Rustに触れる３

ここでは変数の型についてより詳しく解説します。

配列

```
fn main()
{
    let int_hairetu = [1, 3, 5];

    println!("{},{},{}", int_hairetu[0],int_hairetu[1],int_hairetu[2]);
}
```

これは各値を指定する場合の宣言で、すべての値を０にして宣言する場合には以下のようにします。

```
fn main()
{
    let buffer = [0_u8; 256];
}
```

０とは言いましたが０は０でも色々な型の０があるので***u8***（正の整数８bit）の０として要素数256の配列を宣言してみました。

構造体

構造体とは複数の変数を格納しひとまとめにしたもので、Rustではさらにその構造体に専用の関数を定義することが可能です。

まずは基本的な構造体について見ていきましょう。

```
fn main()
{
    let info = Person {age : 18, favorite_num : 3.14};

    println!("Age:{},Favorite Number:{}", info.age, info.favorite_num);
}

struct Person
{
    age : i32,
    favorite_num : f32
}
```

インプリメント

ここでRustで私が最も好きな部分に触れていきます。C言語ではぎりClassに相当するのかなと勝手に思っているものです。

まずは実際のコードを見ていきましょう。

```
fn main()
{
    let mut new_person = Person::new(18, 3.14);

    new_person.info();

    // 「18,3.14」と出力される

    new_person.age = 36;

    new_person.info();

    // 「36,3,14」と出力される
}

struct Person
{
    age : i32,
    favorite_num : f32
}

impl Person
{
    pub fn new(age_ : i32, favorite_number_ : f32)->Person
    {
        Person {age : age_, favorite_num : favorite_number_}
    }

    fn info(&self)
    {
        println!("{},{}", self.age, self.favorite_num);
    }
}
```

ここは重要なところなのでじっくり解説します。まず**インプリメント**とは構造体に対して「振る舞い」として関数を実装するものです。上記コードでは２つの関数をインプリメントしてみました。

インプリメントで構造体に付与する関数は**パブリック**な関数と**プライベート**な関数に分類されます。 ここでいうと**new**関数がパブリックな関数で**info**という関数がプライベートな関数です。

ここでいうパブリックとは**構造体自身じゃなくてもアクセス可能な関数**のことでプライベートとは**構造体自身のみがアクセス可能な関数である**ということです。

これらの違いはメイン関数のはじめのように外部からアクセスする関数であるか、その次のように構造体自体からアクセスするかというところにあります。

Rustではこのように構造体を新規作成する関数を構造体に対してインプリメントするという習慣があります。そのような関数は構造体自身が触れるわけではないため***pub***を関数の先頭につけることによって外部アクセス可能な関数として宣言しているんですね。

そしてプライベートな関数として宣言されている***info***です。こちらは引数に***self***と書いてあります。これは関数の内容で構造体のメンバ変数を取り扱いたいときに用います。今回では構造体自身のメンバ変数である***age***と***favorite_num***を出力したいために***self***が使われています。

selfに&がついている理由ですが、Rustでは関数に変数を入力するときその変数が持つ権利を関数へと譲渡します。するとそれ以降、この構造体で関数を使ったり値を参照したりすることができなくなってしまうためポインタを渡すことで権利ではなく値を参照するだけにしているのです。

インプリメント関数には以下のような場合もあります。

```
fn main()
{
    let mut new_person = Person::new(18, 3.14);

    new_person.info();
    // 「18,3.14」と表示される

    new_person.add_age();
    // ここでageに＋１される

    new_person.info();
    // 「19,3.14」と表示される
}

struct Person
{
    age : i32,
    favorite_num : f32
}

impl Person
{
    pub fn new(age_ : i32, favorite_number_ : f32)->Person
    {
        Person {age : age_, favorite_num : favorite_number_}
    }

    fn info(&self)
    {
        println!("{},{}", self.age, self.favorite_num);
    }

    fn add_age(&mut self)
    {
        self.age += 1;
    }
}
```

今回は***add_age***という関数を追加してみました。Personに含まれる人の年齢の情報を１年増加させる関数です。

これらを実行するとはじめの***info***関数では１８歳と出力されるのに対して***add_age***関数実行後は１９歳と出力されるため関数によって構造体のメンバ変数に対して変更を加えることができていることを確認することができます。

ここでRustの変数作成のときの決まりを思い出してください。可変可能な変数を作成するときには以下のように***mut***を追加するというきまりがありました。

```
let mut value = 3.14;
```

この決まりに従って構造体を新規作成するときにも***mut***キーワードがついています。

さらに、***add_age***関数の引数にも***mut***がついています。これは構造体関数において特によく見られる使い方です。

可変可能な構造体自身を引数にしているため***add_age***関数では構造体のメンバ変数を変えることができていたんですね。

難しかったのでまとめると

- 構造体にはインプリメントによって関数を実装できる
- **self**を引数に用いることによって構造体自身のメンバにアクセスする関数を作成できる
- **mut self**なら関数実行によって構造体のメンバ変数を変更できる
- **pub**をつけた**new**関数によって構造体の新規作成を見やすくしよう