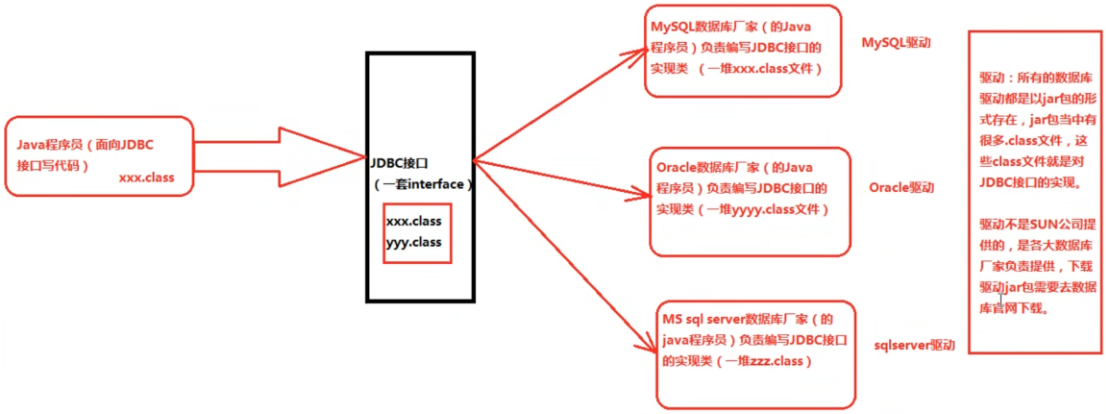
JDBC：Java DataBase Connectivity，Java 语言连接数据库。本质是 SUN 公司制定的一套接口。

java.sql.*；软件包下面有很多接口

为什么要制定接口：

每个数据库的底层实现原理都不一样。不同的数据库公司根据接口的实现类称为驱动。



面向接口编程：

解耦合：降低程序的耦合度，提高程序的扩展力。

多态机制就是非常典型的：面向抽象编程。不要面向具体编程。

JDBC 开发前的准备工作：先从官网下载对应的驱动 jar 包，然后将其配置到环境变量 classpath 当中。

以上的配置是针对于文本编辑器的方式开发，使用 DEA 工具的时候，不需要配置以上的环境变量。IDEA 有自己的配置方式。

JDBC 编程六步:

第一步：注册驱动(作用：告诉 Java 程序，即将要连接的是哪个品牌的数据库)

Connection conn = DriverManager.getConnection(url, user, password);

另一种方法，更常用，参数是一个字符串，字符串可以写到 xxx.properties 文件中，程序不需要重新编译。实际开发中不建议把连接数据库的信息写死在程序里。

反射：

ResourceBundle bundle = ResourceBundle.getBundle("com.ryn.test.jdbc");

String driver = bundle.getString("driver");

String url = bundle.getString("url");

String user = bundle.getString("user");

String password = bundle.getString("password");

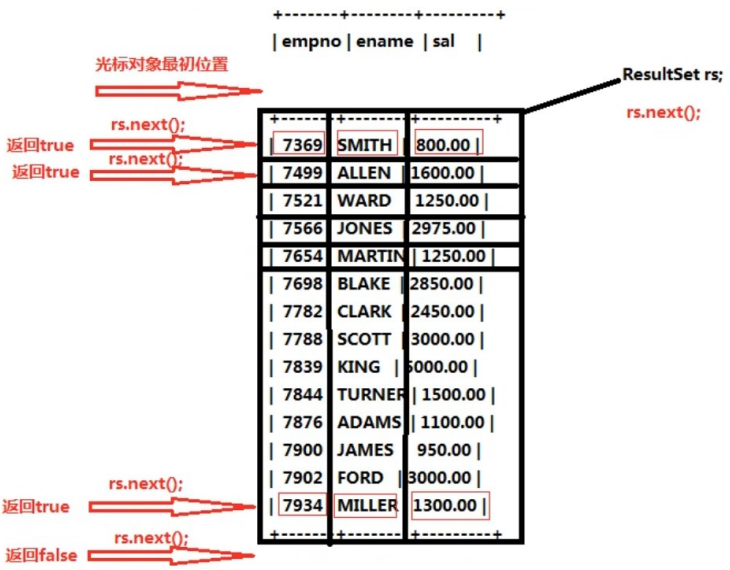
Class.forName("driver");

第二步：获取连接（表示的进程和数据库进程之间的通道打开了，这属于进程之间的通信，重量级的，使用完之后一定要关闭）

第三步：获取数据库操作对象(专门执行 sql 语句的对象)

第四步：执行 SQL 语句(DQL DML…) JDBC 的 sql 不用写分号。

第五步：处理查询结果集(只有当第四步执行的是 select 语句的时候，才有这第五步处理查询结果集。)



遍历结果集

使用 getString(n)获取第 n 列的数据，不管源数据的类型，取出来都是 String

第六步：释放资源(使用完资源之后一定要关闭资源。Java 和数据库属于进程间的通信，开启之后一定要关闭。)

* ur1:统一资源定位符（网络中某个资源的绝对路径）
* https://www.baidu.com/这就是 URL.
* URL 包括哪几部分？：协议，IP，PORT，资源名

http://182.61.200.7:80/index.htm1

* http://通信协议
* 182.61.200.7 服务器 IP 地址
* 80 服务器上软件的端口
* index.html 是服务器上某个资源名

jdbc:mysql //127.0.0.1:3306/tset
cbc:mysql //协议
127.0.0.1 //地址
3306 //数据库端口号
test //具体的数据库实例名。

通信协议：通信之前就提前定好的数据传送格式。数据包具体怎么传数据的。

SQL 注入：

Enter your username: zzz
Enter your password: zzz' or '1'='1

根本原因：

sql = "select name, password from login where name = 'zzz' and password = 'zzz' **or** '1'='1";

```
String sql = "select * from t_user where loginName = '"+loginName+"' and loginPwd = '"+loginPwd+"'";
// 以上正好完成了sql语句的拼接，以下代码的含义是，发送sql语句给DBMS，DBMS进行sql编译。
// 正好将用户提供的“非法信息”编译进去。导致了原sql语句的含义被扭曲了。
rs = stmt.executeQuery(sql);
```

用户输入的信息有 sql 的关键字，并且参与了 sql 语句的编译，导致 sql 的原意被扭曲，达

到 sql 注入。

解决方法：

只要用户提供的信息不参与 SQL 的编译过程，问题就解决了。即使信息里含有 sql 的关键字，不参与编译就不起作用。

使用 java.sql.PreparedStatement。PreparedStatement 接继承了 java.sql.Statement，Preparedstatement 是属于预编译的数据库操作对象，预先对 SQL 框架进行编译，然后再给 SQL 语句传值。

```
//SQL 语句的框架。其中一个？表示一个占位符，一个?将来接收一个"值"，注意：占位符不能使用单引号括起来。
String sql = "select name, password from login where name = ? and password = ?";
// 程序执行到此处，会发送 sql 语句框架给 DBMS，然后 DBMS 进行 sql 语句的预先编译。
ps = conn.prepareStatement(sql);
//给占位符传值（第一个?下标为 1， JDBC 中所有下表从 1 开始）
ps.setString(1, loginName);
ps.setString(2, passWord);
// rs 不需要传递语句
rs = ps.executeQuery();
```

Statement 和 PreparedStatement 区别：

--Statement 存在 sql 注入问题，Preparedstatement 解决了 sqL 注入问题

--Statement 是编译一次执行一次。Preparedstatement 是编译一次，可执行多次，效率较高。

--PreparedStatement 会在编译阶段进行类型的安全检查。

综上所述，PreparedStatement 使用较多。

必须使用 Statement 时：业务方面必须要求支持 SQL 注入。

```
String sql = "select ename from emp order by ename ?";
```

asc 或者 desc 这时候必须注入。

事务：

JDBC 的事务是自动提交的，只要执行一条 DML，则自动提交一次。这是默认的事务行为。但是在实际的业务当中，通常都是条 DML 语句共同联合才能完成的，必须保证他们这些 DM 红语句在同一个事务中同时成功或者同时失败。

事务提交：

```
conn.setAutoCommit (false);
conn.comit ();
conn.rollback();
```

悲观锁与乐观锁：

悲观锁/行级锁：事务必须排队执行，数据被锁，不允许并发。

```
mysql> select ename,job,sal from emp where job='MANAGER' for update;
```

for update 将选中的这几行数据锁定。

乐观锁：支持并发，事务不需要排队，只不过需要一个版本号。

ename job sal version

| BLAKE | MANAGER | 2850.00 | 1.1

事务1-->读取到版本号1.1

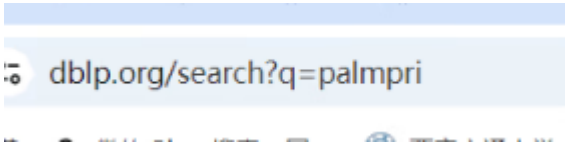
事务2--->读取到版本号1.1

其中事务1先修改了，修改之后看了版本号是1.1，于是提交修改的数据，将版本号修改为1.2

其中事务2后修改的，修改之后准备提交的时候，发现版本号是1.2，和它最初读的版本号不一致。回滚。

数据库写全，不要写第一次
MPD 写全称
图片背景改一下，颜色改深一点

背景：一部分写一页半左右。
ROI 删除
补一些最新的文献 Lu leng　fei kunke　zhang bob



同态加密：
图不要，降重

数据库：　XJTU-UP 篇幅缩减
图片里的文字字体

结论缩减篇幅

相关工作和分析，大段内容可以简写。

写一下和其他方法的不同，区别。

表格转换