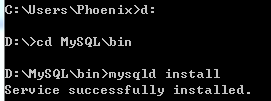
**JAVA小项目——数据库部分**

Shift+Alt+z直接引入try{}catch(),Shift+Ctrl+o导入相关包,Shift+Ctrl+/多行注释,Shift+Ctrl+\取消注释，Ctrl+/单行注释（再按一下就是取消单行注释），Alt+z是代码提示，Ctrl+D删除整行。

MyEclipse中让汉字变大些方法：Windows>Propertise>General>Appearance>Colors and Fonts>Basic>Text Font >Edit 把脚本字符改成“中欧字符”就可以了，其他不用改。

有时候一个变量多处被用到，有时如果想要更改这个变量名，一个个找怕漏掉，这时我们可以按Alt+Shift+O就能让同一变量”发光”，这样就能看到了。

28集 MySQL的安装：

1. 解压MySQL的安装包；
2. 将bin目录添加到path中；
3. 修改my.ini：
4. 复制MySQL目录下其中一个my-xx.ini，并将其重命名为my.ini；
5. 打开my.ini，在该文件中的[mysqld]下面增加两个配置信息BASEDIR=指向mysql的目录，DATADIR=指向MySQL的数据库存储目录。 我的电脑上D盘下有一个MySQL的文件夹，该文件夹里就是MySQL的文件，于是第一个配置信息的值是D:/MySQL，第二个配置信息是：D:/MySQL/data。
6. 通过命令提示符安装MySQL：
7. 首先进入DOS界面，并且进入mysql的bin目录，运行mysqld install（虽然已经配置了path但在安装时仍然建议进入mysql的bin目录）完成安装（输入mysqld remove可以完成卸载）。
8. 使用net start mysql可以启动服务，也可以直接进入“服务”窗口开启MySQL服务。
9. 进入mysql：
10. 使用mysql -u root -p回车之后输入密码就可以进入mysql，没有设密码就直接按回车也能进入；
11. 使用show databases；可以查询mysql中所有的数据库；

（一劳永逸解决MySQL乱码问题，参看网页：MySQL的乱码问题，参照网页：<http://www.shaoqun.com/a/101073.aspx>）

29集 数据库基本操作：

1. 创建数据库：create database 数据库名；删除数据库：drop database 数据库名。
2. 创建好一个数据库之后要在这个数据库中添加表：

use 数据库名；指定对哪个数据库进行操作。

show tables；列出数据库中所有的表。 desc 表名称;查询某个表的结构，也就是列出各个字段的属性。

charset gbk；设置该数据库的编码方式为gbk。

30集 数据库的分析：

每一个类都可以用一张表来表示，类的属性就是表的字段。

主键：对于一个表而言，必须要有一个不能重复的字段作为主键，保证表示一个类的表中的所有实例互相之间都是独一无二的存在（建议将主键设置为一个没有任何意义的字段，通常会使用一个逐渐递增的id字段作为主键）；

外键：一个表的外键可以重复，但必须是另外一个表的主键。

每张表都代表了一个类，现实生活中类与类是有关系的，那么怎么把这种关系体现在表中呢：

首先类与类有三种关系：

1、1对1：A类与B类是一一对应关系，A只能有1个B，B也只能有1个A。这种情况下，把两张表写好后，每张表的字段都包含了各自类的属性以后，我们可以在任意一张表中添加一个外键，以表示这两张表的关系；

2、1对多：一个A可以拥有多个B。举个例子，一张用户表，一张书籍表，一个用户可以借阅多本书籍。我们完成这两张表之后，可以在书籍的表中加入用户的主键做外键。所以应该在“多”的那张表上添加外键；

1. 多对多：举个例子，一个学生可以选修多个课程，一个课程可以有多个学生学。所以当我们完善好学生表和课程表这两张表后，我们可以单独设立一张表示两者关系的关系表。既然重复不可避免，那就由一张表来承担它。

31集 数据表的基本操作(注意，下面所有的操作都是在某个数据库中进行操作，也就是说要使用use \*\*\*；来指明对哪个数据库进行操作)：

1. 基本数据类型：

int表示整型；float表示浮点类型；varchar表示字符串类型(255个字节)；text表示文本类型（没有大小限制，有时我们存储一个网页的时候就可以用这种类型）；binary表示二进制类型（可以用来存照片等一些二进制文件）；

date表日期；time表示时间；datetime表示日期和时间，也就是年月日加时分秒（和timestamp）。

MySQL中没有Boolean类型，但是我们也能将字段类型设置为boolean，比如type boolean，这样是能成功建表的。不过查看一下建表后的语句，就会发现，mysql把boolean替换成tinyint(1)。也就是说mysql把boolean=tinyInt了。所以说boolean在MySQL里的类型为tinyint(1)，MYSQL保存BOOLEAN值时用1代表TRUE,0代表FALSE。我们可以这样插入boolean值：insert into test(type) values(true);，查询type字段此时的值显示为1。

1. 数据表的字段命名规则建议断字使用下划线，比如user表的id字段我们就可以这样user\_id;普通的表名建议使用t\_\*\*\*来命名，如t\_user，临时表表名用temp\_\*\*\*。

举个例子：

|  |
| --- |
| drop table if exists t\_user；//如果存在t\_user就删除这个表  create table if not exists t\_user**(**//如果不存在表t\_user就创建这个表  id int(11) primary key auto\_increment**,**//创建一个id字段，为int类型，11字节容量，自动递增  username varchar(20) not null unique**,** //该字段不能为空，且不能重复  password varchar(20)**,**  nickname varchar(30)**，**  born datetime**,**//设置出生年月  classid int(10)  **)；**  desc t\_user//查询t\_user的结构  drop table t\_user//删除表t\_user |

修改数据：

|  |
| --- |
| alter table t\_user add (cloumn) address varchar(200); //修改表t\_user，添加address这个字段，“（column）”的意思是字段前面随便加不加cloumn，不是字段前面必须加(column)。  alter table t\_user drop address; //修改表t\_user，删除address这个字段  alter table t\_user change nickname newname varchar(50);//修改t\_user，把字段nickname改成newname，字符串50个字节容量。  alter table t\_user modify nickname varchar(30) after id;//把nickname字段移到id字段后面，nickname后面必须指明它的字段描述，也就是varchar(30)。要把某个字段移到第一位，直接在描述后面写first就行了。  alter table 表名 add constraint foreign key(你的外键字段名) REFERENCES 外表表名(对应的表的主键字段名);//设置表中某个字段为外键。  ALTER TABLE 表名 DROP FOREIGN KEY 外键字段名;//取消表中某个字段的外键  set foreign\_key\_checks=0;//当你试图删除表信息时如果mysql不让你删并报错说有该表有外键关联时，你就可以输入这句。等于1则表系统会进行外键检查。 |

插入数据：

|  |
| --- |
| insert into t\_user (username,password,nickname) value (‘张三’,’123’,’小三’);  insert into t\_user values(‘张三’,’123’,null);//不写插入哪几个字段的话系统会默认按顺序插入，不需要插入的字段写null就行了。  insert into t\_user (username,nickname) select no,name from t\_student;//将表t\_student中的no与name字段的数据分别插入到表t\_user中的username和nickname字段下。 |

查询数据：

|  |
| --- |
| select username,password from t\_user;//从表t\_user中选取username和password这两个字段的数据，也就是输出两列；  select \* from t\_user；//输出所有t\_user字段内容；  select \* from t\_user where id>2 and nickname=’小三’;//输出id>2且nickname为小三的所有字段内容；and改成or就表示条件或  select \*from t\_user where id in (1,2,3);//输出id等于1或者等于2或者等于3的所有字段内容；  select \*from t\_user where born between ‘1981-02-01’ and ‘1990-09-01’;//查询born在1981.2.1到1990.9.1中的所有字段内容；  select \* from t\_user where nickname like’张%’;//输出nickname以“张”开头的所有字段内容，%表示任意长度的字符串，a%b表示以a开头b结尾的字符串。而\_表示任意单个字符，a\_b表示以a开头b结尾长度为3的字符串；  year(date)方法中传入date变量可以返回该date变量的年份，now()方法可以返回当前的时间（格式是：年-月-日 时：分：秒）：  select name as ‘姓名’,(year(now())-year(bron)) as ‘年龄’ from t\_user;//输出t\_user中所有人员的name字段和year(now())-year(bron)字段，并分别将字段重命名为姓名和年龄；  select \* from t\_user where born=(select min(born) from t\_user);//输出表t\_user中年龄最大的员工的所有字段内容；而语句select name,min(born) from t\_user；是不行的，这条语句只是输出表中name字段的第一个值，然后再输出born字段中的最小值，而两个值之间并没有什么关系；  select \* from t\_user where born is null;//查询表t\_user中born值为空的所有字段内容，不为空就是is not null；  select count(\*) from t\_user where year(born)=1993;//查询出生在1993年的人数；count()方法中传入字段名来统计该字段中包含的除null以外的数据数量。  select \* from t\_user where classid=3 order by born desc limit 0,5;//取出classid为3的所有字段内容，以born字段的降序排列（从大到小，升序是asc，born后面啥都不加默认是升序），从第一条记录开始取五条记录；  MySQL中语句的执行顺序：先from再 where再group by再having再select再 order by最后 limit  group by的用处：比如说表t\_user中有classid字段表示每个用户所在的班级号，那当我们想知道每个班级有多少人的时候就可以用到group by来对classid进行分组：  select classid,count(\*),max(born) from t\_user group by classid;//整个过程可以这么来想：系统先将表t\_user进行分组，将classid相同的记录分到一组，然后再对每组进行数量统计，最后输出每个classid及其相应的个数以及每个班年龄最小的人的出生日期；  select classid,count(\*) as ‘pn’,max(born) from t\_user group by classid having pn>30;//输出班级人数大于30的classid及其相应的人数以及每个班年龄最小的人的出生日期；整个过程就是先将表t\_user按classid相同的进行分组，然后对每个组进行数量统计，除去数量<=30的分组，输出剩下的分组的相关信息。  having与where的区别：  having是先分组再筛选记录，它总是包含聚集函数（count(),sum()等等），主要用来对聚合函数运算结果的输出进行限制。  where在聚合前先筛选记录．也就是说作用在group by子句和having子句前。where作用于表和视图在分组和聚集计算之前选取输入行（因此，它控制哪些行进入聚集计算），因此，where子句不能包含聚集函数， 因为试图用聚集函数判断那些行输入给聚集运算是没有意义的。  严格说来，你可以写不使用聚集的having子句， 但这样做只是白费劲。同样的条件用where更有效。  总结一句话就是：where语句在groupby语句之前；SQL会在分组之前计算where语句。  having语句在groupby语句之后；SQL会在分组之后计算having语句。  select classid,sex,count(\*) from t\_user group by classid**,**sex;//先将表t\_user按classid分组，然后再对每个分组再按性别分组，然后输出每个分组的classid，sex以及数量值，这就能统计出每个班里男生人数和女生人数。  跨表查询（连接查询）：  select t1.name from t\_clas t1,t\_user t2 where t1.id=t2.classid;//将表t\_clas与表t\_user分别命名为t1与t2，然后利用t1.id=t2.classid的关系将两个表连接起来，然后输出t1表中的name字段值；  select t1.name from t\_clas t1 join t\_user t2 on(t1.id=t2.classid) where...;//将表t\_clas与表t\_user以t1.id=t2.classid的关系连接起来形成一张新表，这张新表的数据不包含不完整的数据，也就是说没有学生的班级记录和没有班级的学生记录不会存在在这张表中。这种连接也称为内连接（inner join）  select \* from t\_clas t1 left join t\_user t2 on(t1.id=t2.classid) where...;//left join就是t\_clas在左边，t\_user在右边，整张表按照左边表的数据来，如果右边的表中没有满足连接关系的内容，就以null来与左边表的内容相连。就比如说一张班级信息表左连接一张学生表，但是班级信息表中的某些班级中没有学生，也就是说学生信息表中没有属于某个班级的学生。这时左连接的结果就是班级信息表中的所有内容都有，某些没有学生的班级元组连接的学生信息的所有字段内容就是null；  select \* from t\_clas t1 right join t\_user t2 on(t1.id=t2.classid) where...;//right join依然还是t\_clas在左边，t\_user在右边，不过此时整张表就是按照右边表的数据来了；  select t1.id, t2.sex, count(t2.cla\_id) from t\_clas t1 left join t\_stu t2 on(t1.id=t2.cla\_id) group by t1.id, t2.sex;//输出每个班里男女生的人数 |

更新数据：

|  |
| --- |
| update 表名 set字段名=新的值 where 条件；  update t\_user set username=’悟空’，nickname=’wukong’ where id=1; |

删除数据：

|  |
| --- |
| delete from 表名 where 条件；(不加条件就是删除所有表中数据)  truncate table 表名;(清空表，与上面不同的是会清除字段的信息，比如我们设置了一个递增的id字段，使用truncate之后在添加数据id就能从1开始，上面使用delete则再添加数据id会从原来的接着往下) |

我们可以在记事本里编辑好一段sql代码，然后将文件重命名以”.sql”结尾。接着在dos界面下用source语句将这个sql文件导入，也能实现数据表的创建。如下图所示：

（注意：导入的路径中的斜杠是“\\”或“/”，因为此时“\”表示转义字符）

外键设置：

|  |
| --- |
| create table if not exits t\_student(  name varchar(20),  snumber varchar(20) primary key,  clas\_id int(10),  constraint foreign key (clas\_id) references t\_classroom (id)//将clas\_id这个字段设置为外键，这个字段就是t\_classroom这张表里的主键id  ); |

权限设置：

|  |
| --- |
| grant all on sb.\* to ’itat’@’localhost’ identified by ‘123’;//创建了一个名为itat的用户，密码是123，该用户能够在本地对数据库sb中的所有表进行任何操作。  登陆就要像下面这样登录：  我们也可以用这个语句修改超级管理员root的密码：  grant all on \*.\* to ‘root’@’localhost’identified by ‘123’ |

关于MySQL中的null：

1. null不能和任何值比较。在SQL中，NULL值与任何其它值的比较（即使是NULL）永远不会为“真”（所以我们在判断是否为null时不能用\*\*=null，这样即使是null=null，也是返回false，我们只能用\*\* is null）。当进行查询操作时如果使用条件如where id <> 2，这样是查不出id为null的记录的，只能这样写where id <> 2 or id is null；
2. null与空("")的比较：
3. 、NULL值是未知的，且占用空间，不走索引，DBA建议建表的时候最好设置字段是NOT NULL 来避免这种低效率的事情的发生。而空值('')是不占用空间的；
4. 、在进行count()统计某列的记录数的时候，如果采用的NULL值，会别系统自动忽略掉，但是空值是会进行统计到其中的；
5. 、判断NULL 用IS NULL 或者 is not null,SQL 语句函数中可以使用ifnull()函数来进行处理，判断空字符用 =''或者 <>''来进行处理；
6. 、对于MySQL特殊的注意事项，对于timestamp数据类型，如果往这个数据类型插入的列插入NULL值，则出现的值是当前系统时间。插入空值，则会出现 '0000-00-00 00:00:00'。

39集：JDBC初步

|  |
| --- |
| **public** **class** testJDBC {  **public** **static** **void** main(String[] args) {  Connection con=**null**;  Statement stat=**null**;  ResultSet rs=**null**;  **try** {  //1、创建Connection  /\*这是url的统一格式，3306是MySQL的默认端口，itat\_test是我要连接的数据库名  \* 一般java连接MySQL默认就是使用utf-8的字符编码去连接，也就是说通过Connection进出数据库的数据都会经过utf-8规则编译  \*但如果我们将数据插入数据库或是从数据库提取数据显示到控制台，数据都显示乱码的话，不妨试一试在下面的url语句最后加入  \*?useUnicode=true&characterEncoding=utf-8  \*?后面跟属性，属性之间通过&来连接  \* \*/  String url="jdbc:mysql://localhost:3306/itat\_test";  String user="root";//用户名和密码就是我们设置的进入MySQL的用户名和密码  String password="123456";  Class.forName("com.mysql.jdbc.Driver");//加载jdbc的connector的driver  con=DriverManager.*getConnection*(url,user,password);  //2、创建sql语句并通过Connection获取Statement  String sql="select \* from employee";  stat=con.createStatement();  /\*这里要分两种情况，如果我们的sql语句是更新语句，那就直接使用stat.executeUpdate(sql);完成更新;  \* 如果是查询语句，也就是说我们要得到相应的查询结果，则需要通过stat.executeQuery(sql)完成，返回值是一个ResultSet的记录集  \* \*/  rs=stat.executeQuery(sql);//这个ResultSet有点像集合中的迭代器Iterator  //遍历记录集的统一方式  **while**(rs.next()){//如果指向下一个记录的指针不为空，那就rs指向下一条记录，并返回true  //如果某个字段中的值是A类型的，就用getA()方法，该方法传入参数可以是字段名，也可以是列数，第一字段的列数是1，但是不建议用列数取数据，因为列数可能会变  System.*out*.println(rs.getInt("id")+","+rs.getString("name"));  }  } **catch** (SQLException e) {  e.printStackTrace();  }**finally**{  /\*释放连接，要先释放ResultSet，再释放Statement，最后释放Connection；  \* \*/  **try** {  **if**(rs!=**null**)rs.close();  **if**(stat!=**null**)stat.close();  **if**(con!=**null**)con.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }  } |

1. 讲解PreparedStatement：

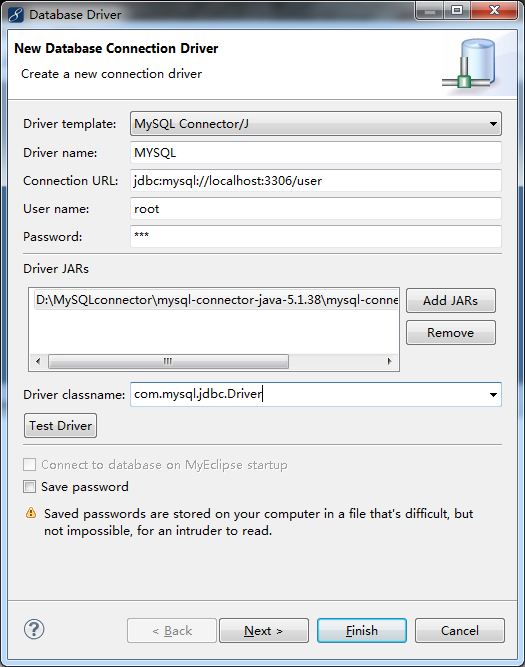
如果我们做网页的登录功能，那么肯定要从前台获得用户输入的数据，然后到后台数据库进行比对，我们常用的语句就是：”select count(\*) from employee where username=’”+username+”’and password=’”+password+”’”。如果返回的值是1就表示用户名密码正确，可以登录。

这时就有一个很明显的漏洞，如果用户输入的用户名是admin’;//，那就把后面的password的验证给注释掉了。这就是SQL注入攻击。

为了解决这个问题我们使用PreparedStatement的方式：

|  |
| --- |
| **public** **class** testJDBC {  **public** **static** **void** main(String[] args) {  Connection con=**null**;  PreparedStatement pstat=**null**;  ResultSet rs=**null**;  **try** {  String url="jdbc:mysql://localhost:3306/itat\_test";  String user="root";  String password="123456";  Class.forName("com.mysql.jdbc.Driver");  con=DriverManager.*getConnection*(url,user,password);    String sql="select \* from employee where id=?";  pstat=con.prepareStatement(sql);  pstat.setInt(1,3);//id字段的数据是int类型，那就用setInt方法将第一个问号的值设置为3  rs=pstat.executeQuery();//执行查询程序    **while**(rs.next()){  System.*out*.println(rs.getInt("id")+","+rs.getString("name"));  }  } **catch** (SQLException e) {  e.printStackTrace();  }**finally**{  **try** {  **if**(rs!=**null**)rs.close();  **if**(pstat!=**null**)pstat.close();  **if**(con!=**null**)con.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }  } |

在MyEcplise中连接MySQL：



关于SQL注入攻击的一些问题：

所谓SQL注入攻击就是在SQL拼接语句的参数中加入单引号，分号等字符来破坏原sql语句的攻击方式。具体的可以参看网页：<http://blog.csdn.net/xbbccx/article/details/7259446>

由于一般等待用户传入的参数都是username，password之类的字符串参数，而这些字符串参数在sql语句中一定是带一对单引号的，所以大部分sql注入语句都是在传入参数中加上一个单引号以求和sql连接语句中的单引号对上从而破坏sql语句结构。所以，我们一般会在sql语句中的传入参数中使用replace方法将字符串中的单引号替换为双引号（username.replace(“’”,”””)），而双引号在sql语句中是正常的字符（username=‘a”c’就表示username为a‘c），这就解决大部分问题了。

而在java中使用PreparedStatement是绝对不会出现问题的，为什么呢？下面是网上的解答：

|  |
| --- |
| 这两天在上课时被同学拿了一段代码问我，这段代码有什么问题，我看了一会说：Connection和PreparedStatement都没关。他说不止这方面的问题，还有sql注入的问题，我就坚决的说使用了占位符不存在sql注入的问题，但是他提出了一种情况，在我看来也很有道理的情况。  pstmt = conn.prepareStatement("delete from user where user.id=?");  pstmt.setString(1, "w");  他认为如果把代码写成这样就有注入问题了  pstmt = conn.prepareStatement("delete from user where user.id=?");  pstmt.setString(1, "w' or '2'='2"); 当时我看了只能告诉他一定不存在注入问题，因为在我的想法中我一直记得的是用占位符能解决注入问题，至于怎么解决的就不知道了，看了上面的代码也很有道理，感觉setString后的sql语句应该是  delete from user where user.id='w' or '2'='2';  回到宿舍我专门写了程序测试一下，事实证明并不想我们想的这样，的确使用占位符不存在注入问题，所以解释是在执行的时候把一些字符给转义了，但这个转义的过程是在什么地方转义的呢，把上面的sql语句在mysql控制台上运行一下，查看一下数据看到所有数据都被删除完，那只能解释成在java程序中转义的，于是我就去看java的源代码，发现在java源码中PreparedStatement只是一个接口，而且是没有子类的接口，我就很纳闷，没实现怎么用的？所以一定有实现的地方，去网上查了一下，jdk直提供接口，而具体实现是由数据库厂商实现的，我们用的就是数据库厂商实现的类。于是我就又去查mysql的jar包源码，发现有个PreparedStatement实现了jdk中的PreparedStatement了。里面的setString方法如下实现：  public void setString(int parameterIndex, String x) throws SQLException {  // if the passed string is null, then set this column to null  if (x == null) {  setNull(parameterIndex, Types.CHAR);  } else {  StringBuffer buf = new StringBuffer((int) (x.length() \* 1.1));  buf.append('\'');  int stringLength = x.length();  //  // Note: buf.append(char) is \_faster\_ than  // appending in blocks, because the block  // append requires a System.arraycopy()....  // go figure...  //  for (int i = 0; i < stringLength; ++i) {  char c = x.charAt(i);  switch (c) {  case 0: /\* Must be escaped for 'mysql' \*/  buf.append('\\');  buf.append('0');  break;  case '\n': /\* Must be escaped for logs \*/  buf.append('\\');  buf.append('n');  break;  case '\r':  buf.append('\\');  buf.append('r');  break;  case '\\':  buf.append('\\');  buf.append('\\');  break;  case '\'':  buf.append('\\');  buf.append('\'');  break;  case '"': /\* Better safe than sorry \*/  if (this.usingAnsiMode) {  buf.append('\\');  }  buf.append('"');  break;  case '\032': /\* This gives problems on Win32 \*/  buf.append('\\');  buf.append('Z');  break;  default:  buf.append(c);  }  }  buf.append('\'');  String parameterAsString = buf.toString();  byte[] parameterAsBytes = null;  if (!this.isLoadDataQuery) {  parameterAsBytes = StringUtils.getBytes(parameterAsString,  this.charConverter, this.charEncoding, this.connection  .getServerCharacterEncoding(), this.connection  .parserKnowsUnicode());  } else {  // Send with platform character encoding  parameterAsBytes = parameterAsString.getBytes();  }  setInternal(parameterIndex, parameterAsBytes);  }  }  到此就告一段落，可以发现在setString时最外面的单引号被转义了，也就是说setString后的sql语句是这样的  delete from user where user.id=\'w' or '2'='2\';  而且仔细看会发现在setString中是一个字符一个字符的解析，该转义的都已经转义，正如他一句注释中写的Better safe than sorry.所以最终，占位符确实不存在注入问题 |

最后我们要大概讲讲转义字符的概念。

ASCII 码使用指定的7 位或8 位[二进制数](http://baike.baidu.com/view/1349.htm" \t "http://baike.baidu.com/view/_blank)组合来表示128 或256 种可能的[字符](http://baike.baidu.com/view/263416.htm" \t "http://baike.baidu.com/view/_blank)，这些字符分为可显示字符和不可显示字符。可显示字符很简单，就是字母、标点符号之类能在屏幕上显示的字符，而不可显示字符就是控制字符，它告诉[编译器](http://baike.baidu.com/view/487018.htm" \t "http://baike.baidu.com/_blank)需要用特殊的方式进行处理。

C以及java等语句中定义了一些字母前加"\"来表示常见的那些不能显示的ASCII[字符](http://baike.baidu.com/view/263416.htm" \t "http://baike.baidu.com/_blank)，如\0,\t,\n等，就称为转义字符，这些[字符](http://baike.baidu.com/view/263416.htm" \t "http://baike.baidu.com/_blank)因为前面的”\”，都不是它本来的ASCII字符意思了。而且值得注意的是\0,\t,\n这些转义字符并不是两个字符，而是一个字符，所以在上面的程序中能使用charAt()方法来遍历字符串以查看是否存在转义字符，而且在使用append()方法的时候所有字符都是用单引号包起来的。



比如说在一个字符串中插入“\n”，输出到时候，字符串就会换行接着输出后面的内容，这就是转义字符的功能。所以说“\”自身是有一定功能的，什么功能呢？就是转义功能嘛，某些字符和它搭配起来能有特殊的功能嘛。如果“\”与某个字符搭起来没有特殊的含义，那是会报错的，编译都不能通过。

除了“\”+字符代表某种功能的转义字符外，在特定的程序语法中某些显式字符也代表特定的意思，比如说双引号，你在一个字符串中插双引号是连编译都通不过的，可我就是要输出双引号怎么办？那就在前面加一个“\”嘛，负负得正，双引号本身有特殊含义，那咱就加一个斜杠再把它转义一下嘛；“\”自己也是一个有特定功能的字符，不能当字符输出，那咱还是加一个斜杠再转义它一次就行了嘛。

总的来说，转义字符就上面图表中的东西，只要斜杠与某个字符搭起来不是上面图表中的转义字符，就一定会报错。只要是上面图表中的转义字符，就一定代表的是图表中相对应的功能。这个道理无论用在哪里都是对的。

后来我对一个符号产生了问题，就是单引号，单引号插入字符串中不会报错，也能正常输出，单引号前加一个“\”，也是能正常输出单引号（上面的图表就很清楚地列出来了，“\’”就是单引号字符）。那为什么上面的程序中要对传入的String最外面的单引号也要转义一下呢？后来我明白了，字符使用单引号包围起来的，字符串是用双引号包围起来的，双引号里的字符串不能用双引号，单引号里的字符也当然不能是单引号喽。后来我试了一下，果然，字符如果是单引号，就必须加斜杠转义，不然编译都不能通过，而且如果字符是双引号不用转义也可以，当然转义了也一样。

所以上面的框中的网页中的最后几句话是有问题的，setString传入的String X经改造后首尾单引号的不应该有转义字符，而是直接的单引号。而X内的单引号我们查源代码的话发现已经被替换成字符‘\\’和字符‘\’’了，而本质上就是加了两个字符：‘\’和‘’’，合到字符串X中的话就是“\’”，也就是说X在sql语句中就变成了‘\*\*\*\’\*\*\*’，X中的单引号已经被转义成单引号字符了，也就是说不再具有字符串包围功能了。

好了，回到我对sql注入产生兴趣的源头：SQL语句是：String sql="select \* from t\_user where username like '%?%'";，但后来我用pstat.setString的时候报错了，错误说没有占位符（也就是问号）给我设置，这是为啥？现在我明白了，这分明是一条完整的SQL语句啊，占位符被单引号包起来，已经成为sql语句字符串的一员了，失去了占位符的功能了。所以我们不能让占位符被单引号包起来。

**数据库的新连接方法**

之前，我们都是这么连接数据库的：

|  |
| --- |
| Connection con=**null**;    String url="jdbc:mysql://localhost:3306/itat\_test";  String user="root";  String password="123456";  Class.forName("com.mysql.jdbc.Driver");  con=DriverManager.*getConnection*(url,user,password); |

现在我们可以新建一个properties配置文件，把url、user、password这些值存在properties中，很简单，下面是我们创建的properties文件jdbc.properties：

|  |
| --- |
| user=root  password=123  url=jdbc:mysql://localhost:3306/msgmanager |

接着我们要用Properties类来连接jdbc.properties文件并且取配置文件中的值，但是为了不老是创建Properties类老是连接properties文件，我们创建单例：

|  |
| --- |
| **public** **class** propertiesUtil {  **private** **static** Properties *p*;  **public** **static** Properties getProperties(){  **if**(*p*==**null**){  *p*=**new** Properties();  **try**{  *p*.load(propertiesUtil.**class**.getClassLoader().getResourceAsStream("jdbc.properties"));  } **catch** (IOException e) {  e.printStackTrace();  }  }  **return** *p*;  }  } |
| 也可以这样创建单例：  **public** **class** propertiesUtil {  **private** **static** Properties *p*;  **static**{  *p*=**new** Properties();  **try** {  *p*.load(propertiesUtil.**class**.getClassLoader().getResourceAsStream("jdbc.properties"));  } **catch** (IOException e) {  e.printStackTrace();  }  }  **public** **static** Properties getProperties(){  **return** *p*;  }  }两种方式我都测试过，没问题 |

这是我们就能这么连数据库：

|  |
| --- |
| Properties p=propertiesUtil.*getProperties*();  Connection con=**null**;  String user=p.getProperty("user");  String password=p.getProperty("password");  String url=p.getProperty("url");  **try** {  Class.*forName*("com.mysql.jdbc.Driver");  con=DriverManager.getConnection(url,user,password);  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** con; |

**Tomcat有效率地连接数据库：**

我们需要在Tomcat的server.xml文件中的Context标签下的添加下面的代码才能实现数据库的连接。不过我们用的是eclipse，eclipse是将项目发布在Tomcat的虚拟空间中的，所以我们必须对workspace中的Servers项目中的server.xml中的对应项目的Context标签进行配置：

|  |
| --- |
| <Context docBase=*"MsgManager"* path=*"/MsgManager"* reloadable=*"true"* source=*"org.eclipse.jst.jee.server:MsgManager"*>  <Resource name=*"jdbc/DatePool"*  auth=*"Container"*  type=*"javax.sql.DataSource"*  username=*"root"*  password=*"123"*  driverClassName=*"com.mysql.jdbc.Driver"*  url=*"jdbc:mysql://localhost:3306/msgmanager"*  maxTotal=*"8"*  maxIdle=*"4"*/><!-- 数据库连接池里最多有8个链接连接到我们的数据库里面 --><!-- 数据库连接池里永远保证有4个链接是空闲的 -->  </Context> |

接下来我们可以这样获得数据库的连接，代码直接复制就行：

|  |
| --- |
| Connection con=**null**;  **try** {  Context initCtx = **new** InitialContext();  Context envCtx = (Context) initCtx.lookup("java:comp/env");  DataSource ds = (DataSource)  envCtx.lookup("jdbc/DatePool");  con=ds.getConnection();  } **catch** (NamingException e) {  e.printStackTrace();  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** con; |

**数据库的事务管理**

事务指的是将所有的对数据库的读写操作保持一定的原子性，要所有操作执行完之后才进行数据库的更新操作，如果在某一操作中出现异常，那么这个异常操作之前的所有操作都要回滚。

1、对于MySQL而言，需要INNODB的表类型才能支持数据库的事务处理，我们可以在配置文件中这样设置：default-storage-engine=INNODB（不过我试了一下，我的mysql好像不用加这句话也能支持事务）

2、在JDBC中，数据会默认提交，所以需要设置自动提交为false，我们这样设置setAutoCommit(false)

3、在所有的更新操作完成之后，使用commit()方法来提交事务

4、如果出现异常，我们在catch中执行rollback()方法执行回滚

|  |
| --- |
| **public** **void** delete(**int** id) {  con=JDBCUtil.*getConnection*();  String sql="delete from t\_comment where msg\_id=?";//先删除评论，再删除留言  **try** {  con.setAutoCommit(**false**);  pstat=con.prepareStatement(sql);  pstat.setInt(1,id);  pstat.executeUpdate();  System.***out***.println(5/0);  sql="delete from t\_msg where id=?";  pstat=con.prepareStatement(sql);  pstat.setInt(1, id);  pstat.executeUpdate();  con.commit();  } **catch** (Exception e) {  **try** {  con.rollback();  } **catch** (SQLException e1) {  e1.printStackTrace();  }  e.printStackTrace();  }**finally**{  JDBCUtil.*closePraparedStatement*(pstat);  JDBCUtil.*closeConnection*(con);  }  } |

我后来发现其实有没有rollback()这个方法都无所谓，只要设置不自动提交事务、提交事务这两句话在，程序执行过程中出现错误都能回滚。