ネットワークアプリケーション

第7回 Javaによるネットワークプログラミング

石井 健太郎
（423研究室・オフィスアワー火3限）

スケジュール

9月27日　第1回「TCP/IPプロトコルスイート」
10月4日　第2回「Javaによるウィンドウプログラミング」
10月11日　第3回「ネットワークアプリケーションのプログラミングモデル」
10月18日　第4回「Javaによるネットワークプログラミング」
10月25日　第5回「Javaによるネットワークプログラミング」
11月8日　第6回「Javaによるネットワークプログラミング」
11月15日　第7回「Javaによるネットワークプログラミング」　最終課題（1）
11月17日　第8回「ウェブプログラミングについて」

スケジュール

11月22日　第9回「JavaScriptによるクライアントサイドウェブプログラミング」
11月29日　第10回「JavaScriptによるクライアントサイドウェブプログラミング」
12月6日　第11回「JavaScriptによるクライアントサイドウェブプログラミング」
12月13日　第12回「JavaScriptによるクライアントサイドウェブプログラミング」
12月20日　第13回「JavaScriptによるクライアントサイドウェブプログラミング」
1月10日　第14回「JavaScriptによるクライアントサイド...」　**最終課題 (2)**
1月17日　第15回「まとめと演習」

やりとりの手順を考える

Server		Client
（カードを振り分ける）		
	<--	(connect)
PlayerCards 0 *	-->	
PlayerCards 1 *	-->	
FieldCards *	-->	
（ほかのクライアントを待つ）		
（クライアントが人数分接続したら）		
Start	-->	
Turn 0 0	-->	
	<--	Hand <id> <card>
Hand <id> <card>	-->	
Turn 1 1	-->	
	<--	Hand <id> <card>
Hand <id> <card>	-->	
Turn 0 2	-->	
	<--	Claim <id>
Claim <id>	-->	（ダウトの吹き出し・カードを開く）
（ダウトの判定）		
（3秒待つ）		
PlayerCards 0 *	-->	
PlayerCards 1 *	-->	
FieldCards *	-->	
Turn 0 2	-->	

どのようなデータ形式にすればよいか？

- 可能な限り，ASCII文字列で1行1コマンド/レスポンスとするのがよい
  - ASCII文字列にすることで，
    - バイトオーダーを気にしなくてよい
    - 空白・カンマといった，アルファベットや数値には出現しない文字を区切りに利用できる
      - 可変長データが表現しやすい
    - 送信/受信したデータそのままで人間が読めるのでデバグが容易
  - 1行1コマンド/レスポンスにすることで，
    - 1つの送信に対して相手からの受信を期待することで，
      アプリケーション同士が足並みをそろえやすい（デッドロックになりにくい）
      - 改行文字までを1つのデータとする送信を常に考える
    - 1つのデータ単位が読みやすいのでやはりデバグが容易

実例

- ASCII文字列で1行1コマンド/レスポンスとする
  - <id>␣<locationX>␣<locationY>␣<orientationX>␣<orientationY>¥n

  - 送信側

```
writer = new PrintWriter(socket.getOutputStream(), true);
writer.println(id + " " + locationX + " " + locationY + " " + orientationX + " " + orientationY);
writer.flush();
```

  - 受信側

```
reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
line = reader.readLine();
tokens = new StringTokenizer(line);
id = Integer.parseInt(tokens.nextToken());
locationX = Double.parseDouble(tokens.nextToken());
locationY = Double.parseDouble(tokens.nextToken());
orientationX = Double.parseDouble(tokens.nextToken());
orientationY = Double.parseDouble(tokens.nextToken());
```

1行1コマンド/レスポンスにできない場合

- 区切り行を設ける
  - 空行
  - ピリオドだけの行
  - 「250␣」で始まる行
  - 「FieldCards ＊」がきたら

実例

- 1行1コマンド/レスポンスにできない場合
  - 区切り行を設ける

（※赤字は入力）

```
> telnet is.is.oit.ac.jp 587
220 is.is.oit.ac.jp ESMTP Postfix (Ubuntu)
EHLO is.is.oit.ac.jp
250-is.is.oit.ac.jp
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
```

バイナリデータを送らなければいけないときは？

- まず考えるべきは，送るデータを固定長にできないか？
  - ただし，たいていの場合，固定長の効率はよくない
    - cf. バイナリデータの必要性

- 固定長にできない(したら意味がない)場合，
  以下のいずれかを考える
  - データには出現しない区切りバイト(バイト列)を使用する
  - 決まったバイト位置にデータサイズを埋め込む

実例

```
class Marker  {
  long id;
  float locationX;
  float locationY;
  float orientationX;
  float orientationY;
}
```

- 送信データを固定長にする
  - 送信側

```
output = new DataOutputStream (socket.getOutputStream () ) ;
output.writeLong (marker.id) ;
output.writeFloat (marker.locationX) ;
output.writeFloat (marker.locationY) ;
output.writeFloat (marker.orientationX) ;
output.writeFloat (marker.orientationY) ;
output.flush () ;
```

  - 受信側

```
input = new DataInputStream (socket.getInputStream () ) ;
id = input.readLong () ;
locationX = input.readDouble () ;
locationY = input.readDouble () ;
orientationX = input.readDouble () ;
orientationY = input.readDouble () ;
```

実例

- データには存在しない区切りバイト(バイト列)を使用する
  - 一枚の画像中のみつかったマーカのIDを列挙する
    - 0のIDを持つマーカはないという前提とすると
    - 35, 2, 43, 119, 15, 0
      というようなバイト列は5つのIDのマーカが見つかったことを意味する

実例

- 決まったバイト位置にデータサイズを埋め込む
  - <id>, <version>, <linkCount>, <link>, <link>, <link>, ...

  - 送信側

```
unsigned char data[1024];
data[0] = id;
data[1] = version;
data[2] = linkCount;
for (int i = 0; i < linkCount; i++)
  data[i + 3] = links[i];
send(socket, data, 3 + linkCount, 0);
```

  - 受信側

```
unsigned char data[1024];
size = recv(socket, data, 3, 0);
id = data[0];
version = data[1];
linkCount = data[2];
size = recv(socket, data + 3, linkCount, 0);
for (int i = 0; i < linkCount; i++)
  links[i] = data[i+3];
```

思ったとおりのデータがこなかったとき

- データがフォーマットにそっていない
  - 無視する
  - 再送を要求する

  - TCPではまず起こらない(プログラムが間違っている場合を除く)が
    UDPやシリアル通信ではありがちなので，対応を考えておく必要がある

- 届くはずのデータが届かない
  - もう1度データを要求する
  - あきらめる

  - タイムアウトにより次の処理を行う
  - プロトコルの設計(やりとりの手順)をもう1度見直す
    - デッドロックになっていないかを確認する

最終課題について（一般的な話）

- 先願優先主義
  - 内容に重複があった場合，（特許と同じ）先願優先主義にて採点する
  - つまり，同じような内容の課題が提出された場合，
    あとに提出された課題の評価を減点する
  - 作成するプログラムの内容は自由とする予定であるので，
    重複を避けるためには，自分オリジナルの要素を可能な限り埋め込むとよい

- 最終提出期限
  - 期末試験日の2～3日あとの深夜までとする予定
  - 正確な提出期限は試験日決定後に案内します

最終課題について（一般的な話）

- プログラムのソースコード＋レポートを提出
  - いずれも評価対象です
- プログラムのソースコード
  - プログラム実行に基づく評価 → こちらでコンパイルのうえ実行した結果により点数をつける
    - どんなに設計がよくできていても機能が充実していても，実行できない場合プログラム点はつきません
      ただし，実行できない場合はメールにて連絡するようにします
- レポート
  - 読み手が理解できれば様式は自由．ただし，理解しやすさは評価の対象となる
  - 以下の項目について記述する
    - 概要: それが，何をするプログラムか？どう役に立つのか？どんな楽しさがあるか？どのような点を工夫したか？
    - プロトコル: 通信プロトコルについて，どのような手順でコマンド・データをやりとりしているのか？をまとめる
    - 方式・データの選択: 以下の項目について理由とともに明記する
      - TCPかUDPか
      - メッセージパッシングかストリーミングか
      - テキストデータかバイナリデータか
  - その他任意で，必要と思う項目を記述してよい
    - 例えば，プログラムの起動方法・必要な外部ライブラリ・設定ファイルの配置方法など（通常と異なる場合）
    - 例えば，正しく動作しているときの画像など

最終課題について（課題の内容）

- Javaによるウィンドウアプリケーションで，
  ソケット通信によるネットワークアプリケーションであり，
  なにかの役に立つ，または，なにかの楽しさをともなう，
  アプリケーションを考えて，作成する
  - どう役に立つか/どんな楽しさがあるかはレポートの概要に記述する

- 下記の内容を，1つのフォルダに格納し，
  それを zip で圧縮したものを，提出フォルダ（Xドライブ）に提出
  - レポート(「<学生番号>.<拡張子>」)　例: N14999.docx
  - プログラムのソースコード(プロジェクトごとに任意のフォルダ名で配置する(*.java))
  - 動作に必要なライブラリなどがある場合はそれも配置する(*.jar など)
  - 複雑でわかりにくそうだったら README.txt に，フォルダ・ファイルの説明を書いてもよい
- X:¥IN科専門¥石井講師¥ネットワークアプリケーション¥最終課題(Java)
- ファイル名は「<学生番号>.zip」とする（ハイフンなし）　例: N14999.zip

- 次回はあさって11月17日（木）です