

riccioni@univali.br

Tópicos Especiais em Computação: Comunicação e Acesso a Banco de Dados em Java

1

- Cliente-Servidor
 - Socket e ServerSocket
 - Interação no MVC: AplicacaoCliente e AplicacaoServidor
- Multithreading
 - classe Thread e
 - interface Runnable
- Acesso a Banco de Dados
 - JDBC

Tópicos Especiais em Computação: Comunicação e Acesso a Banco de Dados em Java

2

- No pacote `java.net`, há duas classes que permitem implementar comunicação cliente-servidor:
 - `Socket` e
 - `ServerSocket`.
- O servidor deve aguardar um pedido de conexão, aceitá-lo e responder ao cliente de acordo com o protocolo implementado.
- Como veremos, no servidor com `multithread` é possível atender a mais de um cliente simultaneamente.

Tópicos Especiais em Computação: Comunicação e Acesso a Banco de Dados em Java

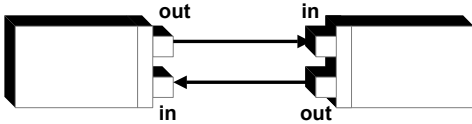
3

A Classe Socket

- Utilizada para representar a conexão entre o cliente e o servidor.
- `Socket(String host, int porta)`
 - cria um socket e o conecta ao servidor definido pelo host e a porta.
- `void close()`
 - libera os recursos alocados.
- `InputStream getInputStream()`
 - obtém o stream de entrada associado ao socket.
- `OutputStream getOutputStream()`
 - obtém o stream de saída associado ao socket.

Cliente-Servidor

- Em Java, os canais de comunicação são unidirecionais (*one-way*).
- Assim, tanto o cliente quanto o servidor precisam de dois streams:

[illegible]

A Classe ServerSocket

- Para aceitar conexões, o servidor deve utilizar um `ServerSocket`.
- `ServerSocket(int porta)`
 - cria um socket nesta porta da máquina local.
- `Socket accept()`
 - aguarda até que uma conexão seja aceita. (bloqueante) Em seguida, retorna o `Socket` que representa a conexão com determinado cliente.
- `void close()`
 - libera os recursos alocados.

Programação em Java: Prof. Ricconi

Teoria e Prática

UNIVALI - São José

http://www.sj.univali.br

Estrutura Geral de um Servidor

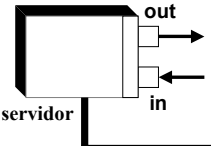
```

public class Servidor {
    private ServerSocket ss;
    private ObjectInputStream in;
    private ObjectOutputStream out;

    public Servidor(int portaServ) {
        ss = new ServerSocket(portaServ);
        Socket cliente = ss.accept(); // aguardando conexão...
        out = new ObjectOutputStream(cliente.getOutputStream());
        out.flush();
        in = new ObjectInputStream(cliente.getInputStream());

        // implementação do protocolo da aplicação...
    }
}

```



Tópicos Especiais em Computação: Comunicação e Acesso a Banco de Dados em Java

7

Programação em Java: Prof. Ricconi

Teoria e Prática

UNIVALI - São José

http://www.sj.univali.br

Estrutura Geral de um Cliente

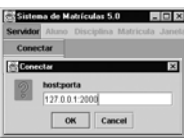
```

public class Cliente {
    private Socket cs;
    private ObjectInputStream in;
    private ObjectOutputStream out;

    public Cliente(String hostServ, int portaServ) {
        // estabelece a conexão...
        cs = new Socket(hostServ, portaServ);
        out = new ObjectOutputStream(cs.getOutputStream());
        out.flush();
        in = new ObjectInputStream(cs.getInputStream());

        // implementação do protocolo da aplicação...
    }
}

```



Tópicos Especiais em Computação: Comunicação e Acesso a Banco de Dados em Java

8

Programação em Java: Prof. Ricconi

Teoria e Prática

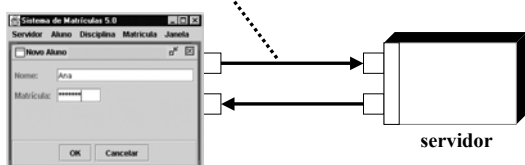
UNIVALI - São José

http://www.sj.univali.br

Exemplo: Cadastrando um Aluno

- Para cadastrar um aluno, vamos definir a seguinte mensagem que será enviada ao servidor:

"CADASTRE_ALUNO nome matrícula"



Tópicos Especiais em Computação: Comunicação e Acesso a Banco de Dados em Java

9

Nome do Método Resolvido Dinamicamente

```
public String perform(FrBean frBean, String operacao, Hashtable
    parametros) try {
    String nomeMetodo = operacao.toLowerCase();
    StringTokenizer st = new StringTokenizer(nomeMetodo, "_");
    nomeMetodo = st.nextToken();           // INSERT → insert
    while(st.hasMoreTokens()) {           // SELECT_COUNT →
selectCount
        String s = st.nextToken();
        nomeMetodo += (" " + s.charAt(0)).toUpperCase() +
s.substring(1);
    }
    Method metodo = estaClasse.getDeclaredMethod(nomeMetodo,
tiposParametros);
    metodo.invoke(this, new Object[] { frBean, parametros } );
    return "OK";
} catch(Exception ex) {
    return ex.toString();
}
```

Método Executado na Subclasse

```
public class AlunoAction extends Action {  
  
    public void insert(FrBean frBean, Hashtable  
        parametros) throws Exception  
  
    {  
        Aluno novo = new Aluno();  
        PropertyUtils.copyProperties(novo, frBean);  
        AlunoSQL.insert(novo,  
            AplicacaoServidor.getConnection());  
    }  
}
```

Multithreading

- Há várias formas de implementação para atender a vários clientes simultaneamente.
- Mas o uso de threads torna a implementação bastante simples.
- Na API do Java, no próprio pacote `java.lang`, está a classe `Thread`, que implementa a seguinte interface:

```
interface Runnable {  
    void run(); ..... Ponto de entrada  
                        de uma thread.  
}
```

Programação em Java:
 Prof. Riccioni
 Teoria e Prática
 UNIVALI - São José
 http://www.sj.univali.br

Acesso a Banco de Dados com JDBC

- JDBC – Java DataBase Connectivity API (pacote java.sql)
- Oferece uma interface em linguagem Java para acesso a banco de dados.
- JDBC 1 (1996)
- JDBC 2 (1998) – oferece *scrollable cursors*,...
- JDBC 3 (2001) – incluído na J2SE API v1.4: avanços em connection pools, ...
- Alguns gerenciadores de banco de dados incluem uma máquina virtual para executar *stored procedures* em Java (SQLJ).

Tópicos Especiais em Computação: Comunicação e Acesso a Banco de Dados em Java

22

Programação em Java:
 Prof. Riccioni
 Teoria e Prática
 UNIVALI - São José
 http://www.sj.univali.br

Caminho da Aplicação ao Banco de Dados

```

graph TD
    App[Aplicação em Java] --> DM[Driver Manager (JDBC)]
    DM --> Bridge[JDBC/ODBC Bridge]
    DM --> JDBC[JDBC driver]
    Bridge --> ODBC[ODBC driver]
    ODBC --> DB[(Banco de Dados)]
    JDBC --> DB
    
```

Fornecido pela Sun.

Fornecido pelo fabricante.

Tópicos Especiais em Computação: Comunicação e Acesso a Banco de Dados em Java

23

Programação em Java:
 Prof. Riccioni
 Teoria e Prática
 UNIVALI - São José
 http://www.sj.univali.br

JDBC (Conceitos Básicos)

- URL(Uniform Resource Locator) o localizador uniforme de recursos de um banco de dados é semelhante ao endereço de algo na Internet, especifica o tipo de protocolo utilizado na transferência, seguido de dois pontos e barra dupla, para depois indicar o nome do computador, o diretório e o nome do documento procurado:
`jdbc:nome do protocolo:informações adicionais`
 Exemplos:
 - `jdbc:odbc://sj.univali.br:3228/MEUBANCO;PWD=riccioni`
 - Acesso ao banco de dados MEUBANCO, na porta 3228 de sj.univali.br, usando o atributo ODBC PWD configurado como riccioni.
 - `jdbc:odbc:SISMAT`
 - acesso a uma fonte de dados ODBC cujo nome é SISMAT.
 - `jdbc:oracle:thin:@150.162.50.201:1521:orcl`
 - acesso ao banco de dados da oracle na porta 1521 da máquina 150.162.50.201 cujo identificador SID = orcl.

Tópicos Especiais em Computação: Comunicação e Acesso a Banco de Dados em Java

24

Estabelecendo uma Conexão

- ```
String url = "jdbc:odbc:SISMAT";
String user = "anonymous";
String password="guest";
Connection con;

try {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

 con = DriverManager.getConnection(url, user, password);
} catch (ClassNotFoundException excecao1) {
 // Impossível carregar o bridge JDBC/ODBC.
} catch (SQLException excecao2) {
 // Impossível estabelecer a conexão.
}
```



## ResultSet

- Para ler o resultado da busca, precisamos obter os valores de cada coluna da linha atual:

## PreparedStatement

- Podemos preparar um comando e utilizá-lo várias vezes, apenas definindo parâmetros:

```
SELECT nome
FROM Alunos
WHERE matricula = matricula digitada
```

```
String busca = "SELECT nome " +
 "FROM Alunos " +
 "WHERE matricula = ?";
PreparedStatement stmt = con.prepareStatement(busca);
stmt.setString(1, jtfMatricula.getText());
ResultSet rs = stmt.executeQuery();
//...
```

## Conjunto de Resultados Roláveis e Atualizáveis

- Valores possíveis de tipo e concorrência:

|                         |                                                                                        |
|-------------------------|----------------------------------------------------------------------------------------|
| TYPE_FORWARD_ONLY       | O conjunto de Resultados não é rolável                                                 |
| TYPE_SCROLL_INSENSITIVE | O conjunto de Resultados é rolável, mas não é sensível as alteração no Banco de Dados. |
| TYPE_SCROLL_SENSITIVE   | O conjunto de Resultados é rolável, e é sensível as alteração no Banco de Dados.       |

|                  |                                                                              |
|------------------|------------------------------------------------------------------------------|
| CONCUR_READ_ONLY | O conjunto de Resultados não pode ser usado para atualizar o Banco de Dados. |
| CONCUR_UPDATABLE | O conjunto de Resultados pode ser usado para atualizar o Banco de Dados.     |

---

---

---

---

---

---

---

---

---

---

---

---

• Por exemplo, se você quer simplesmente rolar por um conjunto de resultados, mas não quer editar seu dados, então use:

• A partir de então todos os conjuntos de resultado que são retornados pela chamada do método:

- Agora são roláveis. Um conjunto de resultados rolável tem um **cursor** que indica a posição corrente do registro na tabela.

---

---

---

---

---

---

## ResultSetMetaData

- Permite obter a estrutura de um *result set*:

Técnicas Especiais em Computação: Comunicação e Acesso a Bancos de Dados em Java 39

## Técnicos Especiais em Computação: Comunicação e Acesso a Banco de Dados em Java

- Técnicas Especiais em Computação: Comunicação e Acesso a Bancos de Dados em Java 38

Técnicas Especiais em Computação: Comunicação e Acesso a Bancos de Dados em Java 38

Técnicas Especiais em Computação: Comunicação e Acesso a Bancos de Dados em Java 38

## Fechando uma Conexão

- Uma conexão que não será mais utilizada deve ser fechada para liberar os recursos utilizados pelo JDBC:

```
try {
```

```
con.close();
```

```
} catch(SQLException excecao) {
```

```
excecao.printStackTrace();
```

}

- Para verificar se uma conexão ainda está aberta ou fechada:

```
boolean fechada = con.isClosed();
```

---

---

---

---

---

---

## Recuperando Chaves (Auto) – JDBC 3.0

```
Statement stmt = con.createStatement();
```

```
stmt.executeUpdate("INSERT INTO " +
```

“Disciplina(nome)” +

“VALUES (‘Prog. em Java’),

```
Statement.RETURN_GENERATED_KEYS);
```

```
ResultSet rs = stmt.getGeneratedKeys();
```

```
if(rs.next()) { // Recupera a chave gerada automaticamente.
```

```
int codigo = rs.getInt(1); // código da disciplina
```

}

...

---

---

---

---

---

---