**JNDI**是 Java 命名与目录接口（Java Naming and Directory Interface），在J2EE规范中是重要的规范之一，不少专家认为，没有透彻理解JNDI的意义和作用，就没有真正掌握J2EE特别是EJB的知识。

那么，JNDI到底起什么作用？  
要了解JNDI的作用，我们可以从“如果不用JNDI我们怎样做？用了JNDI后我们又将怎样做？”这个问题来探讨。

**没有JNDI的做法：**  
程序员开发时，知道要开发访问MySQL数据库的应用，于是将一个对 MySQL JDBC 驱动程序类的引用进行了编码，并通过使用适当的 JDBC URL 连接到数据库。  
就像以下代码这样：

Connection conn=null;  
try {  
  Class.forName("com.mysql.jdbc.Driver",  
                true, Thread.currentThread().getContextClassLoader());  
  conn=DriverManager.getConnection("jdbc:mysql://MyDBServer?user=qingfeng&password=mingyue");  
  /\* 使用conn并进行SQL操作 \*/  
  ......  
  conn.close();  
}   
catch(Exception e) {  
  e.printStackTrace();  
}   
finally {  
  if(conn!=null) {  
    try {  
      conn.close();  
    } catch(SQLException e) {}  
  }  
}

这是传统的做法，也是以前非Java程序员（如Delphi、VB等）常见的做法。这种做法一般在小规模的开发过程中不会产生问题，只要程序员熟悉Java语言、了解JDBC技术和MySQL，可以很快开发出相应的应用程序。  
  
**没有JNDI的做法存在的问题：**  
1、数据库服务器名称MyDBServer 、用户名和口令都可能需要改变，由此引发JDBC URL需要修改；  
2、数据库可能改用别的产品，如改用DB2或者Oracle，引发JDBC驱动程序包和类名需要修改；  
3、随着实际使用终端的增加，原配置的连接池参数可能需要调整；  
4、......  
  
解决办法：  
程序员应该不需要关心“具体的数据库后台是什么？JDBC驱动程序是什么？JDBC URL格式是什么？访问数据库的用户名和口令是什么？”等等这些问题，程序员编写的程序应该没有对 JDBC 驱动程序的引用，没有服务器名称，没有用户名称或口令 —— 甚至没有数据库池或连接管理。而是把这些问题交给J2EE容器来配置和管理，程序员只需要对这些配置和管理进行引用即可。  
  
由此，就有了JNDI。  
  
**用了JNDI之后的做法：**  
首先，在在J2EE容器中配置JNDI参数，定义一个数据源，也就是JDBC引用参数，给这个数据源设置一个名称；然后，在程序中，通过数据源名称引用数据源从而访问后台数据库。  
具体操作如下（以JBoss为例）：  
1、配置数据源  
在JBoss的 D:/jboss420GA/docs/examples/jca 文件夹下面，有很多不同数据库引用的数据源定义模板。将其中的 mysql-ds.xml 文件Copy到你使用的服务器下，如 D:/jboss420GA/server/default/deploy。  
修改 mysql-ds.xml 文件的内容，使之能通过JDBC正确访问你的MySQL数据库，如下：  
<?xml version="1.0" encoding="UTF-8"?>  
<datasources>  
<local-tx-datasource>  
    <jndi-name>MySqlDS</jndi-name>  
    <connection-url>jdbc:mysql://localhost:3306/lw</connection-url>  
    <driver-class>com.mysql.jdbc.Driver</driver-class>  
    <user-name>root</user-name>  
    <password>rootpassword</password>  
<exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter</exception-sorter-class-name>  
    <metadata>  
       <type-mapping>mySQL</type-mapping>  
    </metadata>  
</local-tx-datasource>  
</datasources>  
  
这里，定义了一个名为MySqlDS的数据源，其参数包括JDBC的URL，驱动类名，用户名及密码等。  
  
2、在程序中引用数据源：

Connection conn=null;  
try {  
  Context ctx=new InitialContext();  
  Object datasourceRef=ctx.lookup("java:**MySqlDS**"); //引用数据源  
  DataSource ds=(Datasource)datasourceRef;  
  conn=ds.getConnection();  
  /\* 使用conn进行数据库SQL操作 \*/  
  ......  
  c.close();  
}   
catch(Exception e) {  
  e.printStackTrace();  
}   
finally {  
  if(conn!=null) {  
    try {  
      conn.close();  
    } catch(SQLException e) { }  
  }  
}

直接使用JDBC或者通过JNDI引用数据源的编程代码量相差无几，但是现在的程序可以不用关心具体JDBC参数了。  
在系统部署后，如果数据库的相关参数变更，只需要重新配置 mysql-ds.xml 修改其中的JDBC参数，只要保证数据源的名称不变，那么程序源代码就无需修改。  
  
由此可见，JNDI避免了程序与数据库之间的紧耦合，使应用更加易于配置、易于部署。  
  
**JNDI的扩展：**  
JNDI在满足了数据源配置的要求的基础上，还进一步扩充了作用：所有与系统外部的资源的引用，都可以通过JNDI定义和引用。  
  
所以，在J2EE规范中，J2EE 中的资源并不局限于 JDBC 数据源。引用的类型有很多，其中包括资源引用（已经讨论过）、环境实体和 EJB 引用。特别是 EJB 引用，它暴露了 JNDI 在 J2EE 中的另外一项关键角色：查找其他应用程序组件。  
  
EJB 的 JNDI 引用非常类似于 JDBC 资源的引用。在服务趋于转换的环境中，这是一种很有效的方法。可以对应用程序架构中所得到的所有组件进行这类配置管理，从 EJB 组件到 JMS 队列和主题，再到简单配置字符串或其他对象，这可以降低随时间的推移服务变更所产生的维护成本，同时还可以简化部署，减少集成工作。 外部资源”。   
  
  
**总结：**  
J2EE 规范要求所有 J2EE 容器都要提供 JNDI 规范的实现。JNDI 在 J2EE 中的角色就是“交换机” —— J2EE 组件在运行时间接地查找其他组件、资源或服务的通用机制。在多数情况下，提供 JNDI 供应者的容器可以充当有限的数据存储，这样管理员就可以设置应用程序的执行属性，并让其他应用程序引用这些属性（Java 管理扩展（Java Management Extensions，JMX）也可以用作这个目的）。JNDI 在 J2EE 应用程序中的主要角色就是提供间接层，这样组件就可以发现所需要的资源，而不用了解这些间接性。

在 J2EE 中，JNDI 是把 J2EE 应用程序合在一起的粘合剂，JNDI 提供的间接寻址允许跨企业交付可伸缩的、功能强大且很灵活的应用程序。这是 J2EE 的承诺，而且经过一些计划和预先考虑，这个承诺是完全可以实现的。

**理解JNDI中 java:comp/env/jdbc/datasource 与 jdbc/datasource 的不同之处**

web容器，web.xml数据源的配置

全局环境配置对数据源的引用：

# DataSource

ces.datasource.jndi=java:comp/env/jdbc/cesDs

在描述JNDI，例如获得数据源时，JNDI地址有两种写法，例如同是  jdbc/testDS 数据源：  
A:        Java:comp/env/jdbc/testDS  
B:        jdbc/testDS

这两种写法，配置的方式也不尽相同，第一种方法应该算是一种利于程序移植或迁移的方法，它的实现与“映射”的概念相同，而B方法，则是一个硬引用。  
java:comp/env 是环境命名上下文（environment naming context（ENC）），是在EJB规范1.1以后引入的，引入这个是为了解决原来JNDI查找所引起的冲突问题，也是为了提高EJB或者J2EE应用的移植性。  
在J2EE中的引用常用的有：  
        JDBC 数据源引用在java:comp/env/jdbc 子上下文中声明   
        JMS 连接工厂在java:comp/env/jms 子上下文中声明   
        JavaMail 连接工厂在java:comp/env/mail 子上下文中声明   
        URL 连接工厂在 java:comp/env/url子上下文中声明

可以通过下面的结构示意来发现这两种描述的不同之处：  
A:       java:comp/env/jdbc/testDS(虚地址)   ------>    映射描述符   ------>        jdbc/testDS (实际的地址)  
B:       jdbc/testDS (实际的地址)  
从这种结构上来看，A的确是便于移植的。

再来看一个例子：  
假如你需要获取datasource，例如：dataSource = (DataSource) ctx.lookup("java:comp/env/jdbc/testDS");  
那么在配置文件中进行资源映射时，在web.xml中,  
      <resource-ref>  
        <res-ref-name>jdbc/testDS</res-ref-name>  
        <res-type>javax.sql.DataSource</res-type>  
        <res-auth>Container</res-auth>  
      </resource-ref>  
在相应的资源配置xml中（不同的应用服务器均不同，WSAD中，可以进行可视化的设置），  
    <reference-descriptor>  
      <resource-description>  
        <res-ref-name>jdbc/DBPool</res-ref-name>  
        <jndi-name>OraDataSource</jndi-name>  
      </resource-description>  
    </reference-descriptor>  
实际服务器中的JNDI名字是OraDataSource，逻辑名jdbc/DBPool只是用来和它作映射的，这样做的好处是为了提高可移植性，移植的时候只需要把配置文件改一下就可以，而应用程序可不用改动。

假如你写了一个一般的应用程序，想直接通过JNDI来获取数据源，那么直接lookup(“mytest”)就可以了（假如服务器上的JNDI为mytest），用第一种写法反而会报错的。