ダウンロード・データあります

組み込みRustのライブラリ 便利クレート探偵団

第9回 ビット・フラグの扱いを簡単にするbitflags

中林 智之

リスト1 定義したビット・フラグに対して論理演算を行う例

```
use bitflags::{bitflags};

bitflags! {
    #[derive(Clone, Copy, Debug, PartialEq, Eq,
             Hash)]
    pub struct Flags: u8 {
        const A = 0b00000001;
        const B = 0b00000010;
        const C = 0b00000100;
    }
}

#[test]
fn test_logical_operations() {
    let ac = Flags::A | Flags::C;
    let bc = Flags::B | Flags::C;
    // OR
    assert_eq!((ac | bc), Flags::A | Flags::B
                                   | Flags::C);
    // AND
    assert_eq!((ac & bc), Flags::C);
    // NOT
    assert_eq!(!ac, Flags::B);
    // XOR
    assert_eq!((ac ^ bc), Flags::A | Flags::B);
    // (ac & !bc) と同じ
    assert_eq!((ac - bc), Flags::A);
}
```

（3つのビット・フラグを定義）

組み込みで使えるプログラミング言語として注目されているRustの組み込み開発で役立つライブラリ（クレート）を本連載では紹介します．前回から引き続き，Rustでビット・レベルの操作を行うのに便利なクレートを紹介します．今回はビット・フラグの扱いを簡単にするbitflags[1]です．

プログラム一式はダウンロード・ページからダウンロードできます．

https://www.cqpub.co.jp/interface/download/contents2025.htm

基本的な使い方

組み込み開発では，割り込みフラグや周辺機器のレジスタから読み出したフラグ，通信パケットのフラグなど，さまざまなビット・フラグを使用する機会があります．bitflagsはそのようなビット・フラグを簡単に扱えるようにしてくれます．

まずはbitflags!マクロでフラグを管理する構造体を定義します．このとき#[derive]でCopyやDebug，PartialEqを実装しておくと定義した構造体でコピーや比較ができるようになります．

リスト1の例では，A，B，Cという3つのビット・フラグを定義しています．test_logical_operations()では，bitflags!マクロで定義したビット・フラグを使って論理演算をしています．OR，AND，NOT，XORに加えてフラグの引き算（&!相当の演算）も可能です．

フラグ状態の確認や操作に便利…さまざまなメソッドが使える

論理演算以外で使えるさまざまなメソッドを**リスト2**に示します．フラグが1つも立っていない状態を調べたり，フラグのビット表現を返したり，定義した全てのフラグが立っている状態を調べたりできます．コメントを付けたので，どれも直感的に理解できます．

立っているフラグだけを順番に処理（イテレート）することもできます．**リスト3**のようにforループでmatchを使えば，立っているフラグごとの処理をコンパクトに記述できます．

文字列からの変換/文字列への変換

少し面白い機能として，文字列からの変換と文字列への変換機能があります（**リスト4**）．

parser::from_str()に文字列を渡すとフラグ管理構造体の値が得られます．"A"のような単純なフラグだけでなく，"A | B"のような論理演算や，"0x01"のように16進数の表現を書くこともできます．

逆にto_writer()ではフラグ管理構造体の値か

第4回 メモリ使用量や最悪実行時間が見積もれるheapless（2024年7月号）
第5回 heaplessによるキューとスマート・ポインタ（2024年9月号）
第6回 任意のデータをシリアライズ/デシリアライズするserde（2024年10月号）