Como desenvolver um jogo em Java ME - Parte 7

Agora que todos os recursos para a execução no modo *single-player* do Arkanoid está completa, o jogo pode ser conectado com o resto do mundo. Além de mostrar os seus *scores* a amigos, eles podem ser compatilhados na Internet.

Uma das principais funcionalidades dos dispositivos móveis é a habilidade de ser conectado praticamente em qualquer lugar e a qualquer momento. Com Java ME, você pode facilmente acessar os seguintes recursos de comunição:

- HTTP
- Socket
- SMS
- Bluetooth

Este artigo mostra como usar conexões HTTP para enviar seu histórico de jogos para um servidor central e como usar recursos de SMS para convidar outros amigos a jogar o Arkanoid.

Conector e conexões

O ponto central por onde se obtem recursos de conectividade é a class *Conector*. Esta classe permite criar objetos *Conection* através dos seguintes métodos estáticos:

- open(String name)
- open(String name, int mode)

O parametro *name* é uma URL que identifica o tipo de conexão que você deseja estabelecer. Dependendo do protocolo na URL, um diferente tipo de objeto *Conection* é retornado. Veja a seguinte lista de URLs e os objetos *Connection* correspondentes.

- "http://www.server.com/": retorna uma conexão HTTP usando a classe *HttpConnection*.
- "socket://server.com:8080": retorna uma conexão TCP/TP.
- "btsp://2343434d3434": retorna uma conexão Bluetooth.
- "sms://+351910000000": retorna uma conexão SMS para o número +351910000000.

Dos protocolos mencionados acima, apenas o HTTP é mandatório na especificação dos dispositivos Java ME. Todos os outros protolos são dependentes de implementação. Contudo, a maioria dos novos dispositivos suportam todos os protocolos supracitados.

Note que quando você tenta abrir uma conexão de qualquer, o usuário da aplicação é notificado pelo Sistema de Gerenciamente de aplicações (*Application Management System*) que dá a opção de aceitar ou recusar a requisição por conexão. Se a solicitação é recusada, uma exceção *SecurityException* é lançada e a conexão não é estabelecida.

HTTP

Conexões HTTP estão entre os tipos de conexões mais comumente usadas. Para criar uma conexão HTTP, siga os seguintes passos:

```
import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;

public class Network {

  public byte[] httpConnection(String url, byte[] dataToSend)
    throws IOException
  {
    HttpConnection hc = null;
    // Prepare Connection
    hc = (HttpConnection) Connector.open(url, Connector.READ_WRITE);
    [...]
  }
}
```

Conexões HTTP são representadas pela classe HttpConnection. Esta classe possue três estados:

- **Setup** onde os parametros da requisição são definidos.
- **Connected** onde você pode enviar e receber dados. Você precisa enviar dados antes de receber uma resposta.
- **Closed** Após a leitura dos dados a conexão é finalizada.

Quando você recebe um objeto HttpConnection da classe *Conector*, ele está no estado *Setup*. Você pode então configurar os parametros da requisição. Uma das principais opções que pode ser definida é o método de requisição (POST, GET ou HEAD).

```
  public byte[] httpConnection(String url, byte[] dataToSend) throws IOException {
    [..]
    if (dataToSend == null){
      hc.setRequestMethod( HttpConnection.GET );
    } else {
      hc.setRequestMethod( HttpConnection.POST );
    }
    [...]
  }
```


(C) Copyright Nokia 2013. All rights reserved.

Você pode agora configurar as propriedades da requisição, como o tipo do conteúdo que você está enviando no caso de um requisição do tipo POST.

```
    hc.setRequestProperty("Content-type", "application/octet-stream" );
```

Após você configurar a conexão, você pode iniciar o envio dos dados. Quando você inicia este processo, você não pode mudar os parametros da requisição. Se a conexão foi estabelecida, você começa a receber dados.

```
  public byte[] httpConnection(String url, byte[] dataToSend) throws IOException {
    [...]
    if (dataToSend != null){
      // Write Data
      OutputStream os = hc.openOutputStream();
      os.write( dataToSend );
      os.close();
    }
    // gets answer from  server
    int rc = hc.getResponseCode();
    // check http response
    if (rc == HttpConnection.HTTP_OK){
      // Read Data
      InputStream in = hc.openInputStream();
      ByteArrayOutputStream tmp = new ByteArrayOutputStream();
      int ch;
      while ((ch = in.read()) != -1) {
          tmp.write(ch);
      }
      data = tmp.toByteArray();
      in.close();
      hc.close();
    }

    return data;
  }
```

O método `httpConnection()` está agora completo, e pode ser usado no jogo para enviar as informações de *scores* para o servidor. Inicie adicionando uma nova opção à tela de *scores*:

```
public Displayable initScoreForm() {
  [...]
  cmdSendHighScore = new Command("Send to Server", Command.ITEM, 1);
  highScoreForm.addCommand(cmdSendHighScore);
  [...]
}
```

Agora, crie um método para usa-lo quando este *Command* for selecionado:

```
  public String sendScore(String user, int score) {
    String result = "No Answer";
    // server to send data
    String url = "http://www.sergioestevao.com/midpAdventures/post.php";
    // prepare http request
    String urlTotal = url + "?user=" + user + "&score=" + score;
    byte[] data = null;
    try {
      data = network.httpConnection(urlTotal, null);
    } catch (IOException e) {
      result = "Communication Problems";
      e.printStackTrace();
    } catch (SecurityException s) {
      // user denied access to communication
      result = "You need to allow communications in order to send the highscore to server.";
      s.printStackTrace();
    }
    // check data return.
    if (data != null) {
      result = new String(data);
    }
    return result;
  }
```

E chama quando o *Command comInviteSend* for usado:

```
   if (cmd == comInviteSend) {
        result = sendScore(scores[0].name, scores[0].value);
        display(new Alert("Result", result, null, AlertType.INFO));
   }
```

Se você testar este código, o seguinte alerta será exibido:

Warning: To avoid potential deadlock, operations that may block, such as networking, should be performed in a different thread than the commandAction() handler.

Lembre-se que você está executando este código de acesso a rede como uma resposta para o usuário que está executando uma thread interface gŕafica. Se este código bloquear ou levar um longo tempo para ser executada, a aplicação ficará travada.

Para evitar que tal problema ocorra, uma uma thread para executar as tarefas relacionadas a conectividade. Inicie declarando uma thread e algumas variáveis de estado para controlar as ações da thread.

```
  private static final int ACTION_SEND_HIGHSCORE = 0;
  public Thread workThread;
  public boolean active = false;
```


(C) Copyright Nokia 2013. All rights reserved.

Sempre que há um evento que consome tempo, um método que prepara uma thread e inicia a ação é chamado.

```
  public void doAction(int action) {
    // stores action to do
    this.action = action;
    // check if thread is already created
    if (workThread == null) {
      workThread = new Thread(this);
      workThread.start();
      active = true;
    }
    // wakes up thread to work
    synchronized (this) {
      notify();
    }
  }
```

A nova thread executa o método *run* e executa as tarefas que precisam ser feitas.

```
public void run() {
    while (active) {
      // check what action to do
      switch (action) {
        case (ACTION_SEND_HIGHSCORE):
          // send the first score to the server
          result = sendScore(scores[0].name, scores[0].value);
          commandAction(cmdReceiveHighScore, highScoreForm);
        break;
      }
      // waits for action to do.
      synchronized (this) {
        try {
          wait();
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    }
  }
```

Declare a interface *Runnable* na *MIDlet*:

```
  public class MyMidlet extends MIDlet implements CommandListener, Runnable {
```

Agora invoque o método `doAction()` quando o usuário selecionar a opção 'Send high score'. Note que o método `run()` invoca o comando chamado `cmdReceiveHighScore`.. Este é usado para exibir os resultados da comunicação ao usuário.

```
      if (cmd == cmdSendHighScore) {
        doAction(ACTION_SEND_HIGHSCORE);
      }
      if (cmd == cmdReceiveHighScore) {
        display(new Alert("Result", result, null, AlertType.INFO));
      }
```

SMS

Uma outra forma de comunicação implementada em Java ME é convidar os amigos a a jogar. Você pode também definir uma competição pelos mais altos *scores* em uma tabela.

Primeiro, implemente o método que envia SMS numa classe de rede.

```
  public boolean sendSms(String number, String message){
    boolean result = true;
    try {
      //sets address to send message
      String addr = "sms://"+number;
      // opens connection
      MessageConnection conn = (MessageConnection) Connector.open(addr);
      // prepares text message
      TextMessage msg =
      (TextMessage)conn.newMessage(MessageConnection.TEXT_MESSAGE);
      //set text
      msg.setPayloadText(message);
      // send message
      conn.send(msg);
      } catch (Exception e) {
        result = false;
      }
      return result;
  }
```

Como você pode ver, você precisa apenas invocar o método `open()` na classe *Connector* com uma URL no formato "sms://number". Então crie um objeto *TextMessage* e envie pela rede. O tamanho da mensagem é limitada em torno de 160 caracteres.

Agora, crie um formulário para receber o nome e o número do telefone do usuário.

```
  public Displayable initInviteForm() {
    if (inviteForm == null) {
      inviteForm = new Form("Invite");
      inviteForm.setCommandListener(this);
      inviteForm.addCommand(initBackCommand());

      inviteName = new TextField("Name:", "", 20, TextField.ANY);
```


(C) Copyright Nokia 2013. All rights reserved.

```
      inviteNumber = new TextField("Number:", "", 20, TextField.PHONENUMBER);
      inviteForm.append(inviteName);
      inviteForm.append(inviteNumber);
      comInviteSend = new Command("Invite", Command.ITEM, 1);
      inviteForm.addCommand(comInviteSend);
    }

    return inviteForm;
  }
```

Implemente a ação do comando para mostrar o fomulário e enviar a mensagem usando a thread de envio.

```
public void commandAction(Command cmd, Displayable display) {
   [...]
   else if (display == inviteForm) {
      if (cmd == comBack) {
        display(initMainForm());
      }
      if (cmd == comInviteSend) {
        doAction(ACTION_SEND_INVITE);
      }
    }
  [...]
}
```

Agora, implemente a nova ação no método *run*:

```
public void run() {
  [...]
  case (ACTION_SEND_INVITE):
    // invite another player to play
    String inviteMessage = " invites you to play Transnoid!";
    network.sendSms(inviteNumber.getString(), inviteName.getString()
      + inviteMessage);
    break;
  }
  [..]
}
```

Agora execute o jogo e comece a enviar convites aos seus amigos. **Downloads:**

- Full Source Code
- Jad File
- Jar File

Este artigo faz parte da série *Como desenvolver um jogo em Java ME*. Para ver os outros artigos da série, consulte:

Como desenvolver um jogo em Java ME - Parte 1

Como desenvolver um jogo em Java ME - Parte 2

Como desenvolver um jogo em Java ME - Parte 3

Como desenvolver um jogo em Java ME - Parte 4

Como desenvolver um jogo em Java ME - Parte 5

Como desenvolver um jogo em Java ME - Parte 6

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.