

JDBC, JNDI, DataSource

Что такое JDBC

- JDBC = Java Database Connectivity
- JDBC API описывает интерфейсы и классы для написания приложений работающих с базой данных на языке JAVA
- Приложения написанные на JAVA и JDBC являются «платформо-независимыми» и «независимыми от поставщика»

Что такое JDBC [1]

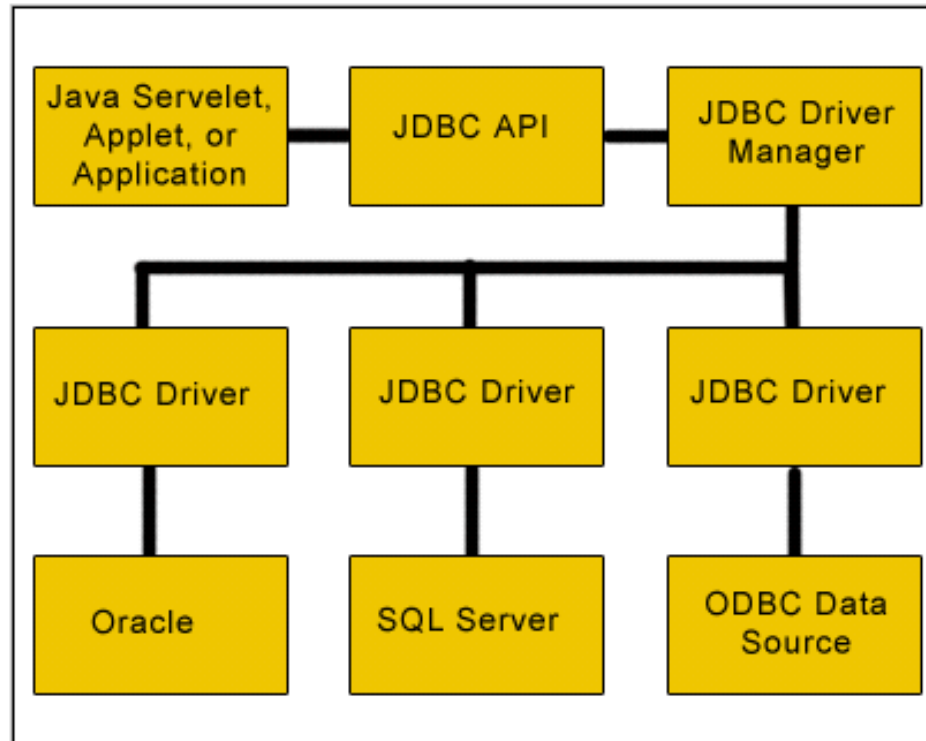
- Расширяет JAVA так что можно общаться с базой данных используя только «чистые» конструкции языка
- А JAVA API может получить доступ к любым табличным данным, в особенности данным сохраненным в реляционной БД:
 - Соединение с источником данных (Data Source), таким как БД
 - Посылка запроса на выборку и обновление данных в БД
 - Извлечение и обработка результатов полученных от БД в ответ на запрос

Что такое JDBC [2]

- JDBC – компоненты
 - API – предоставляет доступ к данным БД через язык JAVA
 - Driver Manager – определяет объекты, которые может подключить JAVA приложение к JDBC драйверу
 - JDBC-ODBC Bridge – предоставляет JDBC доступ через ODBC

Простая архитектура JDBC

- Java приложение вызывает JDBC библиотеку
- JDBC загружает драйвер, который общается с БД
- Движок БД может быть подменен без изменения кода приложения



JDBC программирование

- Загрузка JDBC драйвера БД
- Создание JDBC соединения
- Создание объекта JDBC Statement
- Выполнение SQL выражения через объект JDBC Statement и возвращение ResultSet

Загрузка JDBC драйвера

- Классы которые используются для JDBC программирования содержатся в пакетах *java.sql* и *javax.sql*.

```
import java.sql.*;
```

- Загрузка JDBC драйвера осуществляется вызовом *Class.forName()* с именем класса в качестве аргумента:

```
try {  
    Class.forName("oracle.jdbc.driver.OracleDriver");  
} catch (Exception e) {  
    System.out.println("Unable to load the JDBC driver class");  
    return;  
}
```

Пример популярных JDBC драйверов

Driver class	Start of dbURL	Database
sun.jdbc.odbc.JdbcOdbcDriver	jdbc:odbc:	Bridge to Microsoft ODBC
jdbc.idbDriver	jdbc:idb:	Instant Database (IDB)
oracle.jdbc.Driver.OracleDriver	jdbc:oracle:thin:@server:port#:dbname	Oracle
postgresql.Driver	jdbc:postgres://host/database	PostgreSQL (freeware)
org.gjt.mm.mysql.Driver	jdbc:mysql://host/database	MySQL (freeware)

JDBC – создание соединения (connection)

- JDBC connection – представляет собой сессию/соединение с конкретной БД
- В контексте соединения выполняются SQL запросы и извлекаются результаты
- Объект JDBC connection предоставляет метаданные БД – например информацию о таблицах и их полях

JDBC – создание соединения (connection) [1]

- Driver Manager – класс который управляет JDBC драйверами которые установлены в системе.
- getConnection() – метод используется для установки соединения с БД. Он использует логин, пароль и URL в качестве аргументов.

```
try {  
    Connection con =  
        DriverManager.getConnection(dbURL, name, pass);  
} catch(SQLException x) {  
    System.println("Couldn't get connection !");  
}
```

JDBC – создание выражений (Statement)

- Соединение (Connection) предоставляет возможность взаимодействовать с базой данных
- Используя соединение нужно создать объект Statement путем вызова метода метода ***createStatement()***:

Statement stmt = connection.createStatement();

JDBC – создание выражений (Statement) [1]

- Существуют три типа выражений:
 - **Statement**: простые SQL запросы без параметров
 - **Prepared Statement**: пре-компилируемые SQL запросы с параметрами или без
 - **Callable Statement**: выполнение вызовов хранимых процедур базы данных

JDBC – Prepared Statement

- Более эффективно использовать объект PreparedStatement для отправки SQL выражений
- В отличии от Statement, PreparedStatement передает SQL выражение в базу данных прямо в момент его создания. И в дальнейшем при выполнении SQL в БД он не будет компилироваться.
- Если мы нужно выполнить множество небольших SQL запросов с различными параметрами – использование PreparedStatement наиболее оптимально.

JDBC – Prepared Statement [1]

- Например нужно выполнить несколько запросов:

```
select * from payroll where personnelNo = 12345;  
select * from payroll where personnelNo = 23750;  
select * from payroll where personnelNo = 91720;
```
- Можно использовать PreparedStatement:

```
PreparedStatement ps = con.prepareStatement (  
    "select * from payroll where personnelNo = ?;")
```
- Используем созданное выражение:

```
ps.setInt(1, 12345);  
rs = ps.executeQuery();
```

JDBC – выполнение выражений

- Интерфейс Statement определяет методы для выполнения SQL statement
- *executeQuery(): запросы SELECT*
- *executeUpdate(): запросы на создание и модификацию таблиц*
- *execute(): Выполнение SQL запроса переданного через строку*

Пример SELECT Statement

```
String bar, beer;  
float price;  
ResultSet rs =  
    stmt.executeQuery("SELCET * FROM Sells");  
while(rs.next()) {  
    bar = rs.getString("bar");  
    beer = rs.getString("beer");  
    price = rs.getFloat("price");  
    System.out.println(bar + "sells" + beer +  
        "for" + price + "Dollars");  
}
```

Результат записывается в переменную *rs* типа *ResultSet*. *ResultSet* предоставляет нам **курсор**, который может быть использован для доступа к каждой строке результата.

JDBC – курсор ResultSet

- Курсор при инициализации устанавливается на первую строку
- Каждый вызов метода `next()` приводит к перемещению курсора к следующей строке
- Если существует по крайней мере одна строка, возвращается `TRUE`, иначе вернется `FALSE`

JDBC – Доступ к данным через ResultSet

- Вы можете использовать метод `getXXX` соответствующего типа, чтобы извлечь атрибуты строки: *getString, getFloat*
- Вы можете указать номер или имя колонки для доступа к данным строки курсора:
- `bar = rs.getString("commodity_name");`
- `price = rs.getFloat(2);`
- `beer = rs.getInt("id");`

JDBC - Соответствие типов

SQL type

CHAR, VARCHAR, LONGVARCHAR

NUMERIC, DECIMAL

BIT

TINYINT

SMALLINT

INTEGER

BIGINT

REAL

FLOAT, DOUBLE

BINARY, VARBINARY, LONGVARBINARY

DATE

TIME

TIMESTAMP

Java Type

String

java.math.BigDecimal

boolean

byte

short

int

long

float

double

byte[]

java.sql.Date

java.sql.Time

java.sql.Timestamp

JDBC – Доступ к данным через ResultSet [1]

- Для получения номера строки вашей текущей позиции в результате можно использовать следующие методы: **getRow, isFirst, isBeforeFirst, isLast, isAfterLast**
- Вы получаете скроллируемый курсор для доступа к любой строчке результата. По умолчанию курсор можно скроллировать только вперед только с чтением.
- Режим курсора можно поменять:

```
Statement stmt = con.createStatement(  
    ResultSet.TYPE_FORWARD_ONLY, ResultSet.CURSOR_READ_ONLY);  
ResultSet rs = stmt.executeQuery("SELECT * FROM Sells");
```

JDBC – CREATE/INSERT/UPDATE

- DDL запросы такие как создание и модификация таблиц, также как и изменения данных таблиц выполняются методом *executeUpdate*.

JDBC – CREATE/INSERT/UPDATE [1]

```
Statement stmt = con.createStatement();  
stmt.executeUpdate("CREATE TABLE Sells " +  
    (bar VARCHAR2(40), beer VARCHAR2(40), price REAL)" );  
  
stmt.executeUpdate("INSERT INTO Sells " +  
    "VALUES ('Bar Of Foo', 'BudLite', 2.00)" );  
  
String sqlString = "CREATE TABLE Bars " +  
    "(name VARCHAR2(40), address VARCHAR2(80), license INT)";  
stmt.executeUpdate(sqlString);
```

JDBC - транзакции

- Метод `Connection.setAutoCommit(true/false)` устанавливает/отключает автоматическую фиксацию транзакции
- Режим **AutoCommit** включен по умолчанию
- Метод `Connection.commit()` – фиксирует транзакцию

JDBC – закрытие сессий

*ResultSet.close(), Statement.close(),
Connection.close()*

- Метод **close** позволяет вручную освободить все ресурсы, такие как сетевые соединения и блокировки базы данных, связанные с данным объектом Connection.
- Этот метод автоматически вызывается при сборке мусора; лучше, однако, вручную закрыть Connection, если вы в нем больше не нуждаетесь.

Краткое общение

- Загрузка подходящего драйвера:
`Class.forName("oracle.jdbc.driver.OracleDriver");`
- Получение соединения
`DriverManager.getConnection(dbURL, name, password);`
- Получение объекта Statement (stmt)
`Statement stmt = con.createStatement();`
- Получение ResultSet объекта (rs)
`ResultSet rs = stmt.executeQuery("SELECT * FROM MyTalbe");`
- Итерирование по результатам
`while (rs.next()) {
 int x = rs.getInt("CustNo");
}`
- Обязательное закрытие объектов ResultSet, Statement, и connection.

JDBC – Метаданные (Metadata)

- Варианты получения информации о базе данных или таблице:
 - Обратится к DBA или почитать документацию о базе предоставленную производителем
 - Использовать объект *Metadata* (data about data)
 - *ResultSetMetaData*: объект который используется для получения информации о типах, свойствах столбцов результата.
 - *DatabaseMetaData*: позволяет узнать общую информацию о БД

JDBC – пример получения метаданных

```
DatabaseMetadata meta = con.getMetaData();  
meta.getDatabaseProductName();  
meta.getDatabaseProductVersion();  
meta.getDefaultTransactionIsolation();
```

```
ResultSet rs = st.getResultSet();  
ResultSetMetaData rsmd = rs.getMetaData();  
String name2 = rsmd.getColumnName(2);  
String tableName = rsmd.getTableName(1);
```

JDBC – обработка ошибок

- Клиенту осуществляющему взаимодействие с БД необходимо быть в курсе всех ошибок возвращаемых сервером
 - ***SQLException***: Java исключение – прекращает работу приложения если не будет перехвачено
 - ***SQLWarning***: подкласс ***SQLException*** представляющий собой не критичные ошибки или неожиданные выражения (могут быть проигнорированы)

Пример: перехват ошибок JDBC

```
try{
    con.setAutoCommit(false);
    stmt.executeUpdate( "CREATE TABLE Sells (bar VARCHAR2(40), " +
                        "beer VARHAR2(40), price REAL)");
    stmt.executeUpdate( "INSERT INTO Sells VALUES " +
                        "('Bar Of Foo', 'BudLite', 2.00)");

    con.commit();
    con.setAutoCommit(true);
} catch(SQLException ex) {
    System.err.println( "SQLException: " + ex.getMessage());
    con.rollback();
    con.setAutoCommit(true);
}
```

Исключение сгенерировано так как у BEER указан синтаксически некорректный тип данных

JNDI – Java Naming and Directory Interface

- JNDI – это JAVA API который предоставляет сервис именования и доступа к сервисам объектных каталогов (directory service)
- Использование JNDI предоставляет возможность сохранять и извлекать именованные объекты любых типов

JNDI

- Сам по себе JNDI - просто набор интерфейсов – множество производителей пишут свои реализации
- За интерфейсом JNDI могут скрываться такие сервисы как LDAP, DNS и т.п., а провайдер JNDI в этом случае обеспечивает единый интерфейс к любому сервису каталогов или файловой системе.
- С большинством реализаций JNDI можно работать удаленно, помещая в каталоги различные объекты и получая их оттуда

JNDI - начало

- Получение **InitialContext**

```
1: import javax.naming.InitialContext;  
2: import javax.naming.Context;  
3:  
4: Context ctx = new InitialContext();
```


Если JNDI сервис сетевой, то **ПОЧЕМУ ТАК ПРОСТО?**

- Каждый производитель может реализовать свой класс `javax.naming.InitialContext`, который, во-первых, знает как этот удаленный сервис найти
- Работать такая конструкция будет только внутри сервера приложений, который и предоставляет свою реализацию и где JNDI-сервис работает в той же JVM

JNDI – удаленная работа

- Нужен файл **jndi.properties**, в котором будут указаны параметры подключения к сервису
- Например, для JNDI от сервера приложений Orion параметры могут выглядеть так:

java.naming.factory.initial=

com.evermind.server.ApplicationClientInitialContextFactory

provider.url=ormi://localhost/ejbsamples

java.naming.security.principal=admin

java.naming.security.credentials=123

JNDI – удаленная работа [1]

В примере задаются

- фабрика создания начального контекста, которая и вернет нужную реализацию, зная как работать с сервисом
- ссылка на сервис, формат которой жестко привязан к реализации
- имя и пароль пользователя (как правило, сервера приложений)

Эти параметры опциональны и зависят от сервиса, поэтому для того, чтобы задать их правильно придется заглянуть в документацию.

JNDI – удаленная работа [2]

Как альтернативу можно рассматривать передачу параметров непосредственно в начальный контекст:

```
1: Properties props = new Properties();
2: props.put("java.naming.factory.initial",
3:           "com.evermind.server.ApplicationClientInitialContextFactory");
4: props.put("java.naming.provider.url",
5:           "ormi://localhost/ejbsamples");
6: props.put("java.naming.security.principal", "admin");
7:
8: Context ctx = new InitialContext(props);
```

JNDI – работа с объектами

Публикация объекта в дерево каталогов выполняется следующим образом:

```
1: Context context = new InitialContext();  
2: context.bind("/config/applicationName", "MyApp");
```

Выборку объекта можно выполнить так:

```
1: Context context = new InitialContext();  
2: String appName = (String) context.lookup("/config/applicationName");
```

Большинство реализаций могут публиковать объекты, реализующие интерфейсы `java.io.Serializable` и `java.rmi.Remote`, что дает большие возможности для обмена данными между приложениями, находящимися в разных JVM или даже на разных компьютерах.

Другие возможности JNDI

- установка и модификация атрибутов элементов дерева каталогов
- поиск по дереву каталогов с использованием атрибутов и регулярных выражений
- события, которые могут быть привязаны к созданию, изменению или удалению объектов
- получение списка элементов каталога

JDBC – `java:comp/env`

- Если JNDI может использоваться в любом Java-приложении, то понятие **`java:comp/env`** тесно связано с приложением
- Все что находится в JNDI в каталогах, начинающихся с такой строки **относится только (!) к тому приложению J2EE** (веб-приложению или EJB-компоненту), в котором выполняется работа с деревом каталогов.
- Можно положить, в узел `java:comp/env/jdbc/DS` какой-нибудь источник данных и он будет доступен только из данного приложения.

Декларативная работа с JNDI

Помещаем в пространство имён два объекта:

web.xml

```
1: <env-entry>
2:   <env-entry-name>minvalue</env-entry-name>
3:   <env-entry-type>java.lang.Integer</env-entry-type>
4:   <env-entry-value>12</env-entry-value>
5: </env-entry>
6: <env-entry>
7:   <env-entry-name>maxvalue</env-entry-name>
8:   <env-entry-type>java.lang.Integer</env-entry-type>
9:   <env-entry-value>120</env-entry-value>
10: </env-entry>
```

Получаем их из пространства имен приложения

```
1: InitialContext ictx = new InitialContext();
2: Context myenv = ictx.lookup("java:comp/env");
3: Integer min = (Integer) myenv.lookup("minvalue");
4: Integer max = (Integer) myenv.lookup("maxvalue");
```


JNDI соглашение начального контекста для различных типов объектов

Объект J2EE	Контекст
javax.sql.DataSource	java:comp/env/jdbc
javax.jms.TopicConnectionFactory, javax.jms.QueueConnectionFactory	java:comp/env/jms
javax.mail.Session	java:comp/env/mail
java.net.URL	java:comp/env/url
javax.resource.cci.ConnectionFactory	java:comp/env/eis
javax.xml.registry.ConnectionFactory	java:comp/env/eis/JAXR

JNDI – заключение

- JNDI является мощным средством обмена данными между приложениями.
- J2EE-приложения, использующие в работе EJB или JMS, не могут обойтись без использования JNDI, который хранит ссылки на другие используемые компоненты, очереди сообщений и т.п.

DataSource – источник данных

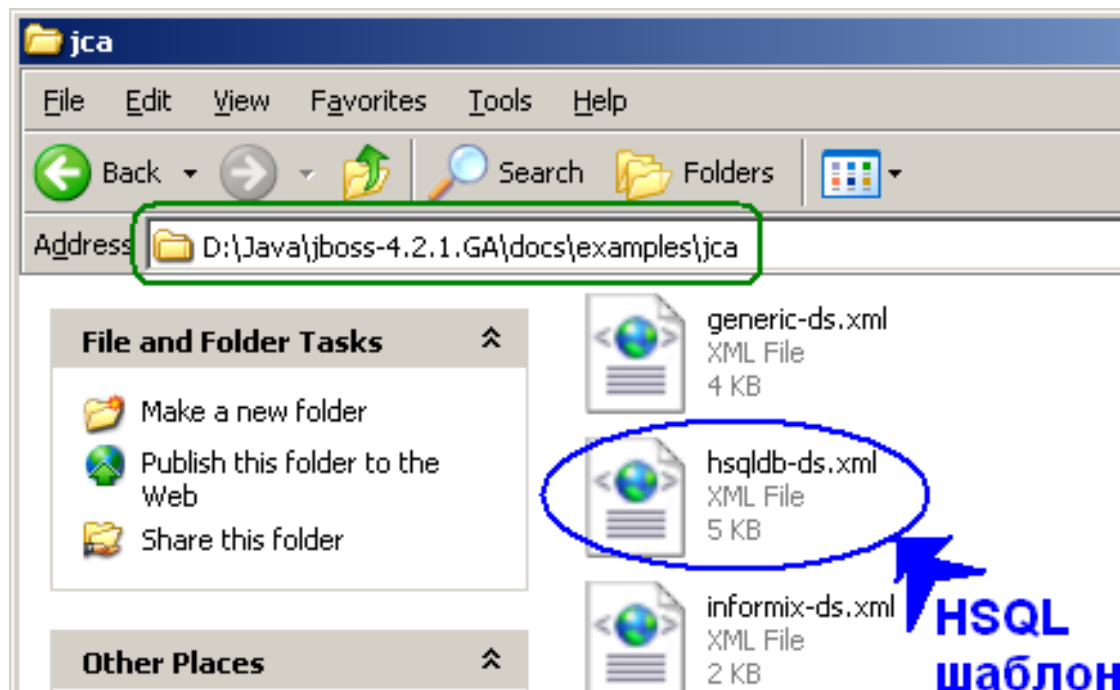
- DataSource (DS) – это фабрика для получения соединения с физическим источником данных который объект DS представляет.
- Является альтернативой DriverManager – т.е. дает возможность получить соединение (Connection)
- Имплементация объекта DataSource обычно регистрируется в службе JNDI

Настройка DataSource

- Настройка DS специфична для каждого из серверов приложений.
- Конфигурация сервера как правило состоит из настройки JDBC драйвера и назначению ему JNDI имени.

Пример настройки DataSource на примере JBoss

- Шаг1. Находим шаблон-пример конфигурационного скрипта для подходящей БД



Пример настройки DataSource на примере JBoss

- Шаг2. Редактируем его в соответствии с нашей базой данных

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- The Hypersonic embedded database JCA connection factory config -->

<datasources>
  <local-tx-datasource>

    <jndi-name>MyHsqlDS</jndi-name>

    <connection-url>jdbc:hsqldb:hsqldb://localhost:1701</connection-url>

    <!-- The driver class -->
    <driver-class>org.hsqldb.jdbcDriver</driver-class>

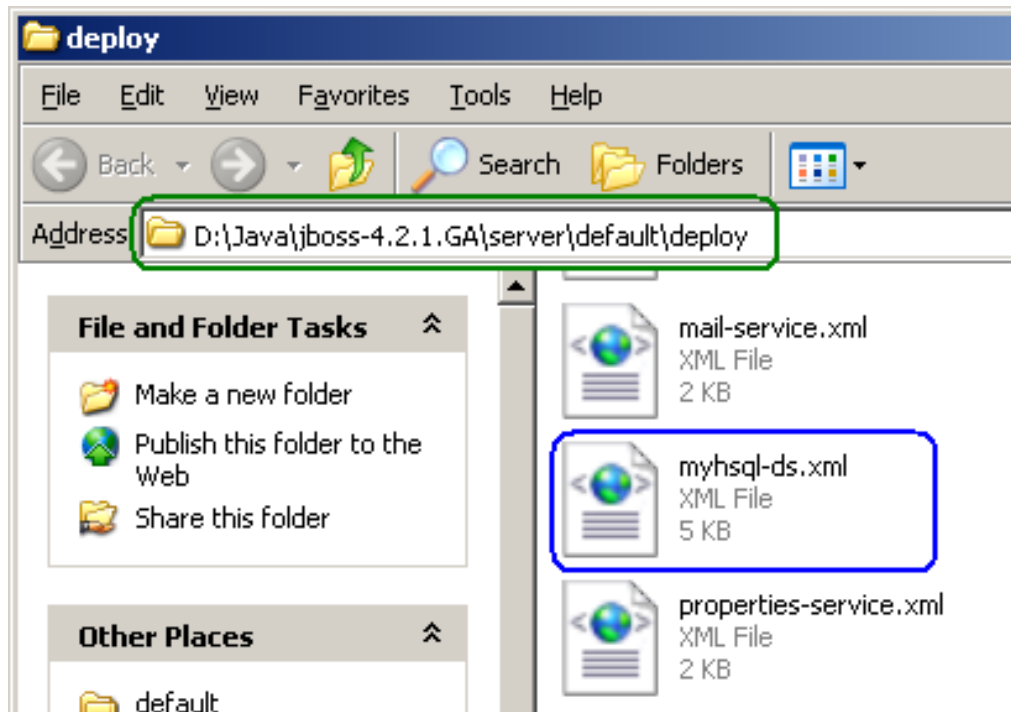
    <!-- The login and password -->
    <user-name>sa</user-name>
    <password></password>

    <!-- The minimum connections in a pool/sub-pool. Pools are lazily constructed on first use -->
    <min-pool-size>5</min-pool-size>
    <!-- The maximum connections in a pool/sub-pool -->
    <max-pool-size>20</max-pool-size>

  </local-tx-datasource>
</datasources>
```

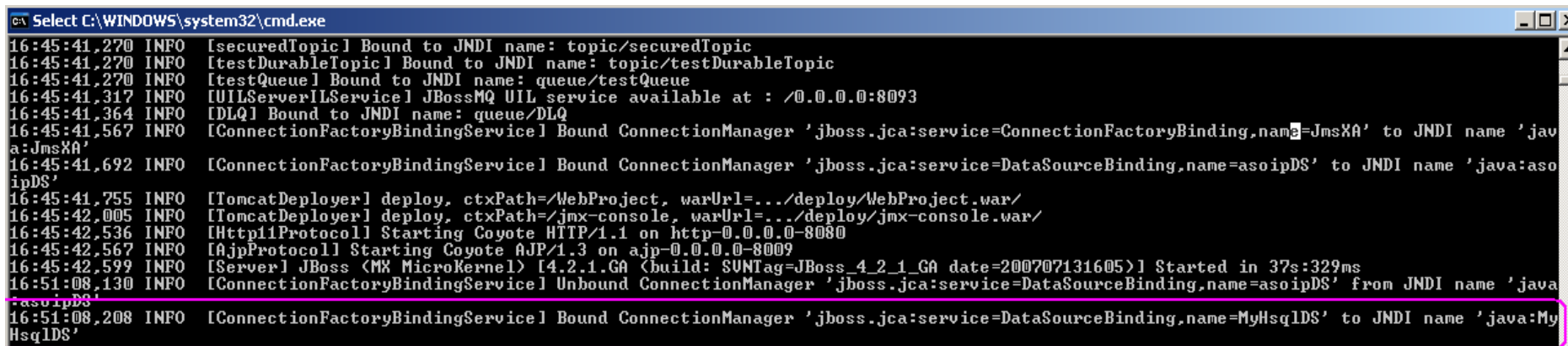
Пример настройки DataSource на примере JBoss

- Шаг3. Копируем его в папку DEPLOY текущей конфигурации нашего сервера приложений



Пример настройки DataSource на примере JBoss

- Шаг5. Запускаем сервер приложений



```
16:45:41,270 INFO [securedTopic] Bound to JNDI name: topic/securedTopic
16:45:41,270 INFO [testDurableTopic] Bound to JNDI name: topic/testDurableTopic
16:45:41,270 INFO [testQueue] Bound to JNDI name: queue/testQueue
16:45:41,317 INFO [UILServerILService] JBossMQ UIL service available at : /0.0.0.0:8093
16:45:41,364 INFO [DLQ] Bound to JNDI name: queue/DLQ
16:45:41,567 INFO [ConnectionFactoryBindingService] Bound ConnectionManager 'jboss.jca:service=ConnectionFactoryBinding,name=JmsXA' to JNDI name 'java:JmsXA'
16:45:41,692 INFO [ConnectionFactoryBindingService] Bound ConnectionManager 'jboss.jca:service=DataSourceBinding,name=asoipDS' to JNDI name 'java:asoipDS'
16:45:41,755 INFO [TomcatDeployer] deploy, ctxPath=/WebProject, warUrl=../deploy/WebProject.war/
16:45:42,005 INFO [TomcatDeployer] deploy, ctxPath=/jmx-console, warUrl=../deploy/jmx-console.war/
16:45:42,536 INFO [Http11Protocol] Starting Coyote HTTP/1.1 on http-0.0.0.0-8080
16:45:42,567 INFO [AjpProtocol] Starting Coyote AJP/1.3 on ajp-0.0.0.0-8009
16:45:42,599 INFO [Server] JBoss (MX MicroKernel) [4.2.1.GA (build: SUNTag=JBoss_4_2_1_GA date=200707131605)] Started in 37s:329ms
16:51:08,130 INFO [ConnectionFactoryBindingService] Unbound ConnectionManager 'jboss.jca:service=DataSourceBinding,name=asoipDS' from JNDI name 'java:asoipDS'
16:51:08,208 INFO [ConnectionFactoryBindingService] Bound ConnectionManager 'jboss.jca:service=DataSourceBinding,name=MyHsqlDS' to JNDI name 'java:MyHsqlDS'
```

- Все вышеуказанные операции можно также производить «на горячую» – т.е. при запущенном сервере JBoss

Получение DataSource из JNDI

Получение объекта источника данных,
который хранится в JNDI:

web.xml

```
1: <resource-ref>
2:   <res-ref-name>jdbc/AccountDS</res-ref-name>
3:   <res-type>javax.sql.DataSource</res-type>
4:   <res-auth>Container</res-auth>
5: </resource-ref>
```

Jboss-web.xml

```
4 <jboss-web>
5   <resource-ref>
6     <res-ref-name>jdbc/AccountsDS</res-ref-name>
7     <jndi-name>jdbc:AccountsDS</jndi-name>
8   </resource-ref>
9 </jboss-web>
```

Java код

```
1: InitialContext ictx = new InitialContext();
2: DataSource ds = ictx.lookup("java:comp/env/jdbc/AccountDS");
```

Спасибо за внимание