聊一聊Rust生命周期参数

学习生命周期参数的意义是: 避免出现悬垂指针

苏林

今天公开课内容

提前阅读: https://kaisery.github.io/trpl-zh-cn/ch10-03-lifetime-syntax.html

1、什么是生命周期参数
2、理解什么是晚限定(late bound)
3、理解什么是早限定(early bound)

什么是生命周期参数

学习生命周期参数的意义是，避免出现悬垂指针.

因为Rust语言它的内存安全是第一原则, 在全球70%的安全漏洞里面, 悬垂指针可能占50%, 所以避免出现悬垂指针是一个很重要的安全保障.

什么是生命周期参数

```rust
fn return_str<'a>() -> &'a str {
    let mut s :String = "Rust".to_string();
    for i :i32 in 0..3 {
        s.push_str( string: "Good ");
    }
    &s[..]
}

fn main() {
    let x :&str = return_str();
}
```

什么是生命周期参数

```rust
fn foo<'a>(x: &'a str, y: &'a str) -> &'a str {
    let result :String = String::from( s: "really long string");
    result.as_str()
}

fn main() {
    let x :&str = "hello";
    let y :&str = "rust";
    foo(x, y);
}
```

什么是生命周期参数

```rust
fn the_longest(s1: &str, s2: &str) -> &str {
    if s1.len() > s2.len() {
        s1
    } else {
        s2
    }
}

fn main() {
    let s1 : String = String::from( s: "Rust");
    let s1_r : &String = &s1;
    {
        let s2 : String = String::from( s: "C");
        let res : &str = the_longest( s1: s1_r, &s2);
        println!("{} is the longest", res);
    }
}
```

理解晚限定(late bound)

```rust
fn the_longest<'a: 'c, 'b: 'c, 'c>(s1: &'a str, s2: &'b str) -> &'c str {
    if s1.len() > s2.len() {
        s1
    } else {
        s2
    }
}

fn main() {
    let s1 :String = String::from( s: "Rust");
    let s1_r :&String = &s1;
    {
        let s2 :String = String::from( s: "C");
        let res :&str = the_longest( s1: s1_r, &s2);
        println!("{} is the longest", res);
    }
}
```

理解晚限定(late bound) 在实际调用的时候, 才限定具体的类型, 这个就叫 晚限定

```rust
fn the_longest<'a: 'c, 'b: 'c, 'c>(s1: &'a str, s2: &'b str) -> &'c str {
    if s1.len() > s2.len() {
        s1
    } else {
        s2
    }
}

fn main() {
    let s1 :String = String::from( s: "Rust");
    let s1_r :&String = &s1;
    {
        let s2 :String = String::from( s: "C");
        let res :&str = the_longest( s1: s1_r, &s2);
        println!("{} is the longest", res);
    }
}
```

理解早限定(early bound)

来看一道迷题, 来开启理解early bound之路. 这段代码来自rust quiz的第11题

rust quiz: https://github.com/dtolnay/rust-quiz

```rust
fn f<'a>() {}
fn g<'a: 'a>() {}

fn main() {
    let pf :fn() = f::<'static> as fn();
    let pg :fn() = g::<'static> as fn();
    print!("{}", pf == pg);
}
```

理解早限定(early bound)

```rust
fn m<T>() {}

fn main() {
    let m1 :fn() = m::<u8>; // ok
    let m2 :fn() = m; // error: cannot infer type for `T`
}
```

理解早限定(early bound)

并不是所有的生命周期函数都是late bound, 这里给出了两条规则:

规则1: 生命周期参数受到它必须超过的某个其他生命周期的限制. ‘a: ‘b

规则2:

理解早限定(early bound)

并不是所有的生命周期函数都是late bound, 这里给出了两条规则:

规则1: 生命周期参数受到它必须超过的某个其他生命周期的限制.‘a: ‘b

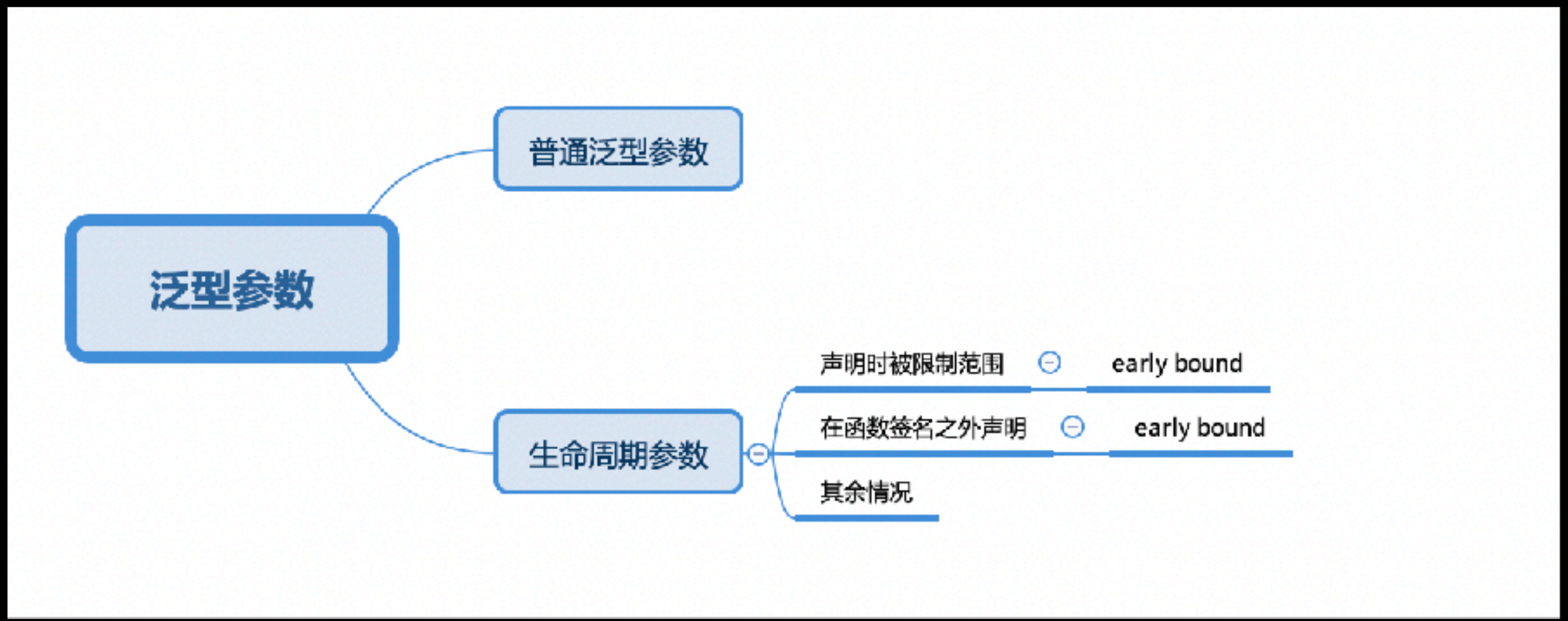
规则2: 生命周期在函数签名之外声明, 例如 在结构体的关联方法中, 它可能来自结构本身.

理解早限定(early bound)

并不是所有的生命周期函数都是late bound, 这里给出了两条规则:

规则1:

规则2:



QA环节

加群一起交流Rust & Datafuse

Datafuse Rust语言中文社区 知数堂 zhishuteng.com