Distributed Computing
Lesson 7: Text Encoding

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
1 Character Encoding
• How can we deal with text?
• How can we know that a sequence of bits stands for “汤卫思” and not for “Öläberschussländer”? 
• Different languages have different characters
• Different languages have different characters
• Originally, storage of text data was mainly designed for US English
• Different languages have different characters
• Originally, storage of text data was mainly designed for US English
• Here, 1B per character is sufficient: ASCII / ISO/IEC 8859–1 [1]
• Different languages have different characters
• Originally, storage of text data was mainly designed for US English
• Here, 1B per character is sufficient: ASCII / ISO/IEC 8859-1
• Original idea: bytes have different meaning, depending on language
• Different languages have different characters
• Originally, storage of text data was mainly designed for US English
• Here, 1B per character is sufficient: ASCII / ISO/IEC 8859-1
• Original idea: bytes have different meaning, depending on language (for German, we can e.g., replace some less important characters with “ä” and “ß”...)
• Different languages have different characters
• Originally, storage of text data was mainly designed for US English
• Here, 1B per character is sufficient: ASCII / ISO/IEC 8859-1
• Original idea: bytes have different meaning, depending on language
• GB2312 encoding especially for Chinese characters (2B for each non-ASCII char)
• Different languages have different characters
• Originally, storage of text data was mainly designed for US English
• Here, 1B per character is sufficient: ASCII / ISO/IEC 8859-1 [1]
• Original idea: bytes have different meaning, depending on language
• GB2312 [2] encoding especially for Chinese characters (2B for each non-ASCII char)
• These approaches are insufficient for other languages
Different languages have different characters

Originally, storage of text data was mainly designed for US English

Here, 1B per character is sufficient: ASCII / ISO/IEC 8859-1

Original idea: bytes have different meaning, depending on language

GB2312 encoding especially for Chinese characters (2B for each non-ASCII char)

These approaches are insufficient for other languages

Universal Coded Character Set (UCS) and Unicode
• Different languages have different characters
• Originally, storage of text data was mainly designed for US English
• Here, 1B per character is sufficient: ASCII / ISO/IEC 8859-1 [1]
• Original idea: bytes have different meaning, depending on language
• GB2312 [2] encoding especially for Chinese characters (2B for each non-ASCII char)
• These approaches are insufficient for other languages
• Universal Coded Character Set (UCS) [3] and Unicode [4–6]
• Encoded as UTF-7, UTF-8 [7] (compatible to ASCII), UTF-16, and UTF-32
Different languages have different characters

Originally, storage of text data was mainly designed for US English

Here, 1B per character is sufficient: ASCII / ISO/IEC 8859-1

Original idea: bytes have different meaning, depending on language

GB2312 encoding especially for Chinese characters (2B for each non-ASCII char)

These approaches are insufficient for other languages

Universal Coded Character Set (UCS) and Unicode

Encoded as UTF-7, UTF-8 (compatible to ASCII), UTF-16, and UTF-32

When sending text data, we need to make sure to use the right encoding!
• In Java, we can use the more general Character Stream API \(^8\) to deal with data conversation
• In Java, we can use the more general Character Stream API\cite{8} to deal with data conversation
• Input
In Java, we can use the more general Character Stream API \cite{8} to deal with data conversation.

- **Input:**
  - Reader's read one or multiple *unicode characters*
In Java, we can use the more general Character Stream API\cite{8} to deal with data conversion.

**Input:**
- Reader's read one or multiple *unicode characters*
- InputStreamReader's are readers which take their data from a byte-based input stream
• In Java, we can use the more general Character Stream API\textsuperscript{[8]} to deal with data conversation

• Input:
  • Reader\textsuperscript{s} read one or multiple \textit{unicode characters}
  • InputStreamReader\textsuperscript{s} are readers which take their data from a byte-based input stream

• Output
In Java, we can use the more general Character Stream API\cite{8} to deal with data conversion

- **Input:**
  - Reader's read one or multiple *unicode characters*
  - InputStreamReader's are readers which take their data from a byte-based input stream

- **Output:**
  - Writer's write one or multiple unicode characters
In Java, we can use the more general Character Stream API\cite{8} to deal with data conversation

**Input:**
- Reader's read one or multiple *unicode characters*
- InputStreamReader's are readers which take their data from a byte-based input stream

**Output:**
- Writer's write one or multiple unicode characters
- OutputStreamWriter's are writers which store all characters written to them in a specified encoding into byte-based output streams
• In Java, we can use the more general Character Stream API\textsuperscript{[8]} to deal with data conversation

• Input:
  • Reader\s read one or multiple \textit{unicode characters}
  • InputStreamReader\s are readers which take their data from a byte-based input stream

• Output:
  • Writer\s write one or multiple unicode characters
  • OutputStreamWriter\s are writers which store all characters written to them in a specified encoding into byte-based output streams

• TCP sockets: plug the InputStreamReader and OutputStreamWriter\s directly into the streams that the socket offers to us
• In Java, we can use the more general Character Stream API\textsuperscript{[8]} to deal with data conversation

• **Input:**
  - Reader\textsuperscript{s} read one or multiple *unicode characters*
  - InputStreamReader\textsuperscript{s} are readers which take their data from a byte-based input stream

• **Output:**
  - Writer\textsuperscript{s} write one or multiple *unicode characters*
  - OutputStreamWriter\textsuperscript{s} are writers which store all characters written to them in a specified encoding into byte-based output streams

• **TCP sockets:** plug the InputStreamReader and OutputStreamWriter\textsuperscript{s} directly into the streams that the socket offers to us

• **UDP sockets:** create the packets in memory
• Usually determined at compile-time
• Usually determined at compile-time
• Different types for different characters and encodings: `char`, `TCHAR`, `wchar_t`, ...
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class MinHTTPServer {
    // this is a minimum web server; see lesson 07 coming later
    public static final void main(final String[] args) {
        ServerSocket server;
        Socket client;
        BufferedReader br;
        int i;
        byte[] bs;
        PrintWriter pw;
        String s;
        FileInputStream fis;
        File f;
        Throwable x;

        try {
            server = new ServerSocket(9995); // create server socket
            for (;;) {
                client = server.accept(); // wait for and accept incoming connections
                br = new BufferedReader(new InputStreamReader(client.getInputStream())); // read character data
                pw = new PrintWriter(new OutputStreamWriter(client.getOutputStream(), "ISO_8859-1")); // chose the right encoding!

                process: { // [9, 10]
                    x = null;
                    try {
                        while ((s = br.readLine()) != null) { // read text from connection line-by-line until end
                            if (s.startsWith("GET \n\n")) { // try to find the GET command in the HTTP request
                                f = new File(s.substring(4, s.indexOf("\n\n", 4)).replace('/', File.separatorChar)); // in a very crude way, extract the requested path from that command
                                bs = new byte[(int) (f.length())]; // allocate a buffer of the right size
                                fis = new FileInputStream(f); // open the file
                                i = fis.read(bs); // read the complete file into memory
                                fis.close(); // close the file
                                pw.write("HTTP/1.1 200 OK\n\n"); pw.flush(); // send "success" according to "ISO_8859-1"
                                client.getOutputStream().write(bs, 0, i); // ... and the file content
                                break process; // ok, we are finished here
                            }
                        }
                    } catch (Throwable t) { x = t; } // if request fails, remember why
                    // hm, we did not find the file or had an error
                    pw.write("HTTP/1.1 404 Not Found\n\n<html><head><title>404</title></head><body><h1>404 - Not found</h1><pre>");
                    if (x != null) { x.printStackTrace(pw); } // write the error message (notice the <pre>...</pre> wrapper)
                    pw.write("</pre></body></html>"); // end the html body
                    pw.flush(); // and flush
                }
            }
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }
}

public class MinHTTPClient { // this is a minimum web client; see lesson 07 coming later

    public static final void main(String[] args) {
        String dest, request, response;  Socket sock;  OutputStreamWriter w;  BufferedReader r;

        dest = "www.baidu.com";  // a random example for a Chinese host
        request = "GET/index.html HTTP/1.1
Host: " + dest + "

// before they used GB2312 encoding

        try {
            sock = new Socket(dest, 80);  // web servers are usually listening at port 80
            w = new OutputStreamWriter(sock.getOutputStream());  // write the HTTP request
            w.flush();  // make sure that all data has been sent
            sock.shutdownOutput();  // closing down the channel for sending data to the server

            r = new BufferedReader(new InputStreamReader(sock.getInputStream(), "UTF-8"));  // Baidu uses UTF-8 encoding
            // before they used GB2312 encoding

            while ((response = r.readLine()) != null) {
                    System.out.println(response);  // print to output
            }

            sock.close();
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }
}
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.Socket;

public class MinHTTPClientJava17 { // this is a minimum web client; see lesson 07 coming later
    public static final void main(String[] args) {
        String dest, request, response;
        dest = "www.baidu.com"; // a random example for a Chinese host
        request = "GET /index.html HTTP/1.1
Host: " + dest + "


    }
    try (Socket sock = new Socket(dest, 80)) { // web servers are usually listening at port 80
            try (OutputStreamWriter w = new OutputStreamWriter(sock.getOutputStream())) {
                w.write(request); // write the HTTP request
                w.flush(); // make sure that all data has been sent
                sock.shutdownOutput(); // closing down the channel for sending data to the server

            } catch (Throwable t) {
                t.printStackTrace();
            }
    }
}
Summary

- Text is a very complex variable-length data structure.
- Historically, there exist many different mappings from characters to bits and bytes.
- Unicode assigns an integer number to a character.
- UTF-8 defines how such a number can be translated to a variable-length list of bits.
- UTF-8 is now the prevalent text encoding in the internet, i.e., you should store all your text-based documents (txt, html, xml, ...) in UTF-8 encoding.
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
Bibliography I


