Distributed Computing

Lesson 6: Data Types and Marshalling

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://www.it-weise.de

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

1. Data Types and Marshalling

website

Overview

- What to consider when exchanging data between hosts in a heterogeneous system?

Introduction

- Heterogeneous distributed system can consist of computers that...

Introduction

- Heterogeneous distributed system can consist of computers that...
  - run different operating systems (Linux, Windows, ...)

Introduction

- Heterogeneous distributed system can consist of computers that...
  - run different operating systems (Linux, Windows, ...)
  - are composed of different hardware (80x86, Alpha, Atmel, Motorola, Power PC, ...)

Introduction

- Heterogeneous distributed system can consist of computers that...
  - run different operating systems (Linux, Windows, ...)
  - are composed of different hardware (80x86, Alpha, Atmel, Motorola, Power PC, ...)
  - run communicating software created with different programming languages

Introduction

- Heterogeneous distributed system can consist of computers that...
  - run different operating systems (Linux, Windows, ...)
  - are composed of different hardware (80x86, Alpha, Atmel, Motorola, Power PC, ...)
  - run communicating software created with different programming languages
- Each of these aspects may influence the way in which data is represented

Introduction

- Heterogeneous distributed system can consist of computers that...
  - run different operating systems (Linux, Windows, ...)
  - are composed of different hardware (80x86, Alpha, Atmel, Motorola, Power PC, ...)
  - run communicating software created with different programming languages
- Each of these aspects may influence the way in which data is represented
- The bit-representation of a `double` on one computer may make no sense at all at another one

Introduction

- Heterogeneous distributed system can consist of computers that...
  - run different operating systems (Linux, Windows, ...)
  - are composed of different hardware (80x86, Alpha, Atmel, Motorola, Power PC, ...)
  - run communicating software created with different programming languages
- Each of these aspects may influence the way in which data is represented
- The bit-representation of a `double` on one computer may make no sense at all at another one
- Marshalling [1]: Transforming data into a representation that can be sent to a different computer/process

- Heterogeneous distributed system can consist of computers that...
  - run different operating systems (Linux, Windows, ...)
  - are composed of different hardware (80x86, Alpha, Atmel, Motorola, Power PC, ...)
  - run communicating software created with different programming languages
- Each of these aspects may influence the way in which data is represented
- The bit-representation of a `double` on one computer may make no sense at all at another one
- Marshalling [1]: Transforming data into a representation that can be sent to a different computer/process
- Unmarshalling: Transforming received marshalled data into the internal representation used in a given computer/process

Endianness

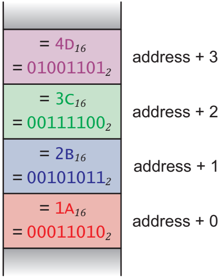
- Byte order in which values are stored in memory that require more than 1 byte

Endianness

- Byte order in which values are stored in memory that require more than 1 byte
- Big Endian

- Byte order in which values are stored in memory that require more than 1 byte
- Big Endian:
  - Store the byte with the most significant bit at lowest address

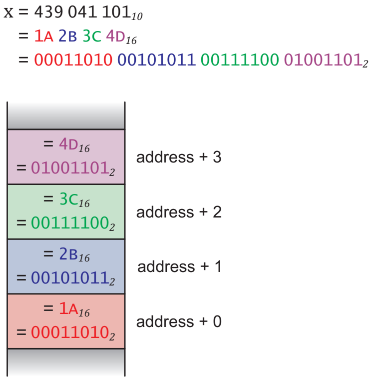
$$
\begin{aligned}
x &= 439\,041\,101_{10}\\
&= 1A\ 2B\ 3C\ 4D_{16}\\
&= 00011010\ 00101011\ 00111100\ 01001101_{2}
\end{aligned}
$$



Big Endian

Endianness

- Byte order in which values are stored in memory that require more than 1 byte
- Big Endian:
  - Store the byte with the most significant bit at lowest address
  - e.g., MIPS, SPARC, PowerPC, Motorola 6800/68k, Atmel AVR32, and TMS9900 processors

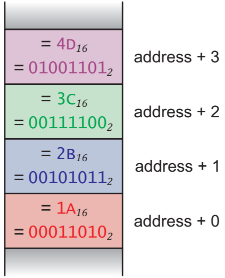


Big Endian

Endianness

- Byte order in which values are stored in memory that require more than 1 byte
- Big Endian:
  - Store the byte with the most significant bit at lowest address
- Little Endian

$$x = 439\,041\,101_{10}$$
$$= 1A\ 2B\ 3C\ 4D_{16}$$
$$= 00011010\ 00101011\ 00111100\ 01001101_{2}$$

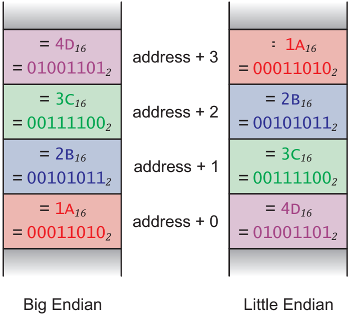


Big Endian

Endianness

- Byte order in which values are stored in memory that require more than 1 byte
- Big Endian:
  - Store the byte with the most significant bit at lowest address
- Little Endian:
  - Store the byte with the least significant bit the lowest address

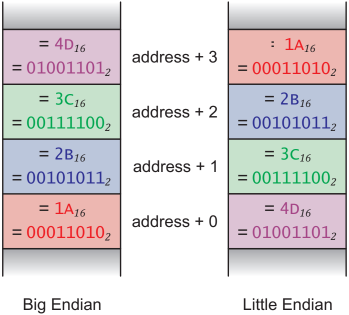
$x = 439\,041\,101_{10}$
$= 1A\ 2B\ 3C\ 4D_{16}$
$= 00011010\ 00101011\ 00111100\ 01001101_{2}$



Endianness

- Byte order in which values are stored in memory that require more than 1 byte
- Big Endian:
  - Store the byte with the most significant bit at lowest address
- Little Endian:
  - Store the byte with the least significant bit the lowest address
  - e.g., 80x86-compatible, Alpha, Altera Nios, Atmel AVR, some SH3/SH4-Systems, and VAX

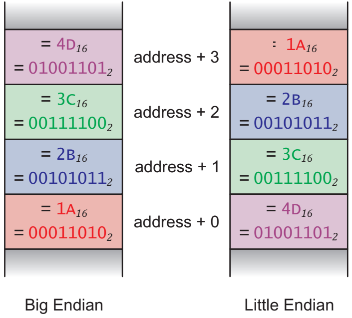
$$x = 439\,041\,101_{10}$$
$$= 1A\ 2B\ 3C\ 4D_{16}$$
$$= 00011010\ 00101011\ 00111100\ 01001101_{2}$$



Endianness

- Byte order in which values are stored in memory that require more than 1 byte
- Big Endian:
  - Store the byte with the most significant bit at lowest address
- Little Endian:
  - Store the byte with the least significant bit the lowest address
- Sending an `int` directly from a Little-Endian machine to a Big-Endian one will not work

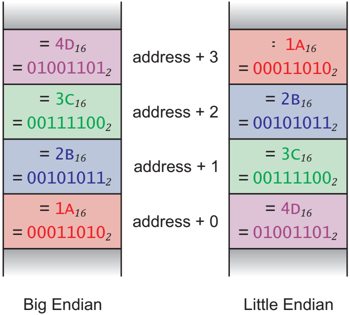
$x = 439\,041\,101_{10}$
$= 1A\ 2B\ 3C\ 4D_{16}$
$= 00011010\ 00101011\ 00111100\ 01001101_{2}$



Endianness

- Byte order in which values are stored in memory that require more than 1 byte
- Big Endian:
  - Store the byte with the most significant bit at lowest address
- Little Endian:
  - Store the byte with the least significant bit the lowest address
- Sending an `int` directly from a Little-Endian machine to a Big-Endian one will not work
- It is usually unknown what endianness a communication partner has

$$
\begin{aligned}
x &= 439\,041\,101_{10}\\
&= \text{1A 2B 3C 4D}_{16}\\
&= 00011010\;00101011\;00111100\;01001101_{2}
\end{aligned}
$$



Endianness

- So each host has a *host byte order*

Endianness

- So each host has a *host byte order*
- In order to ensure proper data exchange, for each protocol, a so-called *network byte order* is defined

Endianness

- So each host has a *host byte order*
- In order to ensure proper data exchange, for each protocol, a so-called *network byte order* is defined
- Before sending data, each computer translates it from host byte order to network byte order

Endianness

- So each host has a *host byte order*
- In order to ensure proper data exchange, for each protocol, a so-called *network byte order* is defined
- Before sending data, each computer translates it from host byte order to network byte order
- Upon arrival, data is translated from network byte order to host byte order

Endianness

- So each host has a *host byte order*
- In order to ensure proper data exchange, for each protocol, a so-called *network byte order* is defined
- Before sending data, each computer translates it from host byte order to network byte order
- Upon arrival, data is translated from network byte order to host byte order
- Internet protocol stack: network byte order = Big Endian

- In C, a set of translation functions is provided

Endianness in C

- In C, a set of translation functions is provided
- These functions translate between host byte order and Big Endian

- In C, a set of translation functions is provided
- These functions translate between host byte order and Big Endian

Function	Datatype	Description

- In C, a set of translation functions is provided
- These functions translate between host byte order and Big Endian

Function	Datatype	Description
htonl()	long (32bit)	host-to-network translation

Endianness in C

- In C, a set of translation functions is provided
- These functions translate between host byte order and Big Endian

Function	Datatype	Description
htonl()	long (32bit)	host-to-network translation
htons()	short (16bit)	host-to-network translation

- In C, a set of translation functions is provided
- These functions translate between host byte order and Big Endian

`Function`	Datatype	Description
`htonl()`	`long` (32bit)	host-to-network translation
`htons()`	`short` (16bit)	host-to-network translation
`ntohl()`	`long` (32bit)	network-to-host translation

- In C, a set of translation functions is provided
- These functions translate between host byte order and Big Endian

Function	Datatype	Description
htonl()	long (32bit)	host-to-network translation
htons()	short (16bit)	host-to-network translation
ntohl()	long (32bit)	network-to-host translation
ntohs()	short (16bit)	network-to-host translation

- In C, a set of translation functions is provided
- These functions translate between host byte order and Big Endian
- Not for 64bit `int`s, as such long integers were not available when API was designed [2]

`Function`	Datatype	Description
`htonl()`	`long` (32bit)	host-to-network translation
`htons()`	`short` (16bit)	host-to-network translation
`ntohl()`	`long` (32bit)	network-to-host translation
`ntohs()`	`short` (16bit)	network-to-host translation

Endianness in Java

- In Java, we can use the more general Stream API [3] to deal with data conversation

Endianness in Java

- In Java, we can use the more general Stream API [3] to deal with data conversation
- Input

Endianness in Java

- In Java, we can use the more general Stream API [3] to deal with data conversation
- Input:
  - `InputStream`s read one or multiple bytes

- In Java, we can use the more general Stream API [3] to deal with data conversation
- Input:
  - `InputStream`s read one or multiple bytes
  - `DataInputStream`s read structured data from an input stream

Endianness in Java

- In Java, we can use the more general Stream API [3] to deal with data conversation
- Input:
  - `InputStream`s read one or multiple bytes
  - `DataInputStream`s read structured data (`int`, `long`, `double`, ...) from an input stream, assuming network byte order
- Output

- In Java, we can use the more general Stream API [3] to deal with data conversation
- Input:
  - `InputStream`s read one or multiple bytes
  - `DataInputStream`s read structured data (`int`, `long`, `double`, ...) from an input stream, assuming network byte order
- Output:
  - `OutputStream`s write one or multiple bytes

Endianness in Java

- In Java, we can use the more general Stream API [3] to deal with data conversation
- Input:
  - `InputStream`s read one or multiple bytes
  - `DataInputStream`s read structured data (`int`, `long`, `double`, ...) from an input stream, assuming network byte order
- Output:
  - `OutputStream`s write one or multiple bytes
  - `DataOutputStream`s write structured data (`int`, `long`, `double`, ...) to an input stream in network byte order

Endianness in Java

- In Java, we can use the more general Stream API [3] to deal with data conversation
- Input:
  - `InputStream`s read one or multiple bytes
  - `DataInputStream`s read structured data (`int`, `long`, `double`, ...) from an input stream, assuming network byte order
- Output:
  - `OutputStream`s write one or multiple bytes
  - `DataOutputStream`s write structured data (`int`, `long`, `double`, ...) to an input stream in network byte order
- TCP sockets: plug the `DataInputStream` and `DataOutputStream`s directly into the streams that the socket offers to us

Endianness in Java

- In Java, we can use the more general Stream API [3] to deal with data conversation
- Input:
  - `InputStream`s read one or multiple bytes
  - `DataInputStream`s read structured data (`int`, `long`, `double`, ...) from an input stream, assuming network byte order
- Output:
  - `OutputStream`s write one or multiple bytes
  - `DataOutputStream`s write structured data (`int`, `long`, `double`, ...) to an input stream in network byte order
- TCP sockets: plug the `DataInputStream` and `DataOutputStream`s directly into the streams that the socket offers to us
- UDP sockets: create the packets in memory

- In Java, we can use the more general Stream API [3] to deal with data conversation
- Input:
  - `InputStream`s read one or multiple bytes
  - `DataInputStream`s read structured data from an input stream
- Output:
  - `OutputStream`s write one or multiple bytes
  - `DataOutputStream`s write structured data to an input stream
- TCP sockets: plug the `DataInputStream` and `DataOutputStream`s directly into the streams that the socket offers to us
- UDP sockets: create the packets in memory:
  - `ByteArrayOutputStream`s are output streams which store all data written to them as `byte`

Endianness in Java

- In Java, we can use the more general Stream API [3] to deal with data conversation
- Input:
  - `InputStream`s read one or multiple bytes
  - `DataInputStream`s read structured data from an input stream
- Output:
  - `OutputStream`s write one or multiple bytes
  - `DataOutputStream`s write structured data to an input stream
- TCP sockets: plug the `DataInputStream` and `DataOutputStream`s directly into the streams that the socket offers to us
- UDP sockets: create the packets in memory:
  - `ByteArrayOutputStream`s are output streams which store all data written to them as `byte`
  - `ByteArrayInputStream`s are input streams which take their data from an array of bytes

Listing: TCPServerStructuredData.java Structured Data: TCP Server / Java

```java
import java.io.DataInputStream;   import java.io.DataOutputStream;
import java.net.ServerSocket;     import java.net.Socket;

public class TCPServerStructuredData {
  public static final void main(final String[] args) {
    ServerSocket      server;       Socket            client;
    DataOutputStream  dos;          DataInputStream   dis;
    String            s;            long              a, b,r;

    try {
      server = new ServerSocket(9996);//1 + 2)

      for (int j = 5; (--j) >= 0;) { //process only 5 clients, so I can show 5) below
        client = server.accept();    //3)

        dis = new DataInputStream(client.getInputStream()); //4 + 3
        s   = dis.readUTF();       //read an UTF-encoded string: the operation
        r   = a = dis.readLong(); //read a 64 bit long integer
        b   = dis.readLong();     //read another 64 bit long int
        if ("add".equalsIgnoreCase(s)) { r += b; } else { // add
          if ("sub".equalsIgnoreCase(s)) { r -= b;  }     // subtract
        }  //4 + 3)

        System.out.println(s + "(" + a + ", " + b + ") = " + r + " to " + client.getRemoteSocketAddress());

        dos = new DataOutputStream(client.getOutputStream()); //marshall output
        dos.writeLong(r); //write 64bit long integer: 4 + 3)
        dos.close(); // flush and close

        client.close(); //4)
      }
      server.close(); //5)
    } catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

Structured Data: TCP Client / Java

Listing: TCPClientStructuredData.java Structured Data: TCP Client / Java

```
import java.io.DataInputStream;    import java.io.DataOutputStream;
import java.net.InetAddress;       import java.net.Socket;

public class TCPClientStructuredData {

  public static final void main(final String[] args) {
    Socket            client;     InetAddress      ia;
    DataOutputStream  dos;        DataInputStream  dis;

    try {
      ia = InetAddress.getByName("localhost");

      client = new Socket(ia, 9996); //1+2)

      dos = new DataOutputStream(client.getOutputStream()); //marshall data
      dos.writeUTF("sub"); //send operation name 3)
      dos.writeLong(9876); //send 64bit long integer
      dos.writeLong(1234); //send another 64bit long integer
      dos.flush();  //flush is important, otherwise stuff may just be buffered!

      dis = new DataInputStream(client.getInputStream()); // unmashall input
      System.out.println("Result: " + dis.readLong());    //3)

      client.close(); //4)
    } catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

Structured Data: UDP Server / Java

Listing: UDPServerStructuredData.java Structured Data: UDP Server / Java

```
import java.io.ByteArrayInputStream;  import java.io.DataInputStream;  import java.io.OutputStream; import
    java.net.DatagramPacket;
import java.io.ByteArrayOutputStream; import java.io.DataOutputStream; import java.net.InetAddress; import
    java.net.DatagramSocket;

public class UDPServerStructuredData {
  public static final void main(final String[] args) {
    DatagramSocket         server;  DatagramPacket    p, answer;
    ByteArrayOutputStream bos;      DataOutputStream  dos;
    ByteArrayInputStream  bis;      DataInputStream   dis;
    byte[]                data;     String            s;
    long                  a, b,r;

    try {
      server = new DatagramSocket(9997); //1)
      data = new byte[2048];

      for (int j = 5; (--j) >= 0;) {
        p = new DatagramPacket(data, data.length); // create package
        server.receive(p);  // receive data 2)

        bis = new ByteArrayInputStream(data, 0, p.getLength());  //wrap in stream 3)
        dis = new DataInputStream(bis); //wrap again for unmarshalling
        s   = dis.readUTF();      //read string with operation id
        r   = a = dis.readLong(); //read 64bit long integer
        b   = dis.readLong();     //read 64bit long integer
        if ("add".equalsIgnoreCase(s)) { r += b; } else { //add
          if ("sub".equalsIgnoreCase(s)) { r -= b;  }     //subtract
        }  //end 3)

        System.out.println(s + "(" + a + ", " + b + ") = " + r + " to " + p.getSocketAddress());

        bos = new ByteArrayOutputStream(); //create buffered stream for answer
        dos = new DataOutputStream(bos); //marshall
        dos.writeLong(r); //write 64bit long with result
        dos.close(); //flush to buffer and close

        answer = new DatagramPacket(bos.toByteArray(), bos.size(), p.getSocketAddress()); //4)
        server.send(answer); //send marshalled answer data
      }
      server.close(); //5)
    } catch (Throwable t) {
      t.printStackTrace();
    }
```

Structured Data: UDP Client / Java

Listing: UDPClientStructuredData.java Structured Data: UDP Client / Java

```
import java.io.ByteArrayInputStream;  import java.io.DataInputStream;  import java.io.OutputStream;
import java.io.ByteArrayOutputStream; import java.io.DataOutputStream; import java.net.InetAddress;
import java.net.DatagramPacket;       import java.net.DatagramSocket;

public class UDPClientStructuredData {
  public static final void main(final String[] args) {
    DatagramSocket  client;         InetAddress           ia;
    ByteArrayOutputStream bos;      DataOutputStream      dos;
    ByteArrayInputStream  bis;      DataInputStream       dis;
    DatagramPacket        p;        byte[]                data;

    try {
      ia = InetAddress.getByName("localhost");
      client = new DatagramSocket();  //create socket 1)

      bos = new ByteArrayOutputStream(); //create buffered stream for building message
      dos = new DataOutputStream(bos);   //mashall data
      dos.writeUTF("add");               //write operation name
      dos.writeLong(1234);               //write 64bit long: 1st operand
      dos.writeLong(9876);               //write 64bit long: 2nd operand
      dos.close();              //flush to buffer and close
      data = bos.toByteArray(); //get array with marshalled data to send

      p = new DatagramPacket(data, data.length, ia, 9997); //create package
      client.send(p);      //send package to server 2)

      client.receive(p); // receive answer
      bis = new ByteArrayInputStream(p.getData(), 0, p.getLength());
      dis = new DataInputStream(bis); //unmarshall
      System.out.println("Result: " + dis.readLong()); //3)

      client.close(); //4)
    } catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

Summary

- All data exists as a sequence of bits and bytes in the memory of a computer.
- Data types and formats are basically contracts regarding how a certain sequence of such bits and bytes is to be interpreted.
- Different programming languages might have different formats for numbers, text (next lesson), and even Boolean values.
- Different CPU architectures might define different formats as well.
- Thus, a sequence of bytes might be interpreted as different number on different computers.
- If data is exchanged, it is thus first marshalled from the sending computer's local format into a network-wide accepted format before sending and then unmarshalled into the receiving computer's local format upon receipt.

谢谢
Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://www.it-weise.de

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog

Bibliography

Bibliography I

1. George F. Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Upper Saddle River, NJ, USA: Pearson Education and Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 4th rev. edition, June 2005. ISBN 0201180596, 0321263545, 9780201180596, and 9780321263544. URL `http://books.google.de/books?id=d63sQPvBezgC`.
2. Tom. Question on stackoverflow.com: 64 bit ntohl() in c++?, May 1, 2009. URL `http://stackoverflow.com/questions/809902/64-bit-ntohl-in-c`.
3. Merlin Hughes, Michael Shoffner, and Derek Hamner. *Java Network Programming: A Complete Guide to Networking, Streams, and Distributed Computing*. Manning Pubs Co. Greenwich, CT, USA: Manning Publications Co., 1999. ISBN 188477749X and 9781884777493. URL `http://books.google.de/books?id=xapQAAAAMAAJ`.