of indirection
- Sytax
  - jdbc:subprotocol:subname
  - example: jdbc:odbc:dbname

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Module 2

Using JDBC

Brighten Your Way And Raise You Up.

Creating a Basic JDBC Application

1. Registering a driver
2. Establishing a connection to the database
3. Creating a statement
4. Executing a SQL
5. Processing the results
6. Closing down JDBC objects

The First Sample—FirstJDBCPro

```
import    java.sql.DriverManager;
import    java.sql.Driver;
import    java.sql.Connection;
import    java.sql.Statement;
import    java.sql.PreparedStatement;
import    java.sql.ResultSet;
import    java.sql.Date;
import    java.sql.SQLException;
import    java.util.GregorianCalendar;
import    java.util.Calendar;

public   class   FirstJDBCPro {
      public  static void main(String[] args){
       if(args.length < 3){
         System.out.println("Usage: java -Djdbc.drivers=<jdbc drivers> "
                        + FirstJDBCPro.class.getName()
                        + " dburl user password"
                        );
         System.exit(1);
       }
            Connection con = null;
            Statement stm = null;
            PreparedStatement pstm = null;
            ResultSet rs = null;
```

The First Sample—FirstJDBCPro(Cont.)

```
try{
                        //1. Register jdbc drivers
                        //
Class.forName("oracle.jdbc.driver.OracleDriver");
                        //Driver driver = new
oracle.jdbc.driver.OracleDriver();
                        //DriverManager.registerDriver(driver);
                        //2. Create database connection through
DriverManager
                        con = DriverManager.getConnection(args[0],
args[1], args[2]);
                        /*con = DriverManager.getConnection(
                                "jdbc:oracle:thin:@localhost:1521:XE",
                                "briup", "briup");
                         */
                  //3. Prepare statement
                        /*pstm = con.prepareStatement(
                          "insert into account(accountno, name, balance,
opendate) values(?, ?, ?, ?)");
                        Calendar calendar = new GregorianCalendar();
                        Date opendate = new
Date(calendar.getTimeInMillis());
                        for(int i = 0; i < 5; i++){
                                pstm.setString(1, "000" + i);
                                pstm.setString(2, "acc0"   + i);
                                pstm.setDouble(3, 10000);
```

The First Sample—FirstJDBCPro (Cont.)

```
        stm = con.createStatement();
                    //4. Execute statement
                    rs = stm.executeQuery(
                      "select accountno, name, balance, opendate  from account"
                    );

                    //5. handle result
                    while(rs.next()){
                          System.out.print("accountno: " + rs.getString(1));
                          System.out.print("\tname: " + rs.getString(2));
                          System.out.print("\tbalance: " + rs.getDouble(3));
                          System.out.println("\topendate: " + rs.getDate(4));
                    }
                }catch(SQLException e){ e.printStackTrace();}
                //catch(ClassNotFoundException ce){ ce.printStackTrace();}
                finally{//6. release resources
                    if(rs != null) try{ rs.close(); }catch(Exception e){}
                    if(stm != null) try{ stm.close(); }catch(Exception e){}
                    if(pstm != null) try{ pstm.close(); }catch(Exception e){}
                    if(con != null) try{ con.close(); }catch(Exception e){}
                }
        }
}
```

Step 1—Registering a Driver

- The driver is used to connect to the database
- The JDBC API uses the first driver it finds that can successfully connect to the given URL
- You can load multiple drivers at the same time
- Register a driver
  - Using the class loader
  - Instantiating a driver
  - Using the jdbc.drivers property

Using the Class Loader

- To communicate with a database engine using the JDBC API, create an instance of the JDBC driver
- Syntax

  Class.forName(driverName);
- Example

  Class.forName(“oracle.jdbc.driver.OracleDriver*”);*

Instantiating a Driver

- **If you need an explicit reference to the driver object, use the new key word**
- **Syntax**

  Driver drv = new DriverConstructor();

  DriverManager.registerDriver(drv);
- **Example**

  Driver drv = new oracle.jdbc.driver.OracleDriver();

  DriverManager.registerDriver(drv);

Defining the JDBC Drivers Property

- **Define the JDBC driver property as a list of driver classes name**
- **Syntax**

  jdbc.drivers = driverName[:driverName]
- **Example**

  jdbc.drivers=oracle.jdbc.driver.OracleDriver

Using the –D Option with the java Command

- **Syntax**

  java –Djdbc.drivers=driverName[:driverName]

- **Example**

  java -Djdbc.drivers=oracle.jdbc.driver.OracleDriver

Often Used JDBC Drivers

- *JDBC-ODBC: sun.jdbc.odbc.JdbcOdbcDriver*
- **Oracle: oracle.jdbc.driver.OracleDriver**
- *Cloudscape: com.cloudscape.cor.RmiJdbcDriver*
- *PointBase: com.pointbase.jdbc.jdbcUniversalDriver*
- *Weblogic MS-SQL driver: weblogic.jdbc.mssqlserver4.Driver*
- **MySQL: com.mysql.jdbc.Driver**

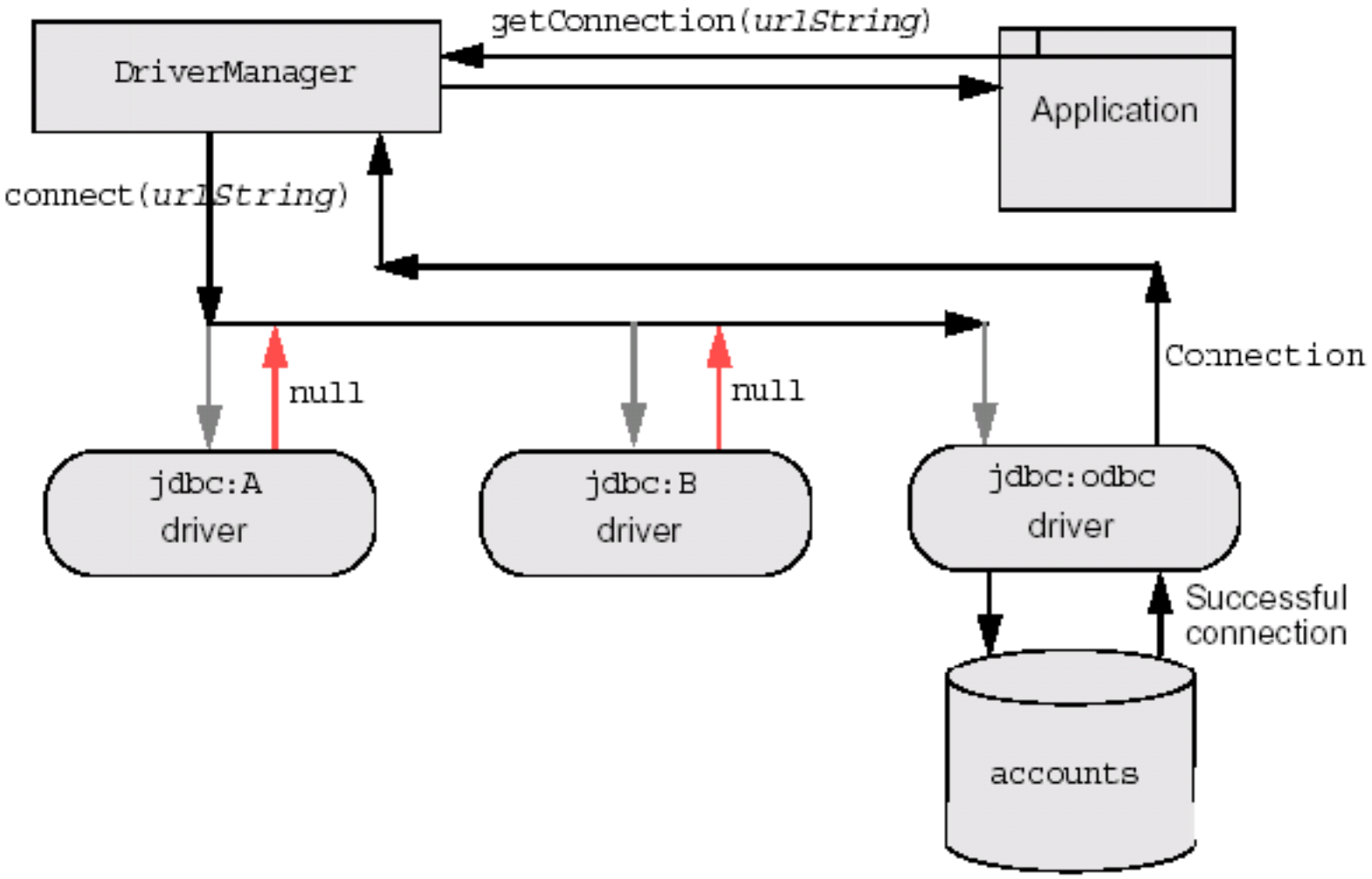
Step 2—Establishing a Connection to the Database

- DriverManager calls getConnection(urlString), which calls Driver.connect(urlString)
- The URL is parsed
- When a driver responds positively to the database URL, the DriverManager makes a connection
- If the driver does not match, null is returned and the next driver in the vector is checked
- If a connection is not made, a SQLException is thrown

Often Used JDBC URLs Syntax

- JDBC-ODBC: *jdbc:odbc:<DB>*
- Oracle: *jdbc:oracle:oci:@<SID> or jdbc:oracle:thin:@<SID>*
- Cloudscape: *jdbc:cloudscape:rmi:<DB>*
- PointBase: *jdbc:pointbase:server://host:[<PORT>]/<DB>*
- Weblogic MS-SQL URL:
  jdbc:weblogic:mssqlserver4:<DB>@<HOST>:<PORT>
- MySQL URL:
  jdbc:mysql://<HOST>:<PORT>/
  <DB>?useUnicode=true&characterEncoding=gb2312

Establishing a Connection to the Database



The DriverManger.getConnection Method

- **Method**

  getConnection(String url)

  getConnection(String url, java.util.properties info)

  getConnection(String url, String user, String passwd)

- **Syntax**

  Connection con = DriverManager.getConnection(arguments)

- **Sample**

  Connection con =

  DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "briup", "briup");

The Driver.connect Method

- Makes a direct call to a specific Driver object
- Avoids the problems of standard approach

The Driver.connect Method

- **Syntax**

  Connection con = Driver.connect(urlString, propertiesInfo)

- **Example**

```
Driver drv = new oracle.jdbc.driver.OracleDriver();
Connection con = null;
Properties userPasswd = new Properties();
userPasswd.setProperty("user", "briup");
userPasswd.setProperty("password", "briup");
try {
con = drv.connect("jdbc:oracle:thin:@localhost:1521:XE", userPasswd);
}catch(SQLException e){}
```

Creating a Statement

- Statement
- PreparedStatement
- CallableStatement

The Statement Object

- Get a Statement object from the Connection.createStatement() method

The Statement Object

- **Syntax**

Statement stm = connection.createStatement();

- **Examples**

```
Statement stm = null;
ResultSet rs = null;
try{
        stm = connection.crateStatement();
        rs = stm.executeQuery("select accountno, name, balance, opendate  from
account");
}catch(SQLException e) {}
```

The PreparedStatement Object

- A precompiled SQL statement that is more efficient than repeatedly calling the same SQL statement
- Extends the Statement interface

The PreparedStatement Object

- **Syntax**

PreparedStatement pstm = connection.prepareStatement(sqlString);

- **Examples**

```
PreparedStatement pstm = null; ResultSet rs = null;
try {
        pstm = connection.prepareStatement("select * from student where id=?");
        pstm.setString(1, "8613034");
        rs = pstm.executeQuery();
}catch(SQLException e){}
```

The CallableStatement Object

- Let you execute Non-SQL statements against the database
- Extends PreparedStatement

The CallableStatement Object

- **Syntax**

CallableStatement cstm =connection.prepareCall(sqlString);

- **Examples**

```
CallabeStatement cstm = null;
try{
  cstm = connection.prepareCall("{call return_student(?, ?)}");
   cstm.setString(1, "8613034");
   cstm.registerOutParameter(2, Types.REAL);
   cstm.execute();
   float gpa = cstm.getFloat(2);
}catch(SQLException e){}
```

Comparing Statement Interfaces

	`Statement`	`PreparedStatement`	`CallableStatement`
Where is code created?	Client	Client	Server
Where is code stored?	Client	Server	Server
Technologies for writing code	Java programming language, SQL operations	Java programming language, SQL operations	The procedural language for the database on the target platform, such as PLSQL (the procedural language for SQL)
Configurability	High	High the first time, then low	Low
Portability	High	High, provided the database supports `PreparedStatement`	Low
Efficiency for transferring data	Low	Low the first time, then high	High

Step 4—Executing the SQL Statement

- The SQL statement is passed unaltered to the underlying database connection
- The result is a table of data accessible through java.sql.ResultSet
- The statement is coupled with the corresponding ResultSet

The Statement Interface

- executeQuery(sqlString)
- executeUpdate(sqlString)
- execute(sqlString)

The Statement Interface

executeQuery(sqlString)

- **Syntax**

Connection con = DriverManger.getConnection(urlString);

Statement stm = con.createStatement();

ResultSet rs = stm.excuteQuery(sqlString);

- **Examples**

Connection con =

DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "briup", "briup");

Statement stm = con.createStatement();

*ResultSet rs = stm.executeQuery("select * from student");*

Step 5—Processing the Results

- To retrieve data from the ResultSet object, use its assessor methods
- Retrieve these values using a column name or index
- A ResultSet
  - Keeps a cursor pointed to the current row
  - Is initially positioned before its first row

Step 5—Processing the Results

- **Syntax**

while(rs.next()){System.out.println(rs.getXXXMethod(column);}

- **Examples**

```
while(rs.next()){
    System.out.print("accountno: " + rs.getString(1));
    System.out.print("\tname: " + rs.getString(2));
    System.out.print("\tbalance: " + rs.getDouble(3));
    System.out.println("\topendate: " + rs.getDate(4));
}
```

Step 6—Closing Down JDBC Objects

- **Connection**
- **Statement**
- **ResultSet**

```
finally{//6. release resources
        if(rs != null) try{ rs.close(); }catch(Exception e){}
        if(stm != null) try{ stm.close(); }catch(Exception e){}
        if(pstm != null) try{ pstm.close(); }catch(Exception e){}
        if(con != null) try{ con.close(); }catch(Exception e){}
}
```

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5-8层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

MyBatis

Brighten Your Way And Raise You Up.

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5-8层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Chap 1

Mybatis入门

Brighten Your Way And Raise You Up.

1.1 mybatis是什么

mybatis是一个简化和实现了**java**数据持久化层的开源框架**My Batis**是一个简化和实现了 **Java** 数据持久化层**(persistence layer)**的开源框架，它抽象了大量的 **JDBC** 冗余代码，并提供了一个简单易用的**API**和数据库交互。

1.2 Git是什么

是一个免费开源的分布式版本控制系统，被用于高速有效地处理大大小小项目中所有文件,在软件开发中使用的其他版本控制软件类似与**SVN**、**VSS**、**CVS**等等

1.3 github是什么

作为一个分布式的版本控制系统，在**Git**中并不存在主库这样的概念，每一份复制出的库都可以独立使用，任何两个库之间的不一致之处都可以进行合并。**github**以托管各种**git**库，并提供一个**web**界面，可以说是一款易于使用的**git**图形客户端。我们熟知的**spring**、**struts**、**Hibernate**等框架的源代码在**github**上面都可以找到其源代码

1.4 iBATIS和MyBatis

iBATIS一词来源于**“internet”**和**“abatis”**的组合，是一个在**2002**年发起的开放源代码项目。于**2010**年**6**月**16**号被谷歌托管，改名为**MyBatis**。

mybatis在**github**中的地址

https://github.com/mybatis/mybatis-3

最新版本的**mybatis**的下载地址

https://github.com/mybatis/mybatis-3/releases

doc文档

http://www.mybatis.org/mybatis-3/

1.5 MyBatis的优势

1.5.1 它消除了大量的**JDBC**冗余代码

1.5.2 它有低的学习曲线

1.5.3 它能很好地与传统数据库协同工作

1.5.4 它可以接受**SQL**语句

1.5.5 它提供了与**Spring**框架的集成支持

1.5.6 它提供了与第三方缓存类库的集成支持

1.5.7 它引入了更好的性能

1.6 mybatis的jar包

mybatis的核心包只有一个**mybatis-3.x.0.jar,**另外还有一些【可选】的依赖包(日志、代理等所需要的),在下载的压缩包中可以找到.

1.7 mybatis框架中的配置文件

第一种:**mybatis**的配置文件: **mybatis-config.xml**,其中包括数据库连接信息，类型别名等等

特点:

名字一般是固定的

位置是**src**下面

第二种:**mybatis**的映射文件:**XxxxxMapper.xml**,其中包括**Xxxx**类所对应的数据库中表的各种增删改查的**sql**语句

特点:

名字一般为**XxxxMapper.xml**,**Xxxx**是对应类的名字

1.8 mybatis中的映射接口

mybatis中除了必须的**jar**包、各种**xml**配置文件之外,一般还需要有调用**sql**语句执行的接口**XxxxMapper.java**

示例:

```
public interface StudentMapper{
        List<Student> findAllStudents();
        Student findStudentById(Integer id);
        void insertStudent(Student student);
}
```

注意接口中的方法的作用是:对**XxxxMapper.xml**中的**sql**语句进行映射

1.9 SqlSession和SqlSessionFactory

SqlSession接口的实现类对象是**mybatis**中最重要的一个对象,我们可以使用该对象来调用**XxxxMapper.java**接口中的方法,从而调用到**XxxxMapper.java**接口中方法所映射的**sql**语句。**sqlSessionFactory**接口的实现类对象是一个工厂对象,专门负责来产生**SqlSession**对象的

例如:

```
InputStream inputStream =
Resources.getResourceAsStream("mybatis-config.xml");

SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);

SqlSession sqlSession =
sqlSessionFactory.openSession();
```

1.9 SqlSession和SqlSessionFactory

```
//第一种 通过XxxxMapper接口的调用

//动态获得XxxxMapper接口的实现类

StudentMapper studentMapper =
sqlSession.getMapper(StudentMapper.class);

studentMapper.insertStudent(newStudent(1,"tom","123@.qq.com
"));


//第二种 执行调用XxxxMapper.xml中写好的sql语句
sqlSession.selectOne("com.briup.pojo.StudentMapper.findStuden
tById",1);
```

1.10 实例练习

根据具体要求，来完成一个插入数据的例子，感受**mybatis**整个过程的各个细节

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5-8层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Chap 2

Mybatis配置文件

Brighten Your Way And Raise You Up.

2.1 mybatis-config.xml文件

配置文件中常见的元素有:

environments元素

dataSource元素

transactionManager元素

properties元素

typeAliases元素

typeHandlers元素

settings元素

mappers元素

2.1.1 environments元素

environments是配置**mybatis**当前工作的数据库环境的地方**MyBatis**支持配置多个**dataSource**环境，可以将应用部署到不同的环境上，如**DEV(**开发环境**)**，**TEST**（测试换将），**QA**（质量评估环境）**,UAT(**用户验收环境**),PRODUCTION**（生产环境），可以通过将默认**environments**值设置成想要的**environment**的**id**值。

2.1.2 dataSource元素

dataSource的类型**type**属性可以配置成其内置类型之一，如**UNPOOLED**，**POOLED**，**JNDI**。如果将类型设置成**UNPOOLED**，**MyBatis**会为每一个数据库操作创建一个新的连接，并关闭它。该方式适用于只有小规模数量并发用户的简单应用程序上。

2.1.3 transactionManager元素

MyBatis支持两种类型的事务管理器：**JDBC** 和 **MANAGED. JDBC**事务管理器被用作当应用程序负责管理数据库连接的生命周期（提交、回退等等）的时候。当你将**TransactionManager** 属性设置成**JDBC**，**MyBatis**内部将使用**JdbcTransactionFactory**类创建**TransactionManager**。例如，部署到**ApacheTomcat**的应用程序，需要应用程序自己管理事务。

2.1.4 properties元素

属性配置元素**properties**可以将配置值写死到**mybatis-config.xml**中,也可以具体到一个属性文件中,并且使用属性文件的**key**名作为占位符.

在上述的配置中，我们将数据库连接属性具体化到了**application.properties**文件中，并且为**driver**，**URL**等属性使用了占位符.

在**applications.properties**文件中配置数据库连接参数，如下所示:

jdbc.driverClassName=oracle.jdbc.driver.OracleDriver

jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:XE

jdbc.username=briup

jdbc.password=briup

2.1.5 typeAliases元素

在**SQLMapper**配置文件中，对于**resultType**和**parameterType**属性值，我们需要使用**JavaBean** 的完全限定名。

我们可以为完全限定名取一个别名（**alias**），然后其需要使用完全限定名的地方使用别名，而不是到处使用完全限定名。如下例子所示，为完全限定名起一个别名：

```
<type Aliases>
 <type Alias alias="Student" type="com.briup.pojo.Student" />
</type Aliases>
```

2.1.6 typeHandlers元素

如果当前插入到数据库中的数据存在一些用户自定义类型的对象的时候,比如**Address**类,为了让**mybatis**能够找到遇到**Address**字段的时候应该把它当做一个什么类型的数据来处理(如当做**String**),那么我们就需要自定义一个**TypeHandler**并将其进行配置

2.1.7 settings元素

为满足应用特定的需求，**MyBatis**默认的全局参数设置可以使用该标签元素进行配置从而将其默认值覆盖掉

2.1.8 mappers元素

SQL Mapper文件中包含的**SQL**映射语句将会被应用通过使用其标签中的**id**值来执行。我们需要在**mybatis-config.xml**文件中配置**SQL Mapper**文件的位置。

2.2自定义MyBatis日志

MyBatis使用其内部LoggerFactory作为真正的日志类库使用的门面。其内部的LaggerFactory会将日志记录任务委托给如下的所示某一个日志实现，日志记录优先级由上到下顺序递减：

- SLF4J
- Apache Commons Logging
- Log4j2
- Log4j
- JDK logging

2.2自定义MyBatis日志

如果**MyBatis**未发现上述日志记录实现，则**MyBatis**的日志记录功能无效,如果你的运行环境中，在**classpath**中有多个可用的日志类库，并且你希望**MyBaits**使用某个特定的日志实现，你可以通过调用以下其中一个方法：

org.apache.ibatis.logging.LogFactory.useSlf4jLogging();

org.apache.ibatis.logging.LogFactory.useLog4JLogging();

org.apache.ibatis.logging.LogFactory.useLog4J2Logging();

org.apache.ibatis.logging.LogFactory.useJdkLogging();

org.apache.ibatis.logging.LogFactory.useCommonsLogging();

org.apache.ibatis.logging.LogFactory.useStdOutLogging();

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Chap 3

Mybatis映射文件

Brighten Your Way And Raise You Up.

关系型数据库和**SQL**是经受时间考验和验证的数据存储机制。和其他的**ORM** 框架如**Hibernate**不同，**My Batis**鼓励开发者可以直接使用数据库，而不是将其对开发者隐藏，因为这样可以充分发挥数据库服务器所提供的**SQL**语句的巨大威力。

与此同时，**MyBaits**消除了书写大量冗余代码的痛苦，它让使用**SQL**更容易。在代码里直接嵌套**SQL**语句是很差的编码实践，并且维护起来困难。**MyBaits**使用了映射器配置文件或注解来配置**SQL**语句。

3.1映射器配置文件和映射器接口

```
<mapper namespace="com.briup.mappers.StudentMapper">
        <select id="findStudentById" parameterType="int" result
Type="Student">
                select stud_id as studId, name, email, dob
                from Students where stud_id=#{studId}
        </select>
</mapper>
```

3.1映射器配置文件和映射器接口

通过下列代码调用**findStudentById**映射的**SQL**语句:

```
SqlSession sqlSession =
MyBatisSqlSessionFactory.openSession();

Student student =
sqlSession.selectOne("com.briup.mappers.StudentMapper.findStudentById", studId);
```

3.1映射器配置文件和映射器接口

我们还可以这样调用:

```
public interface StudentMapper{
        Student findStudentById(Integer id);
}
```

```
SqlSession sqlSession =
MyBatisSqlSessionFactory.openSession();
StudentMapper studentMapper =
sqlSession.getMapper(StudentMapper.class);
```

3.2 映射语句

Insert语句

Update语句

Delete语句

Select语句

3.2.1 INSERT 插入语句

```
<insert id="insertStudent" parameterType="Student">
        INSERT INTO STUDENTS(STUD_ID,NAME,EMAIL, PHONE)
VALUES(#{stud Id},#{name},#{email},#{phone})
</insert>
```

```
public interface Student Mapper{
        int insert Student(Student student);
}
StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
int count = mapper.insertStudent(student);
```

3.2.2 UPDATE更新语句

```
<update id="updateStudent" parameterType="Student">
        UPDATE STUDENTS SET NAME=#{name},EMAIL=#{email},
PHONE=#{phone}
        WHERE STUD_ID=#{studId}
</update>
public interface Student Mapper{
        int updateStudent(Student student);
}
StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
int noOfRowsUpdated = mapper.updateStudent(student);
```

3.2.3 DELETE 删除语句

```
<delete id="deleteStudent" parameterType="int">
        DELETE FROM STUDENTS WHERE STUD_ID=#{stud Id}
</delete>
public interface Student Mapper{
        int deleteStudent(stuId);
}
StudentMapper mapper =
sqlSession.getMapper(StudentMapper.class);
int noOfRowsDeleted = mapper.deleteStudent(studId);
```

3.2.4 SELECT查询语句

```
<select id="findStudentById" parameterType="int"
                        resultType="Student">
SELECT STUD_ID, NAME, EMAIL, PHONE  FROM STUDENTS
            WHERE STUD_ID=#{stud Id}
</select>
 public interface Student Mapper{
             Student findStudentById(Integer studId);
}
StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
Student student = mapper.findStudentById(studId);
```

3.3 结果集映射 ResultMaps

ResultMaps被用来将**SELECT**语句的结果集映射到**JavaBeans**的属性中。我们可以定义结果集映射**ResultMaps**并且在一些**SELECT**语句上引用**resultMap**。**MyBatis**的结果集映射 **ResultMaps**特性非常强大，你可以使用它将简单的**SELECT**语句映射到复杂的一对一、一对多关系的**SELECT**语句上。

3.3.1简单ResultMap

一个映射了查询结果和**Student**这个**JavaBean**的简单的**resultMap**定义如下:

```
<resultMap id="StudentResult" type="com.briup.pojo.Student">
        <id property="studId" column="stud_id" />
        <result property="name" column="name" />
        <result property="email" column="email" />
        <result property="phone" column="phone" />
</resultMap>
```

3.3.1简单ResultMap

```
<select id="findAllStudents" resultMap="StudentResult">
        SELECT * FROM STUDENTS
</select>
<select id="findStudentById" parameterType="int"
resultMap="StudentResult">
        SELECT * FROM STUDENTS WHERE STUD_ID=#{studId}
</select>
```

3.3.1简单ResultMap

resultMap的**id**值应该在此名空间内是唯一的,并且**type**属性是完全限定类名或者是返回类型的别名。

<result>子元素被用来将一个**resultset**列映射到**JavaBean**的一个属性中。

<id>元素和**<result>**元素功能相同，不过**<id>**它被用来映射到唯一标识属性，用来区分和比较对象（一般和主键列相对应）。在**<select>**语句中，我们使用了**resultMap**属性，而不是**resultType**来引用**StudentResult** 映射。当**<select>**语句中配置了**resutlMap**属性**MyBatis**会使用此数据库列名与对象属性映射关系来填充**JavaBean**中的属性。

注意:**resultType**和**resultMap**二者只能用其一，不能同时使用。

3.3.2 拓展 ResultMap

我们可以从从另外一个**<resultMap>**，拓展出一个新的**<resultMap>**，这样，原先的属性映射可以继承过来，以实现:

```
<resultMap type="Student" id="StudentResult">
        <id property="stud Id" column="stud_id" />
        <result property="name" column="name" />
        <result property="email" column="email" />
        <result property="phone" column="phone" />
</resultMap>
```

3.3.2 拓展 ResultMap

```
<!-- Student类中又新增加了一个属性,该属性的类型是Address -->

<!-- 自定义类Address,类中也有多个属性,同时数据库中ADDRESSES表与其对应 -->

<resultMap type="Student" id="StudentWithAddressResult"
extends="StudentResult">

<result property="address.addr Id" column="addr_id" />

 <result property="address.street" column="street" />

<result property="address.city" column="city" />

 <result property="address.state" column="state" />

 <result property="address.zip" column="zip" />

<result property="address.country" column="country" />

</resultMap>
```

3.4一对一映射

元素**<association>**被用来导入“有一个”**(has-one)**类型的关联。在上述的例子中，我们使用了**<association>**元素引用了另外的在同一个**XML**文件中定义的**<resultMap>**。

使用**Student**和**Address**的例子来完成一对一的例子。

3.5 一对多映射

一个讲师**TUTORS**可以教授一个或者多个课程**COURSE**。这意味着讲师和课程之间存在一对多的映射关系。.

使用**TUTORS**和**COURSE**的例子来完成一对一的例子。

<collection>元素被用来将多行课程结果映射成一个课程**Course**对象的一个集合。和一对一映射一样，我们可以使用【嵌套结果**ResultMap**】和【嵌套查询**Select**】语句两种方式映射实现一对多映射。。

3.6 多对多映射

对于在**mybatis**中的多对多的处理,其实我们可以参照一对多来解决。

使用**Student**和**Course**的例子来完成一对一的例子。一个学生可以选择多门课程，一个课程也可以让多个学生选择。

3.7 动态SQL

有时候，静态的**SQL**语句并不能满足应用程序的需求。我们可以根据一些条件，来动态地构建 **SQL**语句。

例如，在**Web**应用程序中，有可能有一些搜索界面，需要输入一个或多个选项，然后根据这些已选择的条件去执行检索操作。在实现这种类型的搜索功能，我们可能需要根据这些条件来构建动态的**SQL**语句。如果用户提供了任何输入条件，我们需要将那个条件添加到**SQL**语句的**WHERE**子句中。**MyBatis**通过使用**<if>,<choose>,<where>,<foreach>,<trim>**元素提供了对构造动态**SQL**语句的高级别支持。

3.7.1 If 条件

当用户点击搜索按钮时，我们需要显示符合以下条件的成列表：

特定讲师的课程

课程名

包含输入的课程名称关键字的课程；如果课程名称输入为空，则取所有课程

在开始时间和结束时间段内的课程

我们可以对应的映射语句，如下所示：

3.7.1 If 条件

```
<select id="searchCourses" parameterType="hashmap" resultMap="CourseResult">
SELECT * FROM COURSES
WHERE TUTOR_ID= #{tutorId}
<if test="courseName != null">
        AND NAME LIKE #{courseName}
</if>
<if test="startDate != null">
        AND START_DATE >= #{startDate}
</if>
<if test="endDate != null">
        AND END_DATE <= #{endDate}
</if>
```

3.7.2 choose,when 和 otherwise 条件

```
SELECT * FROM COURSES
<choose>
<when test="searchBy == 'Tutor'">
        WHERE TUTOR_ID= #{tutorId}
</when>
<when test="searchBy == 'CourseName'">
        WHERE name like #{courseName}
</when>
<otherwise>
        WHERE TUTOR start_date >= now()
</otherwise>
</choose>
```

3.7.3 Where 条件

```
SELECT * FROM COURSES
<where>
<if test="tutorId != null ">
        TUTOR_ID= #{tutorId}
</if>
<if test="courseName != null">
        AND name like #{courseName}
</if>
<if test="startDate != null">
        AND start_date >= #{startDate}
</if>
</where>
```

3.7.4 <trim>条件

```
SELECT * FROM COURSES
<trim prefix="WHERE" prefixOverrides="AND | OR">
<if test=" tutorId != null ">
        TUTOR_ID= #{tutorId}
</if>
<if test="courseName != null">
        AND name like #{courseName}
</if>
</trim>
```

3.7.5 foreach 循环

```
SELECT * FROM COURSES
<if test="tutorIds != null">
<where>
<foreach item="tutorId" collection="tutorIds">
        OR tutor_id=#{tutorId}
</foreach>
</where>
</if>
```

3.7.5 foreach 循环

```
SELECT * FROM COURSES
<if test="tutorIds != null">
<where>
        tutor_id IN
<foreach item="tutorId" collection="tutorIds" open="(" separator="," close=")">
        #{tutorId}
</foreach>
</where>
</if>
```

3.7.6 set 条件

<set>元素和**<where>**元素类似，如果其内部条件判断有任何内容返回时，他会插入**SET SQL** 片段。

```
update students
<set>
<if test="name != null">name=#{name},</if>
<if test="email != null">email=#{email},</if>
<if test="phone != null">phone=#{phone},</if>
</set>
where stud_id=#{id}
```

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5-8层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Chap 4

Mybatis特殊功能

Brighten Your Way And Raise You Up.

除了简化数据库编程外，**MyBatis**还提供了各种功能，这些对实现一些常用任务非常有用，比如按页加载表数据，存取**CLOB/BLOB**类型的数据，处理枚举类型值，等等。

4.1处理枚举类型

My Batis支持开箱方式持久化**enum**类型属性。假设**STUDENTS**表中有一列**gender**（性别）类型为 **varchar**，存储**”MALE”**或者**“FEMALE”**两种值。并且，**Student**对象有一个**enum**类型的**gender** 属性

```
public enum Gender {
        FEMALE,MALE
}
```

4.1处理枚举类型

```
<insert id="insertStudent" parameterType="Student">
insert into students(id,name,email,addr_id, phone,gender)
values(#{id},#{name},#{email},#{address.addrId},#{phone},#{gender})
</insert>
```

4.2处理CLOB/BLOB类型数据

BLOB和**CLOB**都是大字段类型，**BLOB**是按二进制来存储的，而**CLOB**是可以直接存储文字的。通常像图片、文件、音乐等信息就用**BLOB**字段来存储，先将文件转为二进制再存储进去。而像文章或者是较长的文字，就用**CLOB**存储.

BLOB和**CLOB**在不同的数据库中对应的类型也不一样:

MySQL中：**clob**对应**text/longtext**，**blob**对应**blob**

Oracle中 ： **clob**对应**clob**，**blob**对应**blob**

4.3 传入多个输入参数

MyBatis中的映射语句有一个**parameterType**属性来制定输入参数的类型。如果我们想给映射语句传入多个参数的话，我们可以将所有的输入参数放到**HashMap**中，将**HashMap**传递给映射语句。同时**MyBatis**还提供了另外一种传递多个输入参数给映射语句的方法。假设我们想通过给定的**name**和**email**信息查找学生信息，定义查询接口如下：

对于映射器中的方法，**MyBatis**默认从左到右给方法的参数命名为**param1**、**param2…**，依次类推。

4.3 传入多个输入参数

```
Public interface StudentMapper{

List<Student> findAllStudentsByNameEmail(String name, String email);

}


<select id="findAllStudentsByNameEmail" resultMap="StudentResult">

        select stud_id, name,email, phone from Students

        where name=#{param1} and email=#{param2}

</select>
```

4.4多行结果集映射成Map

如果你有一个映射语句返回多行记录，并且你想以**HashMap**的形式存储记录的值，使用记录列名作为**key**值，而记录对应值或为**value**值。我们可以使用**sqlSession.selectMap(),**如下所示：

```
<select id="findAllStudents" resultMap="StudentResult">
        select * from Students
</select>
```

```
Map<Integer, Student> studentMap =
sqlSession.selectMap("com.briup.mappers.StudentMapper.findAll
Students", "studId");
```

这里**studentMap**将会将**studId**作为**key**值，而**Student**对象作为**value**值

4.5 分页

使用**RowBounds**对结果集进行分页

有时候，我们会需要跟海量的数据打交道，比如一个有数百万条数据级别的表。由于计算机内存的现实我们不可能一次性加载这么多数据，我们可以获取到数据的一部分。特别是在**Web**应用程序中，分页机制被用来以一页一页的形式展示海量的数据。

MyBatis可以使用**RowBounds**逐页加载表数据。**RowBounds**对象可以使用**offset**和**limit**参数来构建。参数**offset**表示开始位置，而**limit**表示要取的记录的数目。假设如果你想每页加载并显示**25**条学生的记录，你可以使用如下的代码：

4.5 分页

```
<select id="findAllStudents" resultMap="StudentResult">
        select * from Students
</select>
```

然后，你可以加载如下加载第一页数据（前25条）：

```
int offset =0 , limit =25;
RowBounds rowBounds = new RowBounds(offset, limit);
List<Student> = studentMapper.getStudents(rowBounds);
```

若要展示第二页，使用offset=25,limit=25;第三页，则为offset=50,limit=25。

4.6自定义结果集ResultSet处理

MyBatis在将查询结果集映射到**Java Bean**方面提供了很大的选择性。但是，有时候我们会遇到由于特定的目的，需要我们自己处理**SQL**查询结果的情况。**MyBatis**提供了**Result Handler**插件形式允许我们以任何自己喜欢的方式处理结果集**ResultSet**。

假设想从学生的**stud_id**被用作**key**，而**name**被用作**value**的**HashMap**中获取到**student**信息。

对于**sqlSession.select()**方法，我们可以传递给它一个**ResultHandler**的实现，它会被调用来处理**ResultSet**的每一条记录。

```
public interface ResultHandler{
        void handleResult(ResultContext context);
}
```

4.6自定义结果集ResultSet处理

```
sqlSession.select("com.mybatis3.mappers.StudentMapper.findAll
Students", new ResultHandler(){
        public void handleResult(ResultContext context){
        Student student = (Student) context.getResultObject();

        map.put(student.getStudId(), student.getName());

                }

        }

);
```

4.7 缓存

将从数据库中加载的数据缓存到内存中，是很多应用程序为了提高性能而采取的一贯做法。**MyBatis**对通过映射的**SELECT**语句加载的查询结果提供了内建的缓存支持。默认情况下，启用一级缓存；即，如果你使用同一个**SqlSession**接口对象调用了相同的**SELECT**语句，则直接会从缓存中返回结果，而不是再查询一次数据库。

4.7 缓存

我们可以在**SQL**映射器**XML**配置文件中使用**<cache />**元素添加全局二级缓存。当你加入了**<cache />**元素，将会出现以下情况：

所有的在映射语句文件定义的**<select>**语句的查询结果都会被缓存

所有的在映射语句文件定义的**<insert>,<update>** 和**<delete>**语句将会刷新缓存

缓存根据最近最少被使用（**Least Recently Used**，**LRU**）算法管理

缓存不会被任何形式的基于时间表的刷新（没有刷新时间间隔），即不支持定时刷新机制

缓存将存储**1024**个查询方法返回的列表或者对象的引用

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5-8层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Chap 5

Mybatis的注解

Brighten Your Way And Raise You Up.

之前我们都是在映射器**MapperXML**配置文件中配置映射语句的。除此之外**MyBatis**也支持使用注解来配置映射语句。当我们使用基于注解的映射器接口时，我们不再需要在**XML**配置文件中配置了。如果你愿意，你也可以同时使用基于**XML**和基于注解的映射语句。

5.1 在映射器Mapper接口上使用注解

MyBatis对于大部分的基于**XML**的映射器元素（包括**<select>,<update>**）提供了对应的基于注解的配置项。然而在某些情况下，基于注解配置还不能支持基于**XML**的一些元素。

5.2 @Insert

```
@Insert("INSERT INTO
STUDENTS(STUD_ID,NAME,EMAIL,ADDR_ID, PHONE)

VALUES(#{studId},#{name},#{email},#{address.addr
Id},#{phone})")

int insertStudent(Student student);
```

5.2 @Insert

自动生成主键

可以使用 **@Options**注解的**userGeneratedKeys**和**keyProperty**属性让数据库产生 **auto_increment**（自增长）列的值，然后将生成的值设置到输入参数对象的属性中。

```
@Insert("INSERT INTO STUDENTS(NAME,EMAIL,ADDR_ID,
PHONE) VALUES(#{name},#{email},#{address.addr Id},#{phone})")
@Options(useGeneratedKeys = true, keyProperty = "studId")
int insert Student(Student student);
```

5.2 @Insert

有一些数据库如**Oracle**，并不支持**AUTO_INCREMENT**列属性，它使用序列（**SEQUENCE**）来产生主键的值。

我们可以使用 **@SelectKey**注解来为任意**SQL**语句来指定主键值，作为主键列的值。假设我们有一个名为**STUD_ID_SEQ**的序列来生成**STUD_ID**主键值。

例如:

5.2 @Insert

```
@Insert("INSERT INTO
STUDENTS(STUD_ID,NAME,EMAIL,ADDR_ID, PHONE)

VALUES(#{studId},#{name},#{email},#{address.addrId},#{phone})"
)

@SelectKey(statement="SELECT STUD_ID_SEQ.NEXTVAL FROM
DUAL",  keyProperty="studId", resultType=int.class, before=true)

int insertStudent(Student student);
```

5.3 @Update

我们可以使用 **@Update**注解来定义一个**UPDATE**映射语句，如下所示

```
@Update("UPDATE STUDENTS SET NAME=#{name},
EMAIL=#{email},
PHONE=#{phone} WHERE STUD_ID=#{studId}")
int updateStudent(Student student);
```

```
StudentMapper mapper =
sqlSession.getMapper(StudentMapper.class);
int noOfRowsUpdated = mapper.updateStudent(student);
```

5.4 @Delete

我们可以使用 **@Delete**注解来定义一个**DELETE**映射语句，如下所示

```
@Delete("DELETE FROM STUDENTS WHERE STUD_ID=#{studId}")

int deleteStudent(int studId);
```

5.5 @Select

```
@Select("SELECT STUD_ID AS STUDID, NAME, EMAIL, PHONE
FROM STUDENTS WHERE STUD_ID=#{studId}")

Student findStudentById(Integer studId);
```

为了将列名和**Student bean**属性名匹配，我们为**stud_id**起了一个**studId**的别名。如果返回了多行结果，将抛出**TooManyResultsException**异常。

5.6 结果映射

我们可以将查询结果通过别名或者是 **@Results**注解与**Java Bean**属性映射起来。

```
@Select("SELECT * FROM STUDENTS")
@Results(
{
@Result(id = true, column = "stud_id", property ="studId"),
@Result(column = "name", property = "name"),
@Result(column = "email", property = "email"),
@Result(column = "addr_id", property = "address.addrId")
})
List<Student> findAllStudents();
```

5.7一对一映射

MyBatis提供了 @One注解来使用嵌套select语句（Nested-Select）加载一对一关联查询数据。让我们看看怎样使用 **@One**注解获取学生及其地址信息

```
public interface StudentMapper{
    @Select("SELECT ADDR_ID AS ADDRID, STREET, CITY, STATE, ZIP, COUNTRY
            FROM ADDRESSES WHERE ADDR_ID=#{id}")
    Address findAddressById(int id);
    @Select("SELECT * FROM STUDENTS WHERE STUD_ID=#{studId} ")
    @Results(
    {
        @Result(id = true, column = "stud_id", property = "studId"),
        @Result(column = "name", property = "name"),
        @Result(column = "email", property = "email"),
        @Result(property = "address", cclumn = "addr_id",
        one = @One(select = "com.briup.mappers.Student Mapper.findAddressById"))
    })
    Student selectStudentWithAddress(int studId);
}
```

5.7一对一映射

```
int studId = 1;

StudentMapper studentMapper =
sqlSession.getMapper(StudentMapper.class);

Student student =
studentMapper.selectStudentWithAddress(studId);

System.out.println("Student :"+student);

System.out.println("Address :"+student.getAddress());
```

5.7一对一映射

我们可以在映射器**Mapper**配置文件中配置**<resultMap>**并且使用**@ResultMap**注解来引用它。

```
<mapper namespace="com.briup.mappers.StudentMapper">
  <resultMap type="Address" id="AddressResult">
    <id property="addrId" column="addr_id" />
    <result property="street" column="street" />
    <result property="city" column="city" />
    <result property="state" column="state" />
    <result property="zip" column="zip" />
    <result property="country" column="country" />
  </resultMap>
  <resultMap type="Student" id="StudentWithAddressResult">
    <id property="studId" column="stud_id" />
    <result property="name" column="name" />
    <result property="email" column="email" />
    <association property="address" resultMap="AddressResult" />
  </resultMap>
</mapper>
```

5.7一对一映射

```
public interface StudentMapper{

@Select("select stud_id, name, email, a.addr_id, street, city,

state, zip, country" + " FROM students s left outer join addresses
a on s.addr_id=a.addr_id" + " where stud_id=#{studId} ")

@ResultMap("com.briup.mappers.StudentMapper.StudentWithAd
dressResult")

Student selectStudentWithAddress(int id);

}
```

5.8一对多映射

MyBatis提供了 **@Many**注解，用来使用嵌套**Select**语句加载一对多关联查询。现在让我们看一下如何使用 **@Many**注解获取一个讲师及其教授课程列表信息：

5.8一对多映射

```
@Select("SELECT tutor_id, name as tutor_name, email, addr_id
        FROM tutors where tutor_id=#{tutorId}")
@Results(
{
    @Result(id = true, column = "tutor_id", property = "tutorId"),
    @Result(column = "tutor_name", property = "name"),
    @Result(column = "email", property = "email"),
    @Result(property = "address", column = "addr_id",
    one = @One(select = "ccm.briup.mappers.Tutor Mapper.findAddressById")),
    @Result(property = "courses", column = "tutor_id",
    many = @Many(select = "ccm.briup.mappers.Tutor Mapper.findCcursesByTutorId")})
})
Tutor findTutorById(int tutorId);
```

5.8一对多映射

```
    <resultMap type="Course" id="CourseResult">
      <id column="course_id" property="course Id" />
      <result column="name" property="name" />
      <result column="description" property="description" />
      <result column="start_date" property="startDate" />
      <result column="end_date" property="endDate" />
    </resultMap>
    <resultMap type="Tutor" id="TutorResult">
      <id column="tutor_id" property="tutorId" />
      <result column="tutor_name" property="name" />
      <result column="email" property="email" />
      <association property="address" result Map="AddressResult" />
      <collection property="courses" result Map="CourseResult" />
    </resultMap>
</mapper>
```

5.8一对多映射

```
public interface TutorMapper{
    @Select("SELECT T.TUTOR_ID, T.NAME AS TUTOR_NAME, EMAIL,
            A.ADDR_ID, STREET, CITY, STATE, ZIP, COUNTRY, COURSE_ID, C.NAME,
            DESCRIPTION, START_DATE, END_DATE  FROM TUTORS T LEFT OUTER
            JOIN ADDRESSES A ON T.ADDR_ID=A.ADDR_ID LEFT OUTER JOIN COURSES
            C ON T.TUTOR_ID=C.TUTOR_ID WHERE T.TUTOR_ID=#{tutorId}")
    @Result Map("con.briup.mappers.TutorMapper.TutorResult")
    Tutor selectTutorById(int tutorId);
}
```

Thank You!