Seven Years Later…

Generic Associated Types in Rust

Tim Hegemann

17. Januar 2024

Lightning Talks

Generics

Iterator

```
trait Iterator<T> {
  fn next(&mut self) -> Option<T>;
}
```

Iterator

```
trait Iterator<T> {
  fn next(&mut self) -> Option<T>;
}

struct Iter<'c, T> {
  data: &'c [T],
}
```

Iterator

```
trait Iterator<T> {
  fn next(&mut self) -> Option<T>;
}

struct Iter<'c, T> {
  data: &'c [T],
}

impl<'c, T> Iterator<&'c T> for Iter<'c, T> {
  fn next(&mut self) -> Option<&'c T> {
    if let Some((prefix_elem, suffix)) = self.data.split_first() {
      self.data = suffix;
      Some(prefix_elem)
    } else {
      None
    }
  }
}
```

Count

```
fn count<T>(mut iter: impl Iterator<T>) -> usize {
  let mut cnt = 0;
  while let Some(_) = iter.next() {
    cnt += 1;
  }
  cnt
}
```

Count

```
fn count<T>(mut iter: impl Iterator<T>) -> usize {
  let mut cnt = 0;
  while let Some(_) = iter.next() {
    cnt += 1;
  }
  cnt
}

fn main() {
  let xs = vec![1, 2, 3, 4, 5];
  println!("number of items: {}", count(Iter { data: &xs }));
}
```

Count

```
fn count<T>(mut iter: impl Iterator<T>) -> usize {
  let mut cnt = 0;
  while let Some(_) = iter.next() {
    cnt += 1;
  }
  cnt
}

fn main() {
  let xs = vec![1, 2, 3, 4, 5];
  println!("number of items: {}", count(Iter { data: &xs }));
}
```

```
> number of items: 5
```

Associated Types

Iterator 2.0

```
trait Iterator {
  type Item;
  fn next(&mut self) -> Option<Self::Item>;
}
```

Iterator 2.0

```
trait Iterator {
  type Item;
  fn next(&mut self) -> Option<Self::Item>;
}

struct Iter<'c, T> {
  data: &'c [T],
}

impl<'c, T> Iterator for Iter<'c, T> {
  type Item = &'c T;
  fn next(&mut self) -> Option<Self::Item> {
    // ...
  }
}
```

Count 2.0

```
fn count(mut iter: impl Iterator) -> usize {
  let mut cnt = 0;
  while let Some(_) = iter.next() {
    cnt += 1;
  }
  cnt
}
```

Interlude

Windows

```
let xs = vec![1, 2, 3, 4, 5];

for win in xs.windows(3) {
  println!("{:?}", win);
}
```

Windows

```
let xs = vec![1, 2, 3, 4, 5];

for win in xs.windows(3) {
  println!("{:?}", win);
}
```

```
> [1, 2, 3]
> [2, 3, 4]
> [3, 4, 5]
```

Windows

```
let xs = vec![3.1, 4.1, 5.9, 2.6, 5.3];

for win in xs.windows(3) {
  if let [a, b, c] = win {
    println!("{}", (a + b + c) / 3.0)
  }
}
```

Windows

```
let xs = vec![3.1, 4.1, 5.9, 2.6, 5.3];

for win in xs.windows(3) {
  if let [a, b, c] = win {
    println!("{}", (a + b + c) / 3.0)
  }
}
```

```
> 4.366666666666666
> 4.2
> 4.6000000000000005
```

Write Back

```
for win in xs.windows(3) {
  if let [a, b, c] = win {
    *a = (a + b + c) / 3.0;
  }
}
```

Write Back

```
for win in xs.windows(3) {
  if let [a, b, c] = win {
    *a = (a + b + c) / 3.0;
  }
}
```

```
error[E0594]: cannot assign to `*a`, which is behind a `&` reference
 --> src/main.rs:6:13
  |
6 |             *a = (a + b + c) / 3.0;
  |             ^^^^^^^^^^^^^^^^^^^^^^ `a` is a `&` reference, so the data
  ↪   it refers to cannot be written
```

Write Back

```
for win in xs.windows_mut(3) {
  if let [a, b, c] = win {
    *a = (a + b + c) / 3.0;
  }
}
```

Write Back

```
for win in xs.windows_mut(3) {
  if let [a, b, c] = win {
    *a = (a + b + c) / 3.0;
  }
}
```

```
error[E0599]: no method named `windows_mut` found for struct
↪  `Vec<{float}>` in the current scope
 --> src/main.rs:4:19
  |
4 |     for win in xs.windows_mut(3) {
  |                   ^^^^^^^^^^^ help: there is a method with a similar
  ↪   name: `windows`
```

How Hard Can It Be?

How Hard Can It Be?

```
struct WindowsMut<'t, T> {
  slice: &'t mut [T],
  start: usize,
  window_size: usize,
}
```

How Hard Can It Be?

```
struct WindowsMut<'t, T> {
  slice: &'t mut [T],
  start: usize,
  window_size: usize,
}

impl<'t, T> Iterator for WindowsMut<'t, T> {
  type Item = &'t mut [T];

  fn next(&mut self) -> Option<Self::Item> {
    let res =
        self.slice[self.start..].get_mut(..self.window_size)?;
    self.start += 1;
    Some(res)
  }
}
```

How Hard Can It Be?

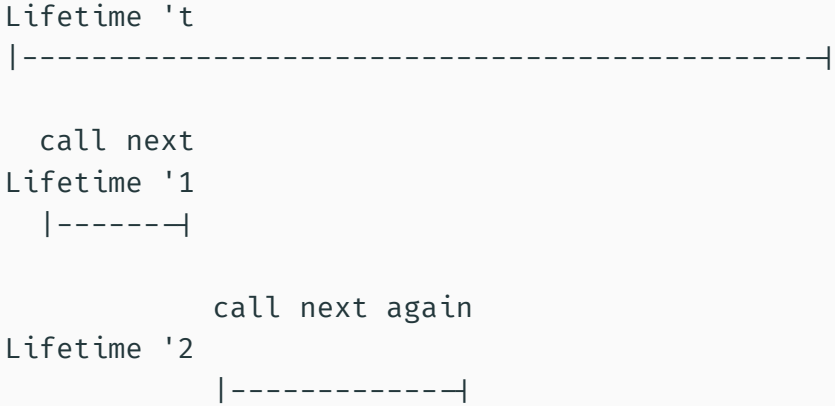
```
let mut xs = vec![3.1, 4.1, 5.9, 2.6, 5.3];

let mut xs_windows_mut = WindowsMut {
    slice: &mut xs,
    start: 0,
    window_size: 3,
};

for win in xs_windows_mut {
    if let [a, b, c] = win {
        *a = (*a + *b + *c) / 3.0;
    }
}
```

Does It Compile?

Does It Compile?

```
error: lifetime may not live long enough
  --> src/main.rs:14:9
   |
8  | impl<'t, T> Iterator for WindowsMut<'t, T> {
   |      -- lifetime `'t` defined here
...
11 |     fn next(&mut self) -> Option<Self::Item> {
   |             - let's call the lifetime of this reference `'1`
...
14 |         Some(retval)
   |         ^^^^^^^^^^^^ method was supposed to return data with lifetime
   ↪  `'t` but it is returning data with lifetime `'1`
```

Does It Compile?



Generic Associated Types

Lending Iterator

```
trait LendingIterator {
  type Item<'a> where Self: 'a;
  fn next<'a>(&'a mut self) -> Option<Self::Item<'a>>;
}
```

Lending Iterator

```
trait LendingIterator {
  type Item<'a> where Self: 'a;
  fn next<'a>(&'a mut self) -> Option<Self::Item<'a>>;
}

impl<'t, T> LendingIterator for WindowsMut<'t, T> {
  type Item<'a> = &'a mut [T] where Self: 'a;

  fn next<'a>(&'a mut self) -> Option<Self::Item<'a>> {
    let res =
        self.slice[self.start..].get_mut(..self.window_size)?;
    self.start += 1;
    Some(res)
  }
}
```

Lending Iterator

```
let mut xs = vec![3.1, 4.1, 5.9, 2.6, 5.3];

let mut xs_windows_mut = WindowsMut {
  slice: &mut xs,
  start: 0,
  window_size: 3,
};

while let Some([a, b, c]) = xs_windows_mut.next() {
  *a = (*a + *b + *c) / 3.0;
}
```

Lending Iterator

```
let mut xs = vec![3.1, 4.1, 5.9, 2.6, 5.3];

let mut xs_windows_mut = WindowsMut {
  slice: &mut xs,
  start: 0,
  window_size: 3,
};

while let Some([a, b, c]) = xs_windows_mut.next() {
  *a = (*a + *b + *c) / 3.0;
}

println!("{:?}", xs);
```

```
> [4.366666666666666, 4.2, 4.6000000000000005, 2.6, 5.3]
```