

Introdução ao Java

Tópicos seleccionados

Programação Distribuída / José Marinho

1

Linguagem Java

- Linguagem de programação baseada no C++
- Pensada de raiz para ambientes distribuídos (WWW, Internet, etc.)
- Principais características:
 - Orientada a objectos
 - Simplicidade (e.g., não existem ponteiros, nem heranças múltiplas)
 - *Garbage collection* automático
 - Transportabilidade
 - Programação concorrente baseada em múltiplas *threads*
 - Segurança (através da abordagem designada por *sandbox*)

2

Programação Distribuída / José Marinho

2

Linguagem Java

- Suporte nativo de programação para redes de dados
 - Endereços IP
 - Datagramas UDP
 - Fluxos TCP
 - Pedidos HTTP
 - Comunicação de grupo
 - ...
- Permite desenvolver aplicações que correm em Web browsers (i.e., *applets*)
- Permite desenvolver aplicações que correm em servidores Web (i.e., *servlets*, *serviços Web*)

3

Programação Distribuída / José Marinho

3

Plataforma Java

- A compilação de código fonte Java resulta em *byte code* Java
- O *byte code* é executado pela designada Máquina Virtual Java
- O objectivo é: *Write Once, Run Anywhere*
- Inconveniente principal da emulação: desempenho inferior ao código máquina

4

Programação Distribuída / José Marinho

4

Java Application Programming Interface

- Conjunto de classes e componentes que permitem a realização de tarefas específicas
 - Aceder a ficheiros
 - Criar interfaces gráficas
 - Interagir com bases de dados
 - Manipular e processar dados
 - Armazenar e manipular dados em memória
 - Aceder a recurso de rede
 - ...

5

Programação Distribuída / José Marinho

5

Java Application Programming Interface

- Existem várias *packages* relacionadas com o acesso a redes de dados
 - java.net (classes para acesso a redes TCP/IP)
 - java.rmi.* (conjunto de *packages* para invocação remota de objectos)
 - org.omg.* (conjunto de *packages* para suporte do *Common Object Request Broker* - CORBA)
 - ...
- Também existem extensões
 - Javax.mail
 - Javax.json
 - ...

6

Programação Distribuída / José Marinho

6

Programação para Redes em Java

- Lado cliente de protocolos/aplicações de rede (e.g., aceder a servidores de e-mail ou de transferência de ficheiros)
- Jogos em rede
- Aplicações para a Web
- Sistemas distribuídos
 - Computação paralela em *clusters* de computadores
 - Acesso integrado a recursos de *hardware* e *software*, possivelmente heterogéneos, distribuídos por várias máquinas (e.g., bases de dados, sistemas secundários de armazenamento de dados, etc.)
 - ...

7

Programação Distribuída / José Marinho

7

Excepções

- Um mecanismo que força o tratamento de erros e de outras situações inesperadas que podem surgir em *runtime*
- Clarifica o código ao separar o essencial do tratamento de situações excepcionais (e.g., erros)
- Uma excepção é encapsulada por um objecto
- Existem vários tipos de excepções
- Todas derivam da mesma classe de base: *java.lang.Throwable*
- Os programadores podem criar novas excepções a partir das existentes (e.g., a classe *java.lang.Exception*)

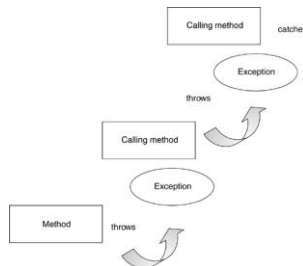
8

Programação Distribuída / José Marinho

8

Excepções

- Uma exceção é passada pelo método onde ocorre o erro/situação reportada ao método que o invocou
- Por sua vez, o método acima pode tratar a exceção ou lançá-la para o nível mais acima e assim sucessivamente
- As exceções não podem ser ignoradas pelos programadores



Java™ Network Programming and Distributed Computing
By David Reilly, Michael Reilly
Publisher : Addison Wesley
ISBN: 0-201-71037-4

9

Programação Distribuída / José Marinho

9

Excepções

```
try {  
    // A block of code where exceptions can be generated  
    //...  
} catch (SocketException e) {  
    System.err.println ("Socket error reading from host : " + e);  
    System.exit(2);  
}  
  
} catch (Exception e) {  
    System.err.println ("Error : " + e);  
    System.exit(1);  
}  
  
} finally {  
    // clean up after try block, regardless of any  
    // exceptions that are thrown  
    //...  
}
```

10

Programação Distribuída / José Marinho

10

Entrada/Saída

- *Package: java.io*
- Fluxos de bytes
 - Classes abstractas de base: ***InputStream*** e ***OutputStream***
 - Os métodos geram excepções do tipo *IOException*
 - Classes derivadas (entrada)
 - ***ByteArrayInputStream*** – permite ler *bytes* a partir de um *array* em memória
 - ***FileInputStream*** – permite ler *bytes* a partir de um ficheiro
 - ***StringBufferInputStream*** – permite ler *bytes* a partir de uma *string*
 - ***System.in*** – permite ler *bytes* a partir da “entrada standard”
 - ...

11

Programação Distribuída / José Marinho

11

Entrada/Saída

- Classes derivadas (saída)
 - ***ByteArrayOutputStream*** – permite escrever *bytes* num *array* de bytes
 - ***FileOutputStream*** – permite escrever *bytes* num ficheiro
 - ***StringBufferOutputStream*** – permite escrever *bytes* numa *StringBuffer* (i.e., semelhante a uma *String*, mas pode ser modificado)
 - ***System.out*** – permite escrever *bytes* na “saída standard”
 - ***System.err*** – permite escrever *bytes* para saída standard de erro
 - ...

12

Programação Distribuída / José Marinho

12

Entrada/Saída

```
FileInputStream in = null;
FileOutputStream out = null;

try {
    in = new FileInputStream("xanadu.txt");
    out = new FileOutputStream("outagain.txt");
    int c;

    while ((c = in.read()) != -1) {
        out.write(c);
    }
} finally {
    if (in != null) {
        in.close();
    }

    if (out != null) {
        out.close();
    }
}
```

13

Programação Distribuída / José Marinho

13

Entrada/Saída

- Fluxos de caracteres
 - Classes de base: **Reader** e **Writer**
 - Conversão automática de formatos (*bytes* vs. *caracteres*)
 - Classes derivadas (entrada)
 - **CharArrayReader** – permite ler **caracteres** a partir de um *array* de caracteres
 - **FileReader** – permite ler **caracteres** a partir de um ficheiro
 - **StringReader** – permite ler **caracteres** a partir de uma *string*
 - Classes derivadas (saída)
 - **CharArrayWriter** – permite escrever **caracteres** num *array* de caracteres
 - **FileWriter** – permite escrever **caracteres** num ficheiro
 - **StringWriter** – permite escrever **caracteres** numa *StringBuffer*

14

Programação Distribuída / José Marinho

14

Entrada/Saída

```
public static void main(String[] args) throws IOException {  
  
    FileReader in = null;  
    FileWriter out = null;  
    int c;  
  
    try {  
        in = new FileReader("xanadu.txt");  
        out = new FileWriter("characteroutput.txt");  
  
        while ((c = in.read()) != -1) {  
            out.write(c);  
        }  
  
    } finally {  
        if (in != null) {  
            in.close();  
        }  
        if (out != null) {  
            out.close();  
        }  
    }  
}
```

15

Programação Distribuída / José Marinho

15

Entrada/Saída

- Linhas de texto
 - Fluxos de caracteres terminados em "\r\n", "\r" ou "\n"
 - Classes de base: *BufferedReader* e *PrintWriter*

```
public static void main(String[] args) throws IOException {  
  
    BufferedReader in = null;  
    PrintWriter out = null;  
    String l;  
  
    try {  
        in = new BufferedReader(new FileReader("xanadu.txt"));  
        out = new PrintWriter(new FileWriter("characteroutput.txt"));  
  
        while ((l = in.readLine()) != null) {  
            out.println(l);  
        }  
  
    } finally {  
        if (in != null) in.close();  
        if (out != null) out.close();  
    }  
}
```

16

Programação Distribuída / José Marinho

16

Entrada/Saída

- classe *Scanner*
 - Extrair palavras de um fluxo de entrada ou de uma *string*

```
import java.io.*;
import java.util.Scanner;

public class ScanXan {
    public static void main(String[] args) throws IOException {
        Scanner s = null;
        try {
            s = new Scanner(new BufferedReader(new FileReader("xanadu.txt")));

            while (s.hasNext()) {
                System.out.println(s.next());
            }
        } finally {
            if (s != null) {
                s.close();
            }
        }
    }
}
```

```
while ((l = in.readLine()) != null){
    s = new Scanner(l);
    while (s.hasNext()) {
        System.out.println(s.next());
    }
}
```

```
while (s.hasNext()) {
    if (s.hasNextDouble()) {
        sum += s.nextDouble();
    } else { s.next(); }
}
```

17

Programação Distribuída / José Marinho

17

Entrada/Saída

- Fluxos de bytes standards
 - *System.in* (*InputStream*), *System.out* (*PrintStream*) e *System.err* (*PrintStream*)
 - Definidos automaticamente
 - Exemplo: `InputStreamReader in = new InputStreamReader(System.in);`
- Consola

```
Console c = System.console();
if (c == null) {
    System.err.println("No console.");
    System.exit(1);
}

String login = c.readLine("Enter your login: ");
char[] oldPassword = c.readPassword("Enter your old password: ");
```

18

Programação Distribuída / José Marinho

18

Entrada/Saída

- Fluxos de dados
 - Todas as classes deste tipo implementam as interfaces *DataInput* ou *DataOutput*

```
double price; int unit; double total = 0.0;
DataInputStream in = new DataInputStream(new BufferedInputStream(
    new FileInputStream("dados.txt")));

try {
    while (true) {
        price = in.readDouble();
        unit = in.readInt();
        System.out.format("You ordered %d units at $%.2f\n",
            unit, price);
        total += unit * price;
    }
} catch (EOFException e) {}

DataOutputStream out = new DataOutputStream(new
    BufferedOutputStream(new FileOutputStream("dados.txt")));

out.writeDouble(10.3);
out.writeInt(34);
```

19

Programação Distribuída / José Marinho

19

Entrada/Saída

- Fluxos/SerIALIZAÇÃO de objectos
 - Associado ao conceito de **objecto persistente**
 - Evita o processo moroso que resulta em lidar directamente com os atributos
 - Os objectos (e *sub-objectos*) que se pretende serializar devem implementar a interface *java.io.Serializable*
 - “*protected transient String password;*” → A inclusão do atributo *password* no objecto serializado é evitada
 - Classes: *ObjectInputStream* e *ObjectOutputStream*
 - Implementam as interfaces *ObjectInput* e *ObjectOutput* (subinterfaces de *DataInput* e *DataOutput*, respectivamente)

20

Programação Distribuída / José Marinho

20

Entrada/Saída

```
public static void main(String[] args) throws IOException, ClassNotFoundException {
    ObjectOutputStream out = null;
    try {
        out = new ObjectOutputStream(new FileOutputStream("dates.bin"));
        out.writeObject(Calendar.getInstance());
    } finally {
        if(out != null) out.close();
    }
    // ...
    ObjectInputStream in = null;
    int nDates = 0; Calendar date = null;
    try {
        in = new ObjectInputStream(new FileInputStream("dates.bin"));
        try {
            while(true) {
                date = null;
                date = (Calendar) in.readObject();
                nDates++;
            }
        } catch (EOFException e) {}
    } finally {
        if (in != null) in.close();
    }
    // ...
}
```

21

Programação Distribuída / José Marinho

21

Bibliografia

- REILLY, David; REILLY, Michael - *Java Network Programming & Distributed Computing* - Addison-Wesley
- <http://download.oracle.com/javase/tutorial/essential/>

22

Programação Distribuída / José Marinho

22