RUST异步编程-透过表象看本质

- 分享人：韩霆军

2022-11-26

Async/await 异步编程模式

内存安全

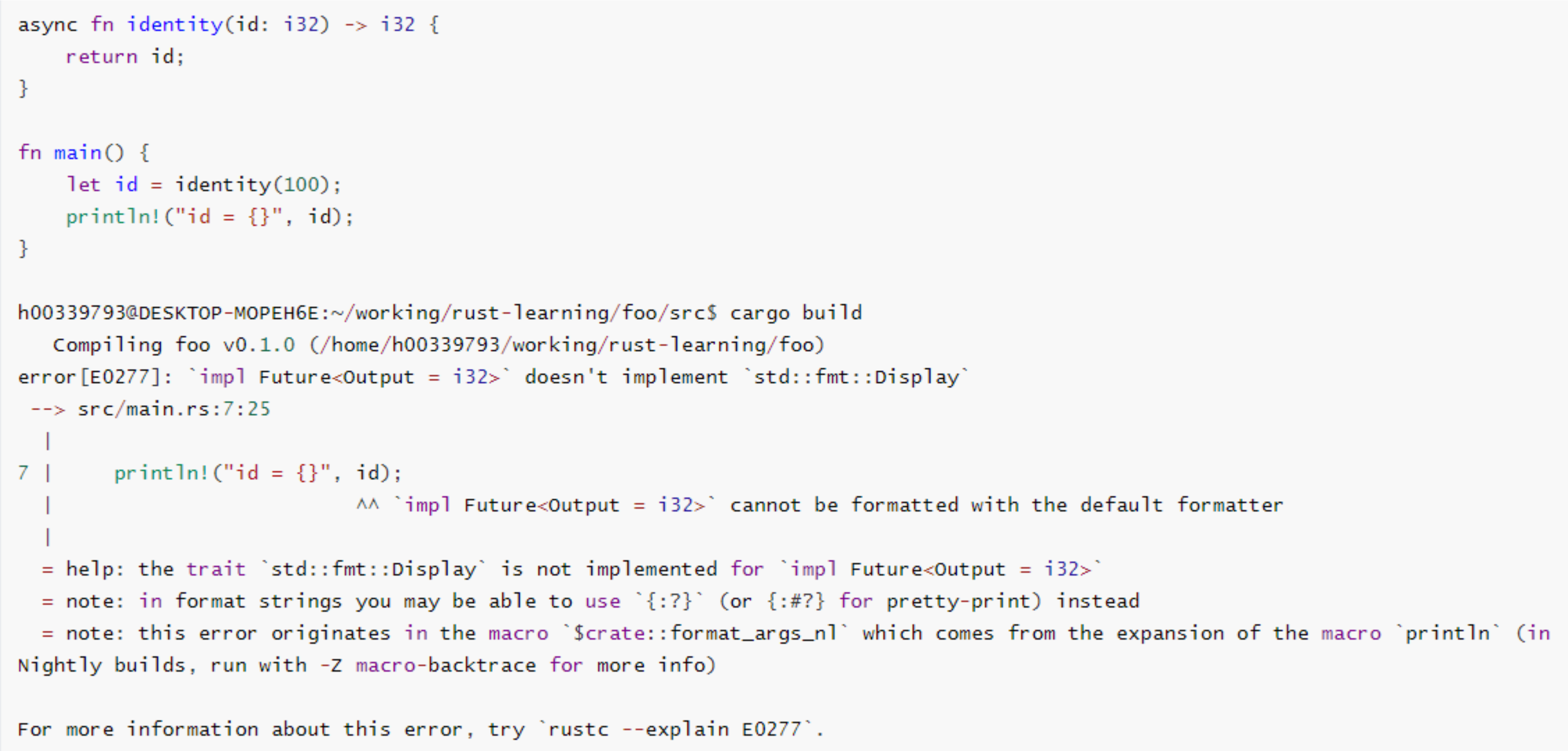
并发安全

无栈协程

无畏并发

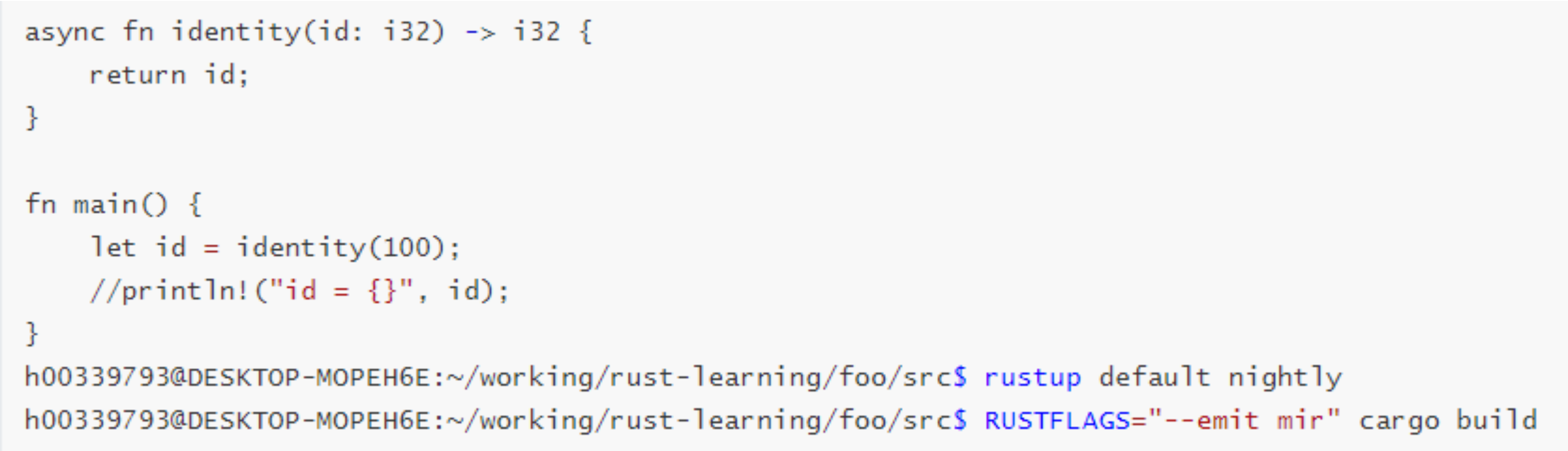
地表最强？

Async表象之下是什么？

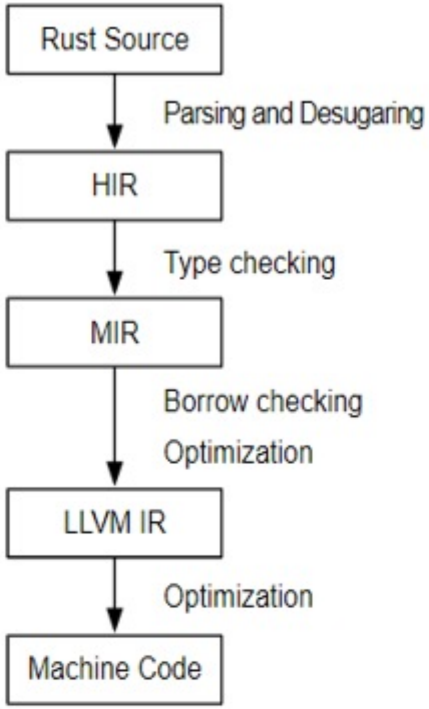


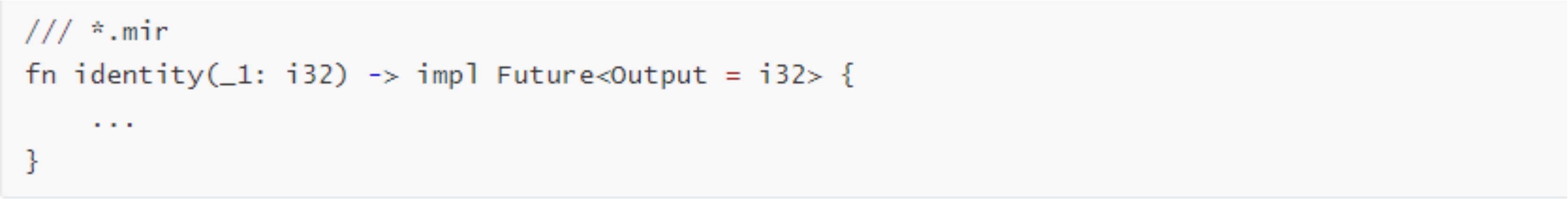
```
async fn identity(id: i32) -> i32 {
    return id;
}

fn main() {
    let id = identity(100);
    println!("id = {}", id);
}

h00339793@DESKTOP-MOPEH6E:~/working/rust-learning/foo/src$ cargo build
   Compiling foo v0.1.0 (/home/h00339793/working/rust-learning/foo)
error[E0277]: `impl Future<Output = i32>` doesn't implement `std::fmt::Display`
 --> src/main.rs:7:25
  |
7 |     println!("id = {}", id);
  |                         ^^ `impl Future<Output = i32>` cannot be formatted with the default formatter
  |
  = help: the trait `std::fmt::Display` is not implemented for `impl Future<Output = i32>`
  = note: in format strings you may be able to use `{:?}` (or {:#?} for pretty-print) instead
  = note: this error originates in the macro `$crate::format_args_nl` which comes from the expansion of the macro `println` (in
Nightly builds, run with -Z macro-backtrace for more info)

For more information about this error, try `rustc --explain E0277`.
```

Async == Future构造

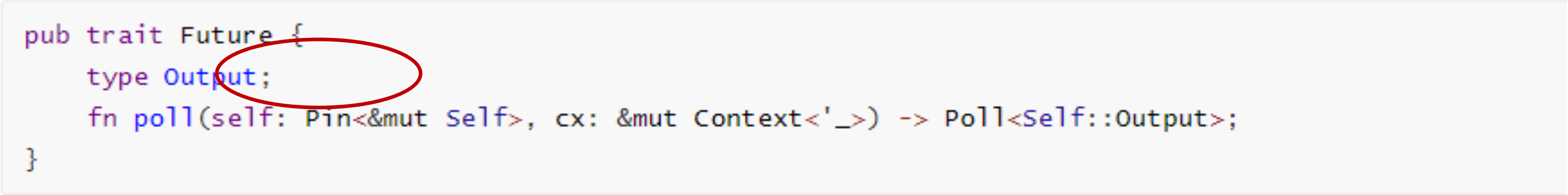


```
async fn identity(id: i32) -> i32 {
    return id;
}

fn main() {
    let id = identity(100);
    //println!("id = {}", id);
}
h00339793@DESKTOP-MOPEH6E:~/working/rust-learning/foo/src$ rustup default nightly
h00339793@DESKTOP-MOPEH6E:~/working/rust-learning/foo/src$ RUSTFLAGS="--emit mir" cargo build
```



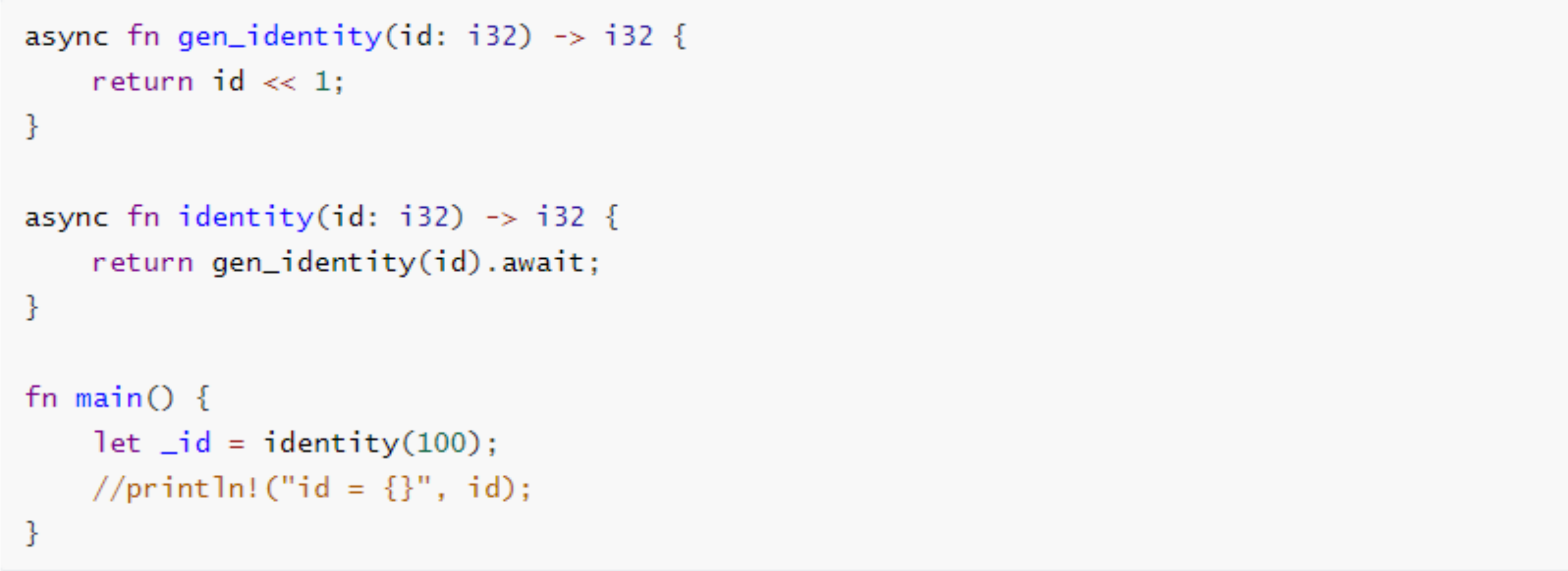


```
/// *.mir
fn identity(_1: i32) -> impl Future<Output = i32> {
    ...
}
```



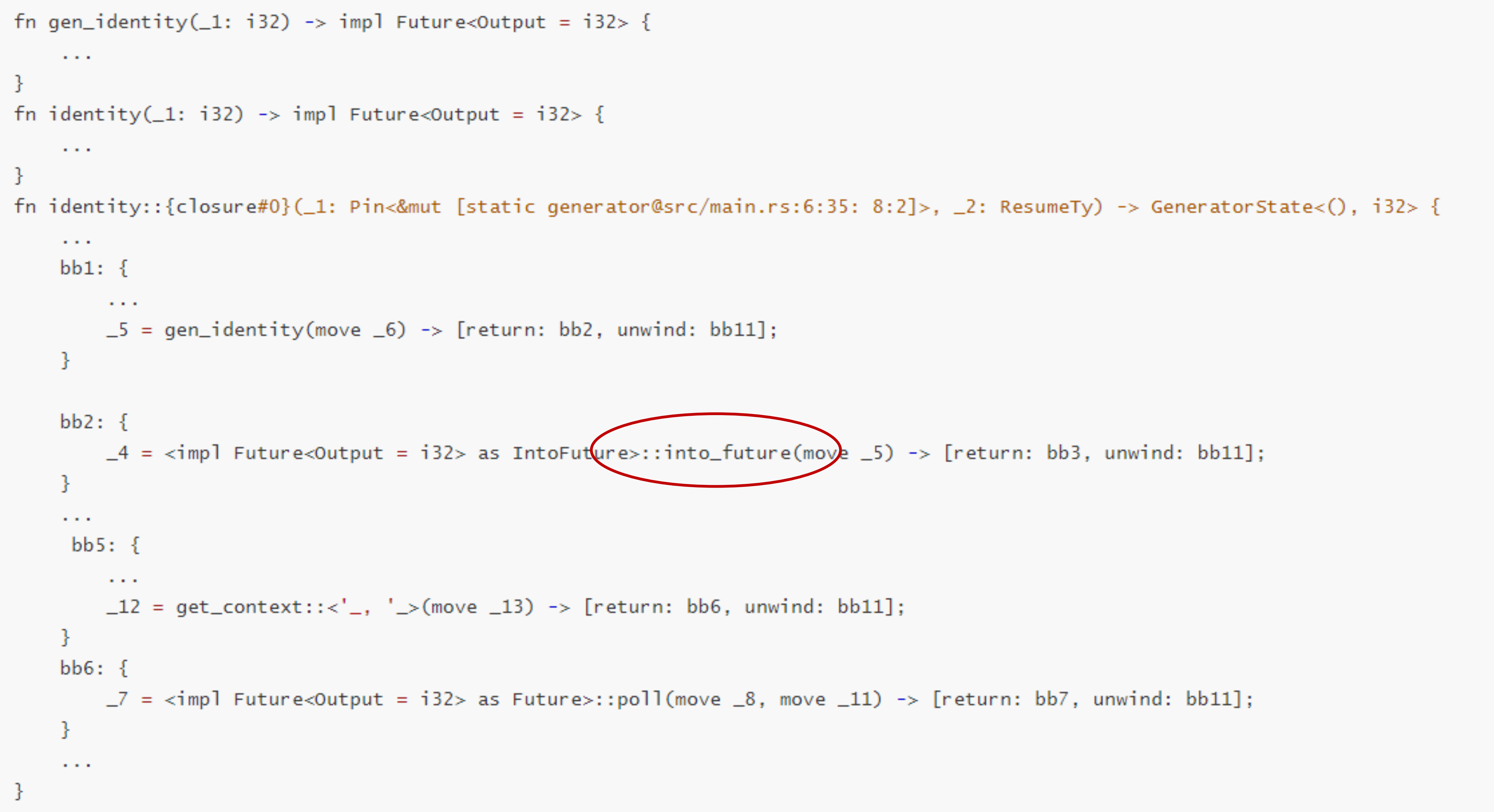
```
pub trait Future {
    type Output;
    fn poll(self: Pin<&mut Self>, cx: &mut Context<'_>) -> Poll<Self::Output>;
}
```

Await表象之下是什么？



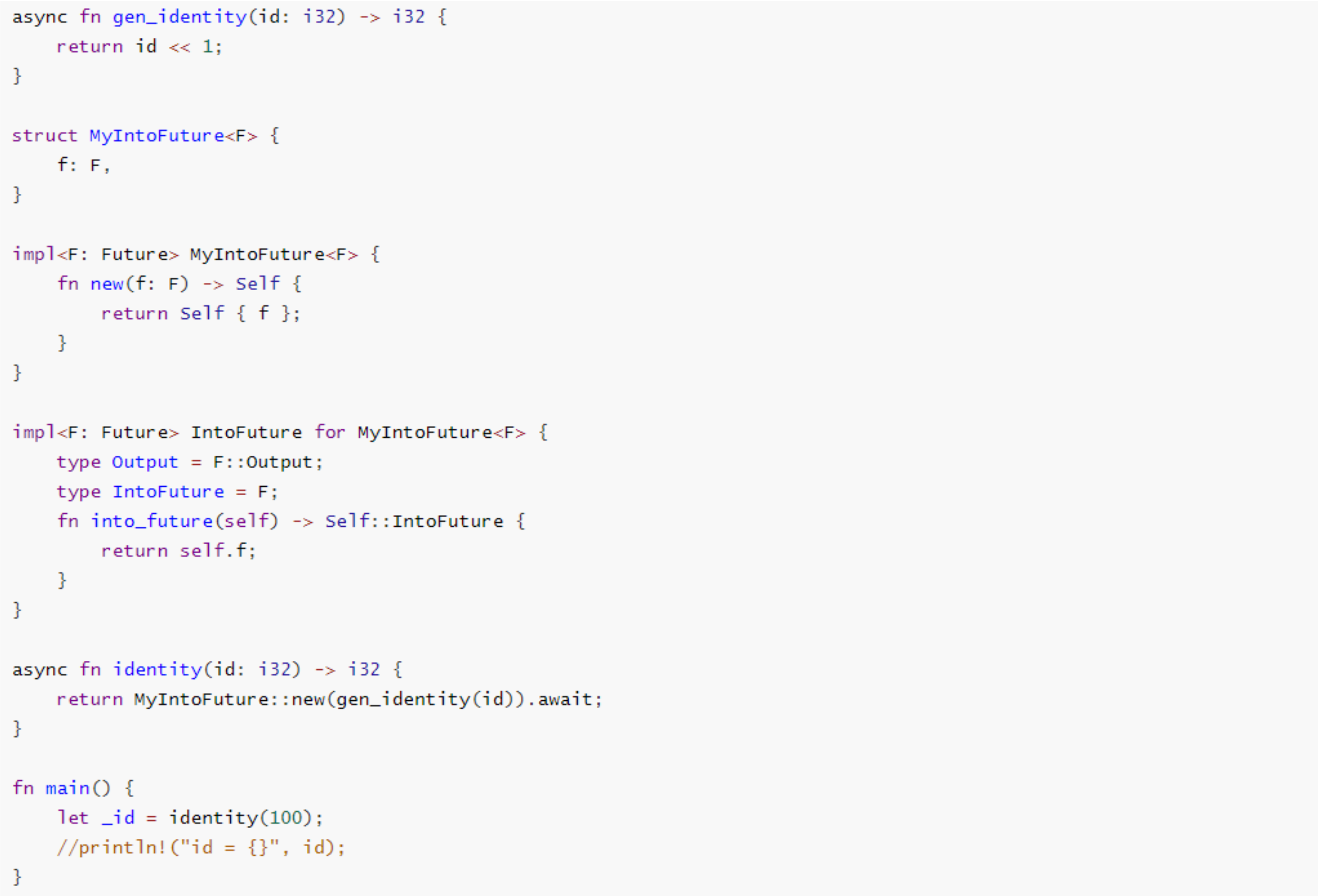
```
async fn gen_identity(id: i32) -> i32 {
    return id << 1;
}

async fn identity(id: i32) -> i32 {
    return gen_identity(id).await;
}

fn main() {
    let _id = identity(100);
    //println!("id = {}", id);
}
```

Await表象之下是什么？



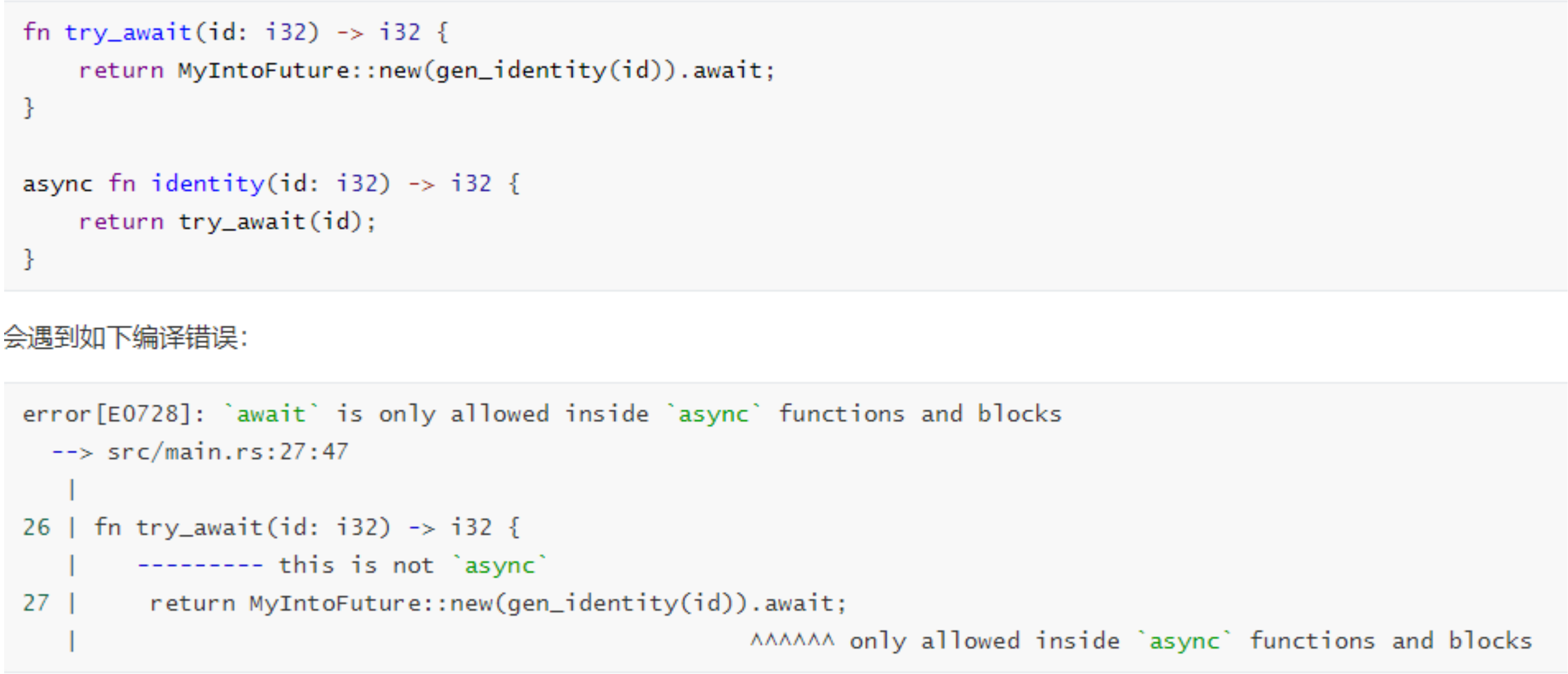
```
fn gen_identity(_1: i32) -> impl Future<Output = i32> {
    ...
}
fn identity(_1: i32) -> impl Future<Output = i32> {
    ...
}
fn identity::{closure#0}(_1: Pin<&mut [static generator@src/main.rs:6:35: 8:2]>, _2: ResumeTy) -> GeneratorState<(), i32> {
    ...
    bb1: {
        ...
        _5 = gen_identity(move _6) -> [return: bb2, unwind: bb11];
    }

    bb2: {
        _4 = <impl Future<Output = i32> as IntoFuture>::into_future(move _5) -> [return: bb3, unwind: bb11];
    }
    ...
     bb5: {
        ...
        _12 = get_context::<'_, '_>(move _13) -> [return: bb6, unwind: bb11];
    }
    bb6: {
        _7 = <impl Future<Output = i32> as Future>::poll(move _8, move _11) -> [return: bb7, unwind: bb11];
    }
    ...
}
```

Await表象之下是什么？



```rust
async fn gen_identity(id: i32) -> i32 {
    return id << 1;
}

struct MyIntoFuture<F> {
    f: F,
}

impl<F: Future> MyIntoFuture<F> {
    fn new(f: F) -> Self {
        return Self { f };
    }
}

impl<F: Future> IntoFuture for MyIntoFuture<F> {
    type Output = F::Output;
    type IntoFuture = F;
    fn into_future(self) -> Self::IntoFuture {
        return self.f;
    }
}

async fn identity(id: i32) -> i32 {
    return MyIntoFuture::new(gen_identity(id)).await;
}

fn main() {
    let _id = identity(100);
    //println!("id = {}", id);
}
```

Await表象之下是什么？

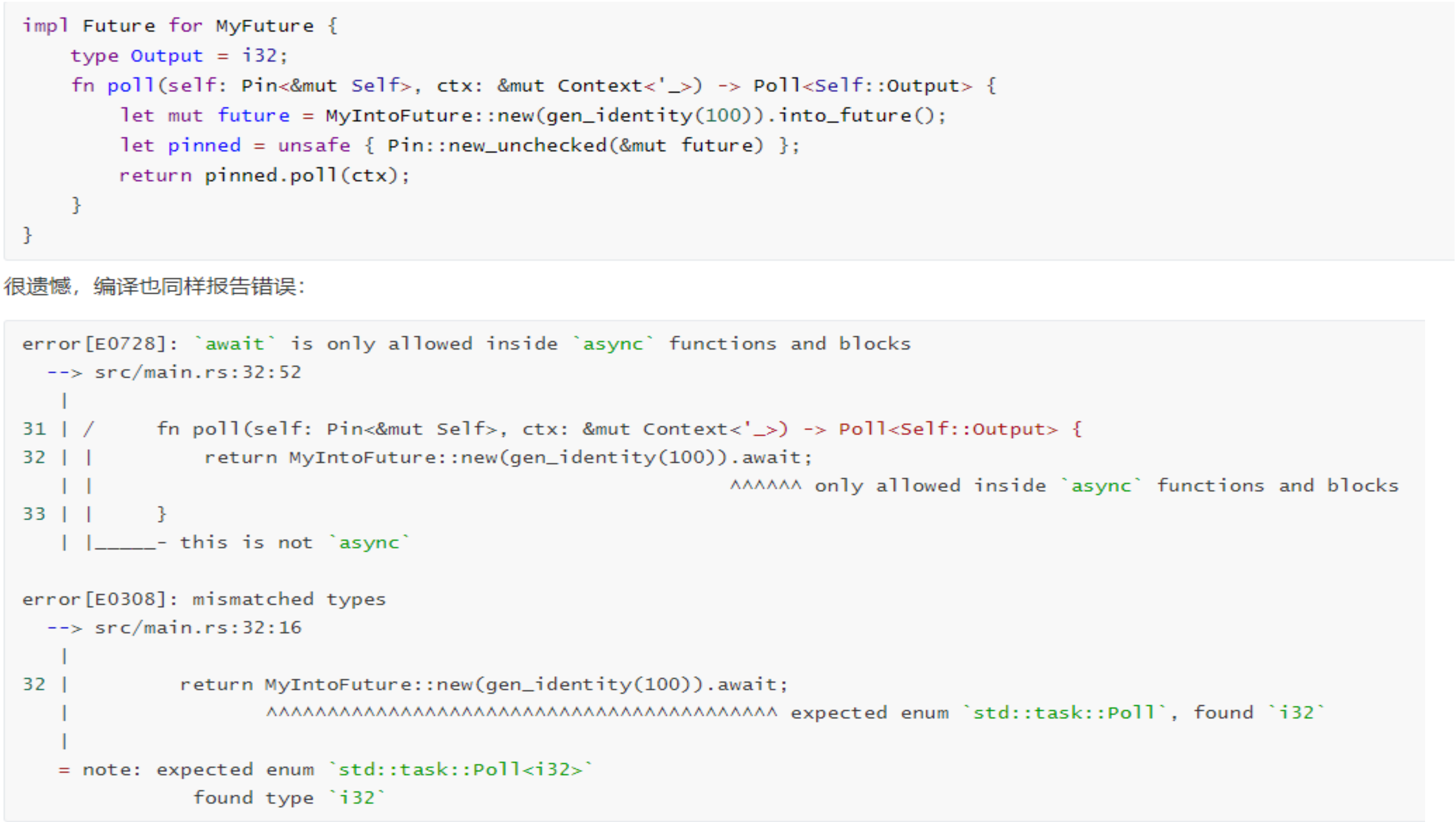


```
fn try_await(id: i32) -> i32 {
    return MyIntoFuture::new(gen_identity(id)).await;
}

async fn identity(id: i32) -> i32 {
    return try_await(id);
}
```

会遇到如下编译错误:

```
error[E0728]: `await` is only allowed inside `async` functions and blocks
  --> src/main.rs:27:47
   |
26 | fn try_await(id: i32) -> i32 {
   |    --------- this is not `async`
27 |     return MyIntoFuture::new(gen_identity(id)).await;
   |                                               ^^^^^^ only allowed inside `async` functions and blocks
```

Await表象之下是什么？

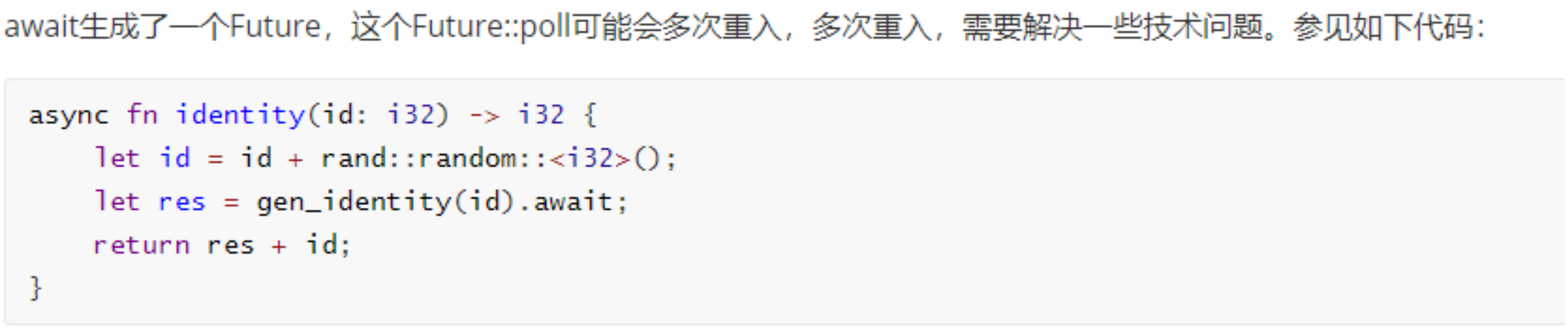


```rust
impl Future for MyFuture {
    type Output = i32;
    fn poll(self: Pin<&mut Self>, ctx: &mut Context<'_>) -> Poll<Self::Output> {
        let mut future = MyIntoFuture::new(gen_identity(100)).into_future();
        let pinned = unsafe { Pin::new_unchecked(&mut future) };
        return pinned.poll(ctx);
    }
}
```

很遗憾，编译也同样报告错误：

```
error[E0728]: `await` is only allowed inside `async` functions and blocks
  --> src/main.rs:32:52
   |
31 | /     fn poll(self: Pin<&mut Self>, ctx: &mut Context<'_>) -> Poll<Self::Output> {
32 | |         return MyIntoFuture::new(gen_identity(100)).await;
   | |                                                    ^^^^^^ only allowed inside `async` functions and blocks
33 | |     }
   | |_____- this is not `async`

error[E0308]: mismatched types
  --> src/main.rs:32:16
   |
32 |         return MyIntoFuture::new(gen_identity(100)).await;
   |                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ expected enum `std::task::Poll`, found `i32`
   |
   = note: expected enum `std::task::Poll<i32>`
              found type `i32`
```

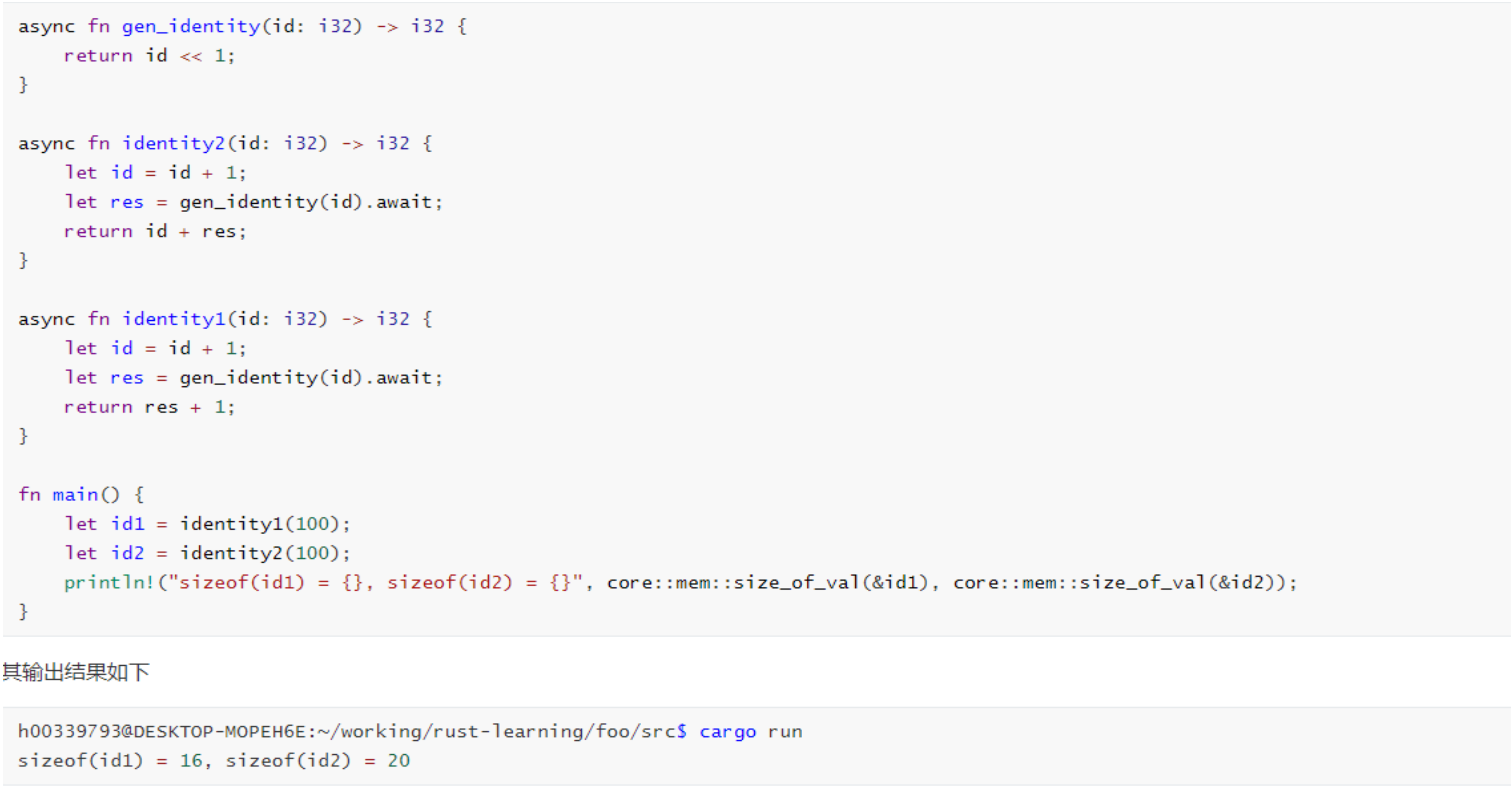
Await和无栈协程



await生成了一个Future，这个Future::poll可能会多次重入，多次重入，需要解决一些技术问题。参见如下代码：

```
async fn identity(id: i32) -> i32 {
    let id = id + rand::random::<i32>();
    let res = gen_identity(id).await;
    return res + id;
}
```

Await和无栈协程

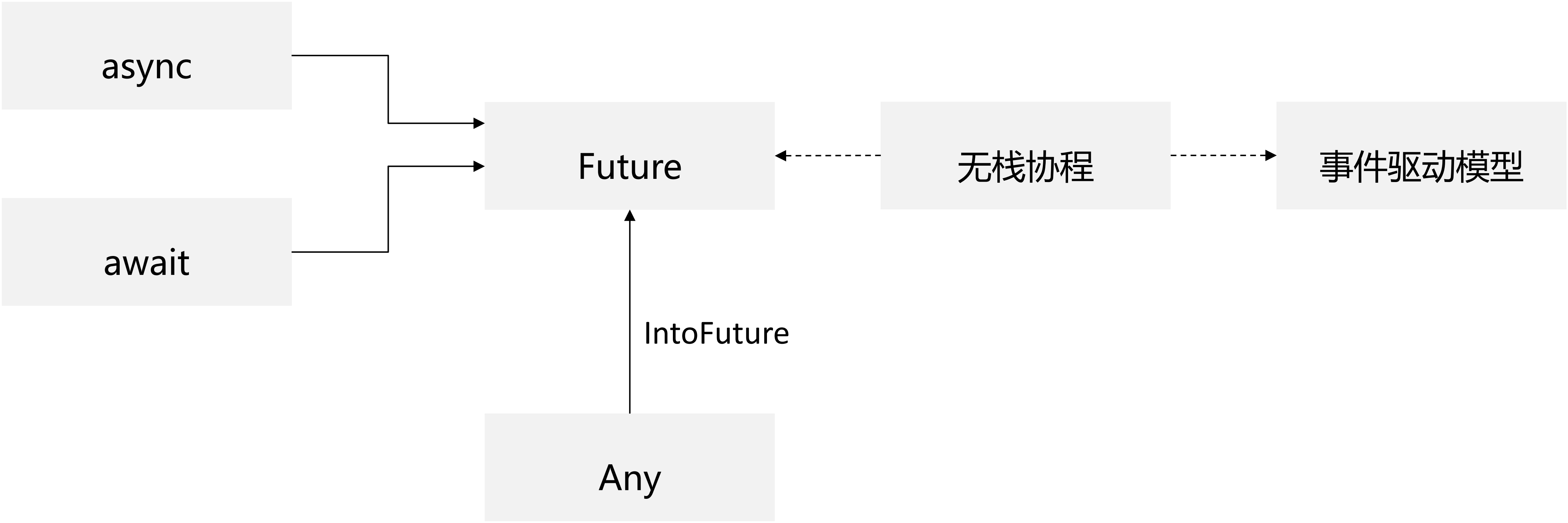


```rust
async fn gen_identity(id: i32) -> i32 {
    return id << 1;
}

async fn identity2(id: i32) -> i32 {
    let id = id + 1;
    let res = gen_identity(id).await;
    return id + res;
}

async fn identity1(id: i32) -> i32 {
    let id = id + 1;
    let res = gen_identity(id).await;
    return res + 1;
}

fn main() {
    let id1 = identity1(100);
    let id2 = identity2(100);
    println!("sizeof(id1) = {}, sizeof(id2) = {}", core::mem::size_of_val(&id1), core::mem::size_of_val(&id2));
}
```

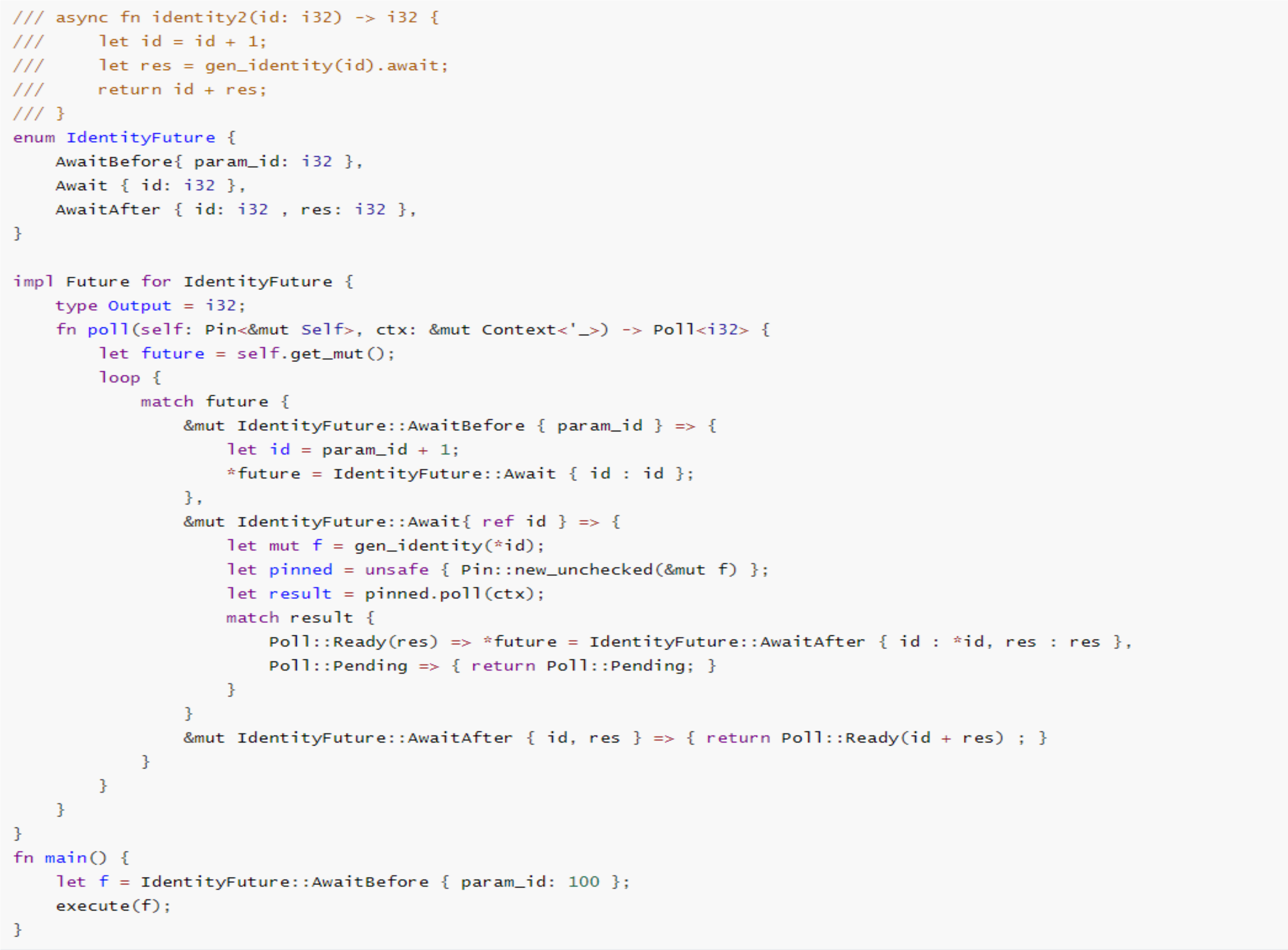
其输出结果如下

```
h00339793@DESKTOP-MOPEH6E:~/working/rust-learning/foo/src$ cargo run
sizeof(id1) = 16, sizeof(id2) = 20
```

初步印象：无栈协程 == 事件驱动模型

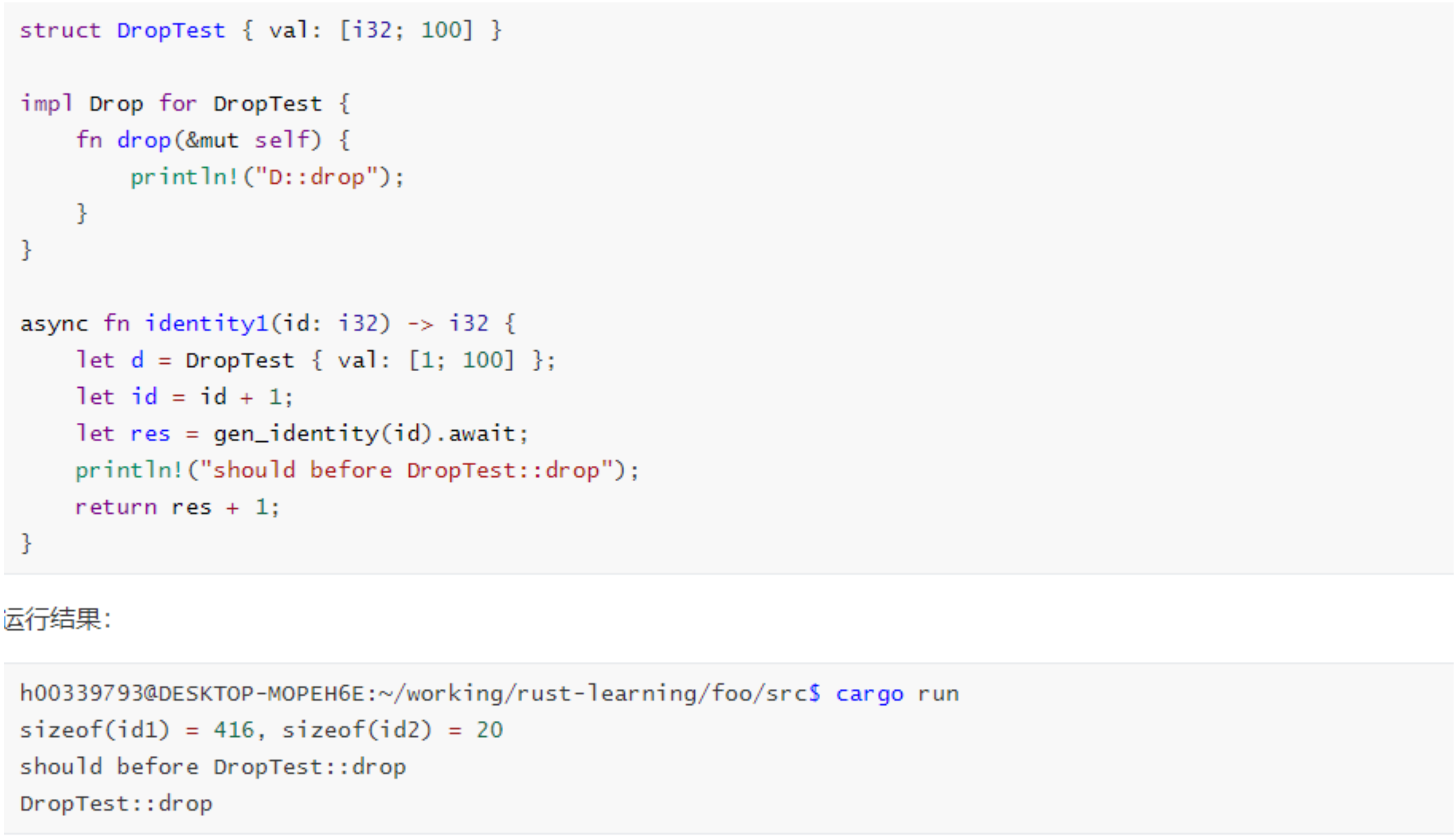


Future状态机版本



```
/// async fn identity2(id: i32) -> i32 {
///     let id = id + 1;
///     let res = gen_identity(id).await;
///     return id + res;
/// }
enum IdentityFuture {
    AwaitBefore{ param_id: i32 },
    Await { id: i32 },
    AwaitAfter { id: i32 , res: i32 },
}

impl Future for IdentityFuture {
    type Output = i32;
    fn poll(self: Pin<&mut Self>, ctx: &mut Context<'_>) -> Poll<i32> {
        let future = self.get_mut();
        loop {
            match future {
                &mut IdentityFuture::AwaitBefore { param_id } => {
                    let id = param_id + 1;
                    *future = IdentityFuture::Await { id : id };
                },
                &mut IdentityFuture::Await{ ref id } => {
                    let mut f = gen_identity(*id);
                    let pinned = unsafe { Pin::new_unchecked(&mut f) };
                    let result = pinned.poll(ctx);
                    match result {
                        Poll::Ready(res) => *future = IdentityFuture::AwaitAfter { id : *id, res : res },
                        Poll::Pending => { return Poll::Pending; }
                    }
                }
                &mut IdentityFuture::AwaitAfter { id, res } => { return Poll::Ready(id + res) ; }
            }
        }
    }
}
fn main() {
    let f = IdentityFuture::AwaitBefore { param_id: 100 };
    execute(f);
}
```

Drop对Future的影响

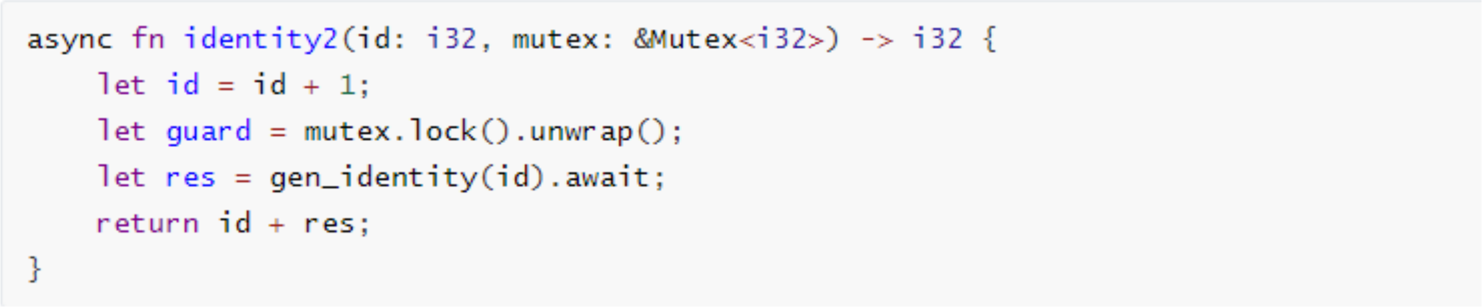


```
struct DropTest { val: [i32; 100] }

impl Drop for DropTest {
    fn drop(&mut self) {
        println!("D::drop");
    }
}

async fn identity1(id: i32) -> i32 {
    let d = DropTest { val: [1; 100] };
    let id = id + 1;
    let res = gen_identity(id).await;
    println!("should before DropTest::drop");
    return res + 1;
}
```

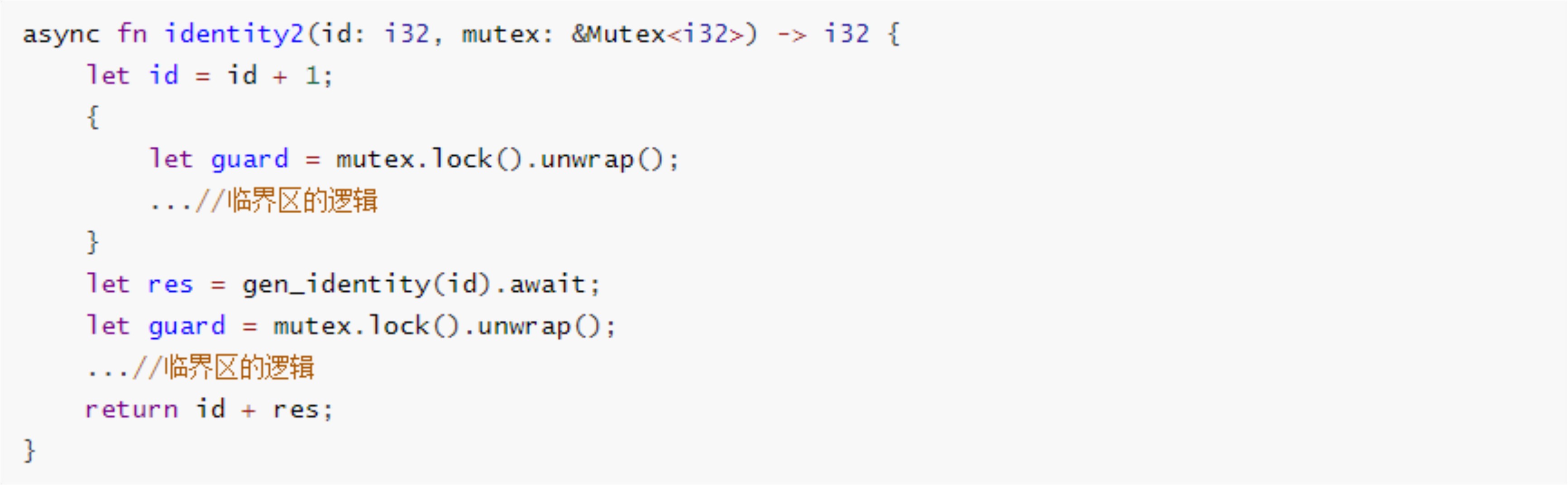
运行结果：

```
h00339793@DESKTOP-MOPEH6E:~/working/rust-learning/foo/src$ cargo run
sizeof(id1) = 416, sizeof(id2) = 20
should before DropTest::drop
DropTest::drop
```

Mutex作用范围

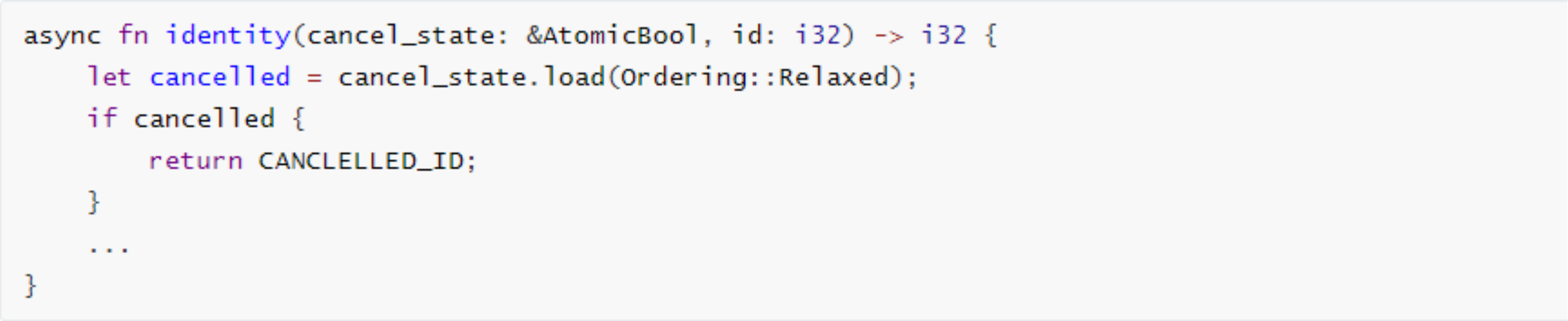


```
async fn identity2(id: i32, mutex: &Mutex<i32>) -> i32 {
    let id = id + 1;
    let guard = mutex.lock().unwrap();
    let res = gen_identity(id).await;
    return id + res;
}
```



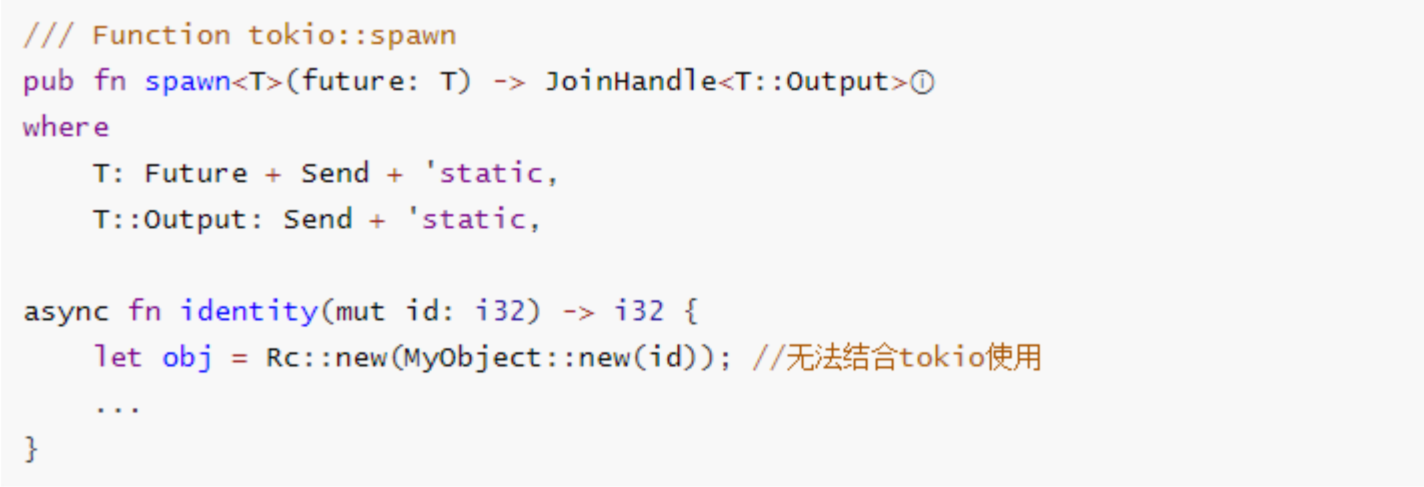
```
async fn identity2(id: i32, mutex: &Mutex<i32>) -> i32 {
    let id = id + 1;
    {
        let guard = mutex.lock().unwrap();
        ...//临界区的逻辑
    }
    let res = gen_identity(id).await;
    let guard = mutex.lock().unwrap();
    ...//临界区的逻辑
    return id + res;
}
```

Future取消



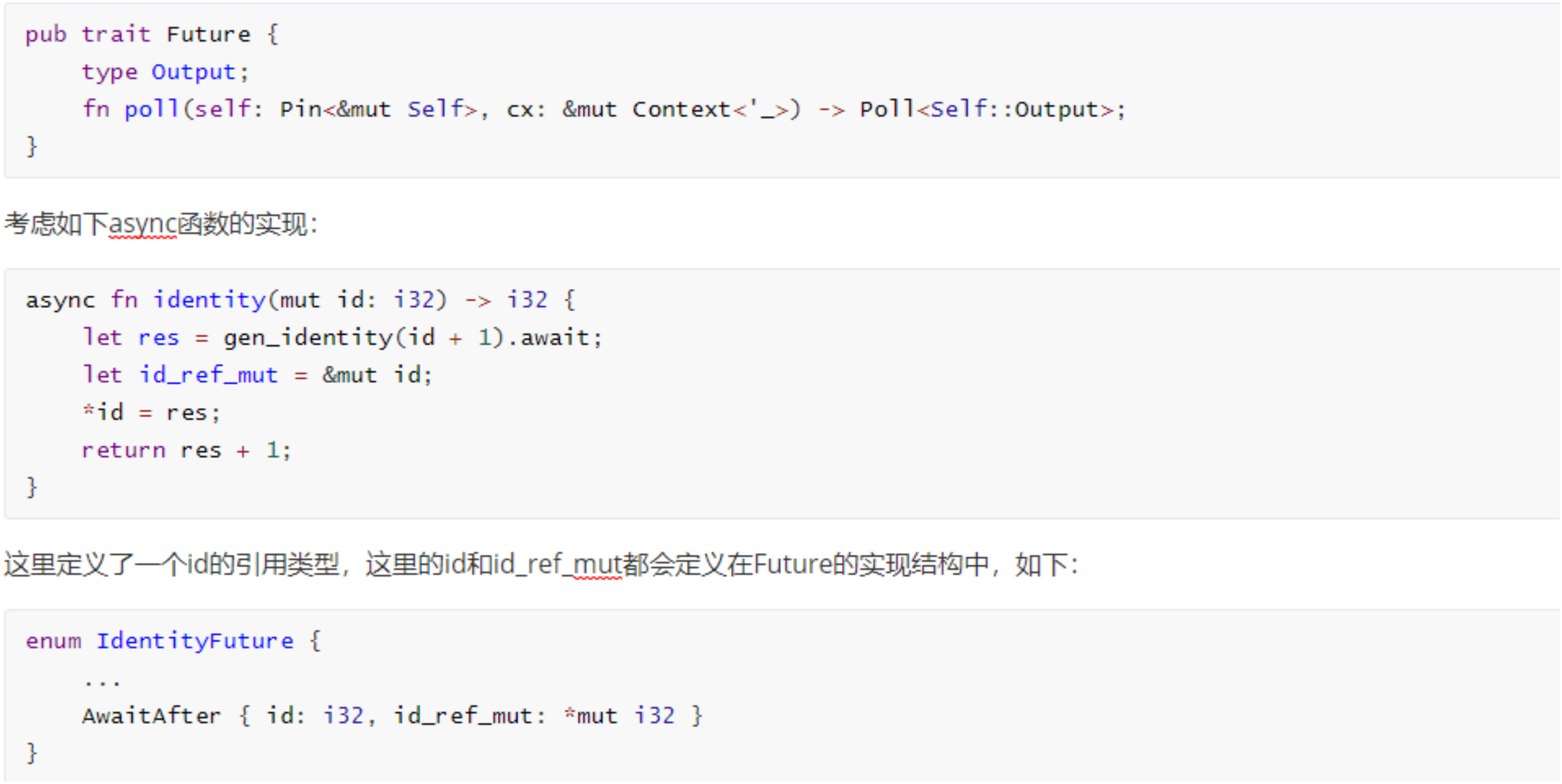
```
async fn identity(cancel_state: &AtomicBool, id: i32) -> i32 {
    let cancelled = cancel_state.load(Ordering::Relaxed);
    if cancelled {
        return CANCLELLED_ID;
    }
    ...
}
```

运行时库的附加约束



```
/// Function tokio::spawn
pub fn spawn<T>(future: T) -> JoinHandle<T::Output>ⓘ
where
    T: Future + Send + 'static,
    T::Output: Send + 'static,

async fn identity(mut id: i32) -> i32 {
    let obj = Rc::new(MyObject::new(id)); //无法结合tokio使用
    ...
}
```

Future和Pin



```
pub trait Future {
    type Output;
    fn poll(self: Pin<&mut Self>, cx: &mut Context<'_>) -> Poll<Self::Output>;
}
```

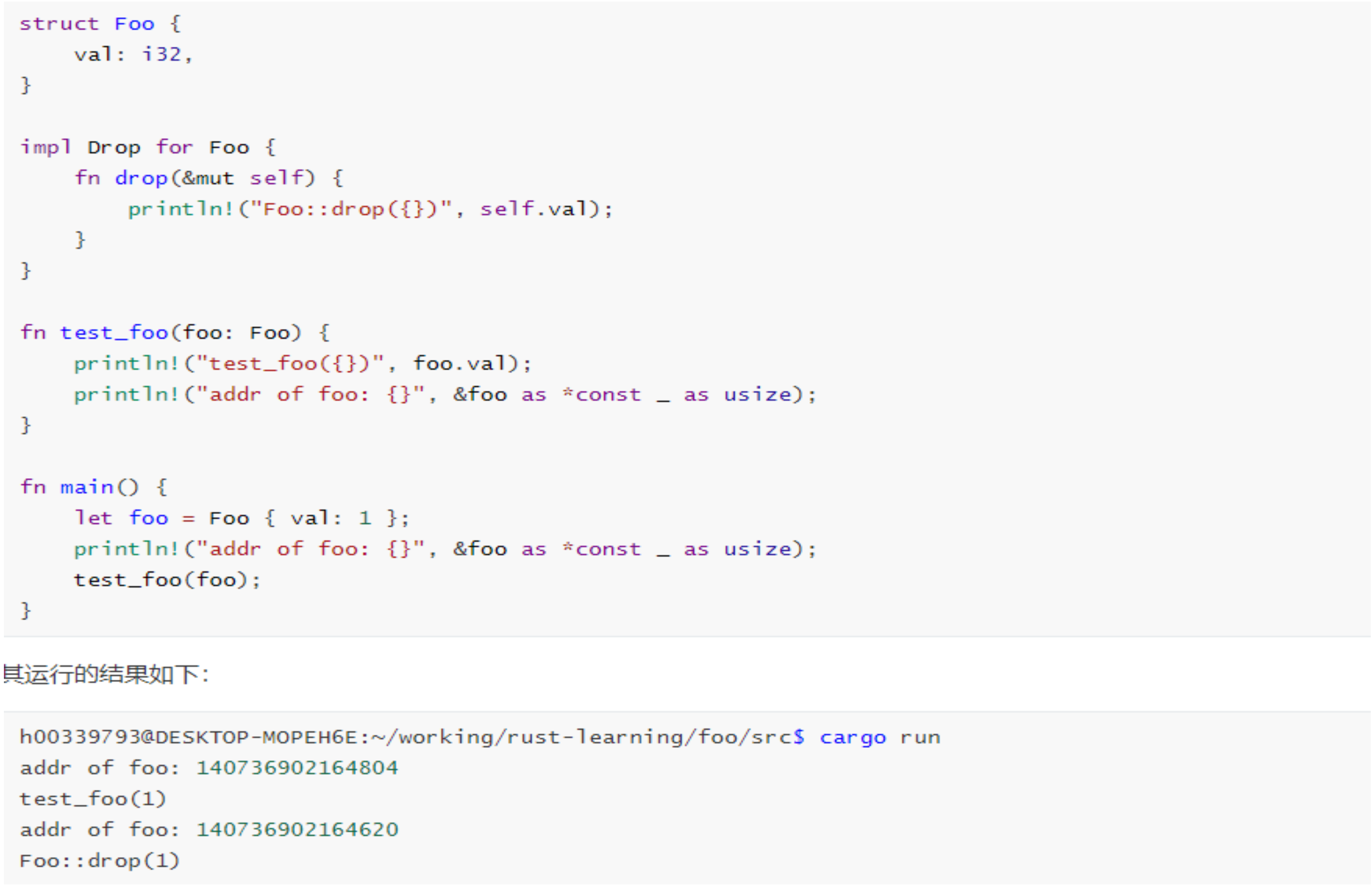
考虑如下async函数的实现：

```
async fn identity(mut id: i32) -> i32 {
    let res = gen_identity(id + 1).await;
    let id_ref_mut = &mut id;
    *id = res;
    return res + 1;
}
```

这里定义了一个id的引用类型，这里的id和id_ref_mut都会定义在Future的实现结构中，如下：

```
enum IdentityFuture {
    ...
    AwaitAfter { id: i32, id_ref_mut: *mut i32 }
}
```

所有权转移：隐式的内存拷贝

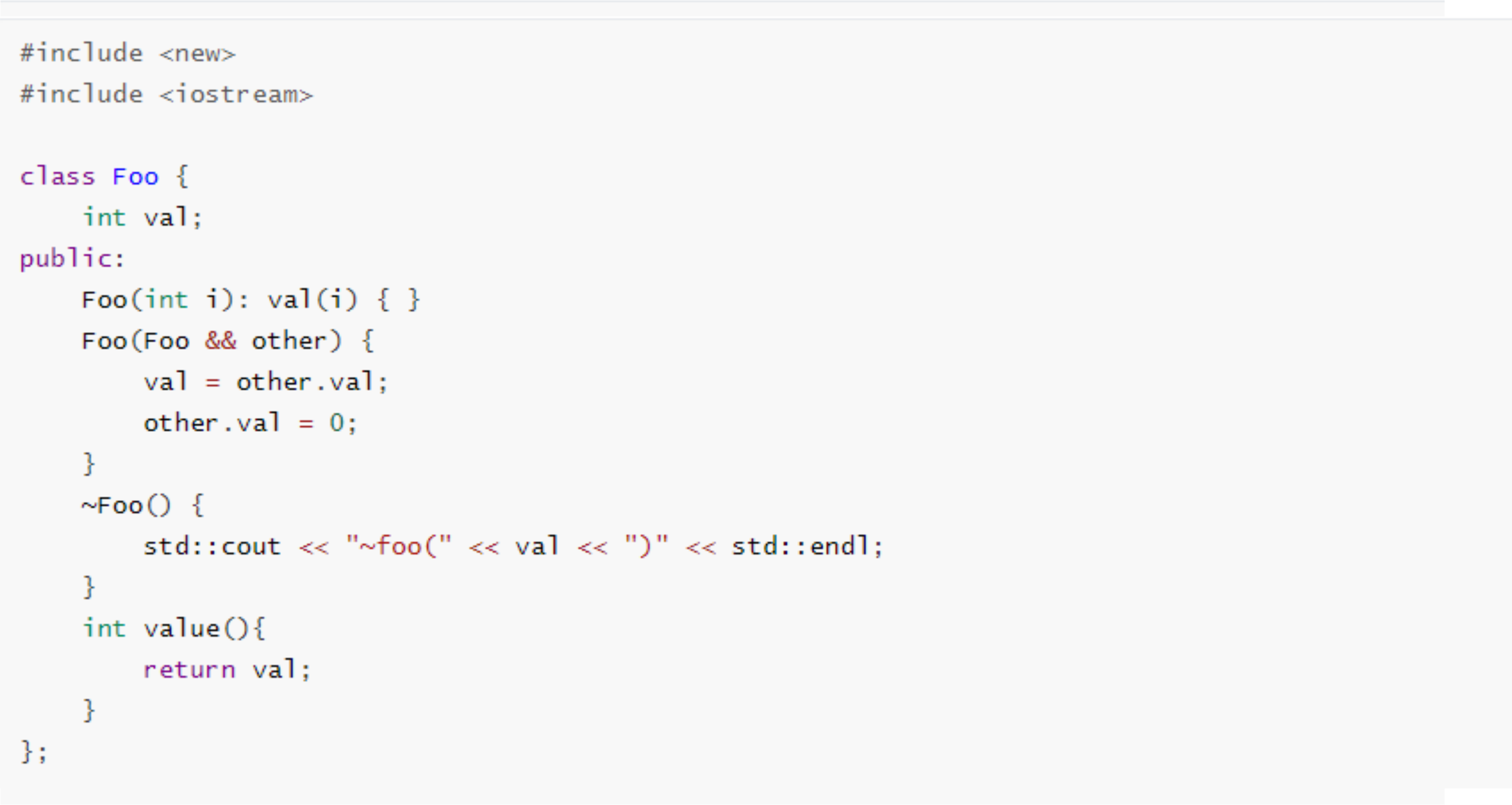


```rust
struct Foo {
    val: i32,
}

impl Drop for Foo {
    fn drop(&mut self) {
        println!("Foo::drop({})", self.val);
    }
}

fn test_foo(foo: Foo) {
    println!("test_foo({})", foo.val);
    println!("addr of foo: {}", &foo as *const _ as usize);
}

fn main() {
    let foo = Foo { val: 1 };
    println!("addr of foo: {}", &foo as *const _ as usize);
    test_foo(foo);
}
```

其运行的结果如下：

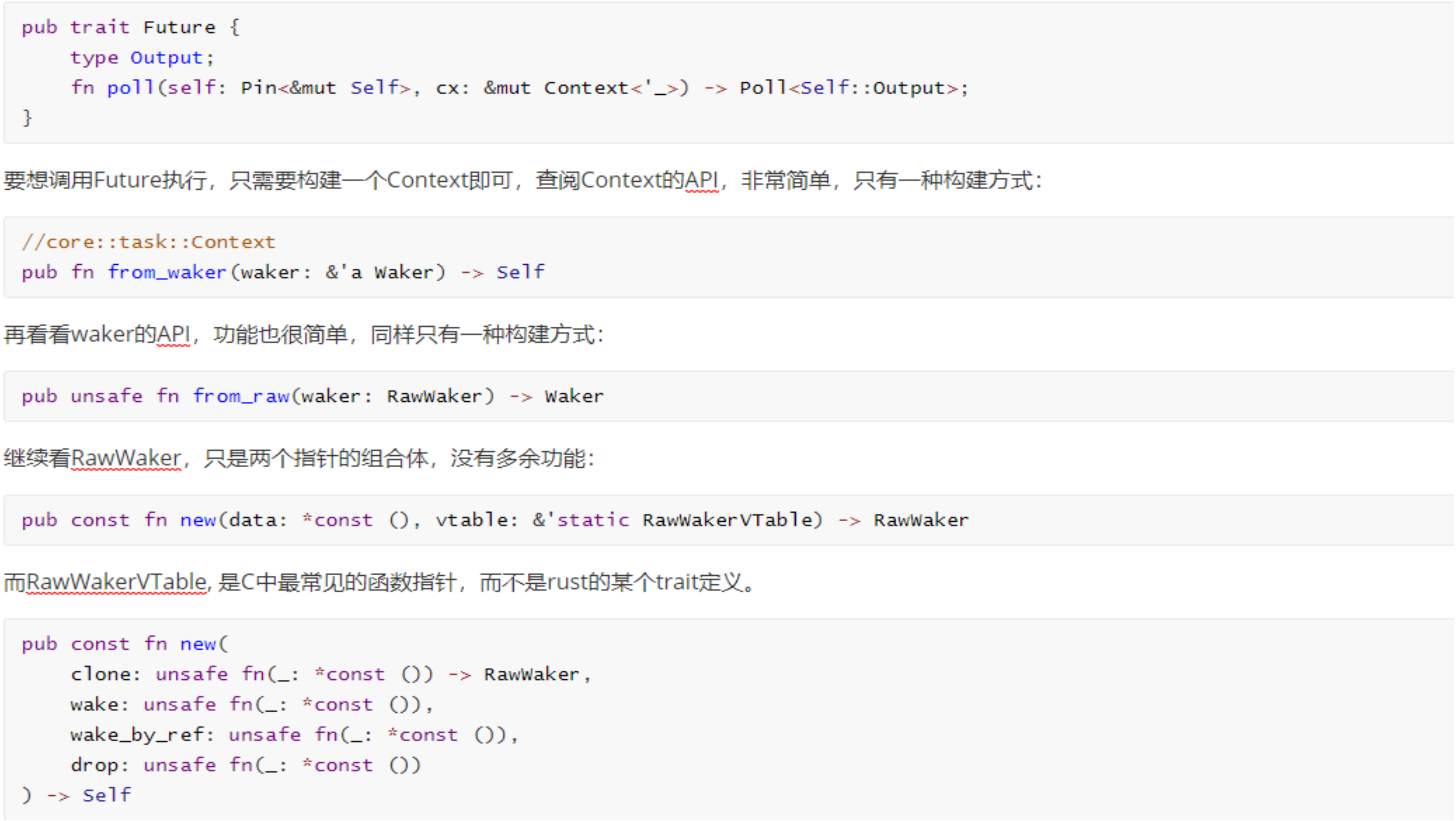
```
h00339793@DESKTOP-MOPEH6E:~/working/rust-learning/foo/src$ cargo run
addr of foo: 140736902164804
test_foo(1)
addr of foo: 140736902164620
Foo::drop(1)
```

Pin：缓解所有权转移的功能缺陷



```cpp
#include <new>
#include <iostream>

class Foo {
    int val;
public:
    Foo(int i): val(i) { }
    Foo(Foo && other) {
        val = other.val;
        other.val = 0;
    }
    ~Foo() {
        std::cout << "~foo(" << val << ")" << std::endl;
    }
    int value(){
        return val;
    }
};
```

Waker和调度框架



```
pub trait Future {
    type Output;
    fn poll(self: Pin<&mut Self>, cx: &mut Context<'_>) -> Poll<Self::Output>;
}
```

要想调用Future执行，只需要构建一个Context即可，查阅Context的API，非常简单，只有一种构建方式：

```
//core::task::Context
pub fn from_waker(waker: &'a Waker) -> Self
```

再看看waker的API，功能也很简单，同样只有一种构建方式：

```
pub unsafe fn from_raw(waker: RawWaker) -> Waker
```

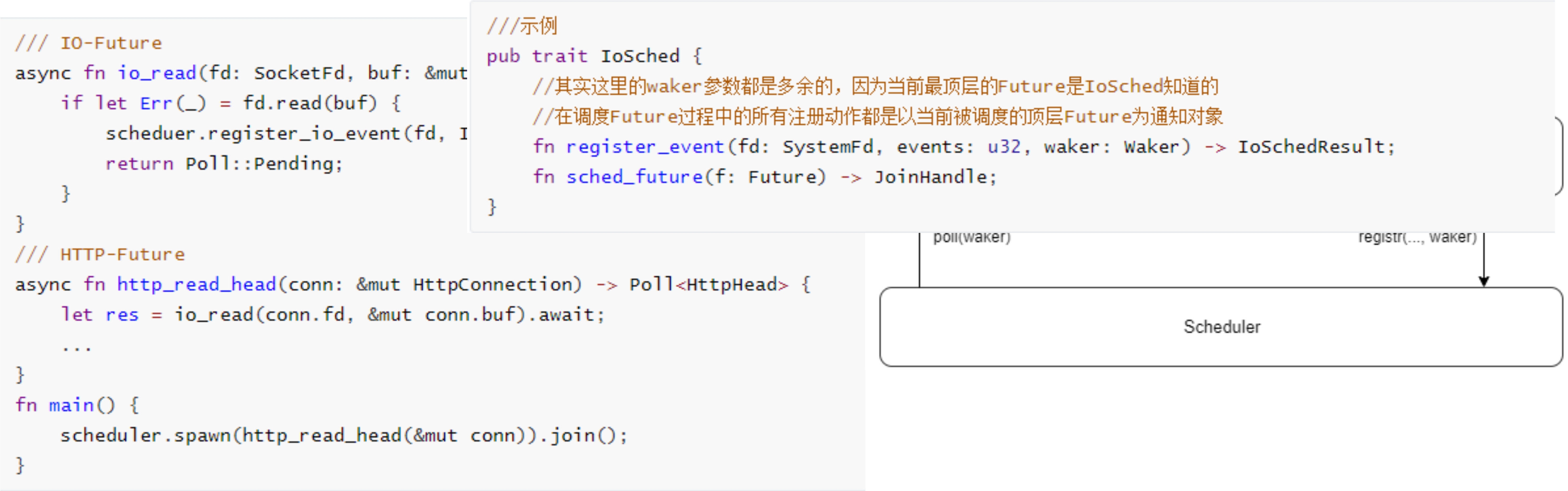
继续看RawWaker，只是两个指针的组合体，没有多余功能：

```
pub const fn new(data: *const (), vtable: &'static RawWakerVTable) -> RawWaker
```

而RawWakerVTable, 是C中最常见的函数指针，而不是rust的某个trait定义。

```
pub const fn new(
    clone: unsafe fn(_: *const ()) -> RawWaker,
    wake: unsafe fn(_: *const ()),
    wake_by_ref: unsafe fn(_: *const ()),
    drop: unsafe fn(_: *const ())
) -> Self
```

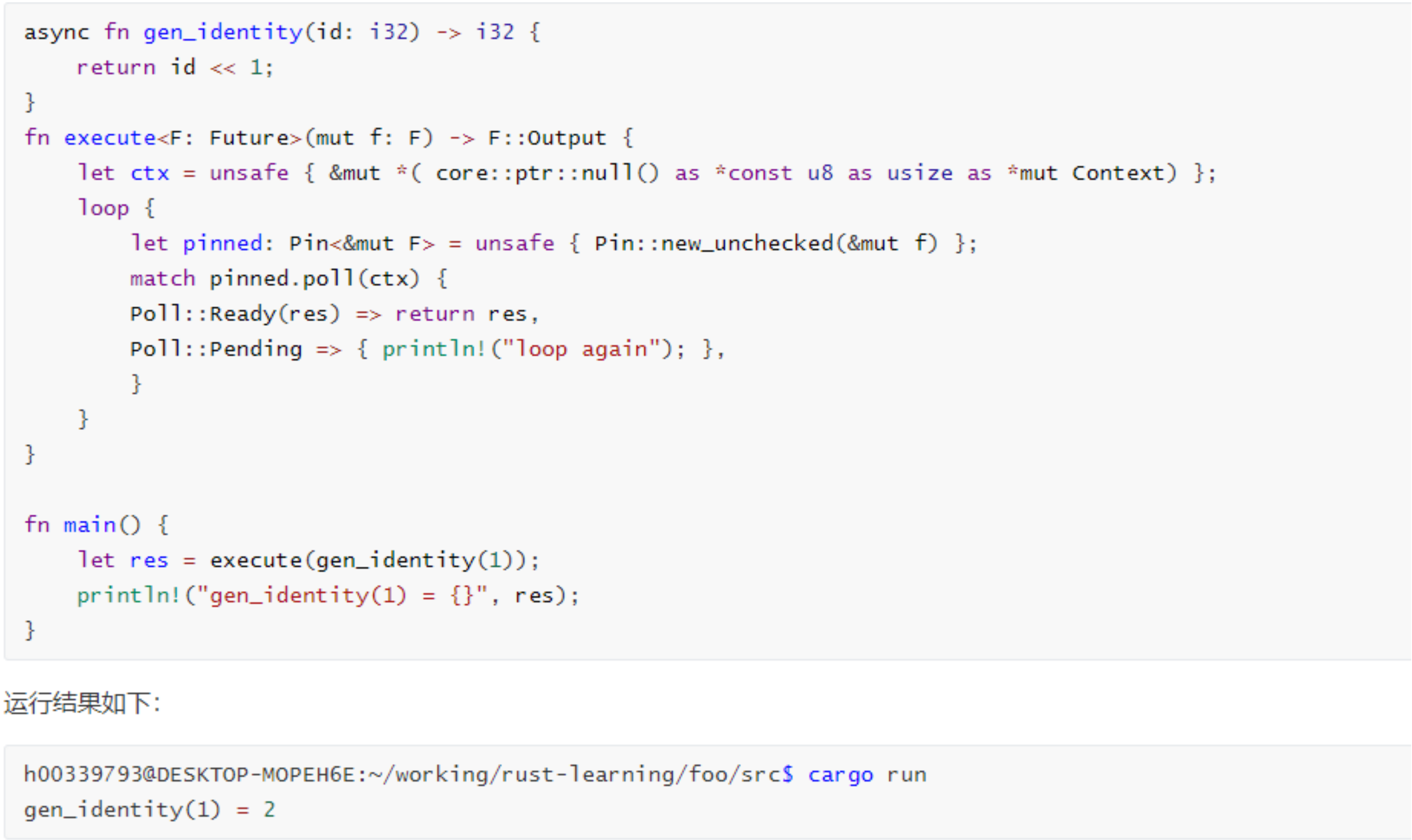
Waker的设计意图



```
/// IO-Future
async fn io_read(fd: SocketFd, buf: &mut
    if let Err(_) = fd.read(buf) {
        scheduer.register_io_event(fd, I
        return Poll::Pending;
    }
}
/// HTTP-Future
async fn http_read_head(conn: &mut HttpConnection) -> Poll<HttpHead> {
    let res = io_read(conn.fd, &mut conn.buf).await;
    ...
}
fn main() {
    scheduler.spawn(http_read_head(&mut conn)).join();
}
```

```
///示例
pub trait IoSched {
    //其实这里的waker参数都是多余的，因为当前最顶层的Future是IoSched知道的
    //在调度Future过程中的所有注册动作都是以当前被调度的顶层Future为通知对象
    fn register_event(fd: SystemFd, events: u32, waker: Waker) -> IoSchedResult;
    fn sched_future(f: Future) -> JoinHandle;
}
```

最简调度器



```
async fn gen_identity(id: i32) -> i32 {
    return id << 1;
}
fn execute<F: Future>(mut f: F) -> F::Output {
    let ctx = unsafe { &mut *( core::ptr::null() as *const u8 as usize as *mut Context) };
    loop {
        let pinned: Pin<&mut F> = unsafe { Pin::new_unchecked(&mut f) };
        match pinned.poll(ctx) {
        Poll::Ready(res) => return res,
        Poll::Pending => { println!("loop again"); },
        }
    }
}

fn main() {
    let res = execute(gen_identity(1));
    println!("gen_identity(1) = {}", res);
}
```

运行结果如下:

```
h00339793@DESKTOP-MOPEH6E:~/working/rust-learning/foo/src$ cargo run
gen_identity(1) = 2
```

观点总结：不神话不贬低，扬其所长，避其所短

- □ 无栈协程即事件驱动模型，async/await开发效率更胜一筹
- □ 运行性能的关键取决于运行时框架的实现，所有语言面临同样挑战
  - ✓ 减少内存拷贝
  - ✓ 提升Cache命中率
  - ✓ 减少上下文切换
  - ✓ 提升内存分配效率
  - ✓ 高性能线程同步机制
  - ✓ …
- □ RUST异步框架的约束
  - ✓ 解耦设计待完善：Waker设计值得商榷，运行时框架接口缺失
  - ✓ 性能隐患：所有权转移带来的隐式内存拷贝，堆数据初始化效率
  - ✓ 功能约束：具体运行时功能接口约束
  - ✓ …

Thank you

www.huawei.com

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.