Code Inspection

Matteo Greselin - Matr. 862908

Version 1.0

Contents

1 Classes and Methods Analyzed

All methods that will be listed, are located in https://github.com/mrniko/netty-socketio/blob/master/src/main/java/com/corundumstudio/socketio/protocol/PacketEncoder.java

- public PacketEncoder(Configuration configuration, JsonSupport json-Support)
  Starting line : 44
- public JsonSupport getJsonSupport()
  Starting line : 49
- public ByteBuf allocateBuffer(ByteBufAllocator allocator)
  Starting line : 53
- public void encodeJsonP(Integer jsonpIndex, Queue<Packet> packets, ByteBuf out, ByteBufAllocator allocator, int limit) throws IOException
  Starting line : 61
- private void processUtf8(ByteBuf in, ByteBuf out, boolean jsonpMode)
  Starting line : 108
- public void encodePackets(Queue<Packet> packets, ByteBuf buffer, ByteBufAllocator allocator, int limit) throws IOException
  Starting line : 123
- private byte toChar(int number)
  Starting line : 144
- static int stringSize(long x)
  Starting line : 170
- static void getChars(long i, int index, byte[] buf)
  Starting line : 176
- public static byte[] toChars(long i)
  Starting line : 211

- public static byte[] longToBytes(long number)
  Starting line : 218
- public void encodePacket(Packet packet, ByteBuf buffer, ByteBufAllocator allocator, boolean binary) throws IOException
  Starting line : 230
- public static int find(ByteBuf buffer, ByteBuf searchValue)
  Starting line : 334
- private static boolean isValueFound(ByteBuf buffer, int index, ByteBuf search)
  Starting line : 343

1.1 Task Performed by class analyzed

The class that i have analyzed is stored into the package com.corundumstudio.socketio.protocol so i suppose that it will perform some task reguarding connection, setting, encoding and so on.
During the insertion of javadoc, that will be presented in the next chapter, following task perfomed by methods analyzed have been brought out:

- check if JavaScript Object Notation is enable
- try to perform utf-8 codifing
- perform encoding between char and byte
- allocate buffer with respect to an indicated choise
- check if a fixed value is into a buffer
- check if, after a search, something has been found

2 Functional Role Of Assigned class

In the assigned class is totally absent any type of documentation. The idea is to provide javadoc or comments step-by-step for each methods in order to allow a better understanding in the future. This assignement is a little bit more difficult due to the total absence of any documentation: this involves that code is, in some part, hardly understandable.

2.1 public PacketEncoder(Configuration configuration, JsonSupport jsonSupport)

2.1.1 Java Doc

```java
/**
*Java constructor, it permitts to defend from further
    modifications by cloning
*
*@param configuration, it defend from further modifications by
    cloning
*@param jsonSupport, it allows to implement custom realizations
    to JSON support operations.
*
*/
```

2.1.2 Code

```java
public PacketEncoder(Configuration configuration, JsonSupport
    jsonSupport) {
  this.jsonSupport = jsonSupport;
  this.configuration = configuration;
}
```

2.2 public JsonSupport getJsonSupport()

2.2.1 Java Doc

```
/**
*Allow to take the value of a variable defined as private
*
*@return jsonSupport, it is the support for JavaScript Object
    Notation
*/
```

2.2.2 Code

```
public JsonSupport getJsonSupport() {
  return jsonSupport;
}
```

2.3 public ByteBuf allocateBuffer(ByteBufAllocator allocator)

2.3.1 Java Doc

```
/**
*It prepares the buffer with respect to the selected type
*We can have two possible returns
*
*@param allocator, it is the buffer that we want to allocate
*
*@return type, it is the type of buffer that has been allocated
*/
```

Note: configuration.isPrefererDirectBuffer is a method implemented in class Configuration. I have reported it's javaDoc in order to increase comprehensibility.

```
/**
* Buffer allocation method used during packet encoding.
* Default is {@code true}
*
* @param preferDirectBuffer    {@code true} if a direct buffer
    should be
*                 tried to be used as target for the encoded
*                 messages. If {@code false} is used it will
*                 allocate a heap buffer, which is backed by
*                 an byte array.
*/
```

2.3.2 Code

```
public ByteBuf allocateBuffer(ByteBufAllocator allocator) {
  if (configuration.isPreferDirectBuffer()) {
    return allocator.ioBuffer();
  }
  return allocator.heapBuffer();
}
```

2.4 public void encodeJsonP(Integer jsonpIndex, Queue<Packet> packets, ByteBuf out, ByteBufAllocator allocator, int limit) throws IOException

2.4.1 Java Doc

```
/**
*
*
*@param jsonIndex, it is an integer used for setting jsonpMode
    that will usefor another process
*@param packets
*@param out, it indicate the type of buffer that has been
    allocated
*@param allocator
*@param limit
*
*/
```

2.4.2 Code

```java
public void encodeJsonP(Integer jsonpIndex, Queue<Packet>
    packets, ByteBuf out, ByteBufAllocator allocator, int limit)
    throws IOException {
  boolean jsonpMode = jsonpIndex != null;
  ByteBuf buf = allocateBuffer(allocator);
  int i = 0;
  while (true) {
    Packet packet = packets.poll();
    if (packet == null || i == limit) {
      break;
    }
    ByteBuf packetBuf = allocateBuffer(allocator);
    encodePacket(packet, packetBuf, allocator, true);
    int packetSize = packetBuf.writerIndex();
    buf.writeBytes(toChars(packetSize));
    buf.writeBytes(B64_DELIMITER);
    buf.writeBytes(packetBuf);

    packetBuf.release();

    i++;

    for (ByteBuf attachment : packet.getAttachments()) {
      ByteBuf encodedBuf = Base64.encode(attachment,
    Base64Dialect.URL_SAFE);
      buf.writeBytes(toChars(encodedBuf.readableBytes() + 2));
      buf.writeBytes(B64_DELIMITER);
      buf.writeBytes(BINARY_HEADER);
      buf.writeBytes(encodedBuf);
    }
  }

  if (jsonpMode) {
    out.writeBytes(JSONP_HEAD);
    out.writeBytes(toChars(jsonpIndex));
    out.writeBytes(JSONP_START);
  }

  processUtf8(buf, out, jsonpMode);
    buf.release();

  if (jsonpMode) {
    out.writeBytes(JSONP_END);
  }
}
```

2.5 private void processUtf8(ByteBuf in, ByteBuf out, boolean jsonpMode)

2.5.1 Java Doc

```
/**
*This methods processes values and decode them with the utf-8
    encoding.
*
*@param in, it indicates the parameters from where i will read
    the
*       value that i will encode
*@param out, it indicates the parameters from where i will write
     the
*        value after that i will encode it
*@param jsonpMode, it idicates if JavaScript OrientedNotation is
*     enabled
*
*/
```

2.5.2 Code

```
private void processUtf8(ByteBuf in, ByteBuf out, boolean
    jsonpMode) {
  while (in.isReadable()) {
    short value = (short) (in.readByte() & 0xFF);
    if (value >>> 7 == 0) {
      if (jsonpMode && (value == '\\' || value == '\'')) {
        out.writeByte('\\');
      }
      out.writeByte(value);
      } else {
        out.writeByte(((value >>> 6) | 0xC0));
        out.writeByte(((value & 0x3F) | 0x80));
      }
    }
  }
```

2.6 public void encodePackets(Queue<Packet> packets, ByteBuf buffer, ByteBufAllocator allocator, int limit) throws IOException

2.6.1 Java Doc

```
/**
*This method encode packets
*
*@param packets , it is a queue of packets
*@param buffer , it rappresent the buffer that is allocated
*@param allocator , it rappresent the byte of buffer that are
    allocated
*@param limit , it rapresent the limit of packets that i can
    remove from
*        the head of the queue
*
*/
```

2.6.2 Code

```
public void encodePackets(Queue<Packet> packets, ByteBuf buffer,
    ByteBufAllocator allocator, int limit) throws IOException {
  int i = 0;
  while (true) {
    Packet packet = packets.poll();
    if (packet == null || i == limit) {
      break;
    }
    encodePacket(packet, buffer, allocator, false);
    i++;
    for (ByteBuf attachment : packet.getAttachments()) {
      buffer.writeByte(1);
      buffer.writeBytes(longToBytes(attachment.readableBytes() +
    1));
      buffer.writeByte(0xff);
      buffer.writeByte(4);
      buffer.writeBytes(attachment);
    }
  }
}
```

2.7 private byte toChar(int number)

2.7.1 Java Doc

```
/**
*It produce a char from a number, it return the char as byte
```

```
*
*@param number
*
*@return byte
*/
```

2.7.2 Code

```
private byte toChar(int number) {
  return (byte) (number ^ 0x30);
}
```

2.8 static int stringSize(long x)

2.8.1 Java Doc

```
/**
*It calculates the correct stringSize with respect
*to values provides in te declaration of  sizeTable
*
*@param x, numeric variable that must be positive
*
*@return i, numeric variable
*/
```

2.8.2 Code

```
static int stringSize(long x) {
  for (int i = 0;; i++)
    if (x <= sizeTable[i])
      return i + 1;
}
```

2.9 static void getChars(long i, int index, byte[] buf)

2.9.1 Java Doc

```
/**
*
*
*@param i,
*@param index,
*@param buf,
*
*/
```

2.9.2 Code

```
static void getChars(long i, int index, byte[] buf) {
  long q, r;
  int charPos = index;
  byte sign = 0;
  if (i < 0) {
    sign = '-';
    i = -i;
  }
  // Generate two digits per iteration
  while (i >= 65536) {
    q = i / 100;
    // really: r = i - (q * 100);
    r = i - ((q << 6) + (q << 5) + (q << 2));
    i = q;
    buf[--charPos] = (byte) DigitOnes[(int)r];
    buf[--charPos] = (byte) DigitTens[(int)r];
  }
  // Fall thru to fast mode for smaller numbers
  // assert(i <= 65536, i);
  for (;;) {
    q = (i * 52429) >>> (16 + 3);
    r = i - ((q << 3) + (q << 1));
    // r = i-(q*10) ...
    buf[--charPos] = (byte) digits[(int)r];
    i = q;
    if (i == 0)
      break;
  }
  if (sign != 0) {
    buf[--charPos] = sign;
  }
}
```

2.10 public static byte[] toChars(long i)

2.10.1 Java Doc

```
/**
*The methods create a variables buffer with a calculated
*size and it will used to call the methods getChars
*
*@param i,
*
*@return buf, variable with the correct size calculated
*/
```

2.10.2 Code

```
public static byte[] toChars(long i) {
  int size = (i < 0) ? stringSize(-i) + 1 : stringSize(i);
  byte[] buf = new byte[size];
  getChars(i, size, buf);
  return buf;
}
```

2.11 public static byte[] longToBytes(long number)

2.11.1 Java Doc

```
/**
*This methods return a byte after that it has recived a
*long variable
*
*@param number
*
*@return res, a byte variables that correspond to the variable
*   number that has been passed
*/
```

2.11.2 Code

```
public static byte[] longToBytes(long number) {   // TODO
    optimize
  int length = (int)(Math.log10(number)+1);
  byte[] res = new byte[length];
  int i = length;
  while (number > 0) {
    res[--i] = (byte) (number % 10);
    number = number / 10;
  }
  return res;
}
```

2.12 public void encodePacket(Packet packet, ByteBuf buffer, ByteBufAllocator allocator, boolean binary) throws IOException

2.12.1 Java Doc

```
/**
*
*
*@param packet, it indicate the packet on which operations are
    done
*@param buffer, it represent the buffer that has been allocated
*@param allocator, it represent the byte of the buffer that has
    been allocated
*@param binary, it isused to decide if buffer is already
    allocated
*     or not
*
*@return
*/
```

2.12.2 Code

```java
public void encodePacket(Packet packet, ByteBuf buffer,
    ByteBufAllocator allocator, boolean binary) throws
    IOException {
  ByteBuf buf = buffer;
  if (!binary) {
    buf = allocateBuffer(allocator);
  }
  byte type = toChar(packet.getType().getValue());
  buf.writeByte(type);
  try {
    switch (packet.getType()) {
      case PONG: {
        buf.writeBytes(packet.getData().toString().getBytes(
    CharsetUtil.UTF_8));
        break;
      }
      case OPEN: {
        ByteBufOutputStream out = new ByteBufOutputStream(buf);
        jsonSupport.writeValue(out, packet.getData());
        break;
      }
      case MESSAGE: {
        ByteBuf encBuf = null;
        if (packet.getSubType() == PacketType.ERROR) {
          encBuf = allocateBuffer(allocator);
          ByteBufOutputStream out = new ByteBufOutputStream(
    encBuf);
          jsonSupport.writeValue(out, packet.getData());
        }
        if (packet.getSubType() == PacketType.EVENT || packet.
    getSubType() == PacketType.ACK) {
          List<Object> values = new ArrayList<Object>();
          if (packet.getSubType() == PacketType.EVENT) {
            values.add(packet.getName());
          }
          encBuf = allocateBuffer(allocator);
          List<Object> args = packet.getData();
          values.addAll(args);
          ByteBufOutputStream out = new ByteBufOutputStream(
    encBuf);
          jsonSupport.writeValue(out, values);
          if (!jsonSupport.getArrays().isEmpty()) {
            packet.initAttachments(jsonSupport.getArrays().size
    ());
            for (byte[] array : jsonSupport.getArrays()) {
              packet.addAttachment(Unpooled.wrappedBuffer(array)
    );
            }
            packet.setSubType(PacketType.BINARY_EVENT);
          }
        }
        byte subType = toChar(packet.getSubType().getValue());
        buf.writeByte(subType);
        if (packet.hasAttachments()) {
          byte[] ackId = toChars(packet.getAttachments().size());
          buf.writeBytes(ackId);
          buf.writeByte('-');
        }
        if (packet.getSubType() == PacketType.CONNECT) {
          if (!packet.getNsp().isEmpty()) {
            buf.writeBytes(packet.getNsp().getBytes(CharsetUtil.UTF_8));
          }
        } else {
        if (!packet.getNsp().isEmpty()) {
          buf.writeBytes(packet.getNsp().getBytes(CharsetUtil.UTF_8));
          buf.writeByte(',');
        }
      }
      if (packet.getAckId() != null) {
        byte[] ackId = toChars(packet.getAckId());
        buf.writeBytes(ackId);
      }
      if (encBuf != null) {
        buf.writeBytes(encBuf);
        encBuf.release();
      }
      break;
    }
  }finally {
  // we need to write a buffer in any case
    if (!binary) {
      buffer.writeByte(0);
      int length = buf.writerIndex();
      buffer.writeBytes(longToBytes(length));
      buffer.writeByte(0xff);
      buffer.writeBytes(buf);
      buf.release();
    }
  }
}
```

2.13 public static int find(ByteBuf buffer, ByteBuf searchValue)

2.13.1 Java Doc

```
/**
*It scans a fixed parameters and with a for cycle checks the
*presence of occurences of another selected parameters
*
```

```
*@param bufferm
*@param searchValue
*
*@return integer that indicate if value has been found and, if
   it
*   is happened, it indicates value positions
*/
```

2.13.2 Code

```java
public static int find(ByteBuf buffer, ByteBuf searchValue) {
  for (int i = buffer.readerIndex(); i < buffer.readerIndex() +
    buffer.readableBytes(); i++) {
    if (isValueFound(buffer, i, searchValue)) {
    return i;
    }
  }
  return -1;
}
```

2.14 private static boolean isValueFound(ByteBuf buffer, int index, ByteBuf search)

2.14.1 Java Doc

```java
/**
*It scans a fixed parameters and with a for cycle checks the
*presence of occurences of another selected parameters
*
*@param buffer, it indicates what i'm looking for
*@param index, it indicates from where i will start
*@param search, it is the buffer into where i'm looking for
*
*@return boolean value, it indicates if the value has been
*   found or not
*/
```

2.14.2 Code

```java
private static boolean isValueFound(ByteBuf buffer, int index,
   ByteBuf search) {
  for (int i = 0; i < search.readableBytes(); i++) {
    if (buffer.getByte(index + i) != search.getByte(i)) {
      return false;
    }
  }
  return true;
}
```

3 | List Of Issues Found By Applying The Checklist

Next sectiond will provide error with respect to an assigned checklist that contains all correct coding rules. The identification of errors was made difficult becouse of the absence of comments that could clarify the code.
Only section with errors have been described.
Reguarding sections **Indentation - Wrapping Lines - Package and Import Statements - Arrays - Methods Calls - Outputformat - Computation, Comparison And Assignment - Files** i haven't found any coding rules violation

3.1 Naming Conventions

Naming conventions, in this class, should create some problem.
Two methods, encodePacket and encodePackets are ambiguous.
Variables DigitTens and DigitOnes start with a work capitalized.
One-character variables is used in a wrong way, as 'q' and 'r' variables in line 177.

3.2 Braces

In this class is used Kernighan and Ritchie style for braces. However curly braces are not always used in the correct way:

- line 170, methods stringSize(...)
  The for cycle in line 171 and the if cycle in line 172 don't use curly braces
- line 203, realted to methods getChars(...)
  The if instruction doesn't use curly baces

3.3 File Organization

In this class blank lines are used to separate different sections, but not all of these are useful.
With respect to maximum line lenght, it is not respected: more lines lenght exceed 80 characters.

3.4 Comments

In the class PacketEncoder, comments are totally absent. In order to increasy the ability to understanding, an upgrade of comments and javadoc has been provided in the previous chapter.

3.5 Java Source Files

PacketEncoder.java provides only one class, as required by checklist, but javadoc is complieely absent.

3.6 Class And Interface Declaration

In lines 148-167 there are variables that should be declareted about in line 40/45.

3.7 Inizialization And Declaration

Reguaring this section, it is partially respected:

- constructor is called in the right position, after variables declaration at the beginning o class;
- variables DigitTens, DigitOnes, digits and sizeTable are correclty define as variables but aren't defined in the correct position (after the beginning of class);
- when required, not all local variable are defined at the beginning of block. For example variable type (line 235)

3.8 Object Comparison

All object comparison are done in the wrong way, with '==' and not with 'equals'

3.9 Exceptions

In different methods arepresent exception. IOException is used in methods:

- encodeJsonP, line 61
- encodePackets, line 123
- encodePacket, line 230

Each of this methods works with input and output command with respect to buffers but in the implementation of exception isn't provide any type of comment. With this approch, when i will have an IOException, i will have an exception (correctly) but without any message.

In methods called processUtf8 there isn't any exception but also it work with some output command for buffers. This methods receive the fixed exeption by the caller methods.
In other methods isn't necessary any type of exception.

3.10 Flows Of Controls

The methods encodePacket, line 230, present a switch instruction, lines 239-318.
All cases are addressed by breack but there isn't a default branch. It's also true that the switch instruction is included in a try/catch instruction; if any case of switch will be done, the execution will pass to the finally branch.
With respect to loops, not all are implemented in the correct way:

- line 198, related to method getCharts
  The for cycle is not correct implemented, it hasn't the contidions

4 | Appendix

4.1 Used Tools

We used the following tools to make the Code Inspection document:

- LYX 2.1: to redact and format the document;
- NetBeans IDE 8.1: to analyze and inspect the source code provided.

4.2 Working Time

In order to complete this document i have used about 24 hours, more of them due to the completely absence of documentation, that is useful on order to understang an unknowing code.