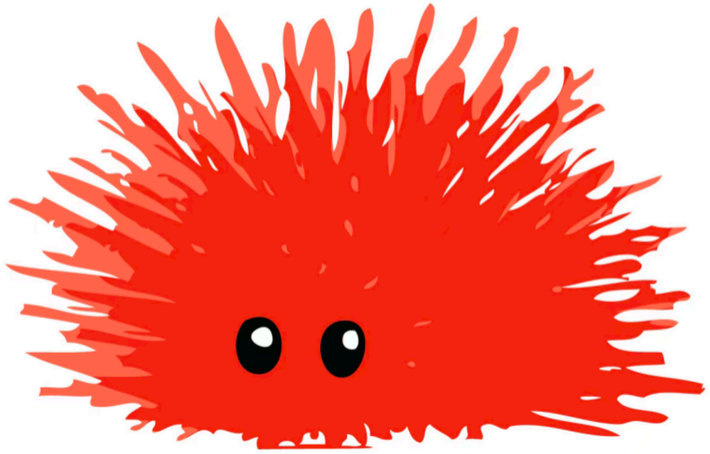


Rust Meetup @ BUPT

2021-04-18

Rust生命周期与变形

Lifetime and Variance of Rust

吴铭钞

About Me

吴铭钞

- 2017级大数据本科生 @ BUPT
- 研0 @ 软件研究所TCSE

- Rust菜狗 @ Brupst
- Github：Jason210314
- 对分布式系统/大数据系统感兴趣

今日内容

- 子类型与变形
- 变形和类型安全
- Rust 生命周期与变形

子类型和变形

Subtyping and Variance

子类型和变型

子类型

If S is a subtype of T, the subtyping relation is often written S <: T, to mean that any term of type S can be safely used in a context where a term of type T is expected.

> *Subtype Requirement*: Let $\phi(x)$ be a property provable about objects x of type T. Then $\phi(y)$ should be true for objects y of type S where S is a subtype of T.

子类型和变形

继承

- 在面向对象编程中，继承是对子类型最直接的体现
- Cat : Animal (Cat extends Animal)，那么 Cat <: Animal
- 由继承关系可以非常清楚地导出子类型关系

子类型和变形

类型构造器

- A type constructor is a feature of a typed formal language that builds new types from old ones

- T[], List<T>

- &‘a T ，Box<T>

- 如果 S <: T，那么 S[] <: T[] ???

子类型和变形

变形 Variance

设类型构造器是 F(T) ，S <: T

- F(S) <: F(T)，协变 Covariance
- F(T) <: F(S)，逆变 Contravariance
- F(T) 和 F(S) 无子类型关系，抗变/不变 Invariance

变形和类型安全

Variance And Type Safe

变形与类型安全

Java 数组协变

```java
class Animal {
    private int name;
}

class Cat extends Animal {
    public Cat() {
    }

    public void meow() {
        System.out.println("meow");
    }
}

class Dog extends Animal {
    public Dog() {
    }

    public void bark() {
        System.out.println("bark");
    }
}
```

编译时允许数组协变，
但是运行时如果进行错误的操作，则抛出异常
其实是很🌶️🐔的设计

```
Exception in thread "main" java.lang.ArrayStoreException: Dog
        at Main.main(Main.java:55)
```

```java
public static void main(String[] args) throws Exception {

    // covariant, compile ok
    Animal[] animals = new Animal[10];
    animals = new Cat[10];
    // runtime error
    animals[0] = new Dog();

}
```

变形与类型安全

Java 泛型容器抗变

编译阶段直接就挂啦⚠️，类型不匹配。

Java其实提供了 super/extend 通配符，可以让 coder 手动定义，
这里不多讲了。

```java
public static void main(String[] args) throws Exception {
    ArrayList<Animal> la = new ArrayList<Cat>();
}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
	Type mismatch: cannot convert from ArrayList<Cat> to ArrayList<Animal>

	at Main.main(Main.java:54)
```

变形与类型安全

变形规则设计不佳导致安全问题

如果 T[] 不是协变的，那么错误的赋值就不会在运行期发生。

```
public static void main(String[] args) throws Exception {

    // covariant, compile ok
    Animal[] animals = new Animal[10];
    animals = new Cat[10];
    // runtime error
    animals[0] = new Dog();

}
```

变形与类型安全

do some evil

一个邪恶的投喂者把🐱换成🐶，然后无辜的调用者让🐶学🐱叫?

完蛋!

万幸Java编译器阻止了这种行为，But Why?

```
static void evil_feed(Cage<Animal> cage) {
    cage.setInner(new Dog());
}

Run | Debug
public static void main(String[] args) throws Exception {
    Cage<Cat> cage = new Cage(new Cat());
    evil_feed(cage);
    cage.inner.meow();
}
```

变形与类型安全

类型构造器变形的依据

- 如果类型构造器对原始类型值只读。应该协变，把 Cage<Cat> 当作 Cage<Animal> 去读，任何时候都合理。
- 如果只写。肯定不能协变，把🐶换成🐱可太惨了。考虑逆变。任何需要 Cage<Cat> 的地方，都可以用 Cage<Animal> 替换，反正只会往里面扔一只🐶，我永远也不会当成具体动物去读它。极少见，一般在函数上体现。
- 可读可写。就剩抗变了，类型必须严格一致。

Rust生命周期和变形

Lifetime and Variance

Rust生命周期和变形

lifetime

- Lifetime 是一种Rust编译器用来保证借用有效的手段，形如 ‘a

  第一位同学讲到的该死的迭代器失效/悬垂引用

- 当编译器不能自动推断出生命周期时，需要coder手动标注。
- ‘a : ‘b 意味着’a至少和’b活得一样长
- 来看个🌰

Rust生命周期和变形

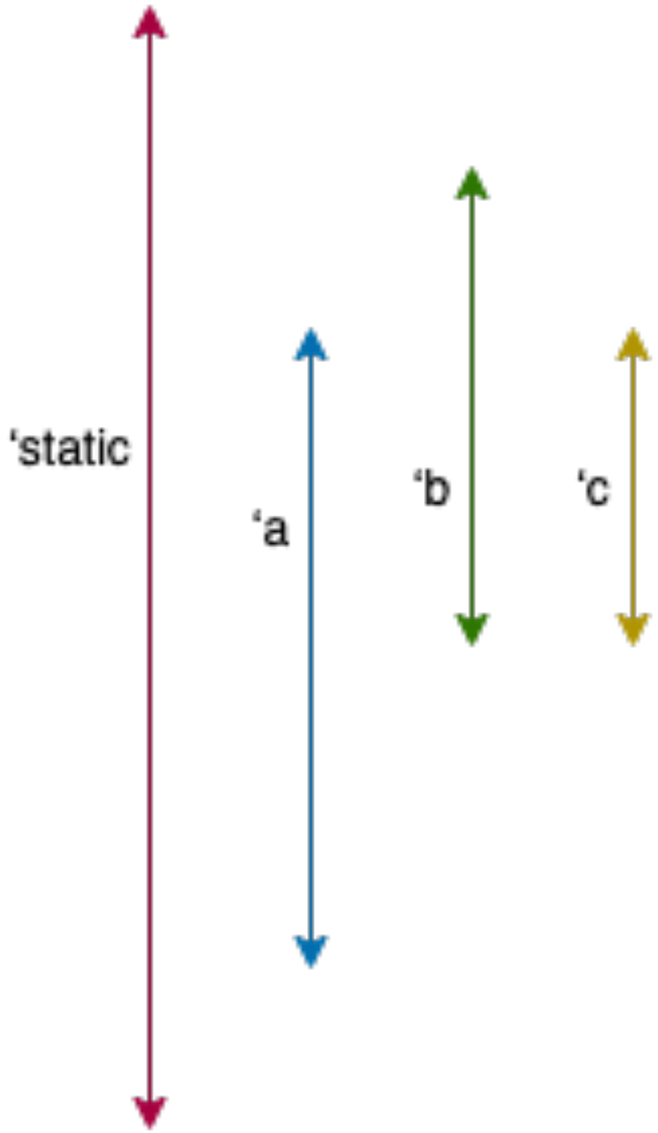
Lifetime and Subtyping

- Rust 妹🈶类型继承啊，但是有 lifetime 啊。
- 从类型继承来看，更特化的是子类型。那么生命周期中哪种更特化呢？当然是活得越长越特化啦。
- ‘a : ‘b 就意味着 ‘a 是 ‘b 的子类型
- ‘static 是任意 ‘a 的子类型

Rust生命周期和变形

Lifetime and Subtyping

- ‘static : ‘a : ‘c
- ‘static : ‘b : ‘c
- ‘a 和 ‘b ? 无亲无故



Rust生命周期和变形

Subtyping and Variance

		'a	T	U
*	`&'a T`	covariant	covariant	
*	`&'a mut T`	covariant	invariant	
*	`Box<T>`		covariant	
	`Vec<T>`		covariant	
*	`UnsafeCell<T>`		invariant	
	`Cell<T>`		invariant	
*	`fn(T) -> U`		**contra**variant	covariant
	`*const T`		covariant	
	`*mut T`		invariant	

Rust生命周期和变形

Subtyping and Variance

- 可能唯一不好理解的是 fn(T) -> U 为啥 对 T 逆变?
- 需要 compute_age(Cat) -> Int，使用 compute_age(Animal) -> Int 替换可行吗?
- 当燃辣。我都能算所有 Animal 的 age 了，肯定能算Cat 的 age 辣，连🐸的都能算。
- 对我们可以使用一个定义域更大的函数代替一个定义域更小的函数，然后在使用的时候，将其定义域安全缩小至和后者一样。即只使用Animal中Cat那一部分。

Rust生命周期和变形

来点🌰

- &mut T，对 T invariant
- 对 input 传递 &mut &‘static str
- 推断 T 为 &‘static str
- 对 val 传 &‘a str
- &‘a str : &’static str
- ‘a : ‘static
- 上一步不能成立，编译失败

```rust
fn evil_feeder<T>(input: &mut T, val: T) {
    *input = val;
}

fn main() {
    // mr. snuggles forever!!
    let mut mr_snuggles: &'static str = "meow! :3";
    {
        let spike = String::from("bark! >:V");
        // Only lives for the block
        let spike_str: &str = &spike;
        // EVIL!
        evil_feeder(&mut mr_snuggles, spike_str);
    }
    // Use after free?
    println!("{}", mr_snuggles);
}
```

Rust生命周期和变形

再来个🌰

- Vec<‘a String> 对 ‘a covariant
- Vec<‘a String>： Vec<‘b String>
- &‘c mut T 对 T covariant
- &‘c mut <‘a String> 和 &‘c mut <‘b String> 無子类型关系
- &‘c T 对 T covariant
- 所以&’c <‘a String> : &’c <‘b String>

```rust
fn covariant<'a: 'b, 'b, 'c>(
    sub: &'c mut Vec<&'a String>,
    sup: &'c mut Vec<&'b String>,
) -> &'c mut Vec<&'b String> {
    sub
}

fn invariant<'a: 'b, 'b, 'c>(
    sub: &'c Vec<&'a String>,
    sup: &'c Vec<&'b String>,
) -> &'c Vec<&'b String> {
    sub
}
```

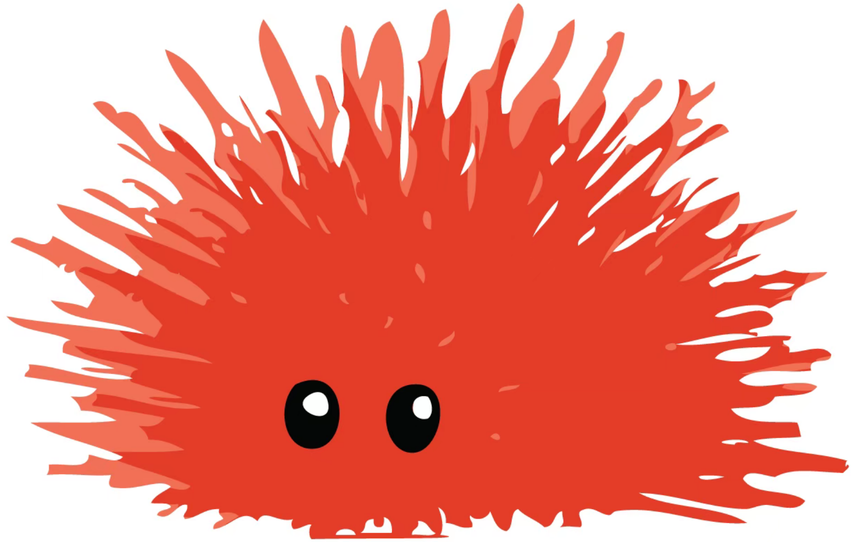
Rust生命周期和变形

还来个🌰

- fn(T) 对 T 逆变
- fn(&‘static i32) : fn(&‘a i32)
- fn(&‘static i32) : fn(&‘static i32)

```
struct ContraVariant<Mixed> {
    f: fn(Mixed),
}

fn test<'a>(
    a: &mut ContraVariant<&'a i32>,
    b: &mut ContraVariant<&'static i32>,
    f1: fn(&'a i32),
    f2: fn(&'static i32),
) {
    a.f = f1;
    a.f = f2;
    b.f = f1;
    b.f = f2;
}
```



感谢大家

参考

- https://doc.rust-lang.org/nomicon/subtyping.html
- https://en.wikipedia.org/wiki/Type_constructor
- https://en.wikipedia.org/wiki/Subtyping
- https://en.wikipedia.org/wiki/Covariance_and_contravariance_(computer_science)
- https://zhuanlan.zhihu.com/p/41814387