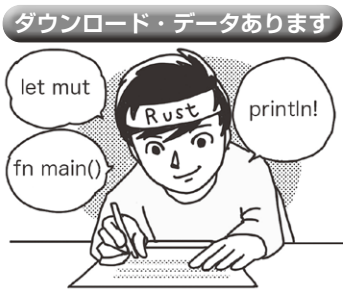
Rustプログラミング問題集



最終回

第6回 はまりがちな落とし穴

中村 仁昭，懸川 岳

本連載ではRustを使った問題と回答形式でRustを学んでいきます．今回は最終回です．連載のまとめとして，Rustを使っていると遭遇する落とし穴について見ていきます．（編集部）

1 ライフタイムの落とし穴

リスト1（a）はnum1とnum2の値を比較して大きい方を標準出力するようなソースコードですが，誤りを含みます．問題のある箇所を見つけて修正してください．

● 回答

回答例を**リスト1**（b）に示します．問題のある箇所は，larger関数が返す参照のライフタイムが変数resultのライフタイムよりも短い点です．そのためresult変数の定義とprintln!を内側のスコープで実行することで解決できます．

またlarger関数定義にある'aのような表記はライフライム注釈と呼ばれるものです．これによりlarger関数は全ての引数と返り値が同じライフタイムを持つことを指定しています．

リスト1　num1とnum2の値を比較して大きい方を標準出力する

```
fn larger<'a>(x: &'a i32, y: &'a i32) -> &'a i32 {
    if x > y {
        x
    } else {
        y
    }
}
fn main() {
    let result;
    {
        let num1: i32 = 12;
        let num2: i32 = 23;
        result = larger(&num1, &num2);
    }
    println!("larger number is: {}", result);
}
```

（a）問題…誤りのある箇所を見つける

```
fn larger<'a>(x: &'a i32, y: &'a i32) -> &'a i32 {
    if x > y {
        x
    } else {
        y
    }
}
fn main() {
    {
        let result;
        let num1: i32 = 12;
        let num2: i32 = 23;
        result = larger(&num1, &num2);
        println!("larger number is: {}", result);
    }
}
```

（b）回答例

2 文字列出力の落とし穴

リスト2（a）は文字列helloを標準出力するソースコードです．動作上問題はありませんが，非効率な処理を含んでいます．その箇所を修正してください．

● 回答

回答例を**リスト2**（b）に示します．問題のある箇所はcloneを使って文字列のコピーを作っている点です．cloneを使用するとデータの新しいインスタンスを作成するため，メモリ使用量も大きくなります．今回の場合は標準出力するのみなので，コピーする代わりに参照を使用すると良いでしょう．

リスト2　文字列helloを標準出力する

```
fn main() {
    let s1 = String::from("hello");
    let s2 = s1.clone();
    println!("s1 = {}, s2 = {}", s1, s2);
}
```

（a）問題…非効率な処理を見つける

```
fn main() {
    let s1 = String::from("hello");
    let s2 = &s1;
    println!("s1 = {}, s2 = {}", s1, s2);
}
```

（b）回答例

rustup（https://www.rust-lang.org/ja/tools/install）でインストールした標準的な環境で動作確認しています．ウェブ・ブラウザの実行環境Rust Playground（https://play.rust-lang.org/）でも試せます．