第4章 アクセス回数をカウントするサーバ作りを例に

PythonとRustで比較するマルチスレッド処理

神戸 淳

マルチスレッド処理でデータの整合性を保つことは難しい

● マルチスレッド処理の問題がRustで解決できる

昨今のプログラム開発においてはシングル・スレッドで動作するプログラムというものは珍しく，高速化などを目的にマルチスレッドで動作させることがほとんどだと思います．

マルチスレッドのプログラムは効率的な動作が可能な一方，複数のスレッドからの競合する操作によりデータが本来望んでいない状態になってしまう，不整合を起こしてしまう可能性があることに注意しなければいけません．

データの整合性を保つことはシステムの完全性にも直結しセキュリティを高めることにもつながるなど，マルチスレッドで動作するプログラムにおいて不整合からデータを守ることは決してプログラミングの基本というだけにとどまらない重要な問題です．

マルチスレッドでデータの操作が競合を起こさないかを完全に把握するのはとても難しいことです．その点Rustには所有権という仕組みがあります．これはRustの特徴の1つとしてよく挙げられますが，その所有権がデータの整合性を保つ上で強みになるのです．

● ウェブ・サーバ開発にもRustは適している

マルチスレッドで動作する環境としては組み込み系だけでなくウェブ・サーバも該当しています．ウェブ系の開発を主に行っている筆者の肌感ではありますが，その分野でも最近Rustの話題を聞くようになっています．

ウェブ・サーバ開発においてRustを選択する理由の1つとして，データの整合性を保ちやすいことが挙げられるでしょう．実際筆者は現在ウェブ・サーバの開発でRustを使っていますが，プログラムを書いてコンパイルさえできれば少なくともRustプロセスのメモリ上ではデータの不整合が起きないことが担保されているという安心感があります．

そこで実際にRustをウェブ・サーバに使用し，マルチスレッドでデータ操作の競合が起こりやすいプログラムを書いてみます．そこで現在ウェブ開発で広く使われているPythonと比較しながらデータの不整合が起こらないことを確認してみましょう．

掲載したプログラムはRust/Pythonそれぞれ標準のインストール方法でインストールしたものだけで動きます．標準ライブラリのみの使用なので依存を取ってくる必要はありません．

なお筆者のバージョンはRust 1.63.0，Python 3.10.8です．

シンプルなマルチスレッド・サーバをPythonとRustで作る

まずは比較対象となるPythonでHTTPリクエストに対してHello Worldを返すだけのシンプルなサーバを作成していきます．任意の場所に以下のPythonプログラムを配置します．

● Python版で試す

リスト1(a)はPython版のサーバです．

```
$ python simple-server.py↵
```

でサーバを起動し，ウェブ・ブラウザで`http://127.0.0.1:8080`へアクセスします．ブラウザの画面にHello, world.の文字が表示されれば成功です．

● Rust版で試す

次にRustで同じ機能を持ったサーバを書きます．任意の場所に，

```
$ cargo new simple-server↵
```

などの名前で新しいRustのプロジェクトを作成し，**リスト1**(b)のプログラムを`src/main.rs`として配置します．

```
$ cargo run src/main.rs↵
```

でサーバを起動し，ウェブ・ブラウザで`http://127.0.0.1:7878`へアクセスします．Hello, world.の文字が表示されれば成功です．

以上でPythonとRustでのシンプルなマルチスレッ