通信処理で使用する一般的な用語

作成日: 2017/12/18
作成者: 西村

更新履歴

更新日	更新概要	作業者
2017/12/18	・ 新規作成	西村
	・	
	・	
	・	
	・	

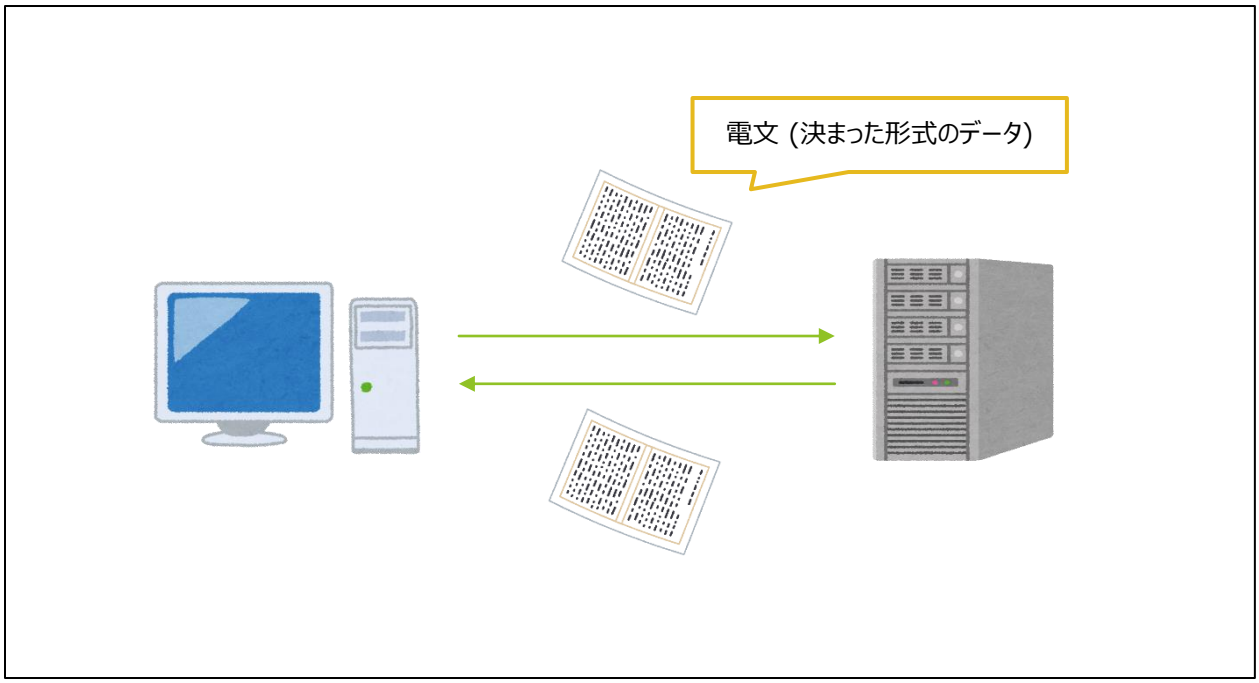
目次

はじめに

この資料では、主に UDP や TCP、RS232C といった通信処理で使用する一般的な用語について簡単に説明します。
通信のプログラムを作成する案件に初めて関わる際に参考にしてください。

電文

「電文」は、機器間で通信を行う際に送信・受信する、形式の決められたデータのことです。電文の形式や種別は、各システムの仕様書などで定義が決められています。



電文の構造 (電文フォーマット)

電文の構造は仕様によってまちまちですが、電文の各種情報(電文種別番号やデータ長など)を統一形式で格納している「ヘッダ部」と、実際のデータ内容を格納する「データ部」に分かれていることが多いです。

電文構造に使われる項目の例

項目	概要
STX	電文の開始位置を示す制御コードです。(0x02)
ヘッダ部	電文の各種情報を格納している部分です。 「SOH」(0x01。ヘッダの開始を示す制御コード) → ヘッダ部 →「STX」という構成になる場合もあります。
電文種別	電文の種別を示す番号です。
データ長	電文の長さを示す値です。
データ部	電文のデータを格納している部分です。
ETX	電文の終了位置を示す制御コードです。(0x03)
チェックサム (SUM)	電文の途中の部分が破損などしていないかを確認する値です。 ETX より前にある場合、後にある場合などバリエーションがあります。

例 1

STX	ヘッダ部	データ部	ETX	SUM

例 2

SOH	ヘッダ部	STX	データ部	ETX	SUM

電文に使用されるデータ形式

電文に使用されるデータ形式はいくつかの種類があります。

形式	概要
バイナリ	「1234」→「0000 0100 1101 0010」 (0x04D2)のように バイナリデータの状態でそのままデータを格納する形式です。
ASCII (テキスト)	「1234」→「0x31 0x32 0x33 0x34」(ASCII コード) のように、 文字としてデータを格納する形式です。
BCD	「1234」→「0x12 0x34」のように、 16 進数にしたときに値がわかりやすい形でデータを格納する形式です。

※「この値は 16 進数です」ということを示すときは、「0x12」「H12」「12H」といった表記が使われることがあります。

(H = Hex の H、0x = Hex の X)

(参考) それぞれの形式の比較

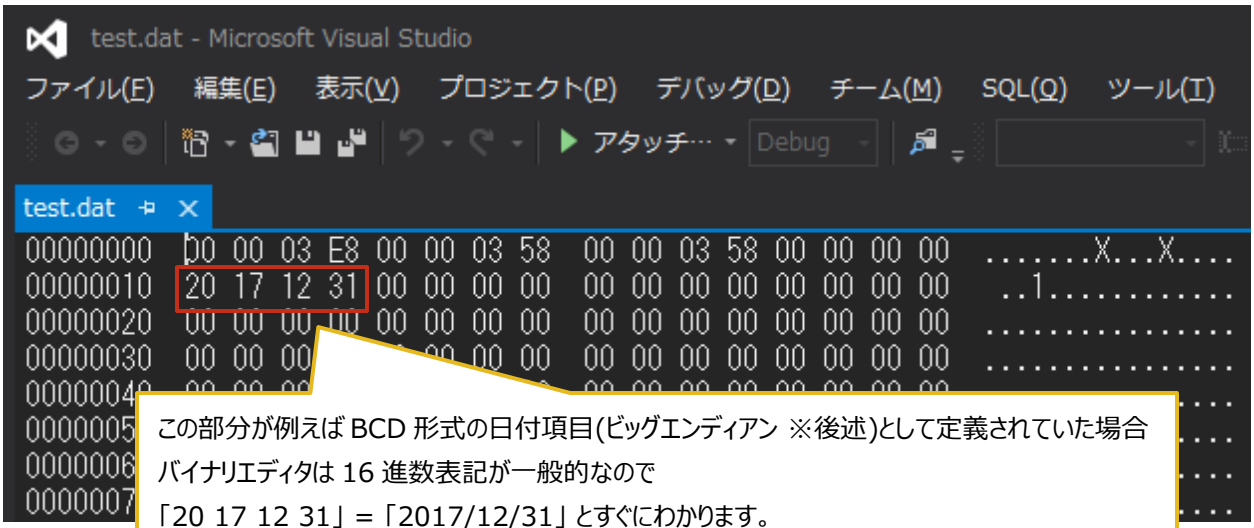
値	バイナリ	ASCII	BCD
0	0x00	0x30	0x00
1	0x01	0x31	0x01
2	0x02	0x32	0x02
3	0x03	0x33	0x03
4	0x04	0x34	0x04
5	0x05	0x35	0x05
6	0x06	0x36	0x06
7	0x07	0x37	0x07
8	0x08	0x38	0x08
9	0x09	0x39	0x09
10	0x0A	0x31 0x30	0x10
11	0x0B	0x31 0x31	0x11
12	0x0C	0x31 0x32	0x12
13	0x0D	0x31 0x33	0x13
14	0x0E	0x31 0x34	0x14
15	0x0F	0x31 0x35	0x15
16	0x10	0x31 0x36	0x16

バイナリと BCD は 10 以降違いがわかると思います。

それぞれの形式には利点や欠点があります。

形式	利点	欠点
バイナリ	・値を加工する必要がない ・データ量が最小限	・制御コード(STX=0x02, ETX=0x03 等)と通常の値が被ることがあるため、回避ルールなどが別途必要になることがある
ASCII (テキスト)	・制御コードと通常値が被らない	・データ量が多くなりやすい
BCD	・制御コードと通常値が被らない ・バイナリエディタで見た時にわかりやすい (バイナリエディタでは 2 桁区切りの 16 進数で表示されるため)	・プログラム上で BCD を通常の数値にするための変換関数が別途必要

例: Visual Studio でバイナリデータが格納されたファイルを表示した例 (フォーマットは架空のものです)



この部分が例えば BCD 形式の日付項目(ビッグエンディアン ※後述)として定義されていた場合
バイナリエディタは 16 進数表記が一般的なので
「20 17 12 31」=「2017/12/31」とすぐにわかります。

バイナリの場合は「01 33 C9 DF」
ASCII の場合は「32 30 31 37 31 32 33 31」です。

エンディアン

「エンディアン」は、「どういう並びの順序でデータを持つか」を決めた種類のことです。

エンディアンには、下記の種類があります。

種類	概要
ビッグエンディアン	「0A0B0C0D」→ 0A 0B 0C 0D のように、上位のデータを上に配置して持つ形式です。 (メモリのアドレスとしては、0A = N 番地、0B = N+1 番地…になり最上位のデータのほうが番地が小さい)
リトルエンディアン	「0A0B0C0D」→ 0D 0C 0B 0A のように、上位のデータを下に配置して並べていく形式です。

エンディアンは、場所によって使われている種類が違うことがあります。

種類	主に使われている場所
ビッグエンディアン	・ HP-UX という OS では、一般的にビッグエンディアンでデータを扱っています。 ・ UDP や TCP の通信では、データをビッグエンディアンに変換して送信するのが一般的です。(ネットワークバイトオーダーと呼ばれます)
リトルエンディアン	・ Linux では、一般的にリトルエンディアンでデータを扱っています。(Intel x86 CPU)

エンディアンの変換 (C 言語)

エンディアンを変換する際は、特に C 言語の通信処理の場合は「htonl()」などを使用すると便利です。

OS がどんなエンディアンだったとしても、ビッグエンディアン (ネットワークバイトオーダー) ⇔ OS 内部のエンディアン (ホストバイトオーダーと呼ばれます) の相互の変換を簡単にしてくれます。

関数	意味
htonl(値)	uint32_t の値をホストバイトオーダーからネットワークバイトオーダーに変換する (Host to Network Long)
htons(値)	uint16_t の値をホストバイトオーダーからネットワークバイトオーダーに変換する (Host to Network Short)
ntohl(値)	uint32_t の値をネットワークバイトオーダーからホストバイトオーダーに変換する (Network to Host Long)
ntohs(値)	uint16_t の値をネットワークバイトオーダーからホストバイトオーダーに変換する (Network to Host Short)

※ 「#include <arpa/inet.h>」が必要です。

制御コード (制御文字)

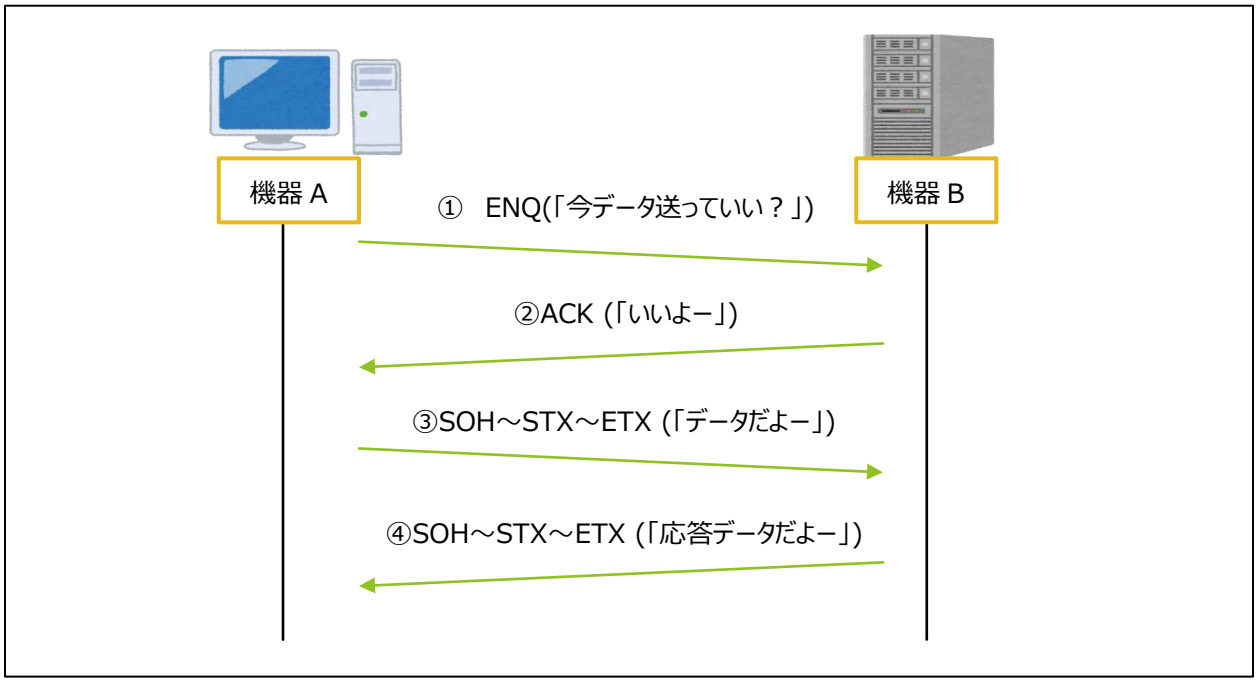
制御コードは、機器同士で円滑に通信を行うために決められた標準的なコードです。ほどんどの通信処理や通信仕様はこういった制御コードをうまく利用して組み立てられています。

主な制御コード

制御コード	値	意味
STX	0x02	電文(データ部または電文全体)の開始位置を示すコード
ETX	0x03	電文(データ部または電文全体)の終了位置を示すコード
ACK	0x06	「OK」「わかりました」のような、肯定する応答(返事)を示すコード
NAK	0x15	「NG」「だめです」のような、否定の応答(返事)を示すコード
SOH	0x01	電文ヘッダ部の開始位置を示すコード
ENQ	0x05	「(今からデータを送っても)大丈夫？」のような、問い合わせ用のコード
SYN	0x16	「同期を取ろう(歩調を合わせよう)」のような、掛け声用のコード (TCP などで使用する)

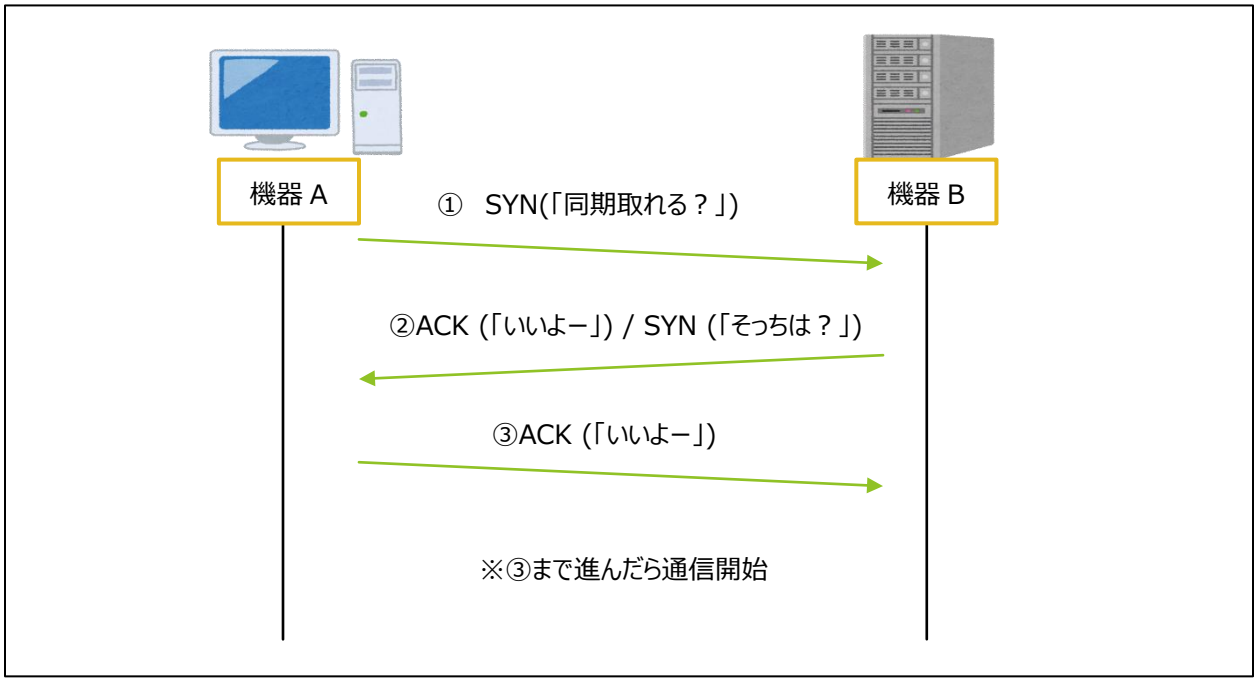
制御コードを使用したやり取りの例

例: 確認を取りながらデータを送り合う



②で何かトラブルがあった場合などは 機器 B は NAK (「だめです」) を返し、機器 A は少し待つなどします。

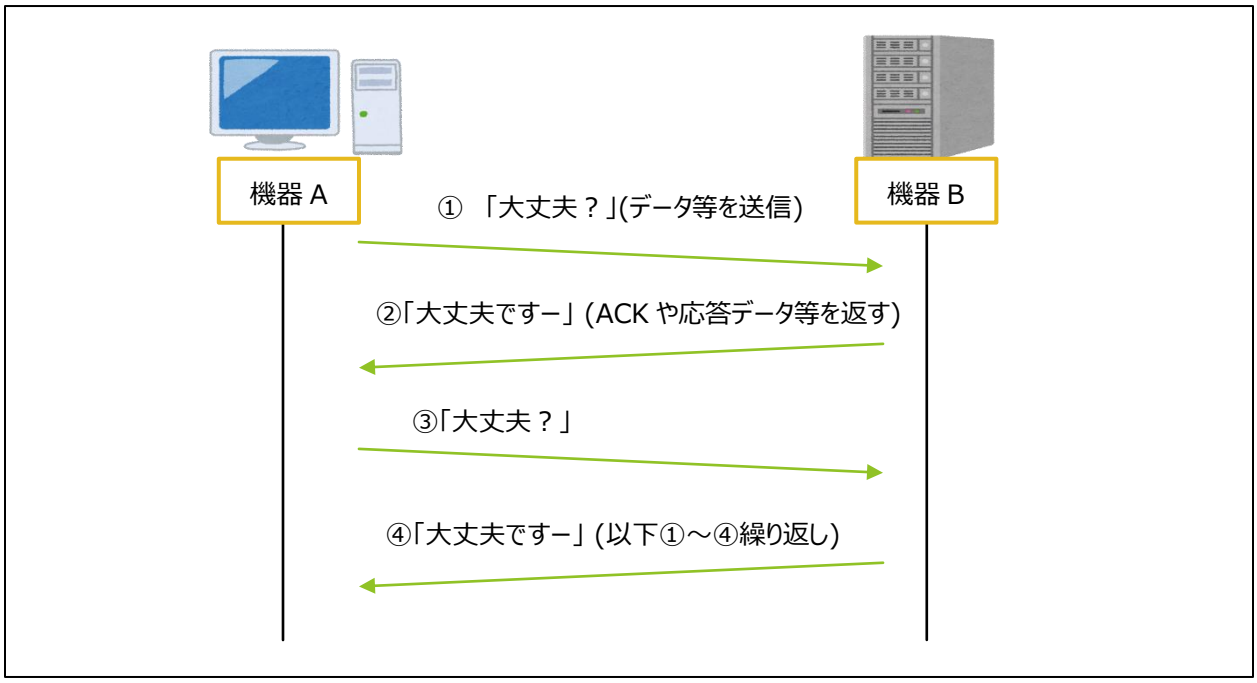
例: TCP 通信の最初に行われるやり取り (3 ウェイハンドシェイク)



ヘルスチェック

「ヘルスチェック」は、他の機器が正常に通信できるかを(主に定期的に)確認することです。「ヘルスチェック ＝ 健康状態をチェックする」という意味です。「死活監視」(生きているか死んでいるか確認する)と呼ばれることもあります。

例

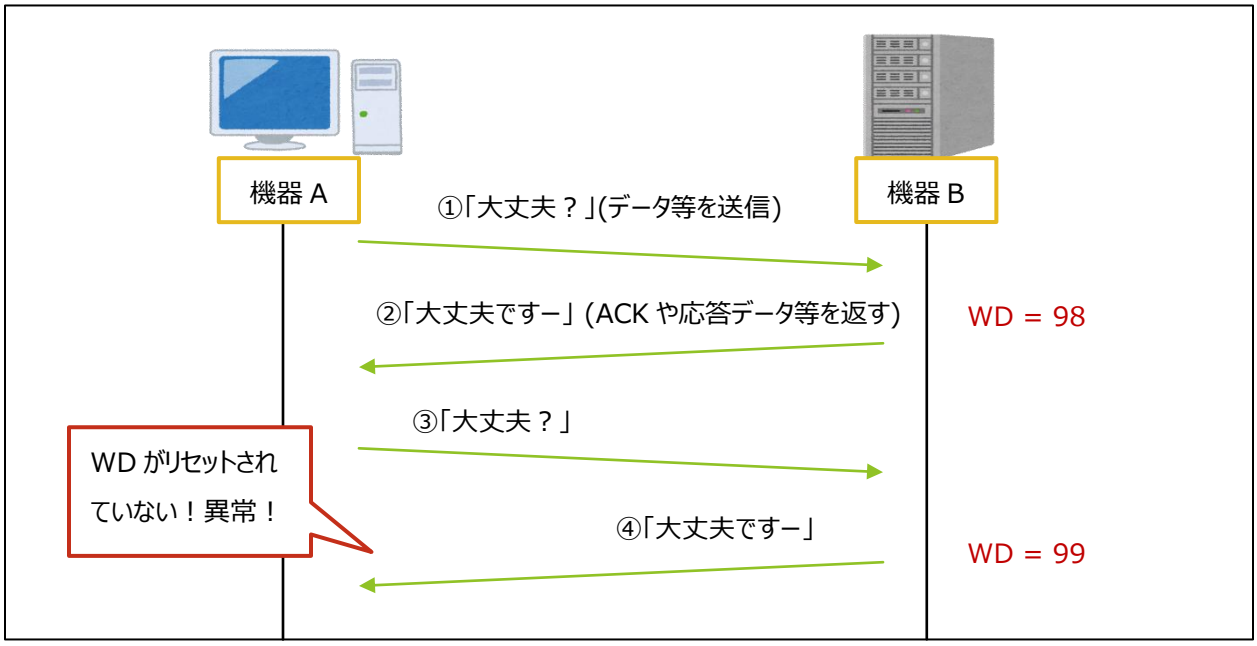


ヘルスチェック(＋異常時の通知)の仕組みをシステムに入れておくと、「いざ重要な通信データを送ろうと思ったら相手先の機器が故障していて通信ができなかった」のようなトラブルを未然に防ぐ事ができます。

ウォッチドッグ (WD)

「ウォッチドッグ」はヘルスチェックに使用するデータのひとつで、異常判定や死活判定をする材料として使用します。「通信データは来ているけど正常な動作をしていないかもしれない」という時、通信データ内などにウォッチドッグの仕組みを入れて正常かどうかを確認します。ウォッチドッグの仕組みには、各システムの仕様によってバリエーションがある場合があります。

例 1: ウォッチドッグデータが一定の値を超えたら、異常とみなす (きちんとリセットできていなかったら異常)



例 2: ウォッチドッグデータが一定時間更新されていなかったら、異常とみなす (値を増やせていなかったら異常)