Sistemas Distribuídos: Conceitos e Projeto
Java Sockets

Francisco José da Silva e Silva

Grupo de Pesquisa em Sistemas Distribuídos (GSD)
Departamento de Informática / UFMA
http://www.lsd.deinf.ufma.br

18 de agosto de 2009

Introdução a Java Sockets

Introdução a Java Sockets

- Socket é o ponto final de um enlace de comunicação estabelecido entre dois programas que se comunicam em rede;
- Sua interface de programação utiliza o conceito de portas de comunicação. O socket é associado a um número de porta de forma a tornar possível à camada TCP localizar a aplicação que deve receber os dados;
- Extensão de um dos conceitos mais fortes do Unix: toda E/S deve parecer ao programador como uma E/S de arquivo;
- O pacote `java.net` disponibiliza as classes necessárias para programação com sockets em Java;
- Leitura recomendada: tutorial da Sun "All About Sockets", disponível em: http://java.sun.com/docs/books/tutorial/networking/sockets/index.html

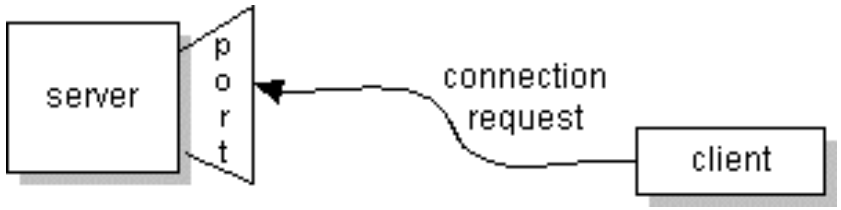
Java Sockets: Classes

- TCP → orientado a conexão
  - **java.net.Socket**
  - **java.net.ServerSocket**
- UDP → sem conexão
  - **java.net.DatagramPacket**
  - **java.net.DatagramSocket**

Java Sockets Orientado a Conexão

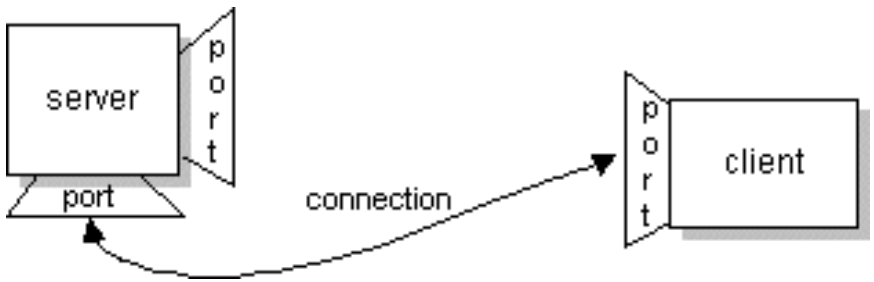
Java Sockets Orientado a Conexão

- O servidor executa em um computador específico e possui um socket associado a um número de porta;
- O servidor apenas escuta, aguardando que um cliente solicite o estabelecimento de uma conexão;
- O cliente deve conhecer em qual máquina o servidor está sendo executado e a porta no qual ele escuta.



Java Sockets Orientado a Conexão

- O servidor aceita a conexão e recebe um novo socket associado à mesma porta;
- Ele necessita do novo socket para poder continuar escutando no socket original por requisições de conexão, enquanto atende o cliente já conectado;
- Do lado cliente, se a conexão foi aceita, um socket é criado para ser utilizado na comunicação com o servidor;
- O cliente e o servidor podem agora se comunicar, escrevendo e escutando em seus respectivos sockets:



Código Cliente

O Cliente:

1. Cria o socket através do construtor da classe `Socket`;
2. Tenta estabelecer uma conexão com o servidor;
3. Uma vez estabelecida a conexão, envia e recebe fluxos de dados;
4. Quando a comunicação for concluida, fecha a conexão.

Código Servidor

O Servidor:

1. Cria um `ServerSocket`;
2. Escuta no socket através do método `accept()`;
3. Uma vez estabelecida a conexão, envia e recebe fluxos de dados;
4. Quando a comunicação for concluída, fecha a conexão.
5. Tipicamente retorna ao passo 2.

Classe Socket: Construtores

Esta classe implementa sockets cliente que utilizam conexão.
Alguns Construtores:

- **Socket(InetAddress address, int port)**: cria um socket e conecta ele a um número de porta específico em um determinado endereço IP;
- **Socket(InetAddress address, int port, InetAddress localAddr, int localPort)**: cria um socket e conecta ele a um endereço remoto e em uma porta remota específica;
- **Socket(String host, int port)**: cria um socket e conecta ele a um número de porta específico em uma máquina com um determinado nome (host);
- **Socket(String host, int port, InetAddress localAddr, int localPort)**: cria um socket e conecta ele a uma máquina remota a partir de um nome e uma porta remota específica.

Classe Socket: Métodos

- **void close()**: fecha este socket;
- **InetAddress getInetAddress()**: retorna o endereço no qual o socket está conectado;
- **InetAddress getLocalAddress()**: retorna o endereço local que o socket está ligado;
- **int getLocalPort()**: retorna a porta local que o socket está ligado;
- **int getPort()**: retorna a porta remota que o socket está conectado;
- **boolean isClosed()**: retorna verdadeiro caso o socket esteja fechado ou falso caso contráio;
- **boolean isConnected()**: retorna o estado de conexão do socket.

Fluxos de Entrada e Saída

- Os dados são enviados e recebidos através de fluxos de entrada e saída.
- Os seguintes métodos são utilizados:
  - **InputStream getInputStream()**: retorna um fluxo de entrada para este socket.
  - **OutputStream getOutputStream()**: retorna um fluxo de saída para este socket.

Classe ServerSocket

- Esta classe implementa socket servidor que utiliza conexão.
- Um socket servidor espera por requisições que venham da rede.
- Ele realiza operações baseadas na requisição e possivelmente envia os resultados ao requerente.
- Um Construtor:
  - **ServerSocket(int port)**: Cria um servidor socket, limitado a uma porta especificada.

Classe ServerSocket: Métodos

Alguns métodos:

- **Socket accept()**: fica escutando uma conexão feita por este socket e aceita ela;
- **void close()**: fecha este socket;
- **InetAddress getInetAddress()**: retorna o endereço local deste socket servidor;
- **int getLocalPort()**: retorna a porta que este socket está escutando;
- **SocketAddress getLocalSocketAddress()**: retorna o endereço ip e porta que este socket está ligado, ou null caso contrário;
- **boolean isClosed()**: retorna verdadeiro caso o socket do servidor esteja fechado ou falso caso contráio;;

Exemplo de Código

Exemplo de código: Banco de Dados de Cotações

Java Sockets Sem Conexão

Java Sockets sem Conexão

- Algumas aplicações não requerem o canal seguro de comunicação ponto-a-ponto provido pelo protocolo TCP;
- Nestes casos, a aplicação pode utilizar um modo de comunicação que entrega pacotes independentes cuja entrega e sequenciamento das mensagens não são garantidos;
- O protocolo UDP provê este serviço.

Java Sockets sem Conexão

- Classes utilizadas:
  - **DatagramPacket**: insere bytes em um pacote UDP denominado datagrama;
  - **DatagramSocket**: envia e recebe datagramas UDP;
- Para enviar dados, insere-se os mesmos em um **DatagramPacket**, enviando-o através do **DatagramSocket**;
- Para receber dados, recebe-se um `DatagramPacket` através de um `DatagramSocket`, procedendo-se em seguida a remoção dos dados a partir do pacote;
- O mesmo tipo de socket é utilizado tanto no cliente quanto no servidor;
- Trabalha com pacotes individuais e não com fluxo: os dados enviados em um datagrama são enviados em um único pacote;
- O socket não é dedicado a uma única conexão.

Classe DatagramPacket

- Pacotes datagramas são usados para implementar um serviço de entrega de pacotes sem conexão.
- Cada mesagem é roteada de uma máquina atá a outra baseada somente na informação contida dentro do pacote.
- Múltiplos pacotes enviados de uma máquina a outra poderiam ser roteados de uma forma diferente e podem chegar em qualquer ordem. A entrega de pacotes não é garantida.

Classe DatagramPacket: Construtores

- **DatagramPacket(byte[] buf, int length)**: constrói um pacote de datagrama (*DatagramPacket*) para receber pacotes com determinado tamanho (*length*).
- **DatagramPacket(byte[] buf, int length, InetAddress address, int port)**: constrói um pacote de datagrama (*DatagramPacket*) para enviar pacotes de tamanho (*length*) para uma máquina (*host*) específica em uma porta específica.
- **DatagramPacket(byte[] buf, int offset, int length)**: Cconstrói um pacote de datagrama (*DatagramPacket*) para receber pacotes com determinado tamanho (*length*), especificando um *offset* dentro do *buffer*.
- **DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)**: constrói um pacote de datagrama (*DatagramPacket*) para enviar pacotes de tamanho (*length*) com um offset, para uma máquina (*host*) específica em determinada porta.

Classe DatagramPacket: Métodos

- **InetAddress getAddress()**: retorna o endereço IP da máquina que este datagrama está sendo enviado ou da onde este datagrama foi recebido.
- **byte[] getData()**: retorna o buffer de dados.
- **int getLength()**: retorna o tamanho dos dados enviados ou tamanho de dados recebidos.
- **int getPort()**: retorna o número da porta do host que este datagrama está sendo enviado ou de onde ele foi recebido.
- **void setAddress(InetAddress iaddr)**: altera o endereço IP da máquina que este datagrama será enviado.
- **void setData(byte[] buf)**: altera o buffer de dados para este pacote. O offset do pacote é zero.
- **void setData(byte[] buf, int offset, int length)**: altera o buffer de dados para este pacote.
- **void setLength(int length)**: altera o tamanho do pacote.

Classe DatagramSocket

- Um socket datagrama é o ponto de envio ou recebimento para um serviço de entrega de pacotes.
- Cada pacote enviado ou recebido em um socket datagrama é individualmente endereçado e roteado.
- Múltiplos pacotes enviados de uma máquina para outra pode ser roteado diferentemente e pode chegar em qualquer ordem.
- Um Construtor:
  - **DatagramSocket(int port, InetAddress laddr)**
    Cria um socket datagrama, ligado a um endereço local *(laddr, port)* específico.

Classe DatagramSocket: Métodos

- **void close()**: fecha o socket deste datagrama.
- **InetAddress getInetAddress()**: retorna o endereço que este socket está conectado.
- **InetAddress getLocalAddress()**: retorna o endereço local que este socket está ligado.
- **int getLocalPort()**: retorna o número de porta da máquina local que este socket está ligado.
- **SocketAddress getLocalSocketAddress()**: retorna o endereço do *endpoint* que este socket está ligado, ou null se ele ainda não estiver ligado a um endereço.
- **int getPort()**: retorna a porta deste socket.

Classe DatagramSocket: Envio e Recebimento de pacotes

- **void receive(DatagramPacket p)**: recebe um pacote de datagrama deste socket.
- **void send(DatagramPacket p)**: envia um pacote de datagrama deste socket.

Exemplo de Código

Exemplo de código: Banco de Dados de Cotações