**Rust tip and trick — Borrow**

[[](https://medium.com/@techhara?source=post_page-----37c8b0426a04--------------------------------)](https://medium.com/@techhara?source=post_page-----37c8b0426a04--------------------------------)

[TechHara](https://medium.com/@techhara?source=post_page-----37c8b0426a04--------------------------------)

·

Follow

4 min read

·

Jun 26

8

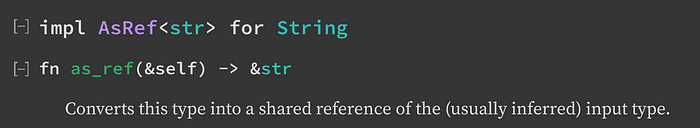
In this blog post, we will explore Rust trait Borrow. We will see how this differs from a similar cousin AsRef trait.

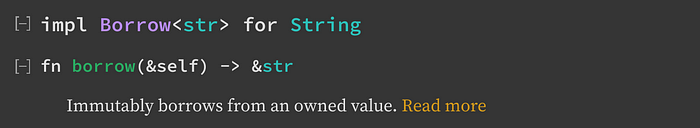
**What are Borrow and AsRef?**

Borrow and AsRef are both traits that allow us to take reference as some other type. For example, suppose we have a String. We can use Borrow or AsRef to get a reference to a str.

let s = String::from("Hello");  
let s\_ref: &str = s.borrow(); // using Borrow  
let s\_ref: &str = s.as\_ref(); // using AsRef

This is possible because String struct implements both Borrow<str> and AsRef<str>.





Looking at the function prototype, they look identical, returning &str type. In fact, they produce the same values too (Rust playground [link](https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&code=use+std%3A%3Aborrow%3A%3ABorrow%3B%0A%0Afn+main%28%29+%7B%0A++++let+s+%3D+String%3A%3Afrom%28%22Hello%22%29%3B%0A++++let+s1%3A+%26str+%3D+s.borrow%28%29%3B+%2F%2F+using+Borrow%0A++++let+s2%3A+%26str+%3D+s.as_ref%28%29%3B+%2F%2F+using+AsRef%0A++++assert_eq%21%28s1%2C+s2%29%3B%0A%7D))

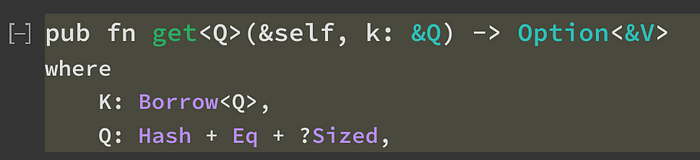
use std::borrow::Borrow;  
  
fn main() {  
 let s = String::from("Hello");  
 let s1: &str = s.borrow(); // using Borrow  
 let s2: &str = s.as\_ref(); // using AsRef  
 assert\_eq!(s1, s2);  
}

So, the question is, how do they differ and when to use Borrow?

**When to use Borrow?**

Borrow is a trait that is intended to be used for types that have a *canonical* way of borrowing as another type. A typical use case for Borrow is when you have a data that wraps around some internal type T for added functionalities. For example, String adds the functionality of mutability to str, so it is a superset of str, in terms of its features. In other words, String can do everything immutable that str can do. Similarly, Vec<T> is essentially a more flexible version of [T], so it again can do everything immutable that [T] can do, and so forth.

**One requirement for implementing Borrow is that Hash, Eq and Ord for a borrowed value are equivalent to those of the owned value**. This is the key difference from AsRef trait. If some data structure implements Borrow trait for T, then it must behave equivalently to T. Wit this requirement, the standard HashMap uses Borrow trait for get() and get\_mut() methods.



Note that K is the key data type, which implements Borrow trait for Q. A typical type for K would be String, whereas a typical type for Q would be str. This means that we can pass any type Q that can be borrowed as a K. In our example, both String and &str implement Borrow<str>, so they can be used as keys. (Rust playground [link](https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&code=use+std%3A%3Acollections%3A%3AHashMap%3B%0A%0Afn+main%28%29+%7B%0A++++let+mut+map+%3D+HashMap%3A%3Anew%28%29%3B%0A++++let+key+%3D+String%3A%3Afrom%28%22key%22%29%3B%0A++++map.insert%28key.clone%28%29%2C+42%29%3B%0A++++%0A++++%2F%2F+we+can+use+a+%26String+or+a+%26str+to+get+the+value%0A++++assert_eq%21%28map.get%28%22key%22%29%2C+Some%28%2642%29%29%3B+%2F%2F+%26str%0A++++assert_eq%21%28map.get%28%26key%29%2C+Some%28%2642%29%29%3B+%2F%2F+%26String%0A%7D))

use std::collections::HashMap;  
  
fn main() {  
 let mut map = HashMap::new();  
 let key = String::from("key");  
 map.insert(key.clone(), 42);  
   
 // we can use a &String or a &str to get the value  
 assert\_eq!(map.get("key"), Some(&42)); // &str  
 assert\_eq!(map.get(&key), Some(&42)); // &String  
}

There is a second key difference between Borrow and AsRef —**Borrow has a blanket implementation for any type T.** That is, you don’t need to trivially implement Borrow<T> for &T. This is more serious than it sounds like. Consider the following code (playground [link](https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&code=struct+MyStruct+%7B%0A++++val%3A+i64%2C%0A%7D%0A%0Aimpl+AsRef%3Ci64%3E+for+MyStruct+%7B%0A++++fn+as_ref%28%26self%29+-%3E+%26i64+%7B%0A++++++++%26self.val%0A++++%7D%0A%7D%0A%0Afn+print_i64%28x%3A+impl+AsRef%3Ci64%3E%29+%7B%0A++++println%21%28%22%7B%7D%22%2C+x.as_ref%28%29%29%3B++++%0A%7D%0A%0Afn+main%28%29+%7B%0A++++let+x+%3D+MyStruct+%7B+val%3A+314+%7D%3B%0A++++print_i64%28%261%29%3B%0A++++print_i64%28x%29%3B%0A%7D))

struct MyStruct {  
 val: i64,  
}  
  
impl AsRef<i64> for MyStruct {  
 fn as\_ref(&self) -> &i64 {  
 &self.val  
 }  
}  
  
fn print\_i64(x: impl AsRef<i64>) {  
 println!("{}", x.as\_ref());   
}  
  
fn main() {  
 let x = MyStruct { val: 314 };  
 print\_i64(&1);  
 print\_i64(x);  
}

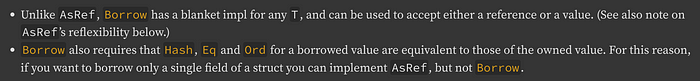
We want to be able to pass a direct reference of the number type i64 as well as any type that can behave like a reference to i64. Unfortunately, this code won’t compile, because, the trivial AsRef<i64> for &i64 is not implemented.

error[E0277]: the trait bound `{integer}: AsRef<i64>` is not satisfied  
 --> src/main.rs:17:16  
 |  
17 | print\_i64(&1);  
 | --------- ^ the trait `AsRef<i64>` is not implemented for `{integer}`  
 | |  
 | required by a bound introduced by this call  
 |  
 = note: required for `&{integer}` to implement `AsRef<i64>`  
note: required by a bound in `print\_i64`  
 --> src/main.rs:11:22  
 |  
11 | fn print\_i64(x: impl AsRef<i64>) {  
 | ^^^^^^^^^^ required by this bound in `print\_i64`

On the other hand, Borrow works right out of the box.

use std::borrow::Borrow;  
  
struct MyStruct {  
 val: i64,  
}  
  
impl Borrow<i64> for MyStruct {  
 fn borrow(&self) -> &i64 {  
 &self.val  
 }  
}  
  
fn print\_i64(x: impl Borrow<i64>) {  
 println!("{}", x.borrow());  
}  
  
fn main() {  
 let x = MyStruct { val: 314 };  
 print\_i64(&1);  
 print\_i64(x);  
}

**Summary**



<https://doc.rust-lang.org/std/convert/trait.AsRef.html#relation-to-borrow>

[Rust](https://medium.com/tag/rust?source=post_page-----37c8b0426a04---------------rust-----------------)

[Borrow](https://medium.com/tag/borrow?source=post_page-----37c8b0426a04---------------borrow-----------------)