devfest 2022

pook-nav-toggle"
dden="" fixed="" aria-label="Hide
Hide side navigation"

Go vs Rust

Google Developer Groups
Korea, Repulic of

장창서
DevOps Engineer, Riiid

소개

Go

Go is expressive, concise, clean, and efficient.

Rust

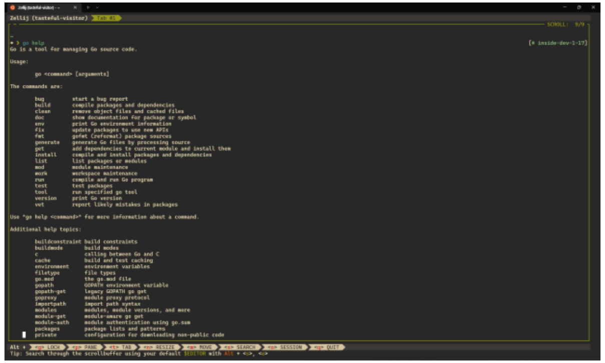
The Rust programming language helps you write faster, more reliable software.

devfest 2022

툴링

Go

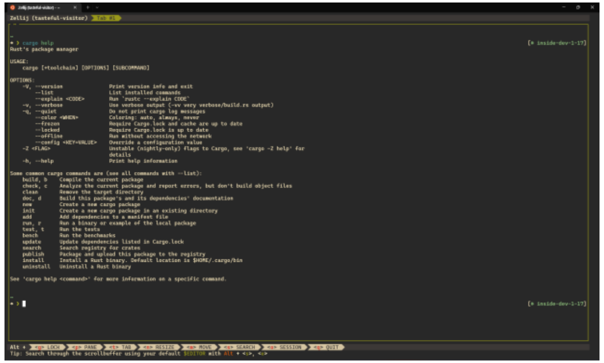
```
go mod init foo/example.com
go get <package>
go run
go build
```



```
Zellij (tasteful-visitor)  Tab #1
                                                                        SCROLL:  9/9
~
* > go help                                                          [* inside-dev-1-17]
Go is a tool for managing Go source code.

Usage:

        go <command> [arguments]

The commands are:

        bug         start a bug report
        build       compile packages and dependencies
        clean       remove object files and cached files
        doc         show documentation for package or symbol
        env         print Go environment information
        fix         update packages to use new APIs
        fmt         gofmt (reformat) package sources
        generate    generate Go files by processing source
        get         add dependencies to current module and install them
        install     compile and install packages and dependencies
        list        list packages or modules
        mod         module maintenance
        work        workspace maintenance
        run         compile and run Go program
        test        test packages
        tool        run specified go tool
        version     print Go version
        vet         report likely mistakes in packages

Use "go help <command>" for more information about a command.

Additional help topics:

        buildconstraint build constraints
        buildmode       build modes
        c               calling between Go and C
        cache           build and test caching
        environment     environment variables
        filetype        file types
        go.mod          the go.mod file
        gopath          GOPATH environment variable
        gopath-get      legacy GOPATH go get
        goproxy         module proxy protocol
        importpath      import path syntax
        modules         modules, module versions, and more
        module-get      module-aware go get
        module-auth     module authentication using go.sum
        packages        package lists and patterns
        private         configuration for downloading non-public code
Alt + <g> LOCK  <p> PANE  <t> TAB  <n> RESIZE  <m> MOVE  <s> SEARCH  <o> SESSION  <q> QUIT
Tip: Search through the scrollbuffer using your default $EDITOR with Alt + <s>, <e>
```

Rust

```
cargo new
cargo add <package>
cargo run
cargo build
```



```
Zellij (tasteful-visitor)  Tab #1
~
* > cargo help                                                       [* inside-dev-1-17]
Rust's package manager

USAGE:
    cargo [+toolchain] [OPTIONS] [SUBCOMMAND]

OPTIONS:
    -V, --version               Print version info and exit
        --list                  List installed commands
        --explain <CODE>        Run `rustc --explain CODE`
    -v, --verbose               Use verbose output (-vv very verbose/build.rs output)
    -q, --quiet                 Do not print cargo log messages
        --color <WHEN>          Coloring: auto, always, never
        --frozen                Require Cargo.lock and cache are up to date
        --locked                Require Cargo.lock is up to date
        --offline               Run without accessing the network
        --config <KEY=VALUE>    Override a configuration value
    -Z <FLAG>                   Unstable (nightly-only) flags to Cargo, see 'cargo -Z help' for
                                details
    -h, --help                  Print help information

Some common cargo commands are (see all commands with --list):
    build, b    Compile the current package
    check, c    Analyze the current package and report errors, but don't build object files
    clean       Remove the target directory
    doc, d      Build this package's and its dependencies' documentation
    new         Create a new cargo package
    init        Create a new cargo package in an existing directory
    add         Add dependencies to a manifest file
    run, r      Run a binary or example of the local package
    test, t     Run the tests
    bench       Run the benchmarks
    update      Update dependencies listed in Cargo.lock
    search      Search registry for crates
    publish     Package and upload this package to the registry
    install     Install a Rust binary. Default location is $HOME/.cargo/bin
    uninstall   Uninstall a Rust binary

See 'cargo help <command>' for more information on a specific command.

~
* >                                                                  [* inside-dev-1-17]

Alt + <g> LOCK  <p> PANE  <t> TAB  <n> RESIZE  <m> MOVE  <s> SEARCH  <o> SESSION  <q> QUIT
Tip: Search through the scrollbuffer using your default $EDITOR with Alt + <s>, <e>
```

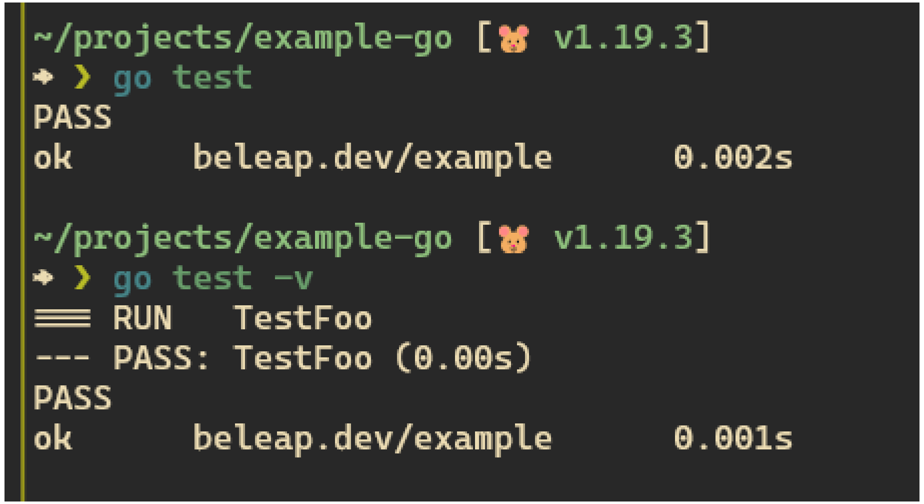
개발환경

기본적으로 go와 rust 모두 lsp 서버를 제공

다만, go는 goland 같은 IDE가 있지만 rust는 아직 없음.
(IntelliJ IDEA에서 Plugin으로 제공되고 있음)

테스트

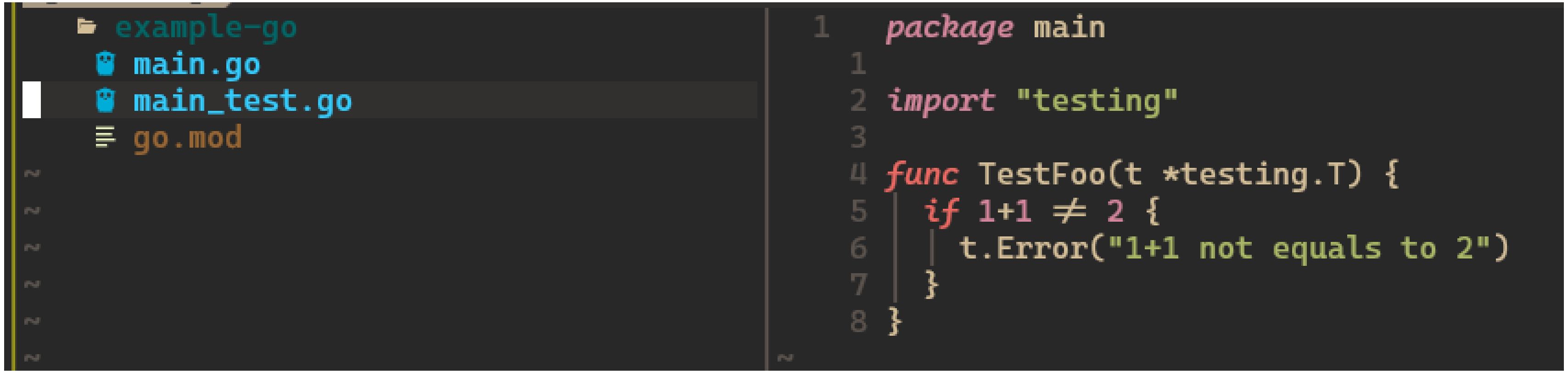
go와 rust 모두 기본으로 test 기능을 제공하고 있습니다.



```
~/projects/example-go [🐹 v1.19.3]
➜ ❯ go test
PASS
ok      beleap.dev/example      0.002s

~/projects/example-go [🐹 v1.19.3]
➜ ❯ go test -v
=== RUN   TestFoo
--- PASS: TestFoo (0.00s)
PASS
ok      beleap.dev/example      0.001s
```

테스트

go와 rust 모두 기본으로 test 기능을 제공하고 있습니다.

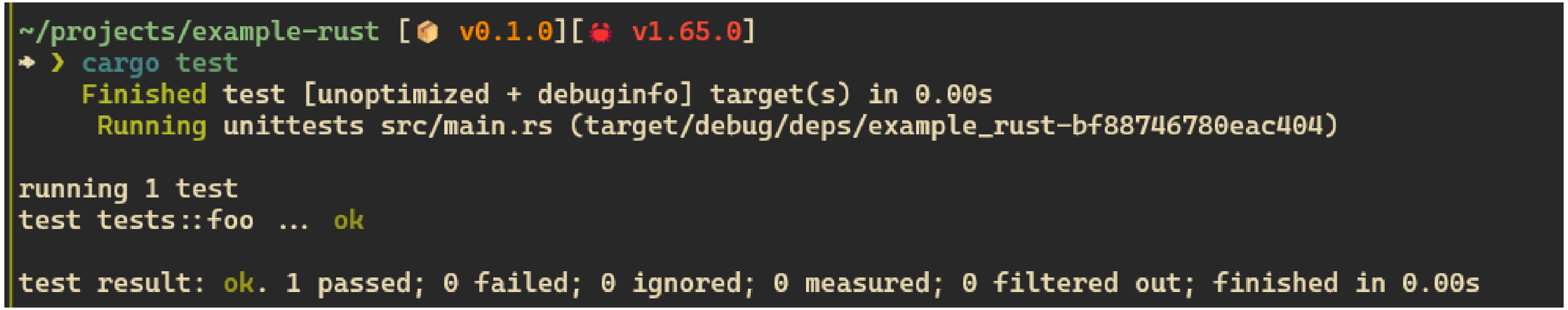


```
example-go
  main.go
  main_test.go
  go.mod
```

```go
package main

import "testing"

func TestFoo(t *testing.T) {
	if 1+1 ≠ 2 {
		t.Error("1+1 not equals to 2")
	}
}
```

테스트

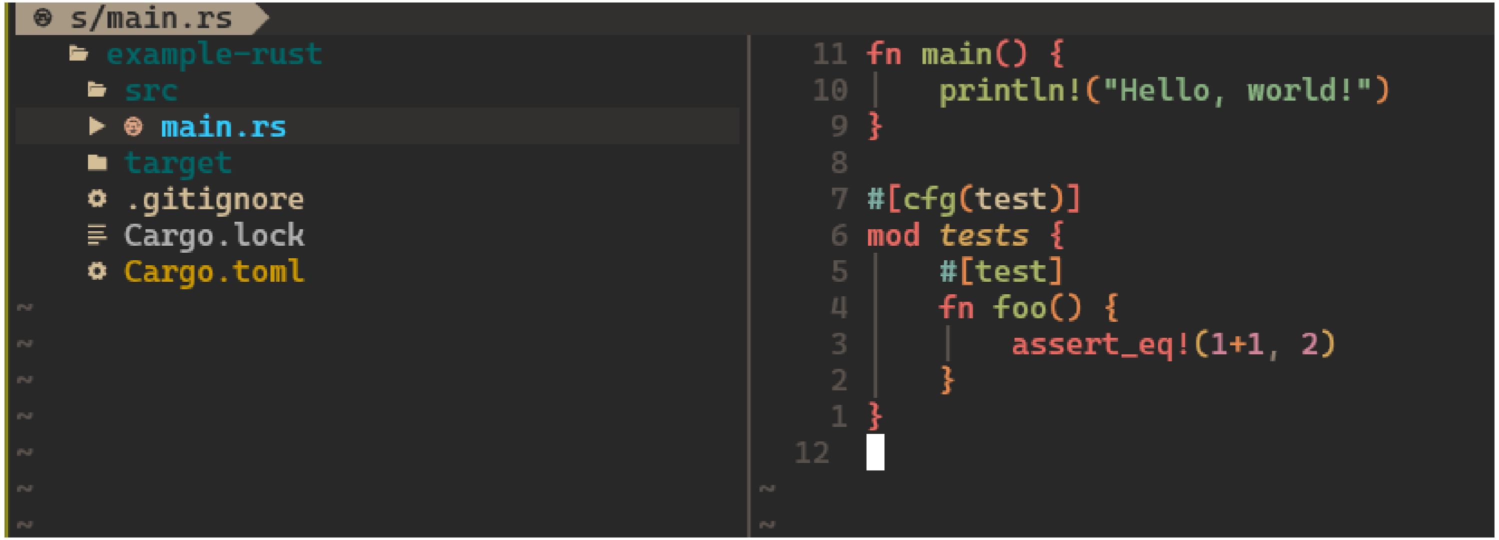
go와 rust 모두 기본으로 test 기능을 제공하고 있습니다.



```
~/projects/example-rust [📦 v0.1.0][🦀 v1.65.0]
➜ ❯ cargo test
    Finished test [unoptimized + debuginfo] target(s) in 0.00s
     Running unittests src/main.rs (target/debug/deps/example_rust-bf88746780eac404)

running 1 test
test tests::foo ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

테스트

go와 rust 모두 기본으로 test 기능을 제공하고 있습니다.



```
s/main.rs
  example-rust
    src
  ▶ main.rs
    target
    .gitignore
    Cargo.lock
    Cargo.toml
```

```rust
fn main() {
    println!("Hello, world!")
}

#[cfg(test)]
mod tests {
    #[test]
    fn foo() {
        assert_eq!(1+1, 2)
    }
}
```

devfest 2022

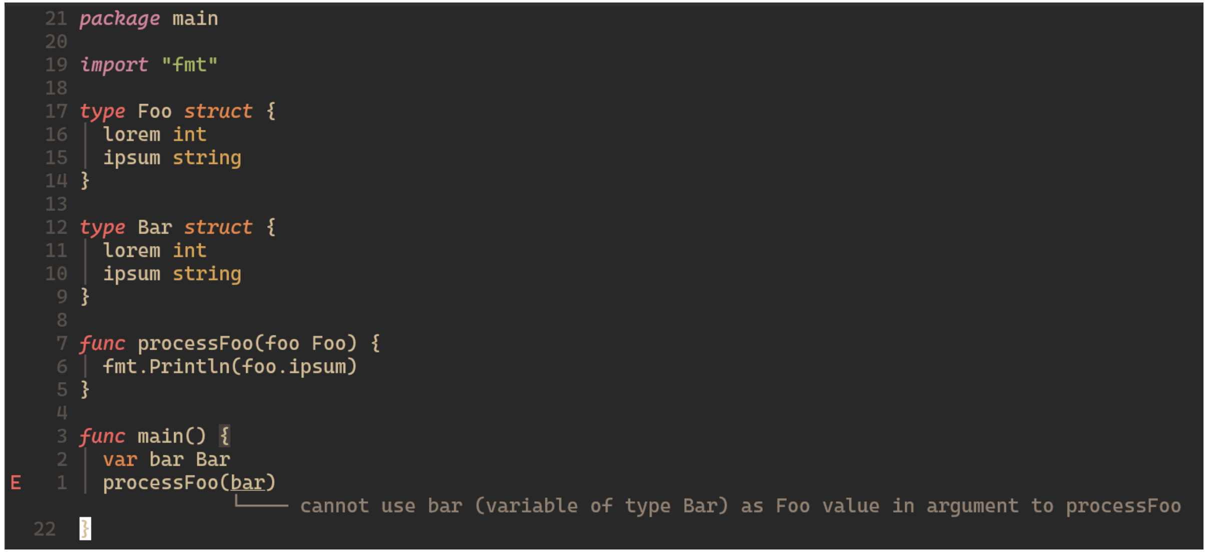
devfest 2022

언어적인 특성

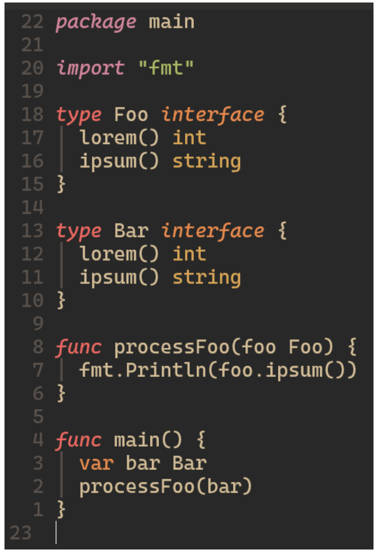
타입

Typing

Go: Structural Typing



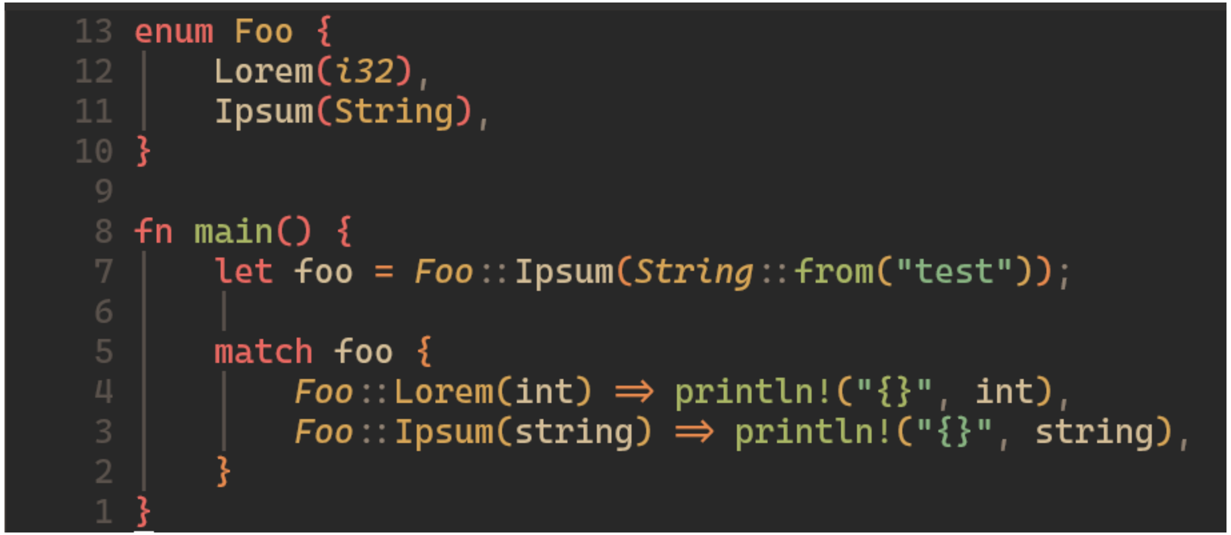
```go
package main

import "fmt"

type Foo struct {
    lorem int
    ipsum string
}

type Bar struct {
    lorem int
    ipsum string
}

func processFoo(foo Foo) {
    fmt.Println(foo.ipsum)
}

func main() {
    var bar Bar
E   processFoo(bar)
              └──── cannot use bar (variable of type Bar) as Foo value in argument to processFoo
}
```



```go
package main

import "fmt"

type Foo interface {
    lorem() int
    ipsum() string
}

type Bar interface {
    lorem() int
    ipsum() string
}

func processFoo(foo Foo) {
    fmt.Println(foo.ipsum())
}

func main() {
    var bar Bar
    processFoo(bar)
}
```

Typing

Rust: Pattern Matching



```rust
enum Foo {
    Lorem(i32),
    Ipsum(String),
}

fn main() {
    let foo = Foo::Ipsum(String::from("test"));

    match foo {
        Foo::Lorem(int) => println!("{}", int),
        Foo::Ipsum(string) => println!("{}", string),
    }
}
```

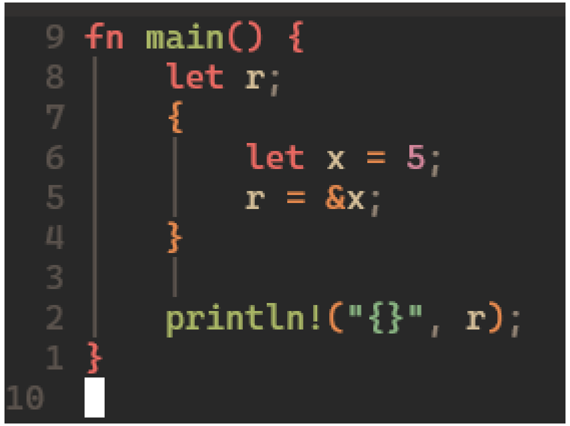
devfest 2022

메모리 관리

Lifetime

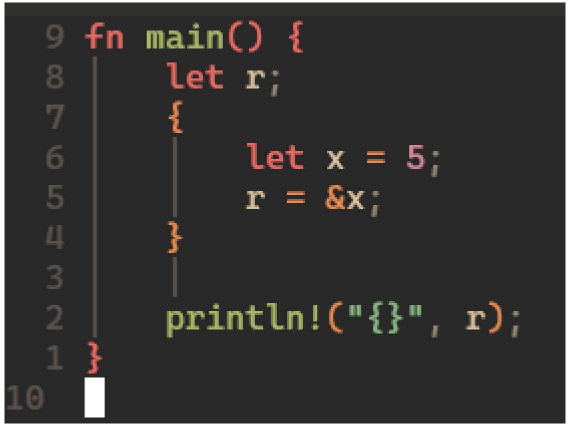
lifetime은 참조가 유효한 범위입니다.

Lifetime

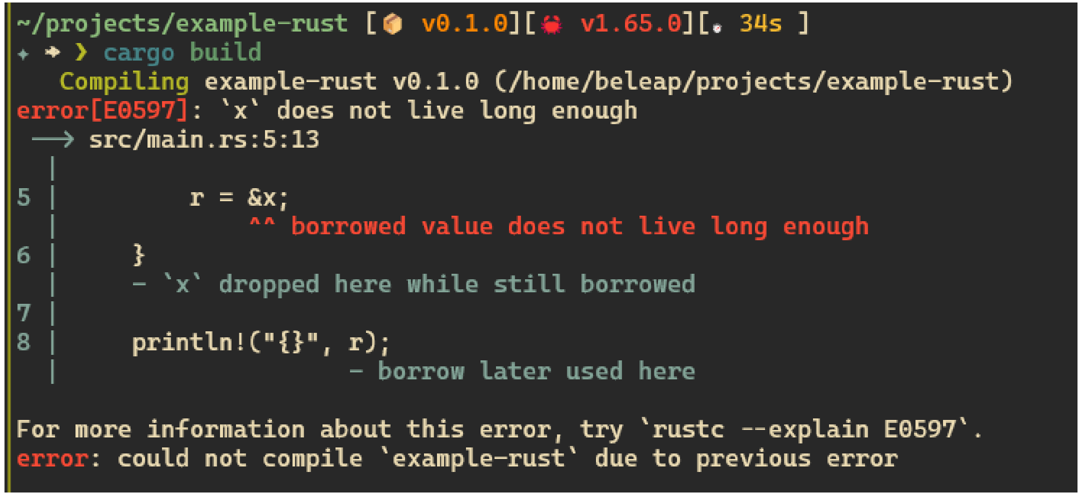


```
fn main() {
    let r;
    {
        let x = 5;
        r = &x;
    }

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

Lifetime



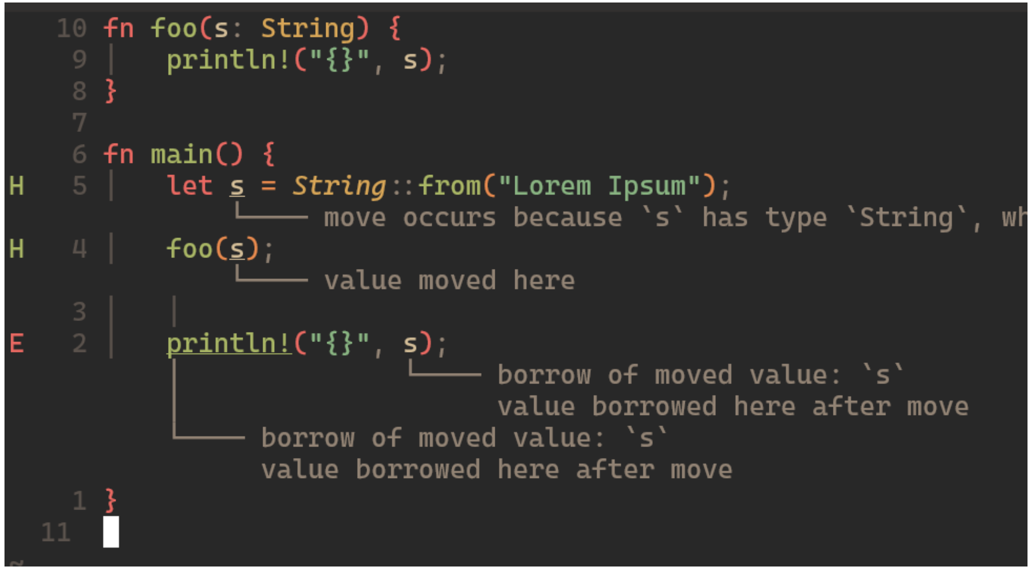
```rust
fn main() {
    let r;
    {
        let x = 5;
        r = &x;
    }

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



```
~/projects/example-rust [📦 v0.1.0][🦀 v1.65.0][• 34s ]
✦ ➜ ❯ cargo build
   Compiling example-rust v0.1.0 (/home/beleap/projects/example-rust)
error[E0597]: `x` does not live long enough
 --> src/main.rs:5:13
  |
5 |         r = &x;
  |             ^^ borrowed value does not live long enough
6 |     }
  |     - `x` dropped here while still borrowed
7 |
8 |     println!("{}", r);
  |                    - borrow later used here

For more information about this error, try `rustc --explain E0597`.
error: could not compile `example-rust` due to previous error
```

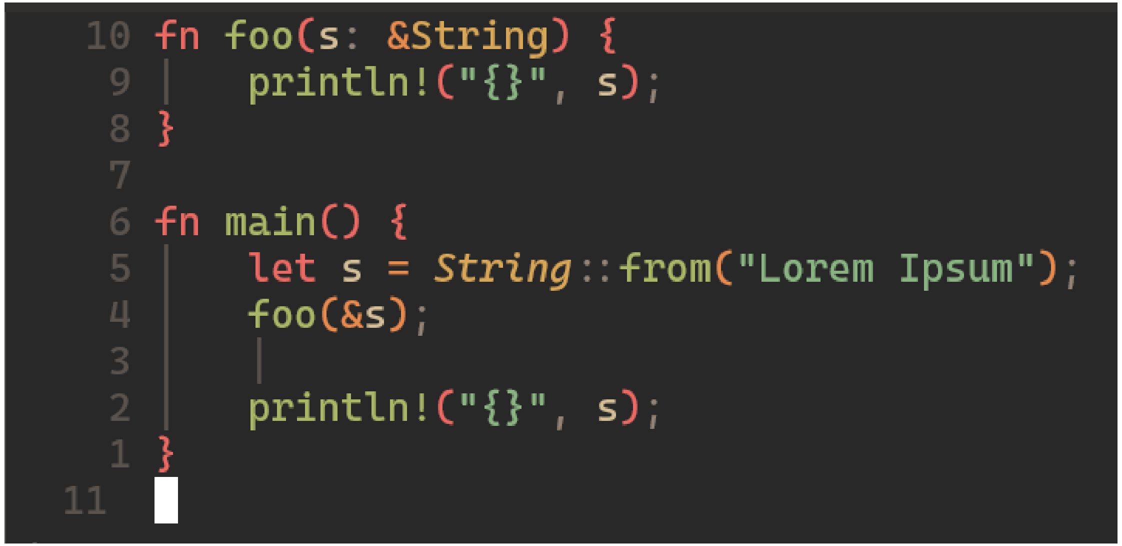
Ownership

소유권은 러스트가 메모리를 관리하는 방법입니다.



```
  fn foo(s: String) {
  │   println!("{}", s);
  }

  fn main() {
H │   let s = String::from("Lorem Ipsum");
          └──── move occurs because `s` has type `String`, wh
H │   foo(s);
          └──── value moved here
  │   │
E │   println!("{}", s);
      │              └──── borrow of moved value: `s`
      │                    value borrowed here after move
      └──── borrow of moved value: `s`
            value borrowed here after move
  }
```

Ownership



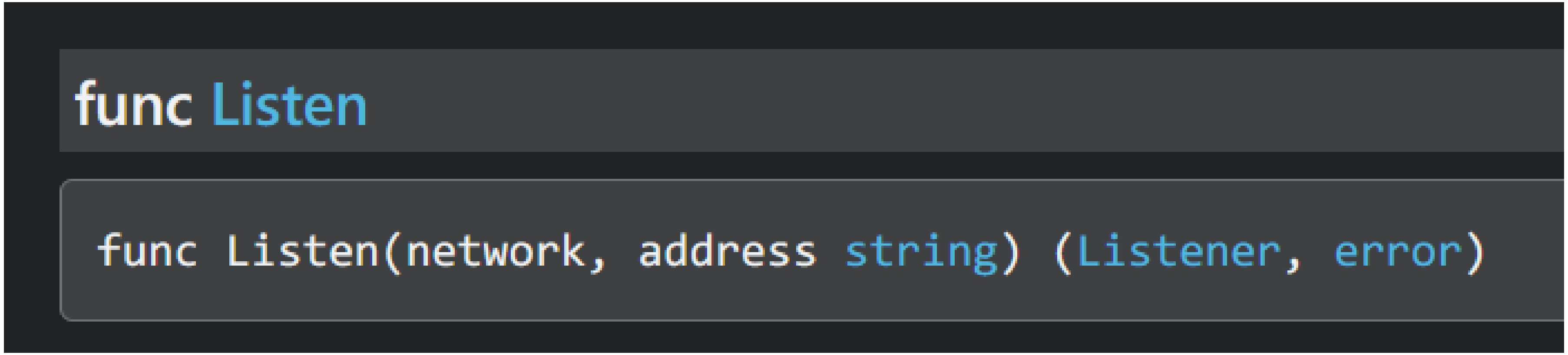
```rust
fn foo(s: &String) {
    println!("{}", s);
}

fn main() {
    let s = String::from("Lorem Ipsum");
    foo(&s);

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

devfest 2022

'' class=''time talk-ended single-
'' class=''talk-name''>...
class=''descrip

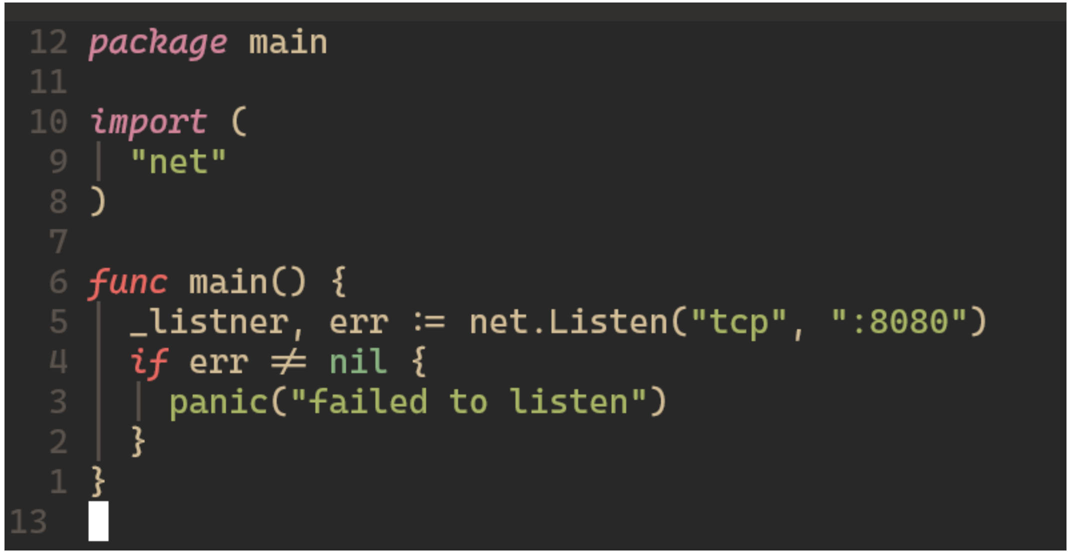
에러처리

Error Handling


func Listen

```
func Listen(network, address string) (Listener, error)
```

Error Handling



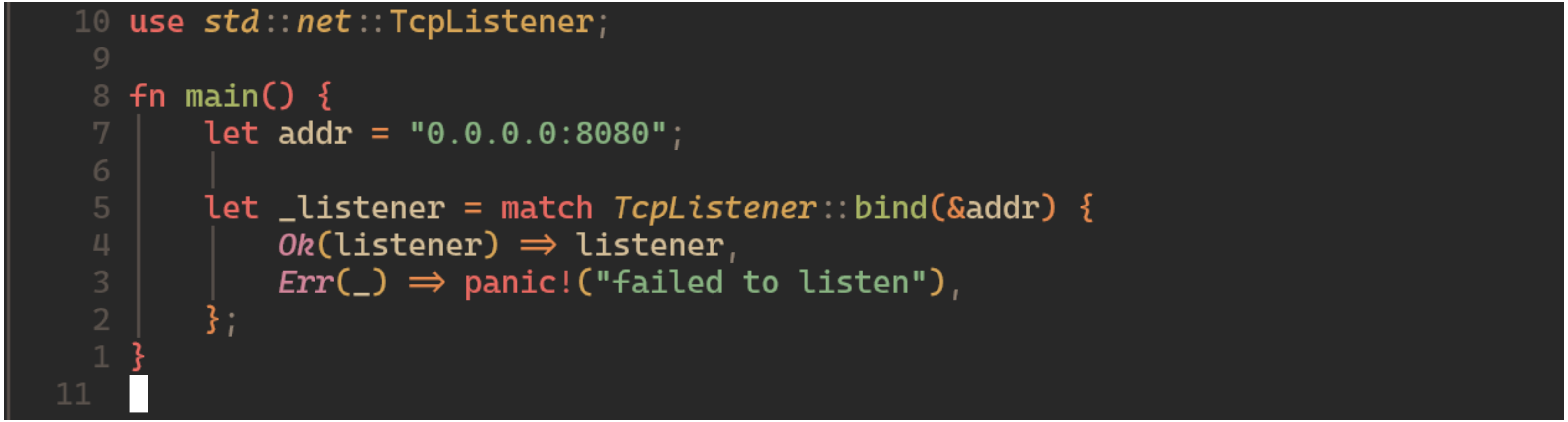
```go
package main

import (
	"net"
)

func main() {
	_listner, err := net.Listen("tcp", ":8080")
	if err ≠ nil {
		panic("failed to listen")
	}
}
```

Error Handling

```
pub fn bind<A: ToSocketAddrs>(addr: A) -> Result<TcpListener>
```

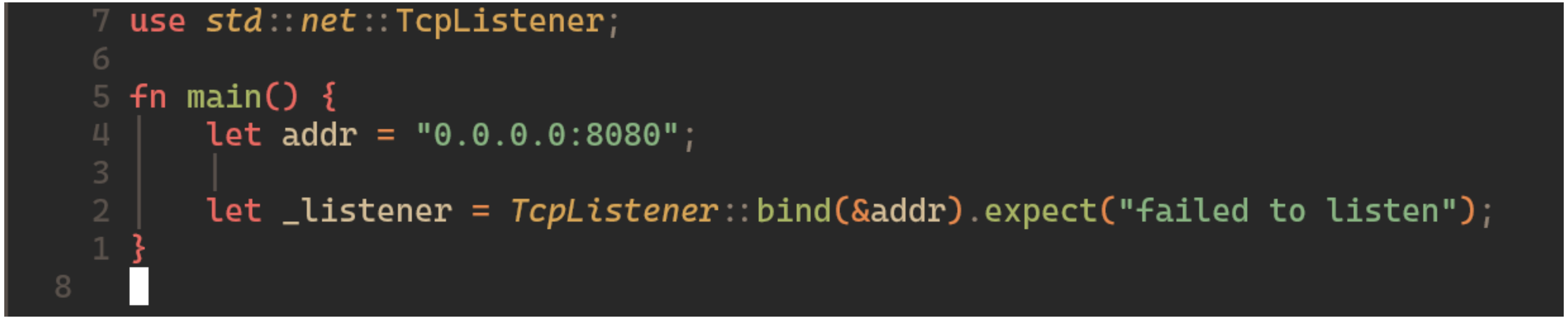
```
pub enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

Error Handling



```rust
use std::net::TcpListener;

fn main() {
    let addr = "0.0.0.0:8080";

    let _listener = match TcpListener::bind(&addr) {
        Ok(listener) => listener,
        Err(_) => panic!("failed to listen"),
    };
}
```

Error Handling

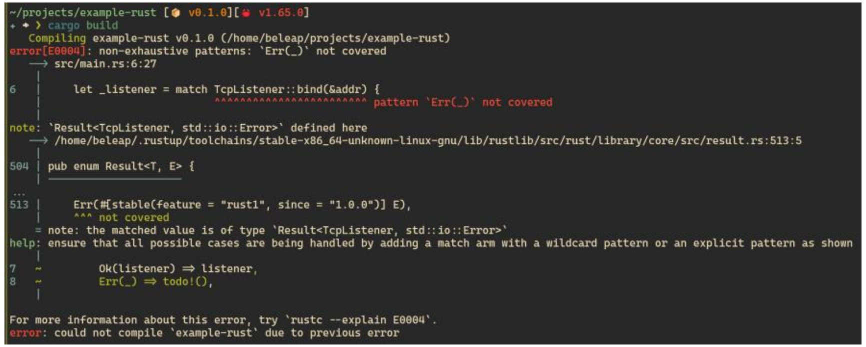


```rust
use std::net::TcpListener;

fn main() {
    let addr = "0.0.0.0:8080";

    let _listener = TcpListener::bind(&addr).expect("failed to listen");
}
```

Error Handling

기본적으로 Go와 Rust의 (result, error)와 Result, panic의 의미는 비슷함

```
Result => Recoverable Error
panic  => Unrecoverable Error
```



```
~/projects/example-rust [ v0.1.0][ v1.65.0]
+ ➜ ❯ cargo build
   Compiling example-rust v0.1.0 (/home/beleap/projects/example-rust)
error[E0004]: non-exhaustive patterns: `Err(_)` not covered
   --> src/main.rs:6:27
    |
6   |     let _listener = match TcpListener::bind(&addr) {
    |                           ^^^^^^^^^^^^^^^^^^^^^^^^ pattern `Err(_)` not covered
    |
note: `Result<TcpListener, std::io::Error>` defined here
   --> /home/beleap/.rustup/toolchains/stable-x86_64-unknown-linux-gnu/lib/rustlib/src/rust/library/core/src/result.rs:513:5
    |
504 | pub enum Result<T, E> {
    | ---------------------
...
513 |     Err(#[stable(feature = "rust1", since = "1.0.0")] E),
    |     ^^^ not covered
    = note: the matched value is of type `Result<TcpListener, std::io::Error>`
help: ensure that all possible cases are being handled by adding a match arm with a wildcard pattern or an explicit pattern as shown
    |
7   ~         Ok(listener) => listener,
8   ~         Err(_) => todo!(),
    |

For more information about this error, try `rustc --explain E0004`.
error: could not compile `example-rust` due to previous error
```

Error Handling

`defer vs drop`

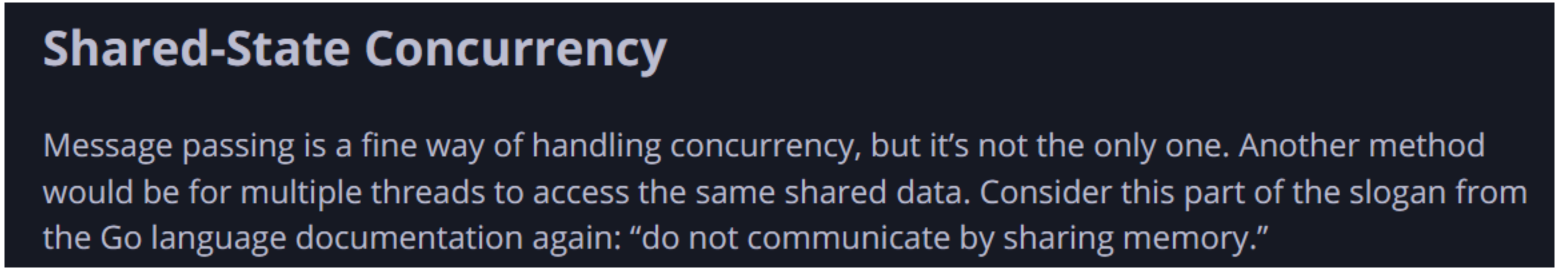
동시성

Concurrency

Go에서는 goroutine이라는 경량 스레드를 사용합니다.
Rust는 OS 스레드를 사용합니다.

Concurrency – Sharing State

Go - chan
Rust - channel, Mutex등 다양한 공유 방식을 제공함. 공유된 상태의 문제를 컴파일 시간에 많이 잡아낼 수 있음.



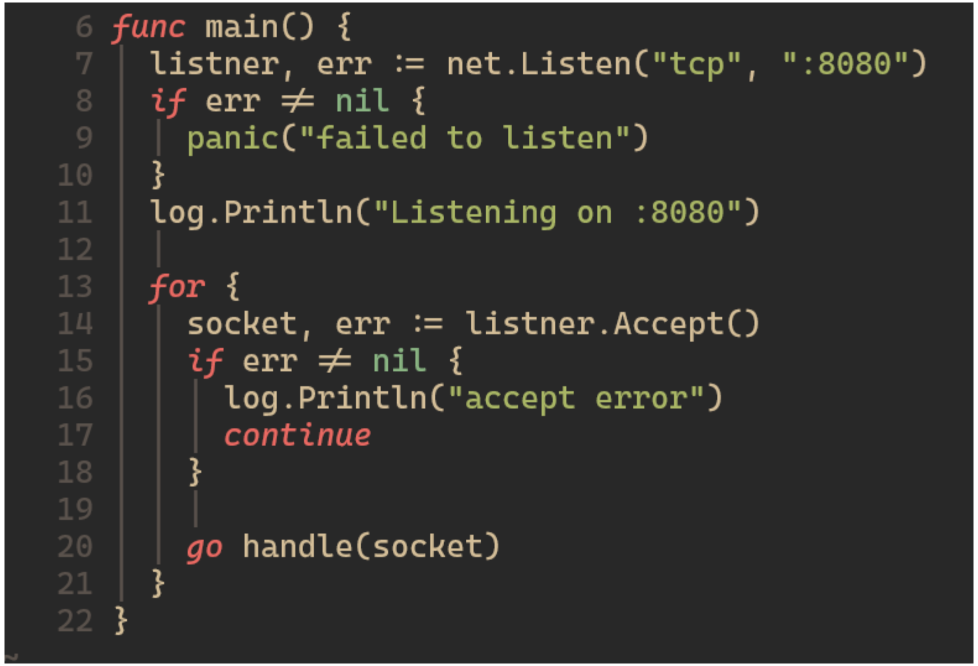
Shared-State Concurrency

Message passing is a fine way of handling concurrency, but it's not the only one. Another method would be for multiple threads to access the same shared data. Consider this part of the slogan from the Go language documentation again: "do not communicate by sharing memory."

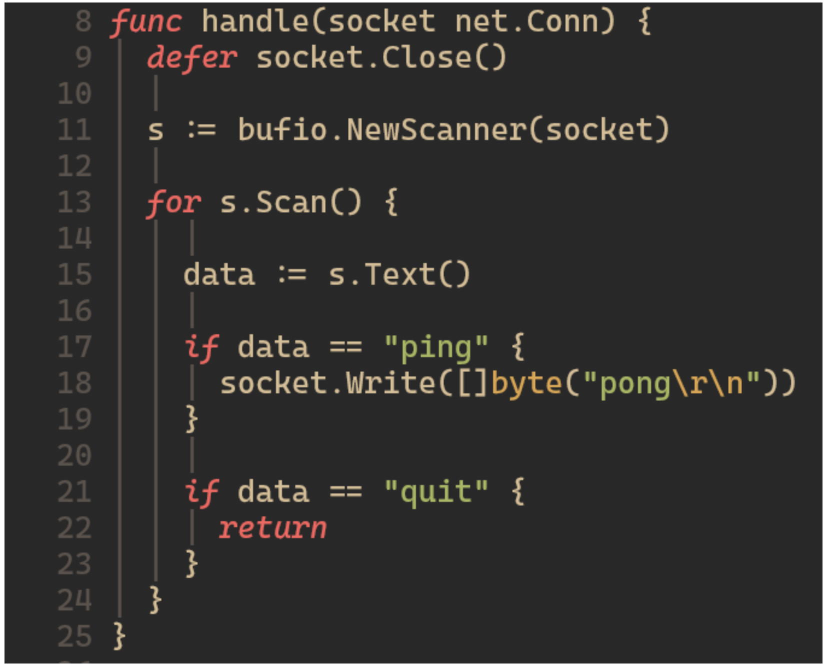
devfest 2022

서버 성능

Go Implementation

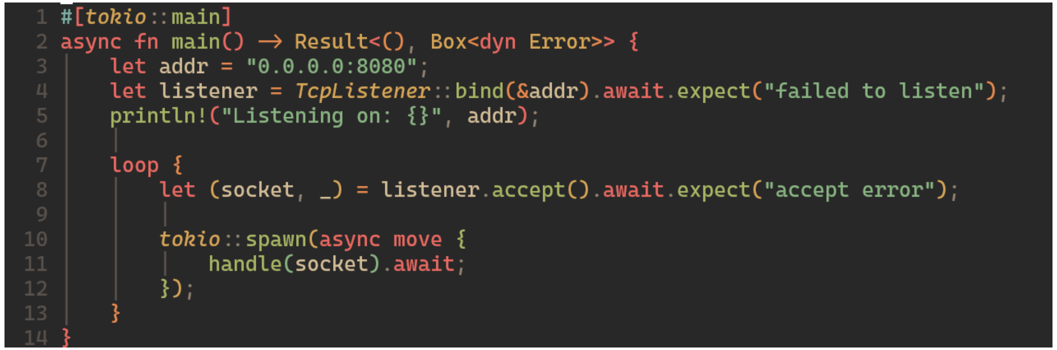


```go
func main() {
	listner, err := net.Listen("tcp", ":8080")
	if err != nil {
		panic("failed to listen")
	}
	log.Println("Listening on :8080")

	for {
		socket, err := listner.Accept()
		if err != nil {
			log.Println("accept error")
			continue
		}

		go handle(socket)
	}
}
```

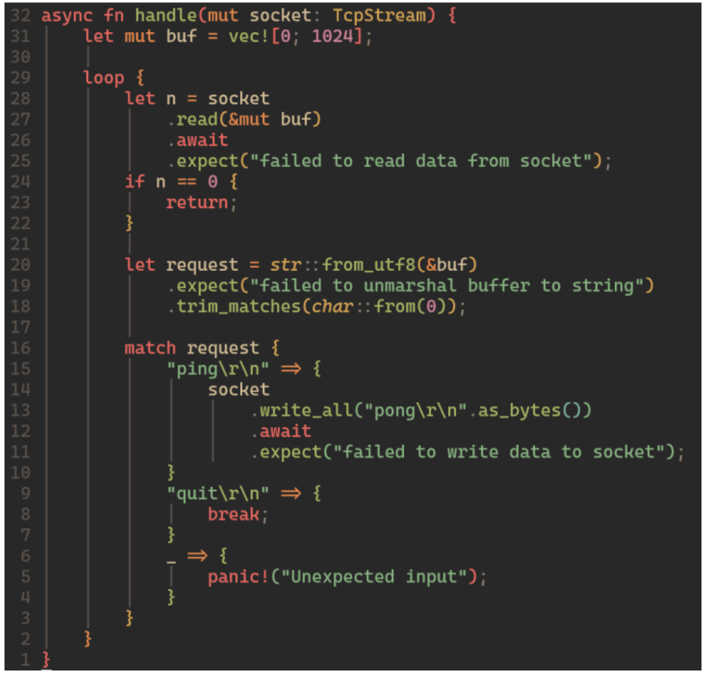


```go
func handle(socket net.Conn) {
	defer socket.Close()

	s := bufio.NewScanner(socket)

	for s.Scan() {

		data := s.Text()

		if data == "ping" {
			socket.Write([]byte("pong\r\n"))
		}

		if data == "quit" {
			return
		}
	}
}
```

Rust Implementation



```rust
#[tokio::main]
async fn main() -> Result<(), Box<dyn Error>> {
    let addr = "0.0.0.0:8080";
    let listener = TcpListener::bind(&addr).await.expect("failed to listen");
    println!("Listening on: {}", addr);

    loop {
        let (socket, _) = listener.accept().await.expect("accept error");

        tokio::spawn(async move {
            handle(socket).await;
        });
    }
}
```



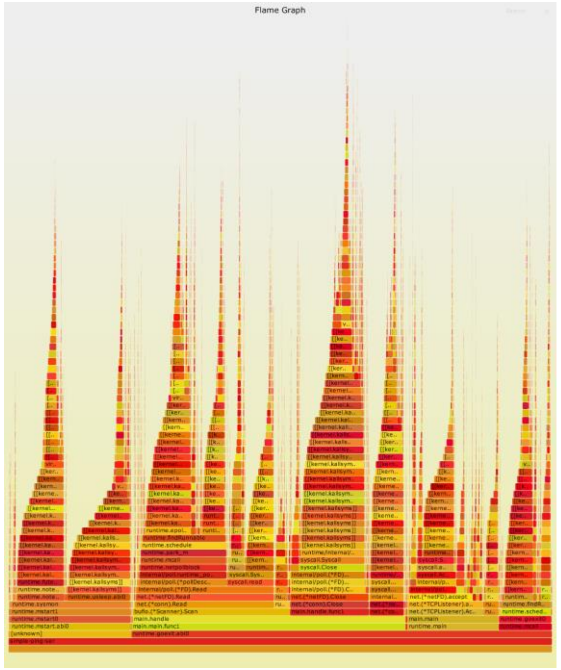
```rust
async fn handle(mut socket: TcpStream) {
    let mut buf = vec![0; 1024];

    loop {
        let n = socket
            .read(&mut buf)
            .await
            .expect("failed to read data from socket");
        if n == 0 {
            return;
        }

        let request = str::from_utf8(&buf)
            .expect("failed to unmarshal buffer to string")
            .trim_matches(char::from(0));

        match request {
            "ping\r\n" => {
                socket
                    .write_all("pong\r\n".as_bytes())
                    .await
                    .expect("failed to write data to socket");
            }
            "quit\r\n" => {
                break;
            }
            _ => {
                panic!("Unexpected input");
            }
        }
    }
}
```

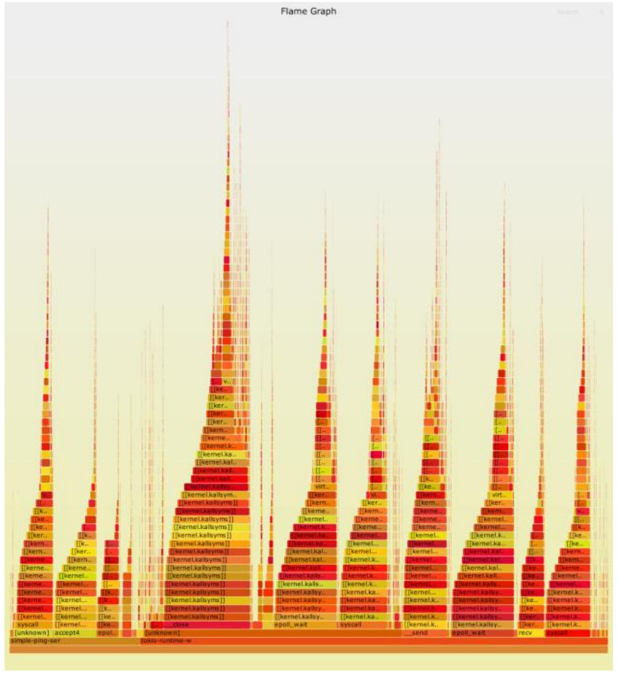
Load Test

Go	avg	min	med	max	p90	p95	rps
#1	507.52	13.59	498.79	1410	695.22	782.2	195.89
#2	497.65	3.43	495.24	1490	683.78	704.77	198.89
#3	494.05	4.44	493.54	1300	689.32	717.97	200.38
avg	499.74	7.153333	495.8567	1400	689.44	734.98	198.3867

Rust	avg	min	med	max	p90	p95	rps
#1	500.47	4.93	497.45	1580	689.67	709.11	197.71
#2	490.47	6.29	492.74	1290	684.55	703.26	202.44
#3	495.46	13.11	494.23	1680	689.15	710.75	199.76
avg	495.4667	8.11	494.8067	1516.667	687.79	707.7067	199.97

Flame Graph





컴파일

컴파일에 걸리는 시간

Go			**Rust (Debug, with Dependency Cache)**			**Rust (Release, with Dependency Cache)**	
#1	0.656		#1	2.206		#1	6.800
#2	0.717		#2	2.156		#2	6.262
#3	0.747		#3	2.082		#3	6.387

바이너리 크기

Go	2675236	2.68MB
Rust(Debug)	25271096	25.3MB
Rust(Release)	4708744	4.71MB
Rust(Release, with lto, strip, panic abort)	522720	522KB