Kurs programowania aplikacji bazodanowych

Wykład 2

Paweł Rajba

Instytut Informatyki
Uniwersytet Wrocławski

Plan wykładu

- Połączenie za pomocą JNDI i DataSource
- Pule połączeń
- Klasy RowSet
- Wiązanie danych

Połączenie za pomocą JNDI i DataSource

- Krótki wstęp do usług katalogowych
- Dotychczas w celu uzyskania obiektu klasy *Connection* wykorzystywaliśmy `DriverManager.getConnection(URL)`
- Lepszym sposobem jest wykorzystanie *JNDI* i interfejsu *DataSource*
- Scenariusz postępowania:
  - Najpierw tworzymy źródło danych z wykorzystaniem interfejsu *DataSource*
  - Następnie, gdzieś w jakichś usługach katalogowych ten DataSource kojarzymy z przyjazną nazwą (robimy tzw. *bind*)
    - Powyższe dwa punkty są zwykle wykonywane przez administratorów bez udziału programistów
  - W aplikacji tworzymy kontekst powiązany z tymi usługami katalogowymi i pobieramy poprzez przyjazną nazwę zarejestrowany tam wcześniej DataSource (robimy tzw. *lookup*)

Połączenie za pomocą JNDI i DataSource

- Co zrobić, żeby za pomocą klasy *DataSource* dostać obiekt klasy *Connection*?
  - Tworzymy obiekt DataSource za pomocą już konkretnej implementacji interfejsu *DataSource*
    - w przypadku DBMS Oracle wykorzystamy klasę *oracle.jdbc.pool.OracleDataSource*
    - w przypadku DBMS SQL Server wykorzystay klasę *com.microsoft.sqlserver.jdbc.SQLServerDataSource*
  - Ustawiamy parametry podstawowe parametry: adres serwera, port, nazwę użytkownika, hasło, nazwę bazy danych
  - Wywołujmy funkcję `DataSource.getConnection()` lub `DataSource.getConnection(String username, String password)`
- Dodatkowo możemy ustawić lub pobrać obiekt, który jest odpowiedzialny za zapisywanie logów
  - `void setLogWriter(PrintWriter out)`
  - `PrintWriter getLogWriter()`

Połączenie za pomocą JNDI i DataSource

- A w jaki sposób pobrać obiekt DataSource z usługi katalogowej?
  - Tworzymy obiekt klasy *InitialContext*
  - Ustawiamy lokalizację usługi katalogowej wykorzystując funkcję `Context.addToEnvironment(String propName, Object propVal)`
  - Pobieramy obiekt klasy *DataSource* za pomocą metody `Context.lookup(nazwa)`
- Zalety rozważanego rozwiązania
  - Dzięki wykorzystaniu inferfejsu *DataSource*, pobranie obiektu *Connection* jest takie samo dla każdego *DBMS*
  - Dzięki *JNDI*, problem utworzenia obiektu DataSource delegujemy najczęściej na administratora usługi katalogowej
  - Kod napisany z wykorzystaniem tego schematu jest bardziej elastyczny i łatwiejszy do napisania

Przykład

Przykład: JNDIExample

- JNDIExample.java

Pule połączeń

Komunikacja aplikacji z DBMS przebiega zwykle w jednym z dwóch schematów:

- Nawiązanie połączenia
  wykonywanie zapytań
  ... (przerwa)
  wykonywanie zapytań
  Zamknięcie połączenia
- Nawiązanie połączenia
  wykonywanie zapytań
  Zamknięcie połączenia
  Nawiązanie połączenia
  wykonywanie zapytań
  Zamknięcie połączenia

Każde w powyższych rozwiązań ma swoje wady. Rozwiązaniem jest użycie puli połączeń.

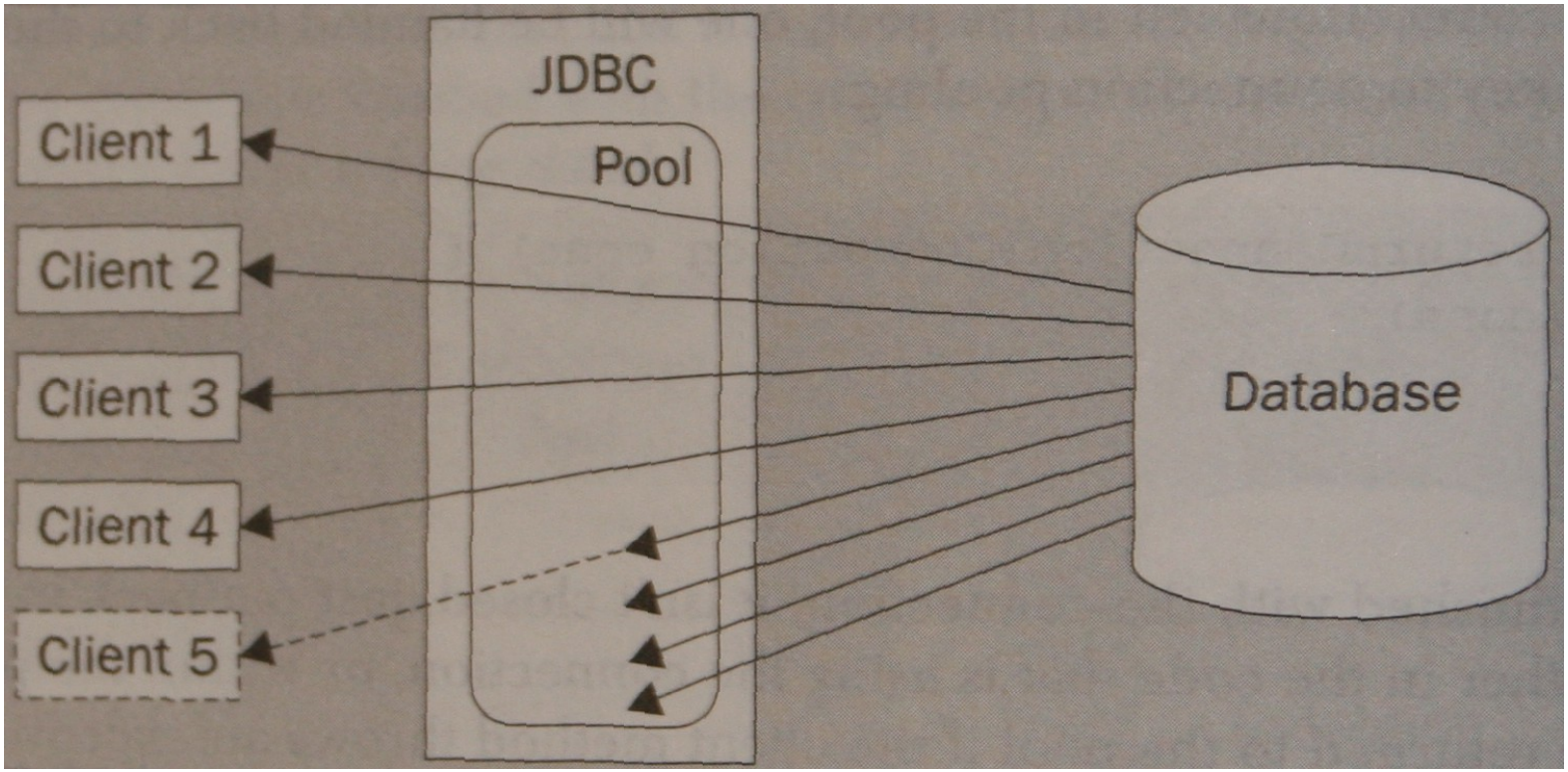
Pule połączeń

Jak mogłaby działać taka pula?

- Tworzymy klasę Manager, która przechowuje stos 20 obiektów klasy Connection
- Klasa X, która chce dostać obiekt Connection łączy się z obiektem Manager i dostaje obiekt Connection
  - `Connection getConnection()`
- Po skończeniu działania, klasa X oddaje obiektowi Manager pobrany wcześniej obiekt Connection
  - `void returnConnection(Connection conn)`

Pule połączeń są zaimplementowane w JDBC Optional Package i korzystanie z nich jest prawie przeźroczyste.

Pule połączeń



Źródło: Dany Ayers, et al., Professional Java Data, Wrox 2001

Przykład

Przykład: ConnectionPooling

- ConnectionPooling.java

Obiekty klasy RowSet

- Wszystkie implementacje RowSet są komponentami JavaBeans
- Dziedziczą z ResultSet
- Mamy dwa rodzaje takich obiektów
  - połączeniowe
  - bezpołączeniowe
- Implementacje połaczeniowe
  - JdbcRowSet
- Implementacje bezpołączeniowe
  - CachedRowSet
  - FilteredRowSet
  - JoinRowSet
- Są też RowSet typowo do zastosowań typu Web (WebRowSet)
  - Temu przyjrzymy się w ostatniej części wykładu
- Będziemy korzystać z implementacji Suna:
  - http://java.sun.com/products/jdbc/download.html

Obiekty klasy RowSet

- Żeby wykorzystać możliwości obiektu *FilteredRowSet* trzeba utworzyć klasę implementującą interfejs *Predicate*
- Interfejs ten ma trzy metody
  - `boolean evaluate(Object value, int column)`
  - `boolean evaluate(Object value, String columnName)`
  - `boolean evaluate(RowSet rs)`

Przykład

Przykład: RowSet

- JdbcRowSetSample.java
- CachedRowSetSample.java
- JoinRowSetSample.java
- FilteredRowSetSample.java, Range1.java, Range2.java
- FilteredRowSetSample2.java, Range3.java
- FilteredRowSetSample3.java, ParityPredicate.java

Wiązanie danych

- Większość tworzonych obecnie aplikacji to aplikacje webowe lub aplikacje typu desktop
- W tego typu aplikacjach występują komponenty wizualne, które trzeba wypełnić danymi
  - Najpopularniejsze komponenty to: *JList*, *JTable*, *JTree*
- Wypełnienie danymi może być wykonane na różne sposoby
  - My wykorzystamy mechanizm polegający na zdefiniowaniu modelu danych
  - Utworzymy także odpowiednich słuchaczy, którzy będą reagować na zmiany w widoku i odpowiednio modyfikować model
- Dalej przyjrzymy się kolejnym komponentom i ich obsłudze

Wiązanie danych

Komponent **JList**

- Reprezentuje dane w postaci listy
- Utworzenie modelu polega implementacji interfejsu *ListModel*
  - `void addListDataListener(ListDataListener l)`
  - `Object getElementAt(int index)`
  - `int getSize()`
  - `void removeListDataListener(ListDataListener l)`
- Wygodne może być użycie jednej z gotowych implementacji:
  - *DefaultComboBoxModel* dla *JComboBox*
  - *DefaultListModel* dla *JList*
- Klasa, która będzie obsługiwać zmiany w modelu musi implementować interfejs *ListDataListener*
  - `void contentsChanged(ListDataEvent e)`
  - `void intervalAdded(ListDataEvent e)`
  - `void intervalRemoved(ListDataEvent e)`

Wiązanie danych

Komponent **JTable**

- Reprezentuje dane w postaci tabelarycznej
- Utworzenie modelu polega na implementacji interfejsu *TableModel*
  - `void addTableModelListener(TableModelListener l)`
  - `Object getValueAt(int row, int col)`
  - `String getColumnName(int col)`
  - `int getColumnCount()`, `int getRowCount()`
  - `void removeTableModelListener(TableModelListener l)`
  - `void setValueAt(Object aValue, int row, int col)`
  - `boolean isCellEditable(int row, int col)`
  - `Class getColumnClass(int column)`
- Wygodne może być użycie implementacji *DefaultTableModel*
- Klasa, która będzie obsługiwać zmiany w modelu musi implementować interfejs *TableModelListener*
  - `void tableChanged(TableModelEvent e)`

Wiązanie danych

Komponent **JTree**

- Reprezentuje dane w postaci drzewa ukorzenionego
- Utworzenie modelu polega na implementacji interfejsu *TreeModel*
  - `void addTreeModelListener(TreeModelListener l)`
  - `Object getChild(Object parent, int index)`
  - `int getChildCount(Object parent)`
  - `int getIndexOfChild(Object parent, Object child)`
  - `Object getRoot()`
  - `boolean isLeaf(Object node)`
  - `void removeTreeModelListener(TreeModelListener l)`
  - `void valueForPathChanged(TreePath path, Object newValue)`
- Wygodne może być użycie implementacji *DefaultTreeModel*
- Elementami drzewa mogą być obiekty klasy *DefaultMutableTreeNode*

Wiązanie danych

- Klasa, która będzie obsługiwać zmiany w modelu musi implementować interfejs *TreeModelListener*
  - `void treeNodesChanged(TreeModelEvent e)`
  - `void treeNodesInserted(TreeModelEvent e)`
  - `void treeNodesRemoved(TreeModelEvent e)`
  - `void treeStructureChanged(TreeModelEvent e)`

Przykład

Przykład: RowSet

- ListComboBinding.java
- TableBinding.java
- TreeBinding.java