DESENVOLVIMENTO DE SISTEMAS WEB

MIDDLEWARE JDBC

Olá!

Ao final desta aula, você será capaz de:

1. Conhecer os fundamentos de Middleware (introdução, princípios básicos; frameworks); 2. Criar aplicativos simples com acesso a banco de dados na Web; 3. Utilizar o NetBeans® na gerência do banco de dados JavaDB.

Nesta aula estudaremos alguns aspectos e definições relativos à implementação do Middleware JDBC. Abordaremos, também, as funcionalidades do NetBeans® para a gerência de banco de dados.

1 Tecnologia *Middleware*

A multinacionalização da economia, das empresas e a ampla concorrência do mercado gerou a necessidade de criação de novos procedimentos para atender aos requisitos de diversas aplicações que devem ser realizadas em um tempo cada vez menor. A *Internet* e a disponibilidade das redes de alta velocidade proporcionaram uma diminuição das distâncias no que se refere à localização das informações. Entretanto, para atender aos novos requisitos das aplicações, tornou-se necessário o acesso a informações armazenadas em diferentes fontes, que podem estar localizadas dentro de uma mesma empresa, em países distantes e em ambientes operacionais diferentes. (*Fonte: BARBOSA*)

BARBOSA, A.C.P. Middleware para Integração de Dados Heterogêneos Baseado em Composição de Frameworks. Rio de Janeiro: PUC, 2001. Tese de Doutorado. Disponível na INTERNET: ftp://ftp.inf.puc-rio.br/pub/docs/theses/01_PhD_barbosa.pdf Arquivo consultado em 9/12/2012.

Neste cenário, surgiu o conceito de *Middleware*. Trata-se de uma infraestrutura projetada para ajudar na gerência da complexidade e da heterogeneidade inerentes a sistemas distribuídos. Tem como objetivo interligar processos clientes a processos servidores, oferecendo um conjunto de serviços que visam diminuir a complexidade do processo de desenvolvimento de uma aplicação.

Fique ligado

De acordo com **HAENDOCHEN FILHO**[1], um middleware situa-se entre vários sistemas operacionais e uma plataforma de programação distribuída, provendo abstrações de alto nível, as quais auxiliam no entendimento da estrutura. Frameworks de middleware disponibilizam serviços de infraestrutura, tornando possível aos desenvolvedores abstrair funcionalidades complexas, como por exemplo: concorrência, gerenciamento, heterogeneidade de plataforma, dentre outros.

[1] HAENDCHEN FILHO, A. Um Framework do tipo Middleware para Sistemas Multiagentes na Internet. Rio de Janeiro: PUC, 2001. Tese de Doutorado. Disponível na INTERNET via www.url: http://www.maxwell.lambda.ele.puc-rio.br/8547/8547_1.PDF Arquivo consultado em: 9/12 /2012.

2 Arquitetura cliente/servidor

Para uma melhor compreensão do assunto abordado, segue, abaixo, uma discussão sobre a sua evolução:

Arquitetura centralizada

Primeiramente, surgiu a arquitetura centralizada *("mainframe")*, onde toda a inteligência é centralizada em um computador central que recebe a informação gerada pela captura da informação do usuário através de um

terminal. Trata-se de uma arquitetura limitada por não suportar facilmente interfaces gráficas com o usuário *("Graphic User Interface"* - GUI) e o acesso a múltiplos bancos de dados geograficamente dispersos. (**Fonte: COSTA**)

COSTA, S.R. Objetos Distribuídos: Conceitos e Padrões. São José dos Campos: INPE, 2000. Dissertação de Mestrado. Disponível na INTERNET via WWW.url: http://mtc-m05.sid.inpe.br/col/sid.inpe.br/deise/2001/04.24.14.21/doc/homepage.pdf

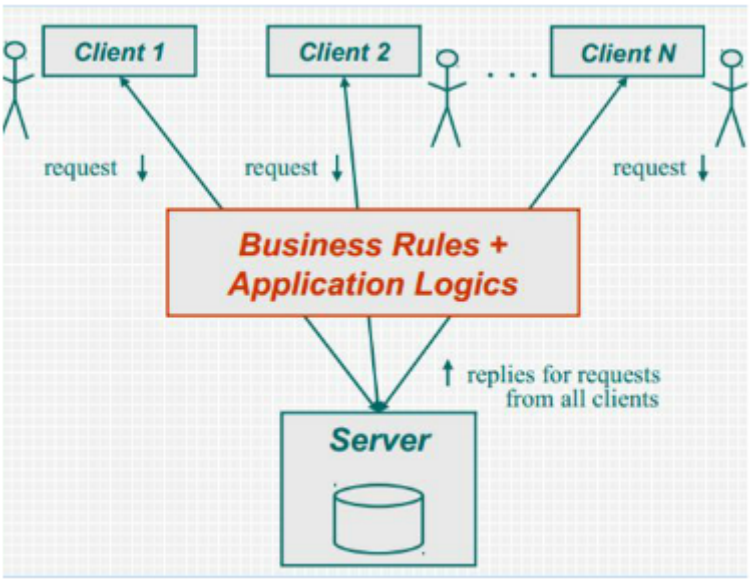
Arquitetura de arquivo compartilhado

Com o aparecimento de redes conectando vários PCs, surgiu a **arquitetura de arquivo compartilhado** (*file sharing*). Nesta arquitetura, o servidor de arquivos envia arquivos da localização compartilhada para o ambiente da estação de trabalho. Neste local, o trabalho requisitado pelo usuário é então executado (incluindo a lógica e os dados). Esta arquitetura também apresenta restrições, pois o bom desempenho está vinculado a um número limitado tanto de compartilhamentos de arquivos como de volume de dados transferidos.

Arquitetura clienta/servidor

Para solucionar estas limitações surgiu a arquitetura cliente/servidor. Nesta arquitetura, um processo é responsável pela manutenção da informação (Servidor), enquanto que outro é responsável pela obtenção dos dados (Cliente). (***Fonte: BATTISTI***) BATTISTI, J. SQL Server 2000: Administração e Desenvolvimento – Curso Completo. Rio de Janeiro: Axcell Books, 2001.

A comunicação cliente/servidor é baseada em troca de mensagens. Segundo COSTA, quando comparada à arquitetura de software centralizada e à arquitetura de compartilhamento de arquivo, apresenta uma melhor usabilidade, flexibilidade, interoperabilidade e escalabilidade.



Arquitetura Cliente/Servidor de duas Camadas (*Two Tier*)

• O cliente comunica-se diretamente com o servidor; • A base de dados fica no servidor; • As regras e a lógica da aplicação ficam no cliente; • Problema de manutenção – toda vez que uma aplicação for alterada, tanto bancos de dados como aplicações clientes precisam ser alteradas; • A aplicação cliente precisa ser instalada em todos os nós.

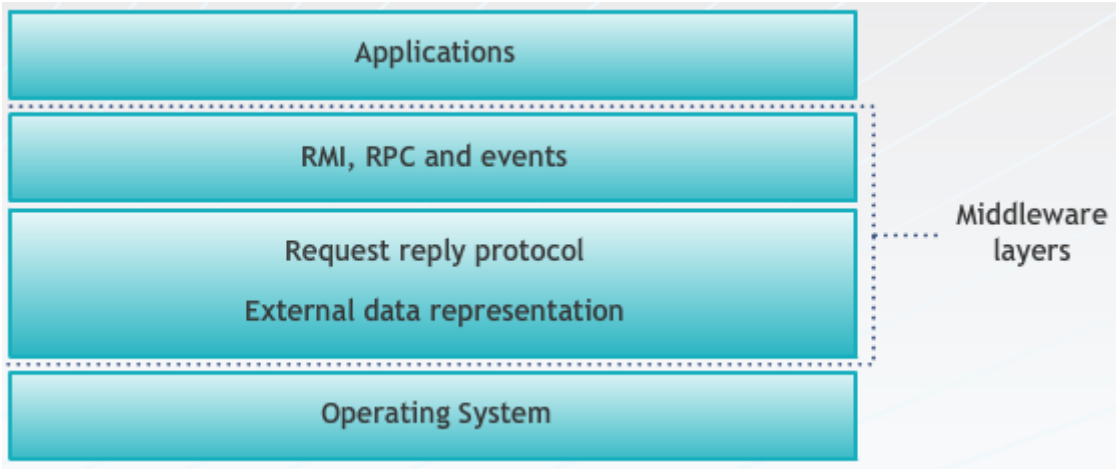
Arquitetura Cliente/Servidor de três Camadas (*Three Tier*)

• Uma camada intermediária é criada entre o servidor e o cliente; • A função da camada intermediária é armazenar as regras do negócio a lógica da aplicação; • O cliente fica responsável apenas pela interface com o usuário; • Qualquer alteração na camada intermediária é imediatamente assumida por todas as aplicações e pelo banco de dados.

3 Ambientes de *Middleware*

Segundo COSTA, uma organização com a necessidade de distribuir uma aplicação pode escolher entre construir um ambiente de trabalho (*framework*) para integração e desenvolvimento próprio ou utilizar produtos existentes no mercado que ofereçam ferramentas de integração e desenvolvimento. Os produtos existentes são baseados nas especificações CORBA da OMG, no DCE (*Distributed Computing Environment*) da OSF, no DCOM da Microsoft®, assim como, no RMI (*Remote Method Invocation*) da linguagem Java.

Estas ferramentas baseiam-se em diversas tecnologias, apresentam diferentes características, mas em alguns pontos elas são similares ou mesmo complementares.



Fonte: Ambientes de Middleware Fonte: COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. Distributed Systems: Concepts and Design. London: Addison Wesley, 2011.

Atenção

O principal objetivo dos serviços *middleware* é permitir que uma plataforma não dependa de APIs específicas, permitindo que aplicações executem em diferentes plataformas e incluem serviços de alto nível que escondam a complexidade de redes e sistemas distribuídos.

4 Middleware para Banco de dados JDBC

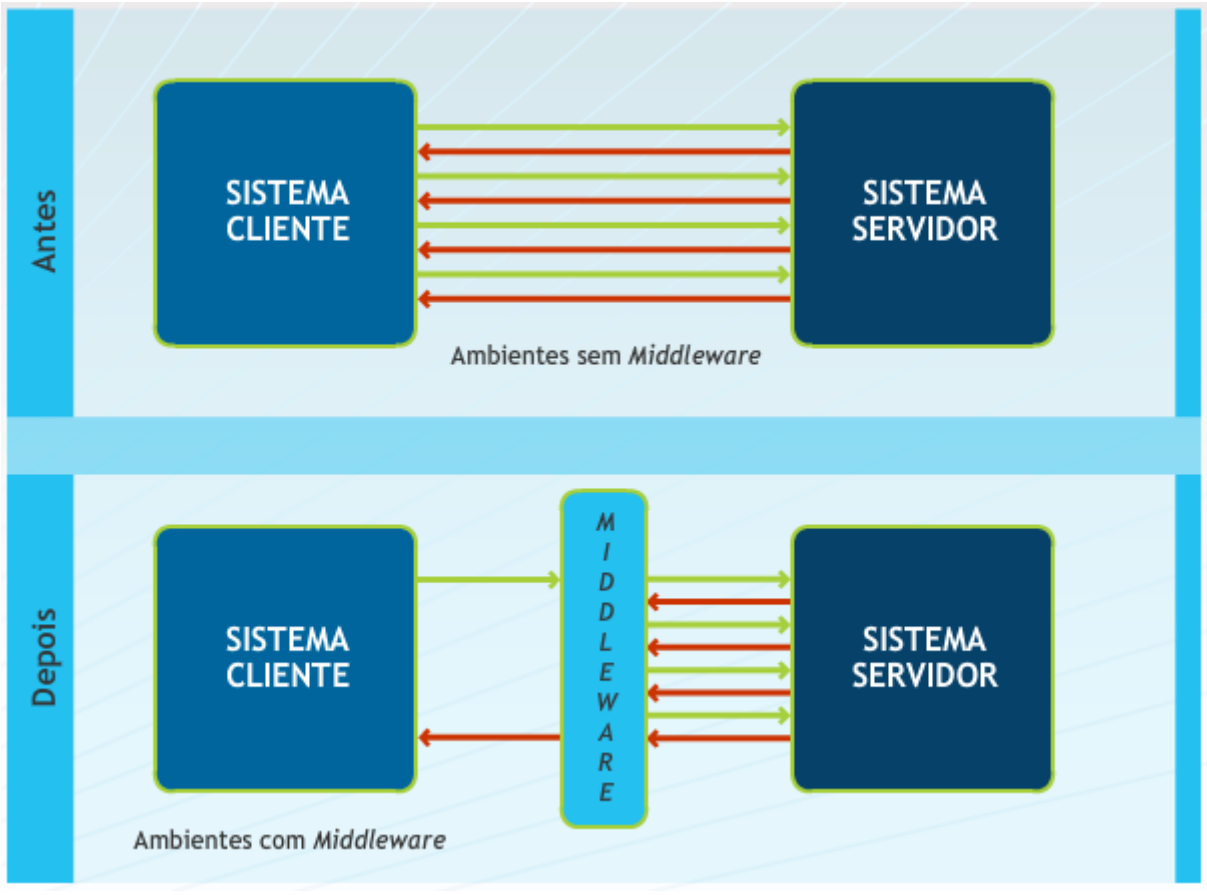
Valos recapitular:

O que é Middleware?

Meio

Middleware é um software intermediário que encapsula complexidade de determinada(s) tarefa(s).

Ou seja:



Fonte: Fonte: CAETANO, D. Middleware JDBC: Usando o Java DB. Disponível na INTERNET via WWW.url: http://www.caetano.eng.br/aulas/2012a/psw/psw_aula07.pdf Arquivo consultado em: 06/12/2012

JDBC (*Java Database Connector*) é uma interface para acesso a banco de dados através de uma API definida em Java.

Sendo assim, aplicações, *Applets, Servlets* ou quaisquer outros programas Java podem

• estabelecer conexão com base de dados;

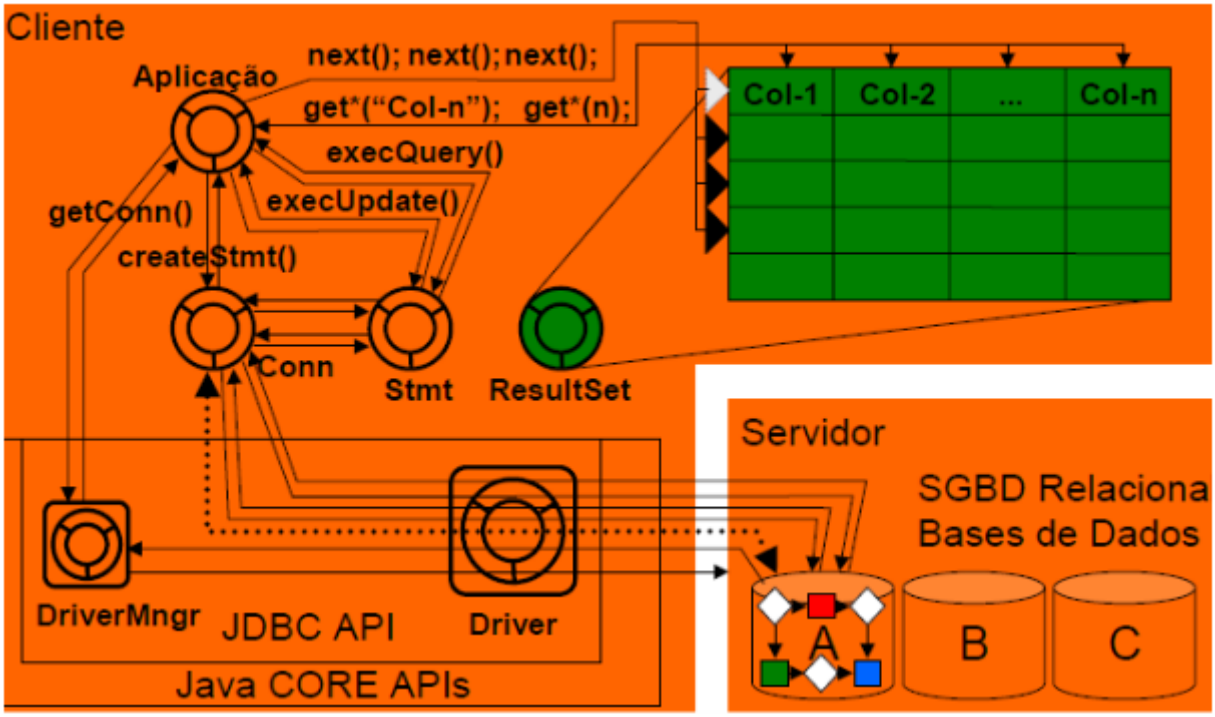
• enviar comandos SQL, e processar resultados.

Segundo CARMO, a biblioteca da JBDC provê um conjunto de interfaces de acesso ao banco de dados. Uma implementação em particular dessas interfaces é chamada de driver. Os próprios fabricantes dos bancos de dados (ou terceiros) são quem implementam os drivers JDBC para cada banco de dados.

Cada banco de dados possui um *driver* JDBC específico (que é usado de forma padrão - JDBC). A API padrão do Java já vem com o *driver* JDBC-ODBC, que é uma ponte entre a aplicação Java e o banco através da configuração de um recurso ODBC na máquina.

Drivers de outros fornecedores devem ser adicionados ao CLASSPATH da aplicação para que seja possível a sua utilização.

Sendo assim, é possível alterar o *driver* utilizado e não afetar a aplicação.



Fonte: JDBC e SQL – Modelo Computacional Fonte: FERNANDES, J.H.C. Integrando Java e Banco de Dados.

5 Tipos de Drivers JDBC

- **Tipo 1**

  Driver Ponte JDBC-ODBC: implementação nativa que conecta uma aplicação Java a um banco de dados através de ODBC configurado na máquina.

- **Tipo 2**

  Driver API-Nativa Parcialmente Java: É uma "casca" sobre uma implementação nativa de um driver de acesso ao banco de dados.

- **Tipo 3**

  Driver Java c/ Net-Protocol: Utiliza um *middleware* para a conexão com o banco de dados.

- **Tipo 4**

  Driver Java Puro: *Driver* totalmente implementado em Java.

6 Acessando dados com JDBC

As principais classes e interfaces do pacote ***java.sql*** são:

• **DriverManager** - gerencia o *driver* e cria uma conexão com o banco; • **Connection** - classe que representa uma **conexão** com ou bando de dados; • **Instrução** - controla e executa uma instrução SQL; • **PreparedStatement** - controla e executa, também, uma instrução SQL; • **ResultSet** - contém o conjunto de dados retornados por uma consulta SQL; • **ResultsetMetaData** - classe que trata dos metadados do banco.

Uma interface *Connection* possui os métodos para criar um *Statement* , fazer o *commit* ou *rollback* de uma transação, dentre outros. Como interfaces, *Statement* e *PreparedStatement* possuem métodos para executar comandos SQL. *ResultSet* possui método para recuperar os dados consolidados de uma consulta, além de retornar os metadados da consulta. *ResultsetMetaData* apresenta métodos para recuperar como meta informações do banco.

7 Usando JDBC

Primeira Etapa

Para que uma aplicação Java possa interagir com um banco de dados, uma conexão deve ser estabelecida da seguinte forma:

• Carregamento do *driver* JDBC específico;

• Criação da conexão com o banco de dados.

Para um melhor entendimento:

Carregamento do *driver*:

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

A String passada ao método forName() é o nome específico da classe que implementa o *Driver* JDBC de cada banco de dados. No exemplo acima está o nome do *driver* da ponte JDBC-ODBC.

Cada *driver* possui um nome diferente. Para utilizar, deve-se consultar a documentação do fabricante.

2. Criação da conexão:

Connection conn = DriverManager.getConnection(

"url",

"usuario",

"senha"

);

Após o carregamento do *driver*, a classe DriverManage*r* é a responsável pela conexão ao banco e pela devolução do objeto Connection, o qual representa a conexão com o banco de dados.

O parâmetro "url" também é específico de cada fornecedor de driver. Consulte a documentação do fabricante. Normalmente, na url são informados o IP ou nome do servidor, porta e o nome da base de dados.

Os outros argumentos são o nome do usuário do banco e a senha de acesso.

Segunda Etapa

Com a conexão estabelecida, pode-se interagir com o banco de dados de diversas maneiras: Inserção, alteração e retirada de registros;

• Busca de registros; • Criação de tabelas.

A seguir, seguem alguns exemplos. Todos eles estão baseados no uso do driver JDBC fornecido pela Oracle®. Entretanto, os mesmos exemplos podem ser utilizados com qualquer banco de dados relacional que possua um driver JDBC, necessitando apenas trocar o nome do driver e a URL de conexão.

Não esqueça que o JAR do driver JDBC (fornecido pelo fabricante) precisa ser disponibilizado no CLASSPATH da aplicação.

Saiba mais

Saiba mais sobre Middleware JDBC:

ANTONIO NETO. Java na Web. Rio de Janeiro: Ciência Moderna, 2011.

ORACLE CORPORATION – Tecnologia Java SE / Banco de Dados – //www.oracle.com /technetwork/java/javase/jdbc/index.html

O que vem na próxima aula

Na próxima aula, você vai estudar:

- Arquitetura MVC: conceito, tipos e padrões complementares para uma implementação da arquitetura de forma adequada.

CONCLUSÃO

Nesta aula, você:

- Compreendeu o conceito do Middleware JDBC;
- Aprendeu o mecanismo para a criação de banco de dados JavaDB com a ferramenta NetBeans®.