Rust开发干货集

GB

guoxbin@gmail.com

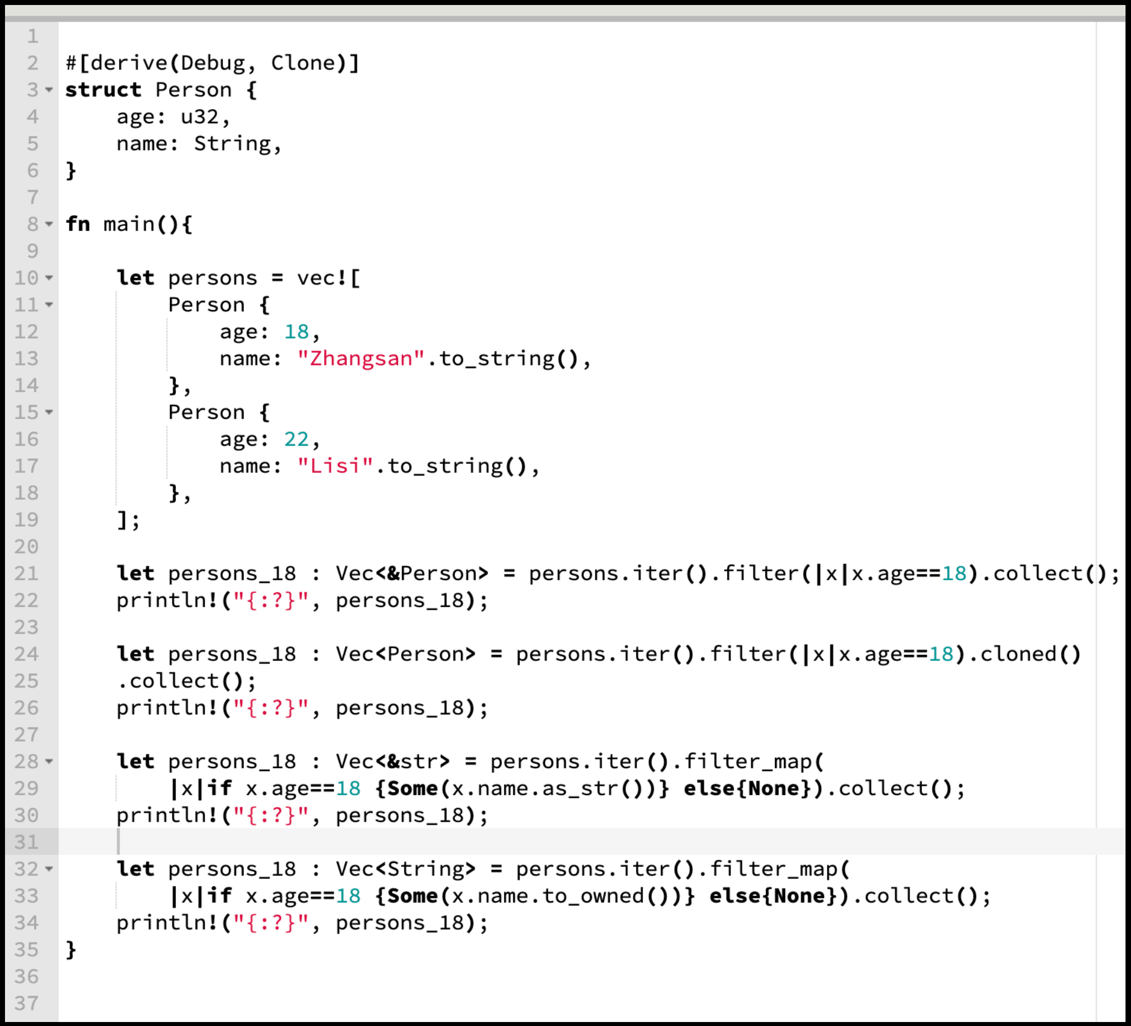
https://github.com/yeeco (YeeCo Team)

2019-12-16

YeeCo是基于PoW出块共识和CRFG最终确定性共识的全分片公链项目，使用**Rust**作为开发语言

迭代器

- iter() 不转移所有权
- cloned() 可以实现复制
- filter_map() 可以同时完成转换和过滤

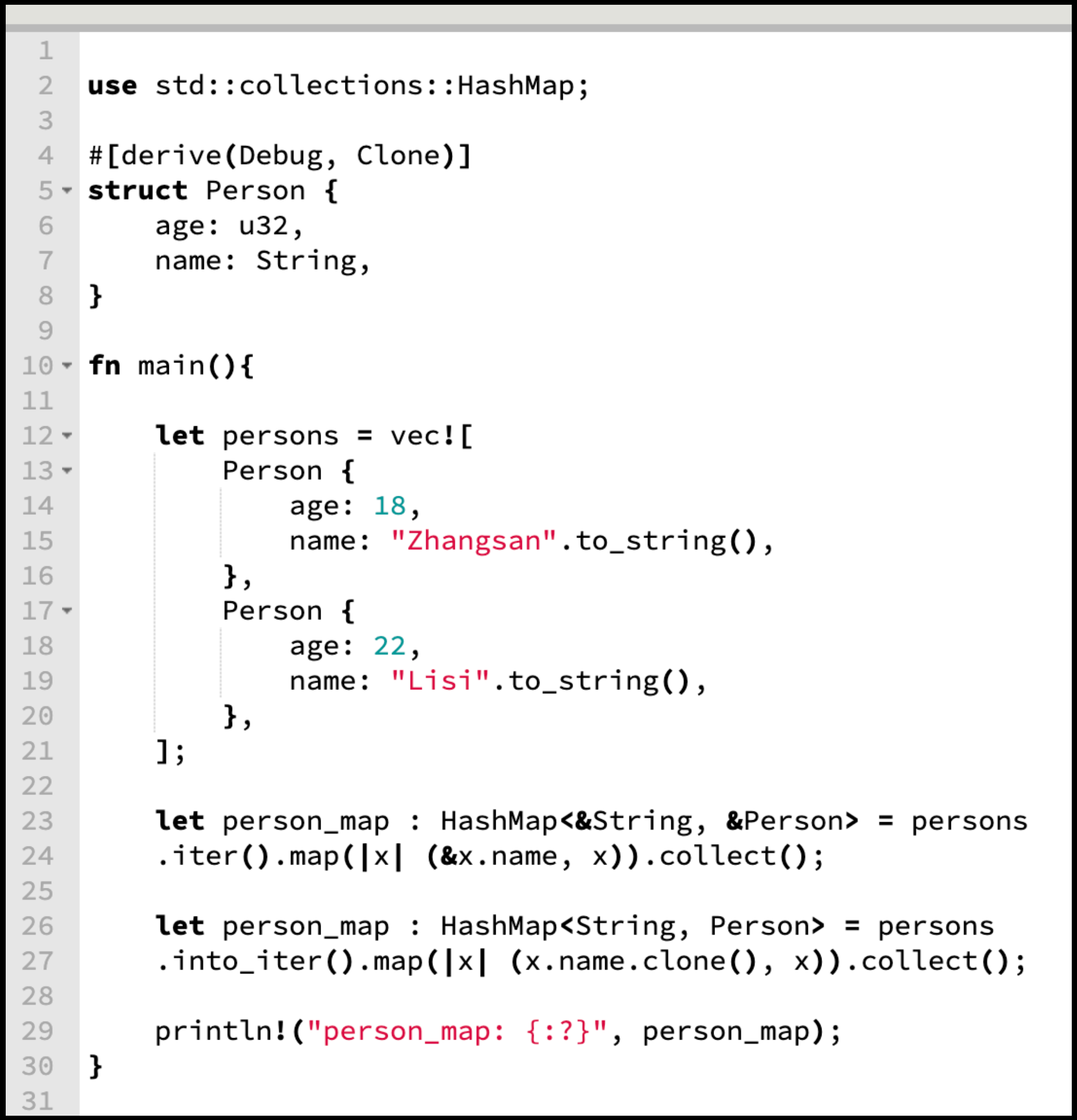


```rust

#[derive(Debug, Clone)]
struct Person {
    age: u32,
    name: String,
}

fn main(){

    let persons = vec![
        Person {
            age: 18,
            name: "Zhangsan".to_string(),
        },
        Person {
            age: 22,
            name: "Lisi".to_string(),
        },
    ];

    let persons_18 : Vec<&Person> = persons.iter().filter(|x|x.age==18).collect();
    println!("{:?}", persons_18);

    let persons_18 : Vec<Person> = persons.iter().filter(|x|x.age==18).cloned()
    .collect();
    println!("{:?}", persons_18);

    let persons_18 : Vec<&str> = persons.iter().filter_map(
        |x|if x.age==18 {Some(x.name.as_str())} else{None}).collect();
    println!("{:?}", persons_18);

    let persons_18 : Vec<String> = persons.iter().filter_map(
        |x|if x.age==18 {Some(x.name.to_owned())} else{None}).collect();
    println!("{:?}", persons_18);
}


```

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=24dde9019b32871ba19b074aa3fe1038

迭代器

- into_iter() 转移所有权



```rust

use std::collections::HashMap;

#[derive(Debug, Clone)]
struct Person {
    age: u32,
    name: String,
}

fn main(){

    let persons = vec![
        Person {
            age: 18,
            name: "Zhangsan".to_string(),
        },
        Person {
            age: 22,
            name: "Lisi".to_string(),
        },
    ];

    let person_map : HashMap<&String, &Person> = persons
    .iter().map(|x| (&x.name, x)).collect();

    let person_map : HashMap<String, Person> = persons
    .into_iter().map(|x| (x.name.clone(), x)).collect();

    println!("person_map: {:?}", person_map);
}

```

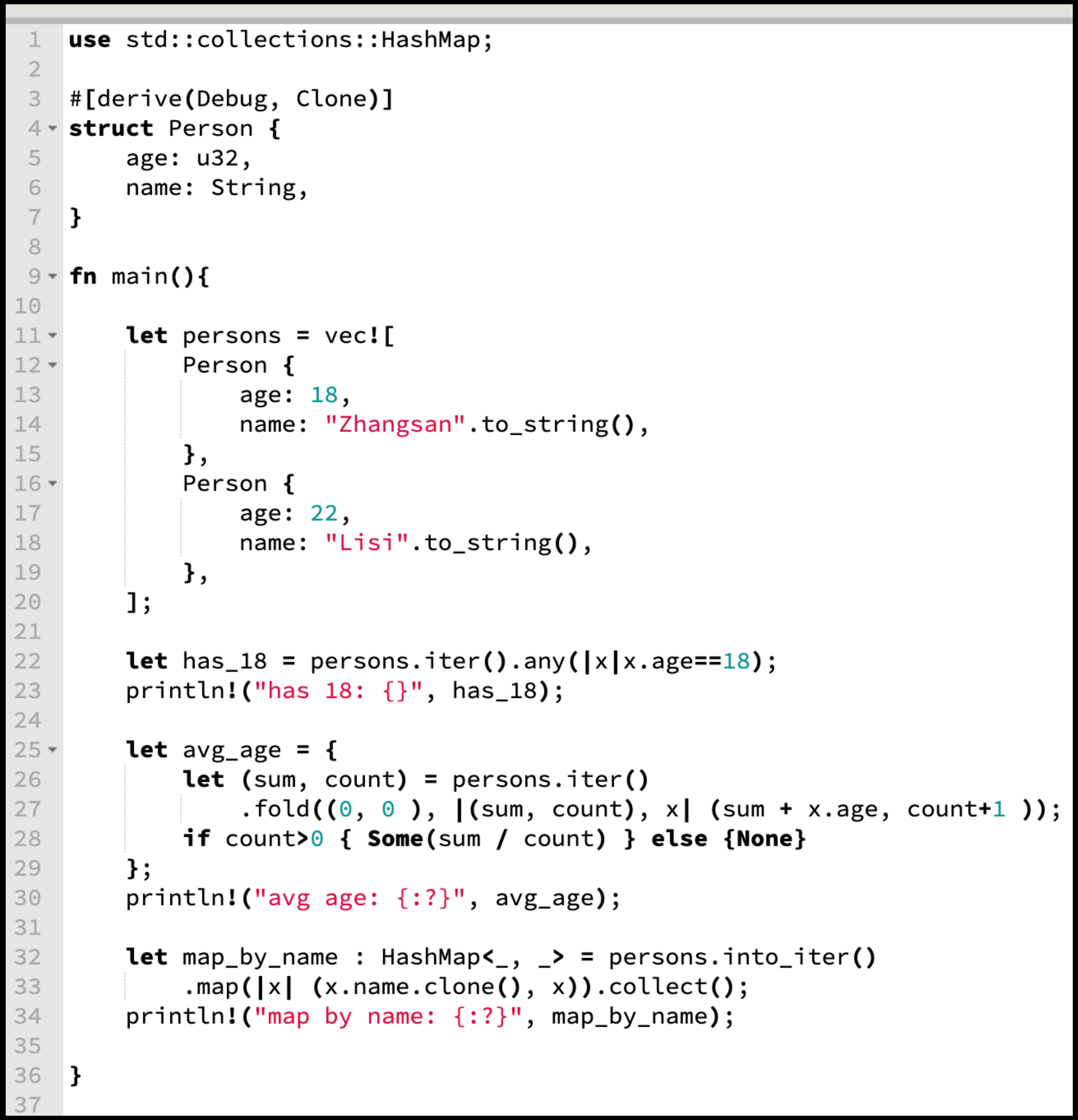
https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=cb5e1a2f8dcdb0e4c66a80cf9bac640b

迭代器

- any()消费器可以查找是否存在满足条件的元素，迭代器是惰性的，any消费器可能不需要遍历Iterator

- fold()消费器可以实现reduce逻辑

- collect()消费器可以实现集合类型转化

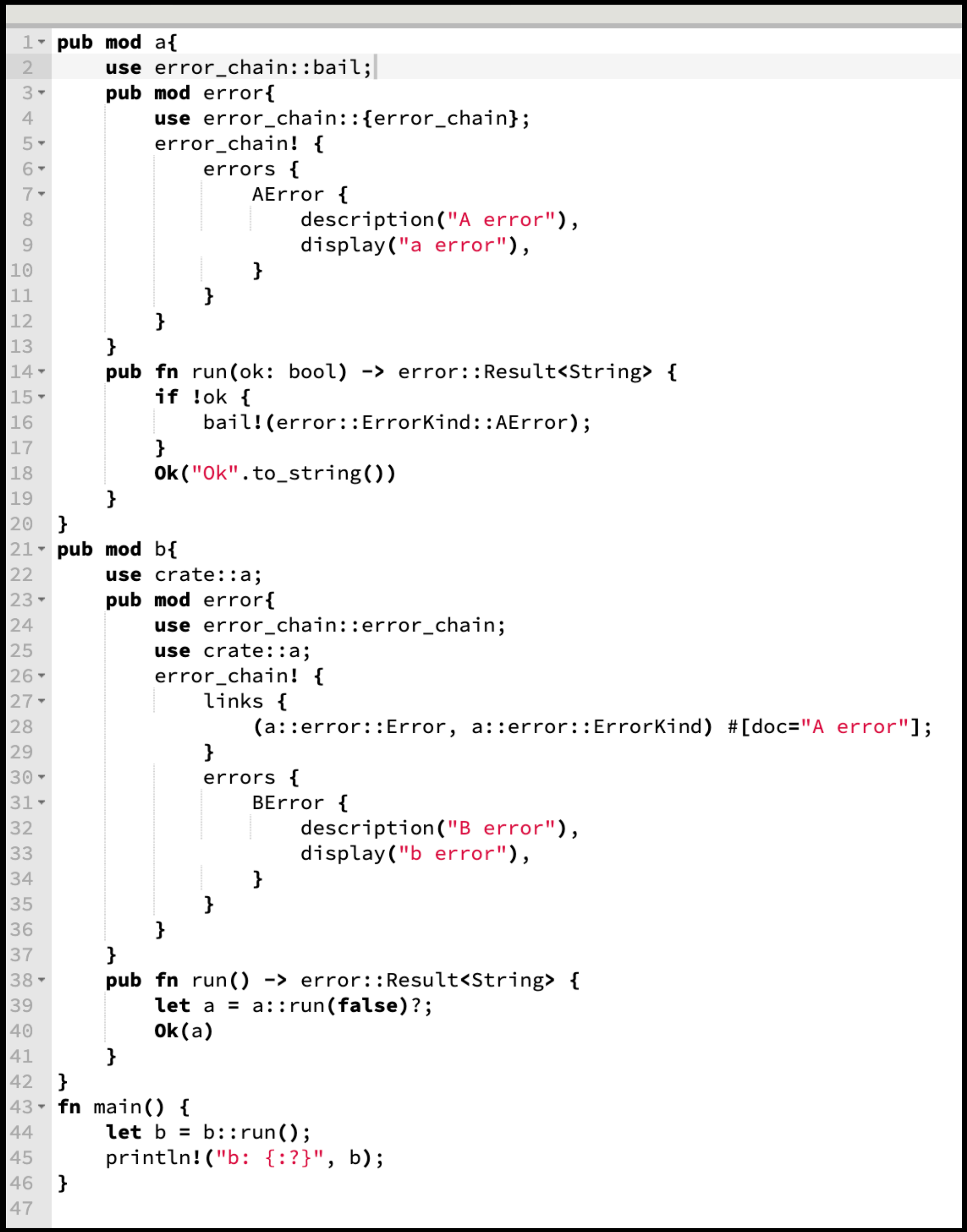


```rust
use std::collections::HashMap;

#[derive(Debug, Clone)]
struct Person {
    age: u32,
    name: String,
}

fn main(){

    let persons = vec![
        Person {
            age: 18,
            name: "Zhangsan".to_string(),
        },
        Person {
            age: 22,
            name: "Lisi".to_string(),
        },
    ];

    let has_18 = persons.iter().any(|x|x.age==18);
    println!("has 18: {}", has_18);

    let avg_age = {
        let (sum, count) = persons.iter()
            .fold((0, 0 ), |(sum, count), x| (sum + x.age, count+1 ));
        if count>0 { Some(sum / count) } else {None}
    };
    println!("avg age: {:?}", avg_age);

    let map_by_name : HashMap<_, _> = persons.into_iter()
        .map(|x| (x.name.clone(), x)).collect();
    println!("map by name: {:?}", map_by_name);

}
```

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=7376f6766ce7aa584b6c66717c206aeb

错误处理

- 每个模块可以声明错误
- 每个模块可以声明对其他模块的错误的link，可实现自动的err_map

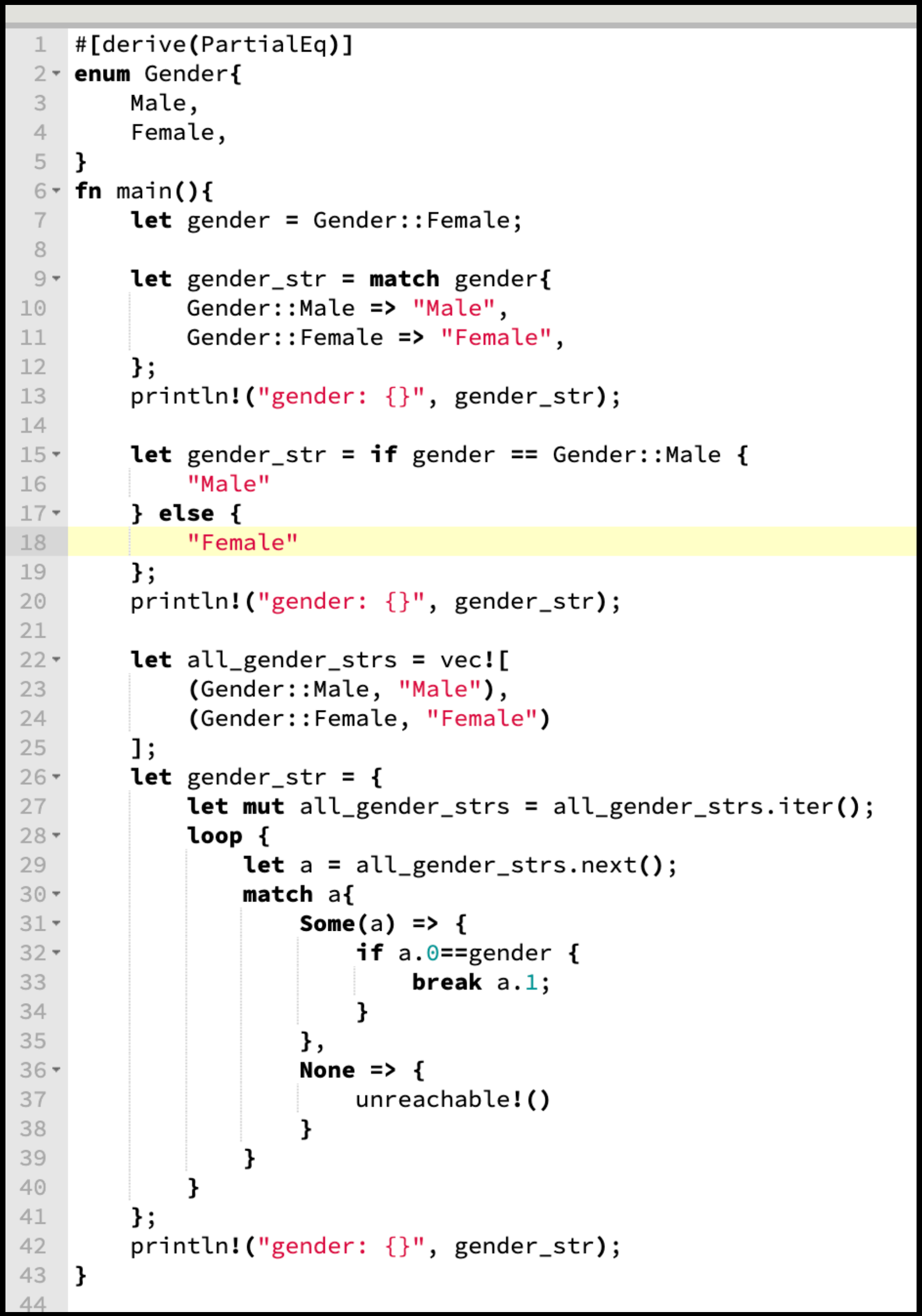


```rust
pub mod a{
    use error_chain::bail;
    pub mod error{
        use error_chain::{error_chain};
        error_chain! {
            errors {
                AError {
                    description("A error"),
                    display("a error"),
                }
            }
        }
    }
    pub fn run(ok: bool) -> error::Result<String> {
        if !ok {
            bail!(error::ErrorKind::AError);
        }
        Ok("Ok".to_string())
    }
}
pub mod b{
    use crate::a;
    pub mod error{
        use error_chain::error_chain;
        use crate::a;
        error_chain! {
            links {
                (a::error::Error, a::error::ErrorKind) #[doc="A error"];
            }
            errors {
                BError {
                    description("B error"),
                    display("b error"),
                }
            }
        }
    }
    pub fn run() -> error::Result<String> {
        let a = a::run(false)?;
        Ok(a)
    }
}
fn main() {
    let b = b::run();
    println!("b: {:?}", b);
}
```

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=181ebbee6b0e2369720099faed7c4271

表达式

- 使用match或if时，尽量用表达式，而不是赋值
- 使用match或if时，可使用元组实现多个变量的求值
- 使用loop时，break可以提供返回值
- 使用for时，break不可以提供返回值

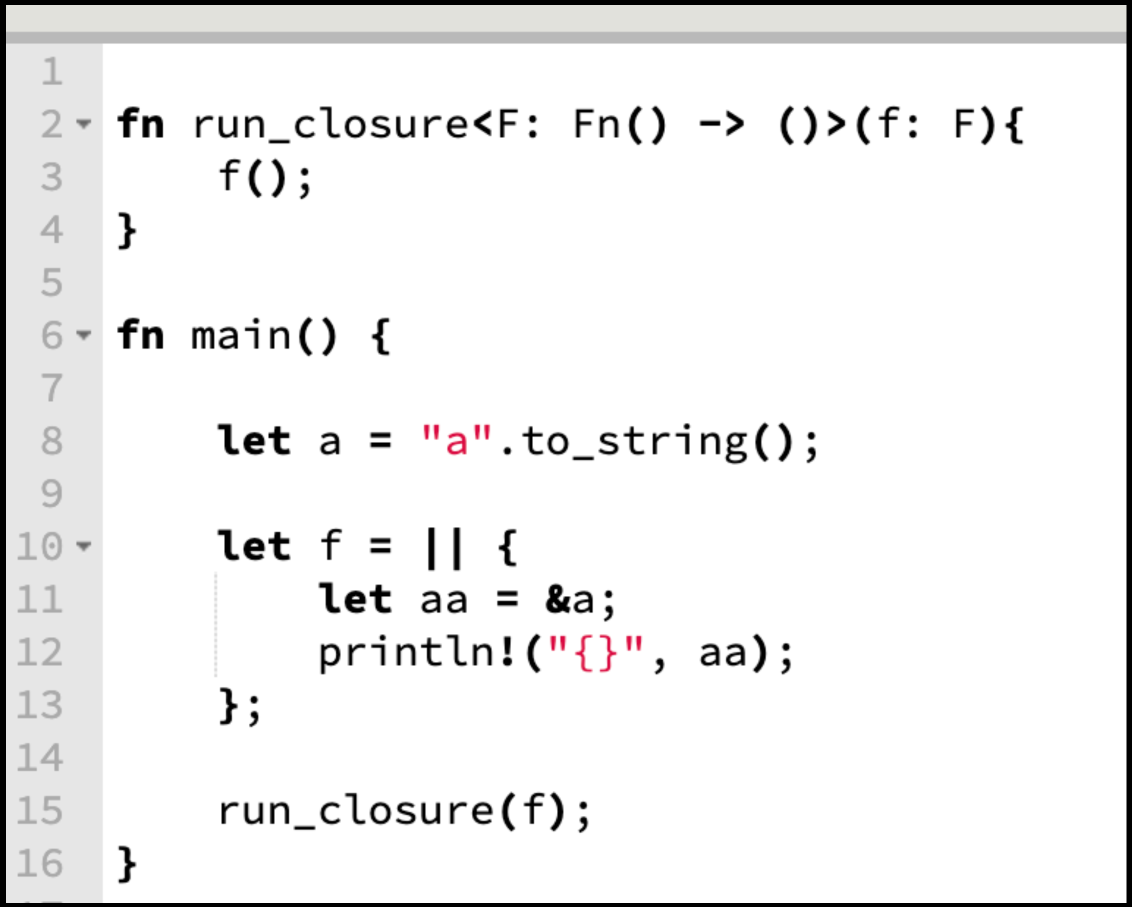


```rust
#[derive(PartialEq)]
enum Gender{
    Male,
    Female,
}
fn main(){
    let gender = Gender::Female;

    let gender_str = match gender{
        Gender::Male => "Male",
        Gender::Female => "Female",
    };
    println!("gender: {}", gender_str);

    let gender_str = if gender == Gender::Male {
        "Male"
    } else {
        "Female"
    };
    println!("gender: {}", gender_str);

    let all_gender_strs = vec![
        (Gender::Male, "Male"),
        (Gender::Female, "Female")
    ];
    let gender_str = {
        let mut all_gender_strs = all_gender_strs.iter();
        loop {
            let a = all_gender_strs.next();
            match a{
                Some(a) => {
                    if a.0==gender {
                        break a.1;
                    }
                },
                None => {
                    unreachable!()
                }
            }
        }
    };
    println!("gender: {}", gender_str);
}
```

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=ee3a71dd397cf38ff75b4685c6e7885a

闭包

- Fn() 闭包
- 对应借用语义
- 捕获的变量是借用

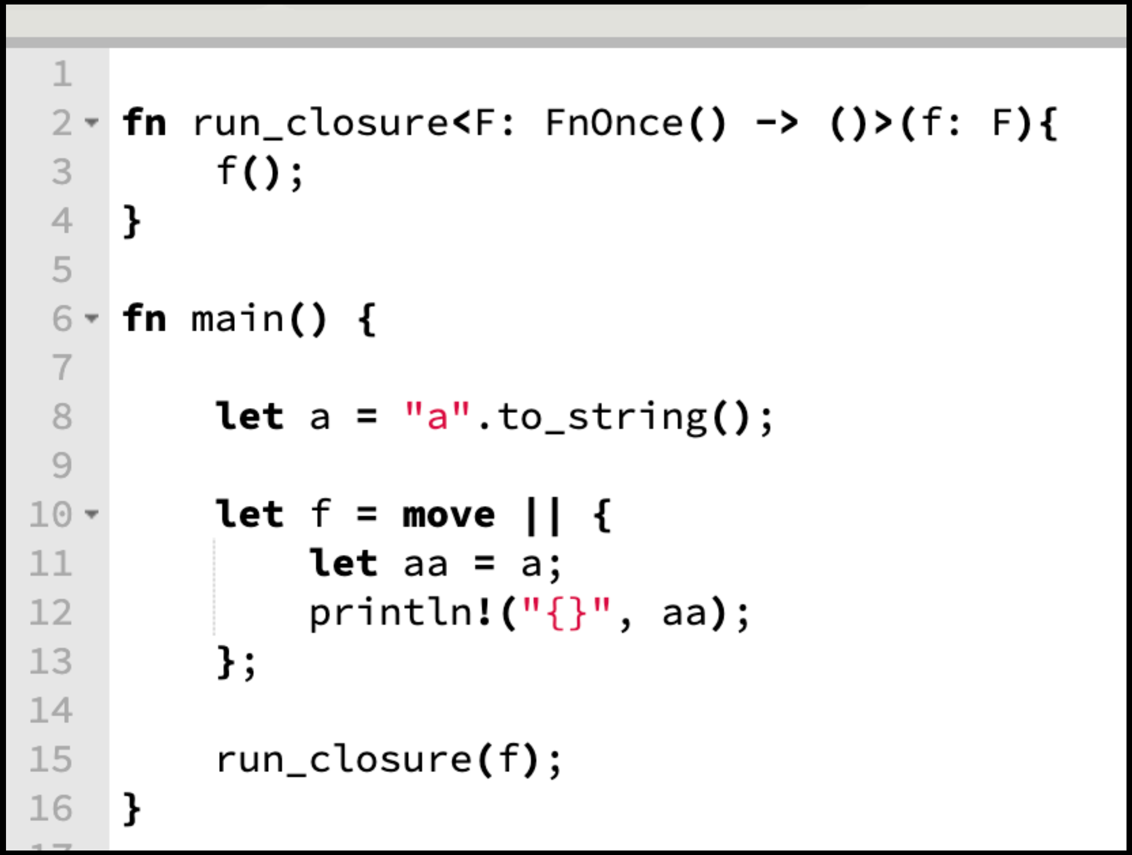


```rust

fn run_closure<F: Fn() -> ()>(f: F){
    f();
}

fn main() {

    let a = "a".to_string();

    let f = || {
        let aa = &a;
        println!("{}", aa);
    };

    run_closure(f);
}
```

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=70d1dcdc9d4fb7303e7606017d493ee4

闭包

- FnOnce() 闭包
- 对应所有权转移语义
- 捕获的变量是所有权转移

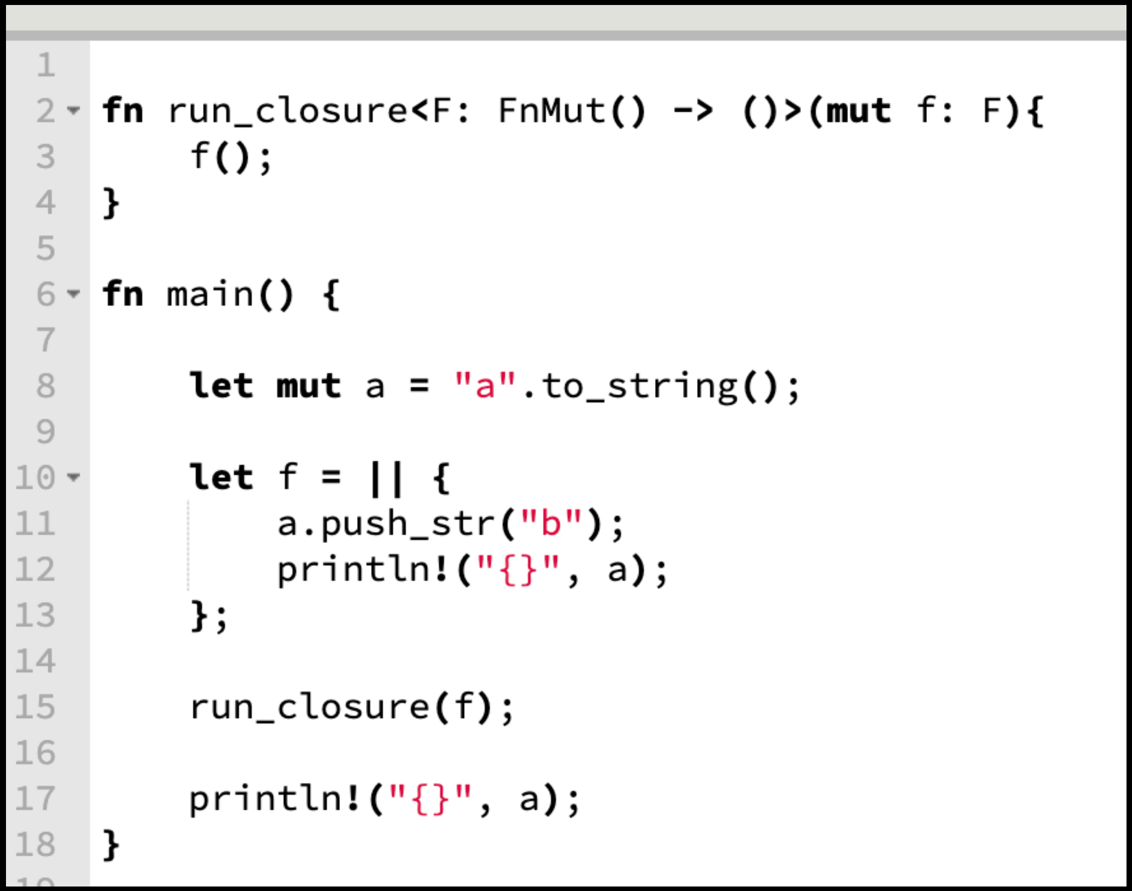


```rust

fn run_closure<F: FnOnce() -> ()>(f: F){
    f();
}

fn main() {

    let a = "a".to_string();

    let f = move || {
        let aa = a;
        println!("{}", aa);
    };

    run_closure(f);
}
```

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=8598889e61ee12a8598ac0d42ba246d6

闭包

- FnMut() 闭包
- 对应可变借用语义
- 捕获的变量是可变借用

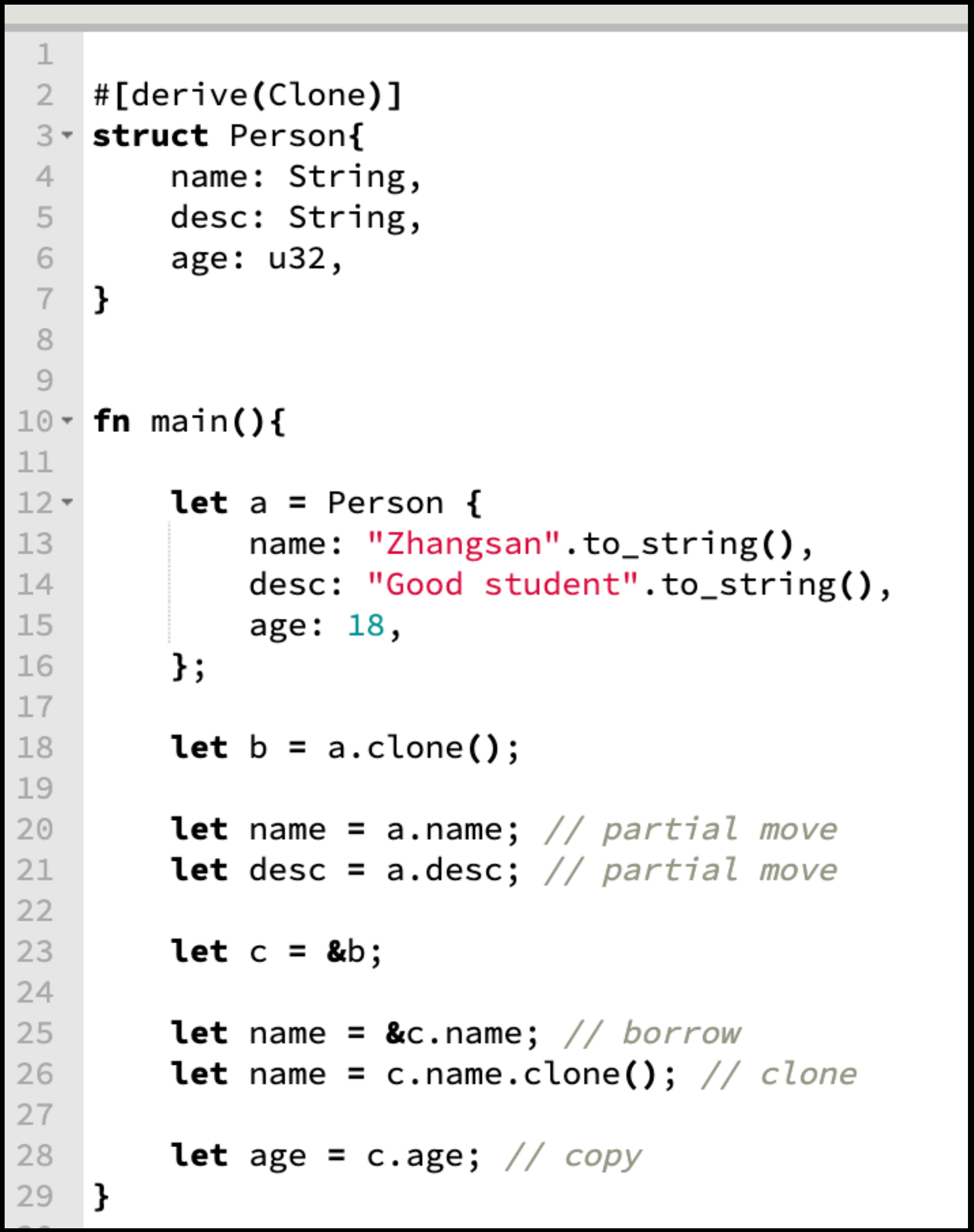


```rust

fn run_closure<F: FnMut() -> ()>(mut f: F){
    f();
}

fn main() {

    let mut a = "a".to_string();

    let f = || {
        a.push_str("b");
        println!("{}", a);
    };

    run_closure(f);

    println!("{}", a);
}
```

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=1e2c5e88f36088fa8b8550abf75722a3

所有权

- 多个Partial move之间可以并存
- Partial move 和move 之间不可并存
- 对借用的成员进行绑定操作是move语义，应该加&或clone
- 对借用的成员进行绑定操作，如果成员实现了copy，则是copy语义，可以不加&和clone

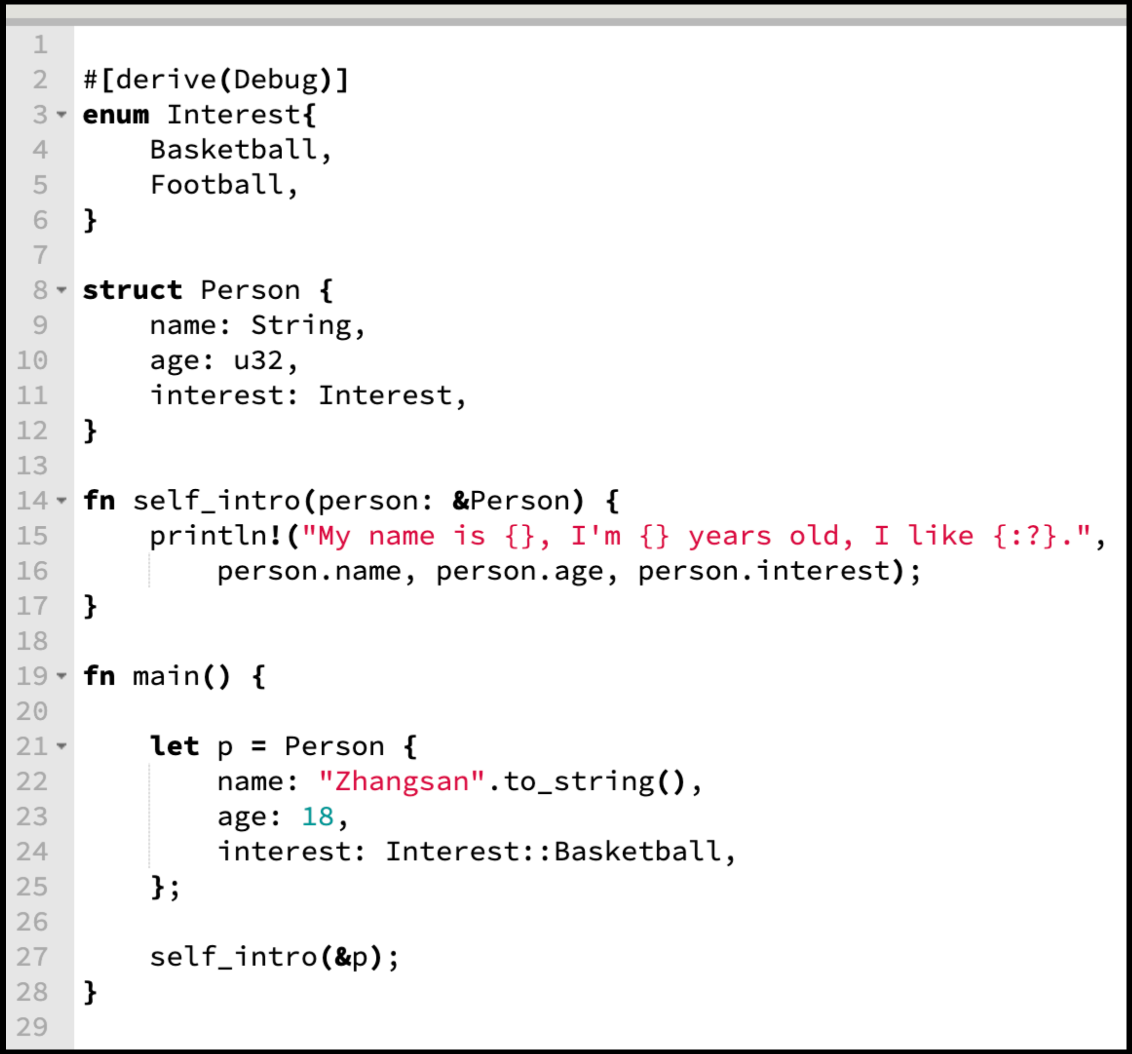


```rust

#[derive(Clone)]
struct Person{
    name: String,
    desc: String,
    age: u32,
}


fn main(){

    let a = Person {
        name: "Zhangsan".to_string(),
        desc: "Good student".to_string(),
        age: 18,
    };

    let b = a.clone();

    let name = a.name; // partial move
    let desc = a.desc; // partial move

    let c = &b;

    let name = &c.name; // borrow
    let name = c.name.clone(); // clone

    let age = c.age; // copy
}
```

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=60e2245c50bd75ddbcd760e4db8c2cce

泛型

- 泛型的目的有
  - 多态
  - 契约编程



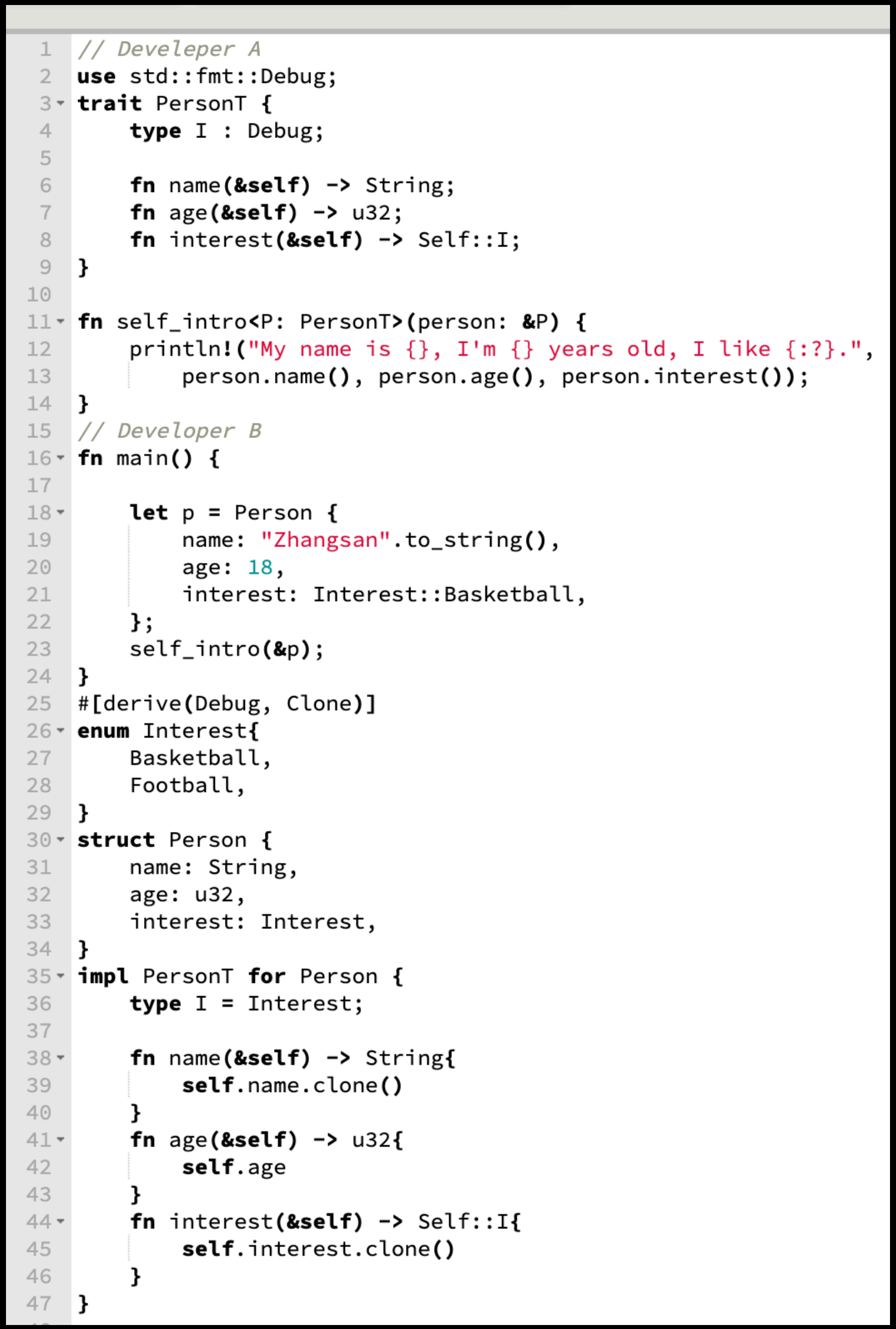
```rust
#[derive(Debug)]
enum Interest{
    Basketball,
    Football,
}

struct Person {
    name: String,
    age: u32,
    interest: Interest,
}

fn self_intro(person: &Person) {
    println!("My name is {}, I'm {} years old, I like {:?}.",
        person.name, person.age, person.interest);
}

fn main() {

    let p = Person {
        name: "Zhangsan".to_string(),
        age: 18,
        interest: Interest::Basketball,
    };

    self_intro(&p);
}
```

- 代码演示了不使用泛型的情况

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=76d1fa9958f5ae7b780b1f521cbb7b2c

泛型

- 在契约编程中，两个模块有依赖关系，由不同人在开发，开发节奏也不同
- 范型是依赖方定义抽象概念，使得自己的模块自洽，可以独立编译的重要工具。



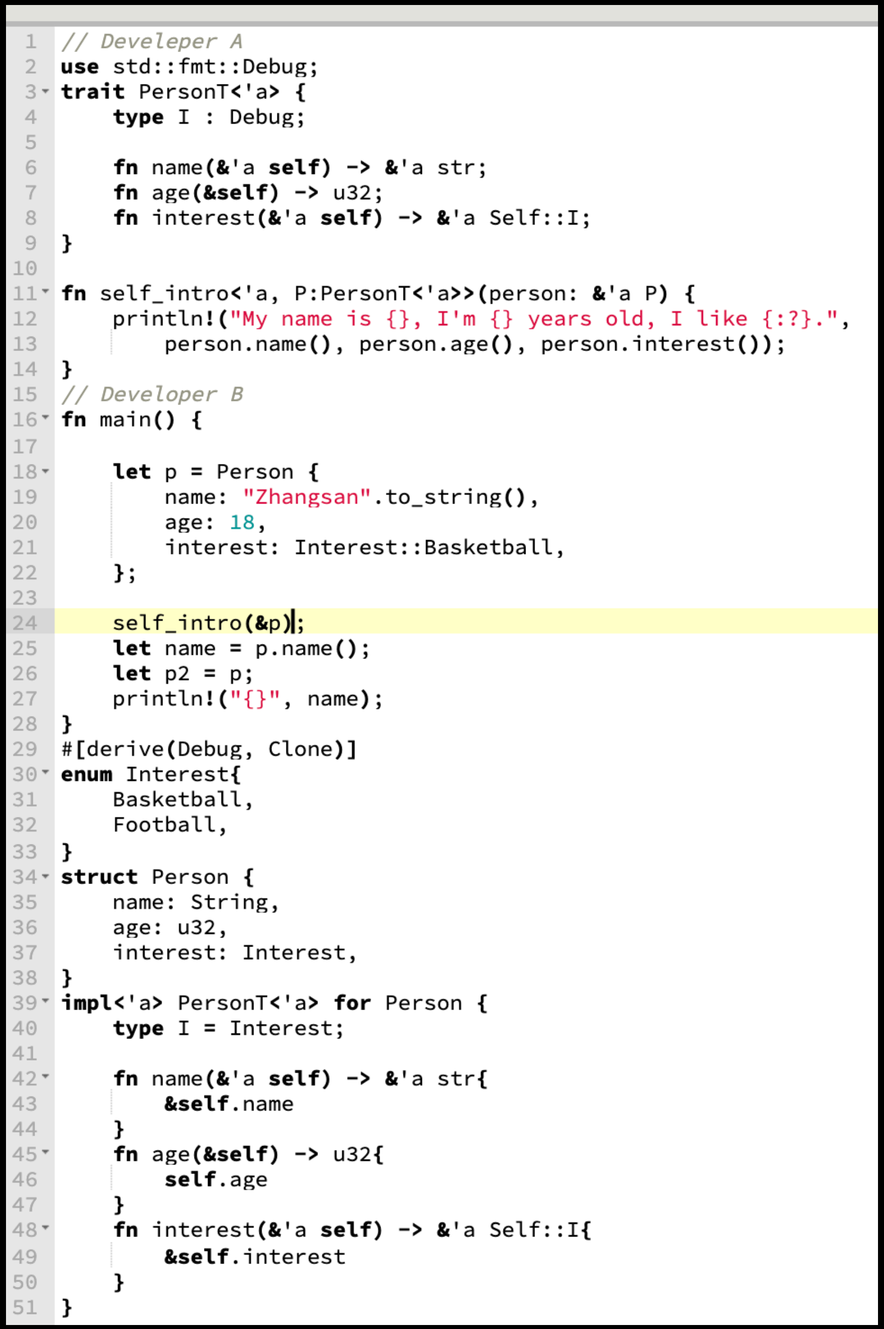
```
// Develeper A
use std::fmt::Debug;
trait PersonT {
    type I : Debug;

    fn name(&self) -> String;
    fn age(&self) -> u32;
    fn interest(&self) -> Self::I;
}

fn self_intro<P: PersonT>(person: &P) {
    println!("My name is {}, I'm {} years old, I like {:?}.",
        person.name(), person.age(), person.interest());
}
// Developer B
fn main() {

    let p = Person {
        name: "Zhangsan".to_string(),
        age: 18,
        interest: Interest::Basketball,
    };
    self_intro(&p);
}
#[derive(Debug, Clone)]
enum Interest{
    Basketball,
    Football,
}
struct Person {
    name: String,
    age: u32,
    interest: Interest,
}
impl PersonT for Person {
    type I = Interest;

    fn name(&self) -> String{
        self.name.clone()
    }
    fn age(&self) -> u32{
        self.age
    }
    fn interest(&self) -> Self::I{
        self.interest.clone()
    }
}
```

- 代码演示了使用泛型的情况

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=8af54f21067684d03f048673ac4c46ca

生命周期

- 生命周期是借用引入的
- 基于性能的考虑，应该尽量用借用而不是clone
- 生命周期定义了关联的变量之间的生命周期相关性
    - struct和成员变量
    - 函数参数和返回值（包括trait中的）
        - 代码演示了使用生命周期优化泛型性能

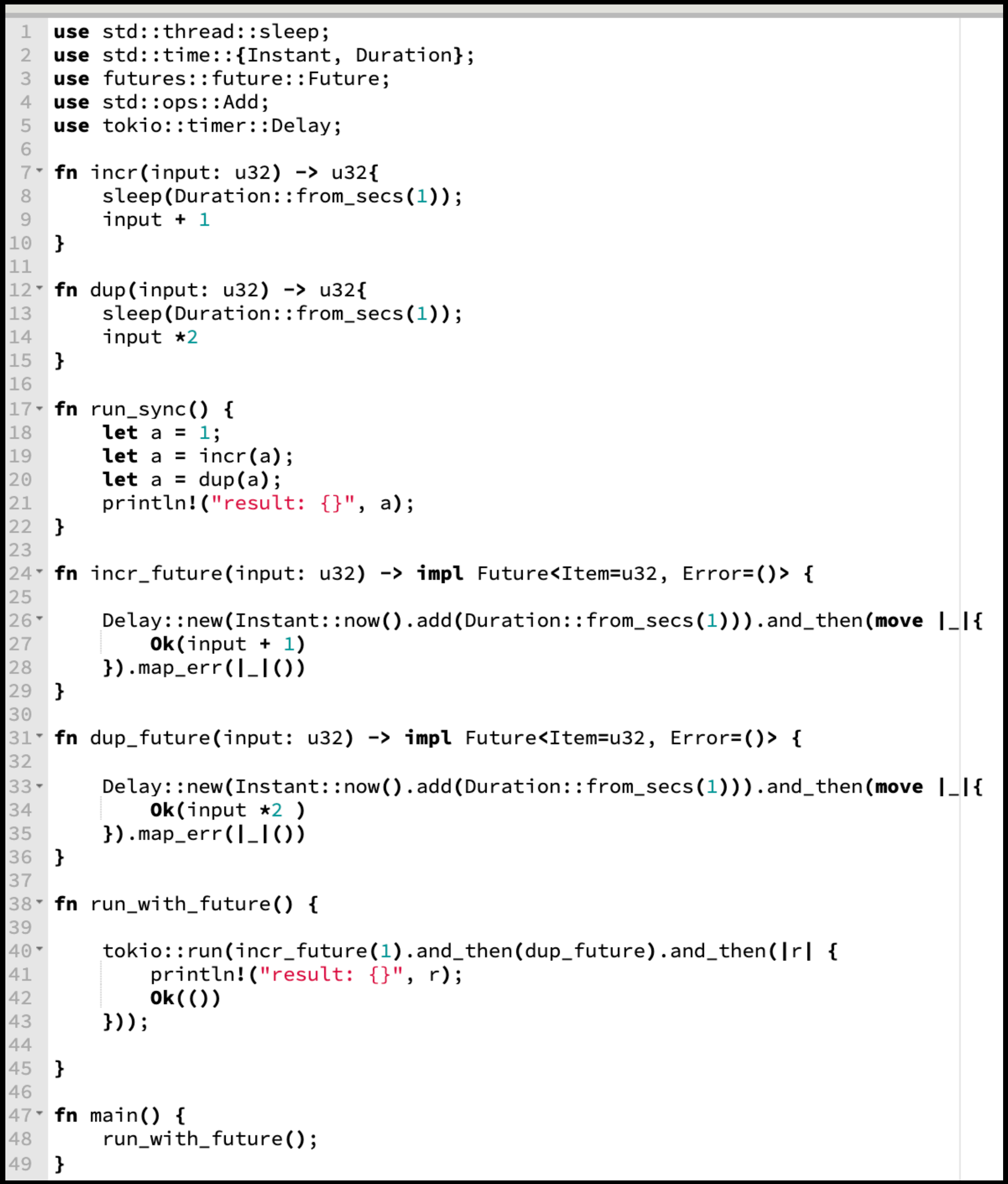


```rust
// Develeper A
use std::fmt::Debug;
trait PersonT<'a> {
    type I : Debug;

    fn name(&'a self) -> &'a str;
    fn age(&self) -> u32;
    fn interest(&'a self) -> &'a Self::I;
}

fn self_intro<'a, P:PersonT<'a>>(person: &'a P) {
    println!("My name is {}, I'm {} years old, I like {:?}.",
        person.name(), person.age(), person.interest());
}
// Developer B
fn main() {

    let p = Person {
        name: "Zhangsan".to_string(),
        age: 18,
        interest: Interest::Basketball,
    };

    self_intro(&p);
    let name = p.name();
    let p2 = p;
    println!("{}", name);
}
#[derive(Debug, Clone)]
enum Interest{
    Basketball,
    Football,
}
struct Person {
    name: String,
    age: u32,
    interest: Interest,
}
impl<'a> PersonT<'a> for Person {
    type I = Interest;

    fn name(&'a self) -> &'a str{
        &self.name
    }
    fn age(&self) -> u32{
        self.age
    }
    fn interest(&'a self) -> &'a Self::I{
        &self.interest
    }
}
```

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=bce361d578b606d7e85e8a191fe2e09e

异步编程

- 函数式编程
  - and_then / map
  - or_else / map_err
  - then
  - select
  - join



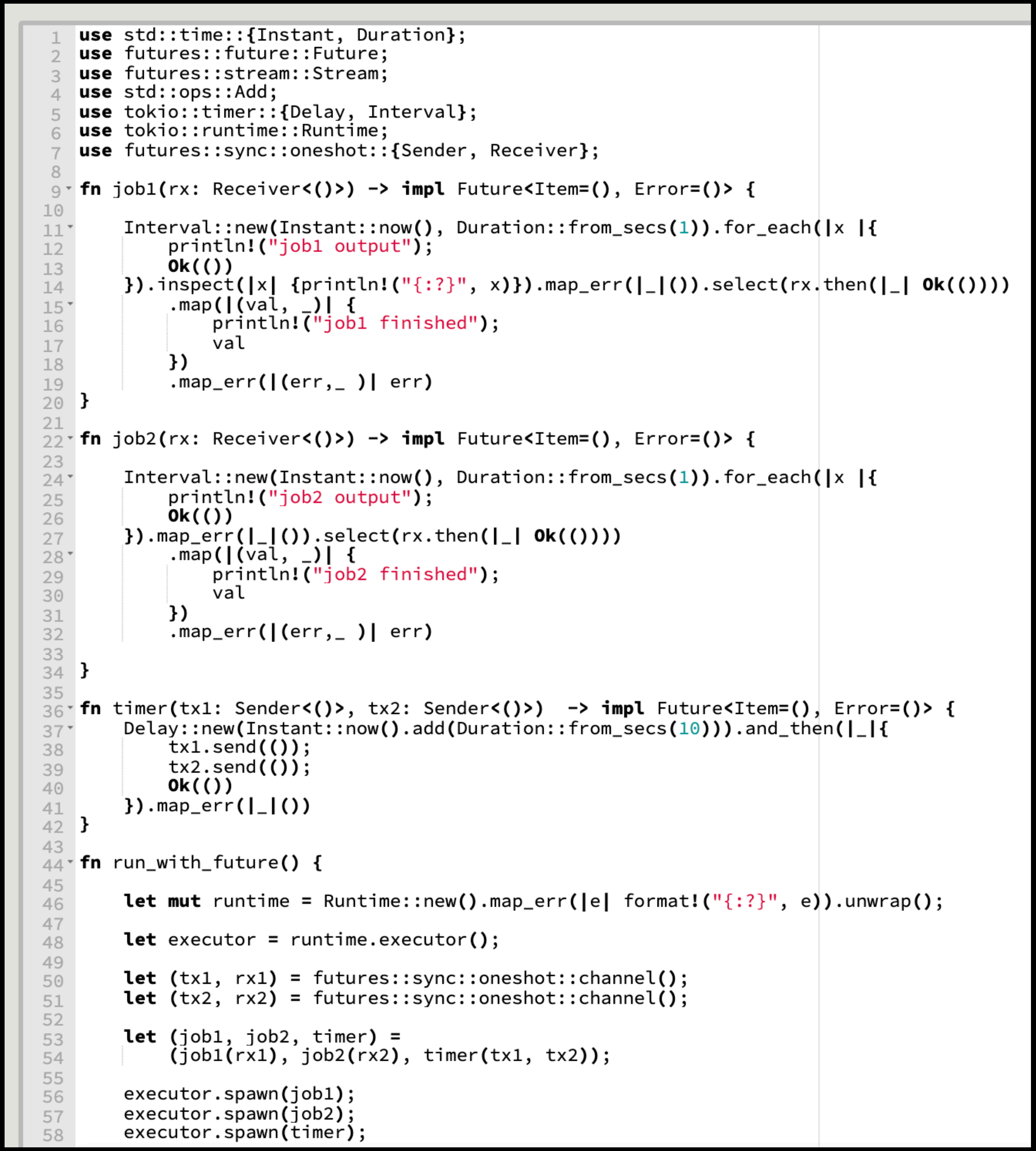
```rust
use std::thread::sleep;
use std::time::{Instant, Duration};
use futures::future::Future;
use std::ops::Add;
use tokio::timer::Delay;

fn incr(input: u32) -> u32{
    sleep(Duration::from_secs(1));
    input + 1
}

fn dup(input: u32) -> u32{
    sleep(Duration::from_secs(1));
    input *2
}

fn run_sync() {
    let a = 1;
    let a = incr(a);
    let a = dup(a);
    println!("result: {}", a);
}

fn incr_future(input: u32) -> impl Future<Item=u32, Error=()> {

    Delay::new(Instant::now().add(Duration::from_secs(1))).and_then(move |_|{
        Ok(input + 1)
    }).map_err(|_|())
}

fn dup_future(input: u32) -> impl Future<Item=u32, Error=()> {

    Delay::new(Instant::now().add(Duration::from_secs(1))).and_then(move |_|{
        Ok(input *2 )
    }).map_err(|_|())
}

fn run_with_future() {

    tokio::run(incr_future(1).and_then(dup_future).and_then(|r| {
        println!("result: {}", r);
        Ok(())
    }));

}

fn main() {
    run_with_future();
}
```

- 代码无法在playground中运行

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=8df205a949f2779839ad8f3f75575493

异步编程

- 低开销并发
- 消息传递优于共享内存



```rust
use std::time::{Instant, Duration};
use futures::future::Future;
use futures::stream::Stream;
use std::ops::Add;
use tokio::timer::{Delay, Interval};
use tokio::runtime::Runtime;
use futures::sync::oneshot::{Sender, Receiver};

fn job1(rx: Receiver<()>) -> impl Future<Item=(), Error=()> {

    Interval::new(Instant::now(), Duration::from_secs(1)).for_each(|x |{
        println!("job1 output");
        Ok(())
    }).inspect(|x| {println!("{:?}", x)}).map_err(|_|()).select(rx.then(|_| Ok(())))
        .map(|(val, _)| {
            println!("job1 finished");
            val
        })
        .map_err(|(err,_ )| err)
}

fn job2(rx: Receiver<()>) -> impl Future<Item=(), Error=()> {

    Interval::new(Instant::now(), Duration::from_secs(1)).for_each(|x |{
        println!("job2 output");
        Ok(())
    }).map_err(|_|()).select(rx.then(|_| Ok(())))
        .map(|(val, _)| {
            println!("job2 finished");
            val
        })
        .map_err(|(err,_ )| err)

}

fn timer(tx1: Sender<()>, tx2: Sender<()>)  -> impl Future<Item=(), Error=()> {
    Delay::new(Instant::now().add(Duration::from_secs(10))).and_then(|_|{
        tx1.send(());
        tx2.send(());
        Ok(())
    }).map_err(|_|())
}

fn run_with_future() {

    let mut runtime = Runtime::new().map_err(|e| format!("{:?}", e)).unwrap();

    let executor = runtime.executor();

    let (tx1, rx1) = futures::sync::oneshot::channel();
    let (tx2, rx2) = futures::sync::oneshot::channel();

    let (job1, job2, timer) =
        (job1(rx1), job2(rx2), timer(tx1, tx2));

    executor.spawn(job1);
    executor.spawn(job2);
    executor.spawn(timer);
```

- 代码无法在playground中运行

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=e289a33a6716b383d1a4f80575c5da71

送的

- 1_000_000：更易读的数字
- {:#?}：更易读的debug格式
- dbg!：原地打印debug但不影响变量上下文
- match @：模式匹配时指代被匹配变量
- match if：模式匹配可以附带if条件
- Iterator::inspect：迭代中输出
- Future::inspect：future中输出

谢谢

欢迎交流

guoxbin@gmail.com