26生命周期制约问题

2020年5月9日 23:04

```
usestd::fmt::Debug;

Fn main(){
Let wawa=Wawa{
 age:"hades".to_string(),
};

Let name=Name{
name:&wawa,
};

sub(&name);
}



Struct Name<'a>{
name:&'aWawa,
}

#[derive(Debug)]
Struct Wawa{
age:String,
}

Trait Hello{
Type Out;
fnhello(&self)->Self::Out;
}

impl<'a> Hello for &Name<'a>{
Type Out=&'aWawa;
fnhello(&self)->Self::Out{
self.name
}
}

fnsub<T>(s:T)
where
T:Hello+'static,
<T as Hello>::Out:std::fmt::Debug,
{
println!("{:?}",s.hello());
}
```

可以看到编译器的抱怨:

name的引用不是'static的,那么成员的wawa的引用不是'static的.

如果

```
impl<'a> Hello for Name<'a>{
Type Out=&'aWawa;
fnhello(&self)->Self::Out{
self.name
}
}
```

sub(name);编译器会知道name是一个类型的所有权转移到了这里,只会抱怨wawa的引用不是static的了.

```
T:Hello+'static,
```

值得是结构体T本身满足约束Hello, 结构体内部的成员需要满足'static约束,

如果T是一个引用的话,也要满足'static约束.

因为 T 可以是任意类型，T 自身也可能是一个引用，或者是一个存放了一个或多个引用的类型，而他们各自可能有着不同的生命周期。Rust 不能确认 T 会与 'a 存活的一样久，现在代码可以编译了，因为 T: 'a 语法指定了 T 可以为任意类型，不过如果它包含任何引用的话，其生命周期必须至少与 'a 一样长。

我们可以选择不同的方法来解决这个问题，如示例 19-18 中 StaticRef 的结构体定义所示，通过在 T 上增加 'static 生命周期约束。这意味着如果 T 包含任何引用，他们必须有 'static 生命周期.

```
S:Fn(Task<T>)+Send+Sync+'static;
```

闭包的约束,Send + Sync +'static指的是闭包结构体内部捕获的成员满足的约束.

最后一个是trait对象的生命周期约束:

```
fnmain(){
letn=Nihao{name:&"hades".to_string()};
n.hello();
nihao(Box::new(n)asBox<dynWomen>);
}

structNihao<'a>{
name:&'aString,
}

traitWomen{
```

```
fnhello(&self);
}

impl<'a>WomenforNihao<'a>{
fnhello(&self){
println!("{:?}",self.name);
}
}



fnnihao(a:Box<dynWomen+'static>){
a.hello();
}
```