

# Golang과 웹소켓을 활용한 서버프로그래밍

---

- 장애없는 서버 런칭 도전기

배상익 | Aidan  
카카오게임즈 퀸스튜디오

kakao**games**

## 퀸스튜디오와 스낵게임



카카오톡 게임별(4번째 탭) 설정에서 추가가능  
설치없이 즐기는 스낵게임 HTML5게임 개발

팀장님(Dennis) 철학

“빨리 만들어야해, 높은 퀄리티 ”

---

Aidan\_배상익

## 클라이언트 개발자에서 서버개발자

카카오에서 경험 할 수 있는 많은 유저풀과 대용량 트래픽  
객체지향을 넘어 함수형 프로그래밍, 액터 모델, CSP 등  
동시성 프로그래밍에 대한 궁금증

Malloc에 고통받기엔... 인생은 짧고 공부할게 너무 많고 만들어야될게 많다.

Golang, Python, javascript, DevOps  
실수를 통해서 배우고 공유하자.

물들어올때 노를 저어라



SUP  
ERC  
ELL

kakao games

kakao games



**50만 명이 넘는 사전예약자  
카카오톡 플랫폼의 힘  
다운로드가 필요없는 스낵게임**



---

## 상황

서버개발은 혼자 해야되는 상황,  
코드 첫 줄부터

Admin 페이지 까지  
개발 기간 6주

자동화 서버 배포 환경을 구축완료  
Golang에 대한 불안감이 어느정도 해소된 상태  
(Golang을 접한지 6~7개월)

**생산성과 퍼포먼스 두마리 토끼를 잡기 위해 Golang을 믿어보자**

---

이제 우리 팀이 망한다면 서버라는 변수빼고 없어...  
게임회사는 보통 클라이언트 때문에 망하지않아...  
아이단...



죽지않는 서버를 위해  
코어로직 살 불일 때마다  
테스트를 중심으로

# 죽지않는 서버개발을 위한 거시적인 시스템구성

## 단위테스트

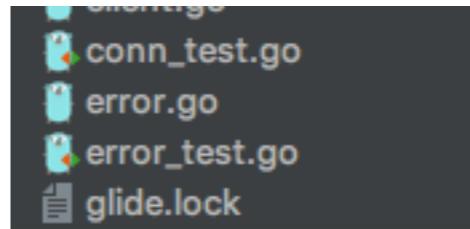
## Git - GoCD (CI/CD tool)

## mesos - marathon (DKOS)

## ngrinder 부하테스트 (5분, 10분)

## 메모리 고루틴 회수 체크

Go test -race option



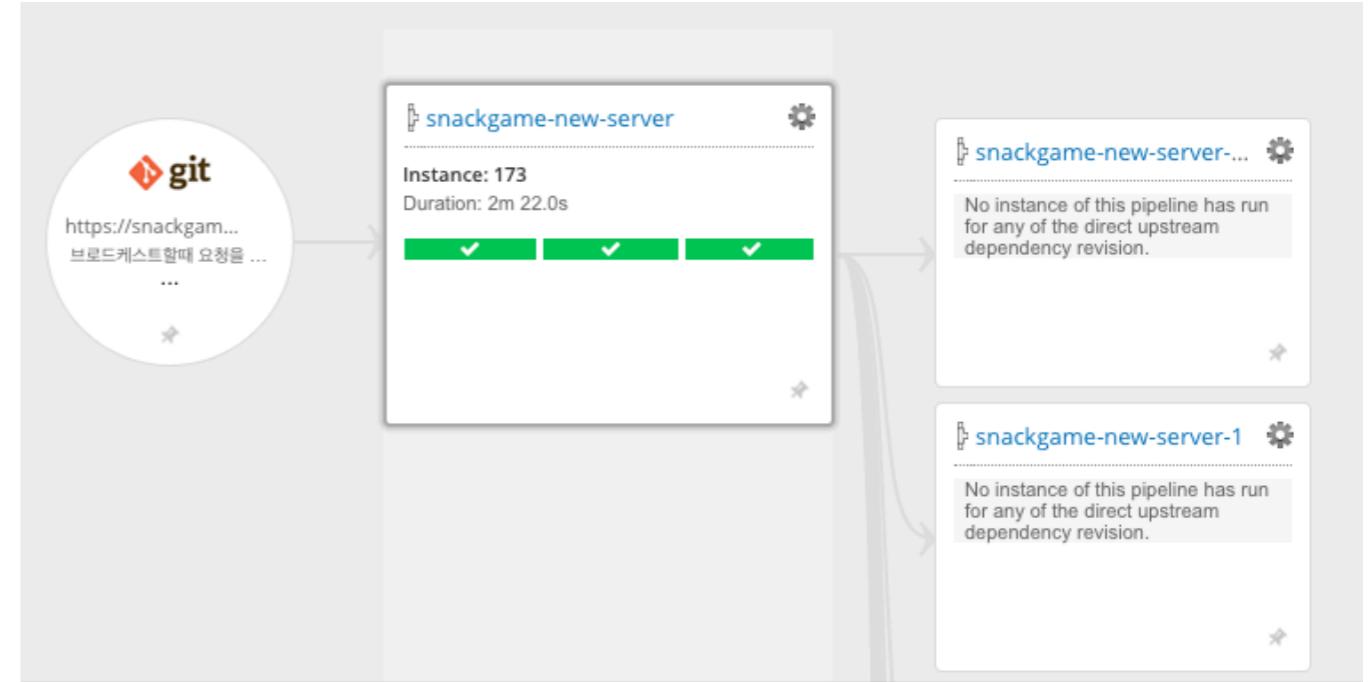
```
Alloc = 47916
TotalAlloc = 368791
Room(1986): Join 작업 후 현재인원 1
Room(1986): waitMatching goroutine 실행
Sys = 139646
NumGC = 42
NumGoroutine=11968
Room(1984): Join 작업 전 현재인원 2
Room(1984): Join 작업 후 현재인원 3
Room(1983): Join 작업 전 현재인원 3
Room(1983): Join 작업 후 현재인원 4
Room(1986): Join 작업 전 현재인원 1
Room(1986): Join 작업 후 현재인원 2
Room(1983): Join 작업 전 현재인원 4
Room(1983): Join 작업 후 현재인원 5
Room(1983): waitMatching goroutine 종료
```

# 죽지않는 서버개발을 위한 거시적인 시스템구성

단위테스트

Git - GoCD (CI/CD tool)

mesos - marathon (DKOS)



ngrinder 부하테스트 (5분, 10분)

메모리 고루틴 회수 체크

도커 이미지 빌드

Private snackgame docker hub에 푸시

# 죽지않는 서버개발을 위한 거시적인 시스템구성

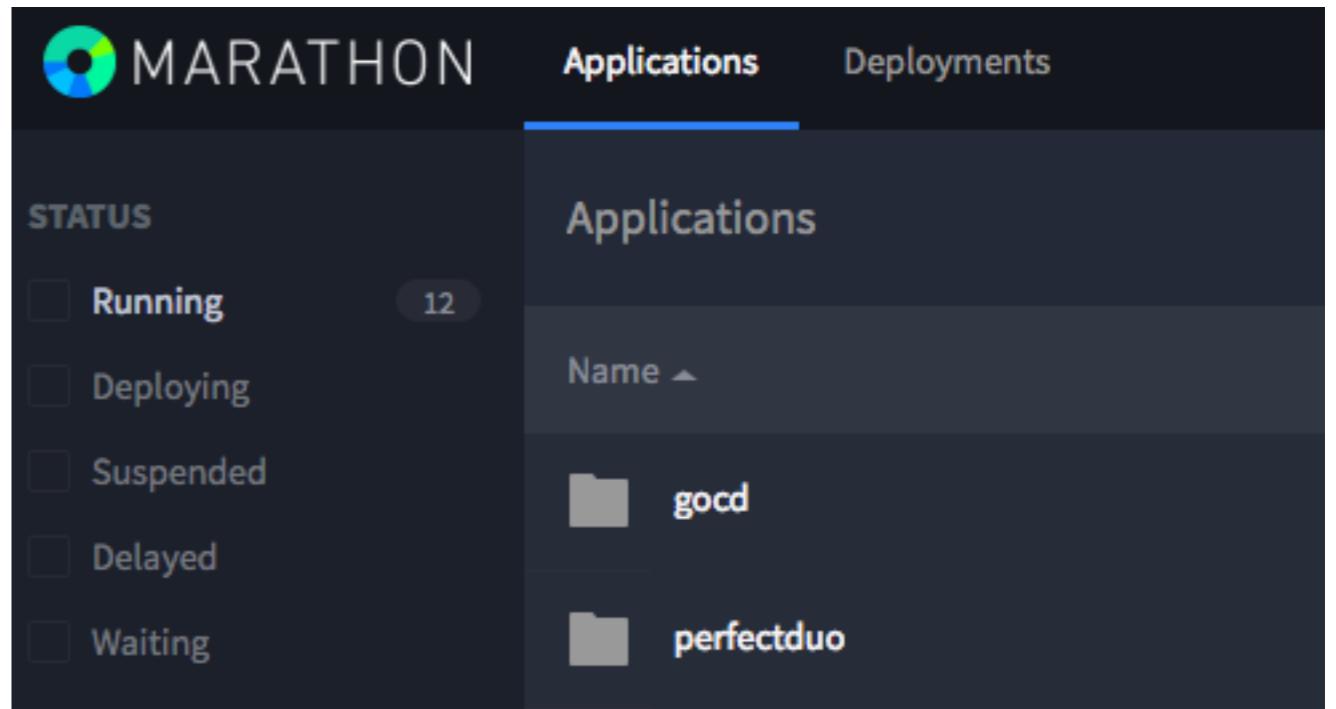
단위테스트

Git - GoCD (CI/CD tool)

mesos - marathon (DKOS)

ngrinder 부하테스트 (5분, 10분)

메모리 고루틴 회수 체크



인스턴스의 비정상 종료시 폐일오버 전략  
스케일링, 고가용성

# 죽지않는 서버개발을 위한 거시적인 시스템구성

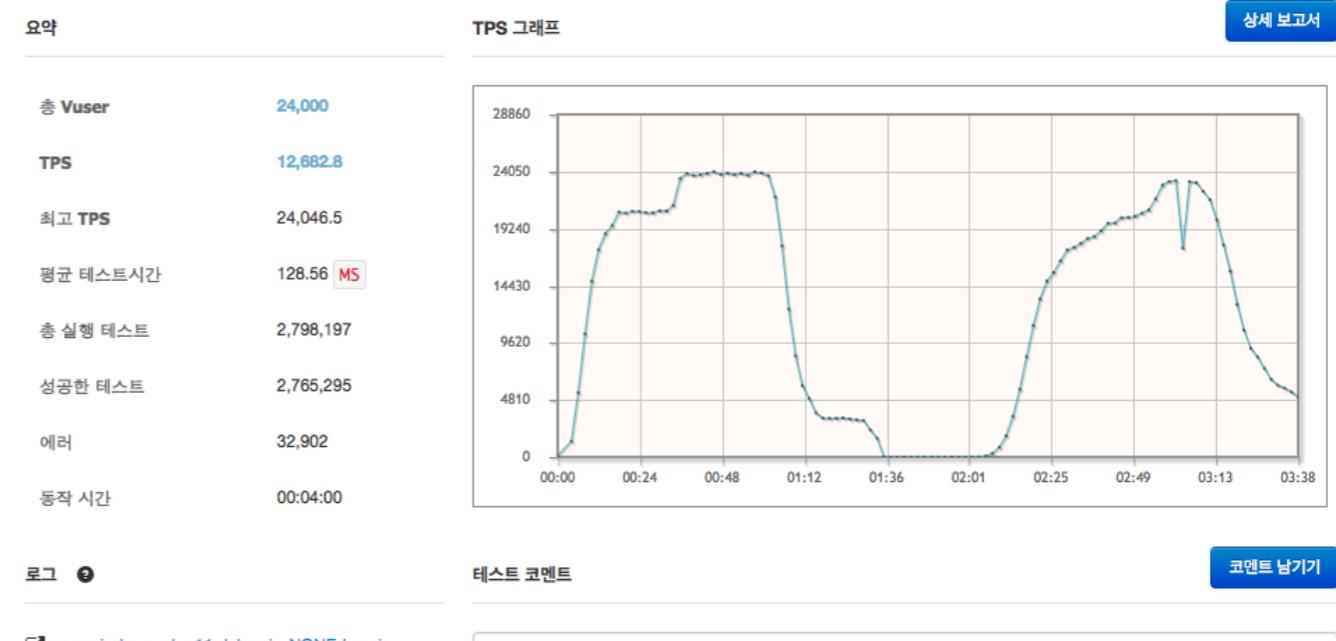
단위테스트

Git - GoCD (CI/CD tool)

mesos - marathon (DKOS)

ngrinder 부하테스트 (5분, 10분)

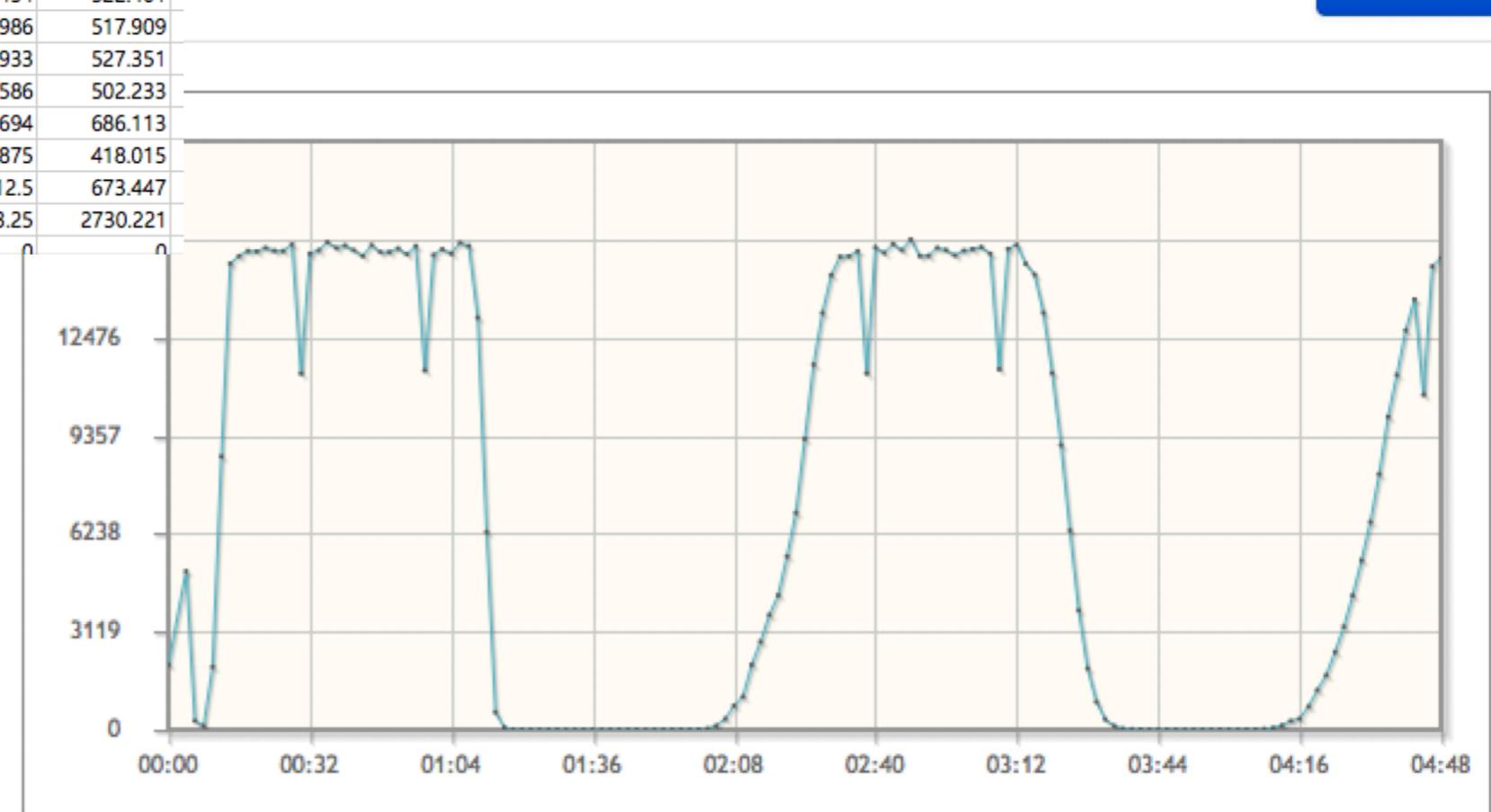
메모리 고루틴 회수 체크



Test Script는 Jython. 에러율 확인  
기능 추가때마다 테스트 스크립트도 추가  
20000명이 넘으면... 알고보니 linux에 max con 제한

0	messageJoin	0	0	0
0	messageJoin	3	0	8289.333
0	messageJoin	21	0	8390.476
0	messageJoin	47	0	7980.191
0	messageJoin	147	0	7962.857
0	messageJoin	376	0	7944.88
0	messageJoin	741	0	7913.031
0	messageJoin	1183	0	7928.757
0	messageJoin	1742	0	7950.757
0	messageJoin	2016	0	7988.956
0	messageJoin	1968	0	8006.39
0	messageJoin	1533	0	8101.847
0	messageJoin	1141	0	8116.434
0	messageJoin	572	0	8125.986
0	messageJoin	313	0	8233.933
0	messageJoin	99	0	8490.586
0	messageJoin	62	0	8793.694
0	messageJoin	16	0	8768.875
0	messageJoin	4	0	9112.5
0	messageJoin	4	0	10753.25
0	messageJoin	0	0	2730.221

상세 보고서



어떤 테스트에서 에러가 많았는지

시간이 흐름에도 지속적으로 TPS를 유지하는지

# 죽지않는 서버개발을 위한 거시적인 시스템구성

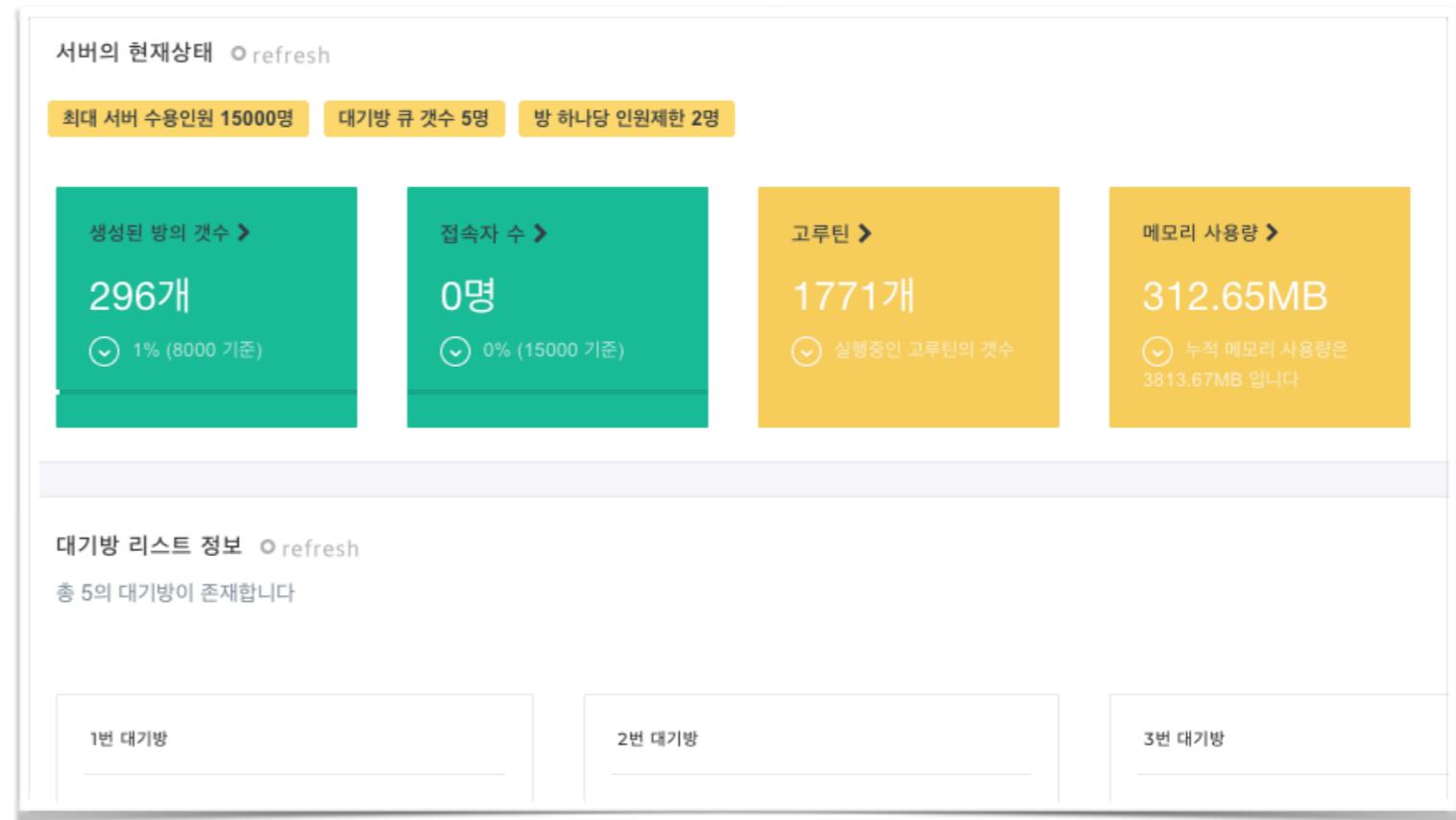
단위테스트

Git - GoCD (CI/CD tool)

mesos - marathon (DKOS)

ngrinder 부하테스트 (5분, 10분)

메모리 고루틴 회수 체크

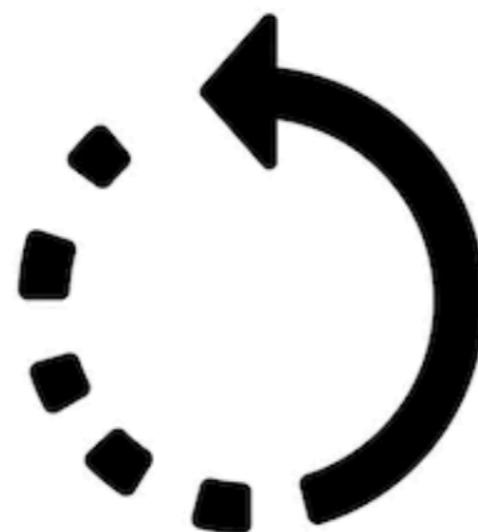


Admin Page(Vue)  
runtime.NumGoroutine()  
runtime.MemStats

---

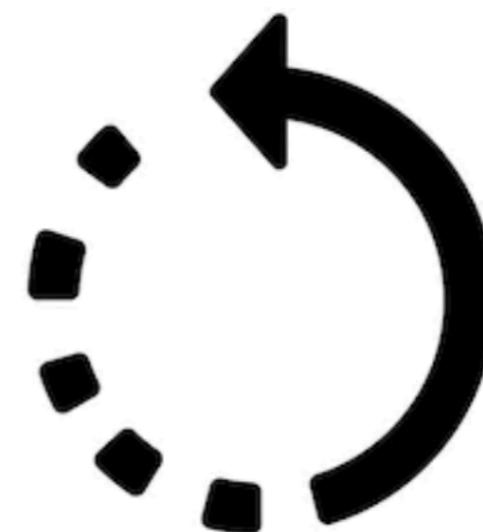
## Golang 디자인 철학

CSP( 순차 프로세스 통신 ), 객체지향에서 관점전환(꽤나 혼란)  
프로세스의 커뮤니케이션에 초점을 맞춘 프로그래밍 언어



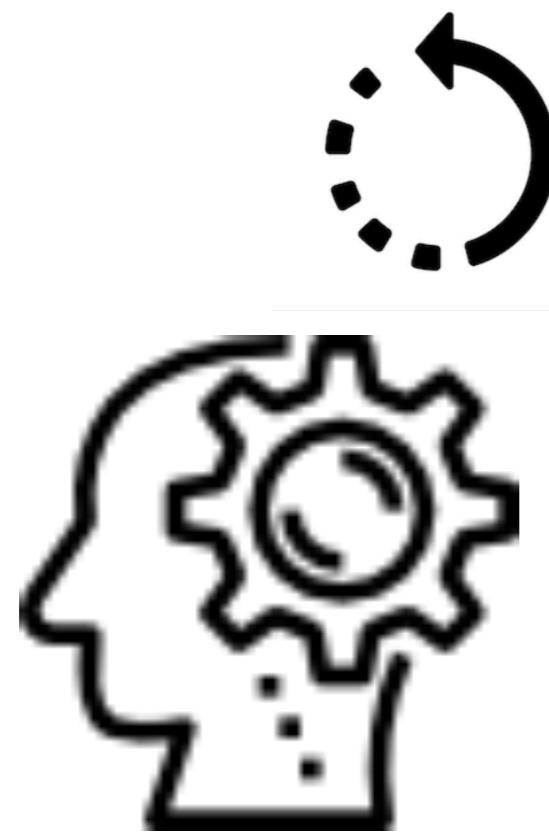
채널

---



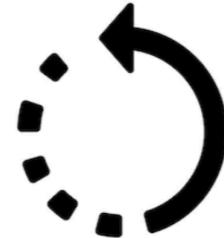
## Golang 디자인 철학

Rob Pike : Concurrency is not Parallelism



점심 메뉴 고민...

카톡 대기중..



도대체 왜 잘돌아가는걸까..

동시성은  
한꺼번에 많은 일들을  
다루는 것

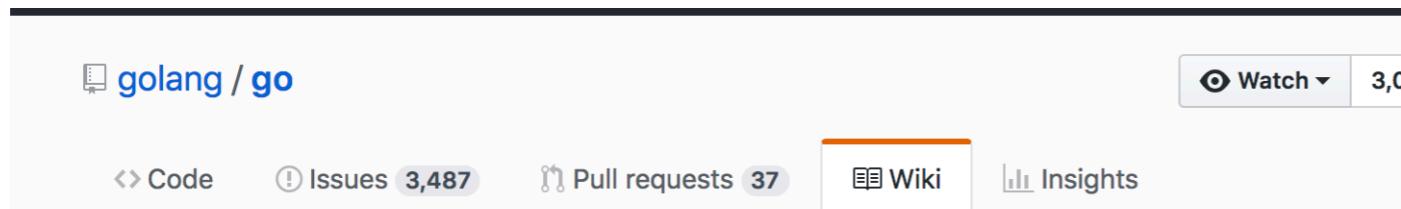
병렬성은  
많은 일을 한꺼번에 실행

---

# Concurrency Golang Programming

- ◆ 적당한 방법론을 택하고  
Mutex vs Channel
- ◆ 나의 실수를 얼른 찾자  
Goroutine Leak  
HandlePanic

# Mutex vs Channel



## MutexOrChannel

Rick Beton edited this page on 20 Mar 2015 · 5 revisions

### Use a sync.Mutex or a channel?

One of Go's mottos is "*Share memory by communicating, don't communicate by sharing memory.*"

공유 자원에 대한 관리를 어떻게 할 것인가

goroutine간 소통 관리를 어떻게 할까

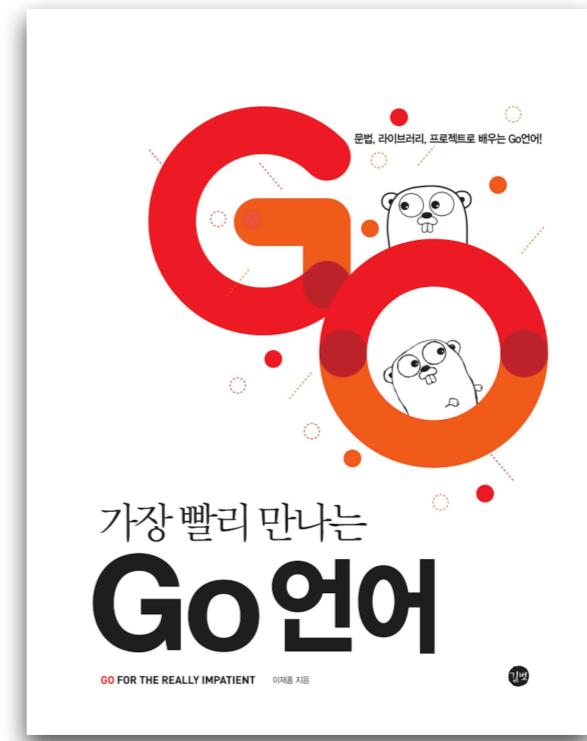
---

## Mutex vs Channel

초기

공유 자원에 대한 접근에 있어서 gosched, mutex, channel  
뭘 활용하는게 좋을까

가장 빨리 만나는 Go 언어 책의 예제



## Mutex vs Channel

### 상황

공유하는 data slice에  
1000번 append하는  
**goroutine 4개 실행**

일정 시간 후  
마지막에 data len 체크

```
/*  
 * 2312개 - 아무것도 안사용했을경우  
 * 3612개 - Gosched  
 * Gosched 뮤텍스 4000개  
 * 뮤텍스 4000개  
 * 고채널 4000개  
 */
```

> 뮤텍스, 채널 둘 다 좋다 (?)

## Mutex vs Channel

### Mutex로 동기화

```
func main() {
    runtime.GOMAXPROCS(runtime.NumCPU())
    var data = []int{}
    var mutex = new(sync.Mutex)

    go func() {
        for k := 0; k < 1000000; k++ {
            mutex.Lock()
            data = append(data, k)
            mutex.Unlock()
        }
    }()
}
```

x 4개

전통적인 방법으로  
Data append하는 부분을  
mutex로 감싸준다.

```
    time.Sleep(500 * time.Millisecond)
    fmt.Println(len(data))
}
```

500ms 이 후  
Data Slice length 확인

## Mutex vs Channel

### Unbuffered Channel로 동기화

```
func main() {
    runtime.GOMAXPROCS(runtime.NumCPU())
    var data = []int{}

    var c = make(chan int)

    go func() {
        for k := 0; k < 1000000; k++ {
            <-c
            data = append(data, k)
            c <- 1
        }
    }()
}
```

x 4개

```
c <-1

time.Sleep(500 * time.Millisecond)
fmt.Println(len(data))
}
```

Unbuffered channel을 하나 준비하고

Data append 작업은  
해당 채널에 int값이 수신되었을 때 실행

작업이 끝나면 int값을 송신해서  
다른 고루틴에게 제어권 양보

## Mutex vs Channel

```
/*  
아무것도 사용안하고 경합발생  
1952832개  
  
뮤텍스 4000000개  
  
gochannel - 4만개 40만개도 문제없 400만개는 문제잇었습니  
2333505개, 2321505 2224415 ...  
*/
```

500ms 이후 확인해보니 뮤텍스는 400만개 모두 Append작업완료!

Channel 동기화의 경우 230만개

아무것도 사용하지 않은 경우 180만개~ 190만개 (자주 error 발생)

---

## Mutex로 Room Join처리 등 많은 동시성 작업을 케어



시간, 많은 부하 테스트

잘 돌아가던 녀석이 Goroutine 회수가 안되고

**Race condition 발생**

알 수 없는 오류들이 발생

서버를 띄워 놓고 퇴근 다음 날

**Deadlock 발생**

---

Rob Pike

*Do not communicate by sharing memory;  
instead, share memory by communicating*

공유 메모리로 소통하지마세요.

대신 Communicating을 통해 메모리를 공유하세요



---

뮤텍스는 신용카드와 같아...

많은 뮤텍스의 사용 -> 데드락과의 필연

goroutine간의 대화를 뮤텍스 보호에 너무 많이 의존하지 말자

“뮤텍스는 70년대 포드자동차를 모는 것과 같다.  
멀리가기엔 안정성이 떨어진다.”

미래를 품은 7가지 동시성 모델



---

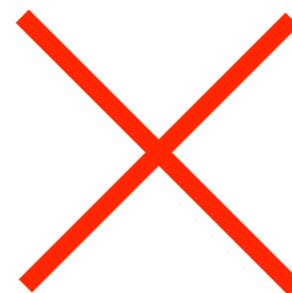
## Stateless한 코드 작성

stateful한 부분이 있다면

최대한 제거, 가변상태를 줄이자

- channel을 통해 해당 상태별 Select문으로 분기

```
func (w *World) enter(client *Client) {  
    if client.isEntered {  
        return  
    }  
    client.isEntered = true  
}
```



## Stateless한 코드 작성

stateful한 부분이 있다면

최대한 제거, 가변상태를 줄이자

- channel을 통해 해당 상태별 Select문으로 분기

```
for {
    select {
        case client := <-w.ChanEnter:
            fmt.Println("클라이언트 입장")
            w.clientMap[client] = true
        case client := <-w.ChanLeave:
            if _, ok := w.clientMap[client]; ok {
                delete(w.clientMap, client)
                fmt.Println("클라이언트 퇴장")
                close(client.send)
            }
    }
}
```

---

**Channel을 잘 활용해야 고스러운 코딩이 가능하다**

range, close, select

**Go blog에서도 추천하는 Concurrency pattern**

Pipeline pattern

Fan in - fan out

---

## channel의 기능들을 잘 접합시킨 - Pipeline pattern

```
func generatePipeline(in <-chan int) <-chan int {
    out := make(chan int)
    go func() {
        for n := range in {
            fmt.Println(n)
            out <- n
        }
        close(out)
    }()
    return out
}

func squareNumber(in <-chan int) <-chan int {
    out := make(chan int)
    go func() {
        for n := range in {
            out <- n * n
        }
        close(out)
    }()
    return out
}
```

---

## 나의 실수를 얼른 찾자

### Goroutine Leak

**뮤텍스 쌍이 맞지 않는다.**

**Sender가 있는데, Receiver가 없다.**

**Receiver가 있는데, Sender가 없다.**

채널 중심으로 코드를 살펴보면 디버깅이 편리하다.

Runtime/trace는 관심있게 지켜보는 중

## 나의 실수를 얼른 찾자

### HandlePanic

```
func HandlePanic(msg string, watch bool) {
    if r := recover(); r != nil {
        var err error
        switch x := r.(type) {
        case string:
            err = errors.New(x)
        case error:
            err = x
        default:
            err = errors.New("Unknown panic: " + x)
        }
        // ...
    }
}
```

defer HandlePanic()

[서버 패닉 발생]

에러내용: runtime error: invalid memory address or  
nil pointer dereference  
패닉원인: Room has panic!  
{"room\_id":"R8411dba6bc3d4ff99deb1a1ec0ee8  
d0f","client\_list":  
[{"id":"C87acb86d830e4b6da414d64c7bc3a679  
","is\_connected":true,"score":  
167,"player\_id":"51762192377743","nickname":"U  
2FsdGVkX1/2QdIXcrGvFw4f0JrzC4ECMAnT9eTvj  
I4=","profile\_image":""}], "user\_limit\_count":  
2,"is\_matched":false,"room\_sequence":  
1,"created\_at":"2018-06-15T11:22:20.233183583  
+09:00"}

-----

호스트:

컨테이너ID: KAKAOui-MacBook-Pro-5.local

오전 11:23

---

나의 실수를 얼른 찾자

HandlePanic

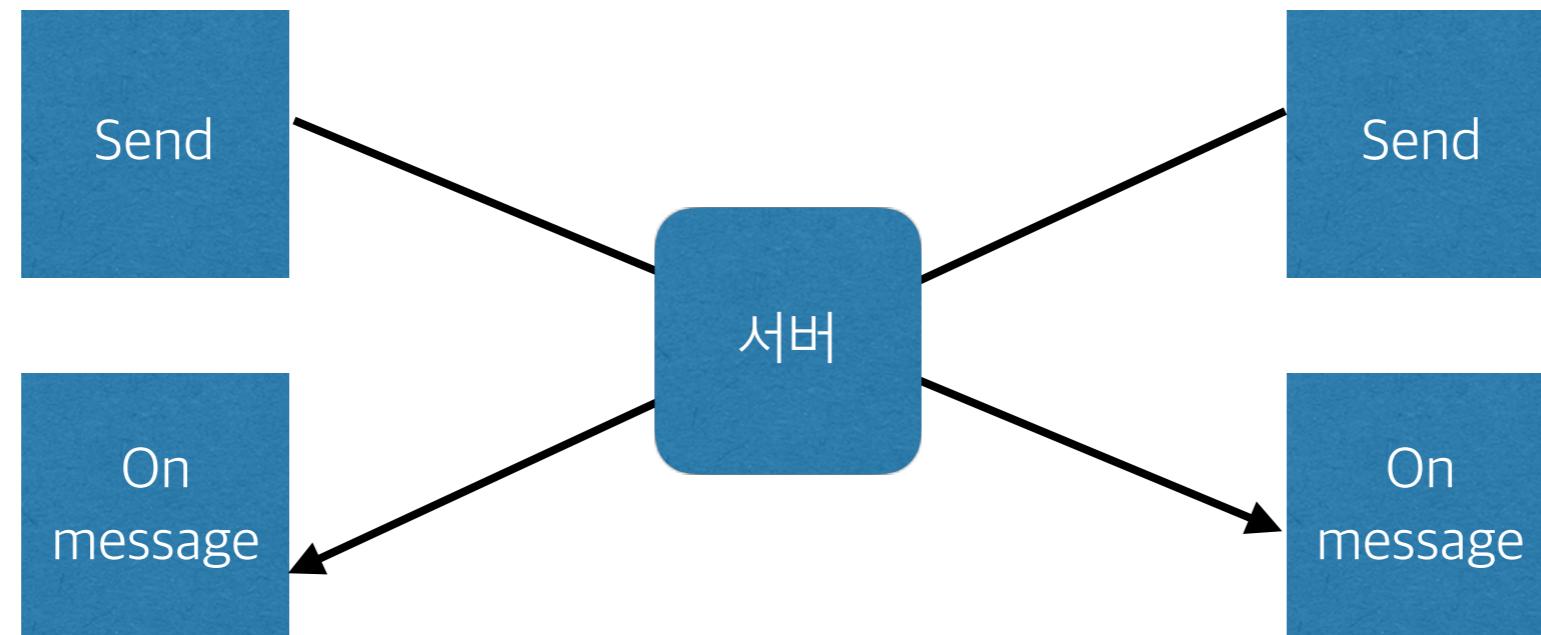
## **Send on Closed Channel**

Channel이 고루틴의 소통에 중점적으로 가볍게 설계되어 데이터를 채널에 보내기 전에 closed를 확인하는 기능은 없음.  
그래서 닫는 타이밍을 잘 확인해야합니다.

# Websocket

Ws프로토콜을 사용해 클라이언트간 양방향 스트림 통신을 제공  
응답을 기다릴 필요가 없는 고오급 기술, 이벤트를 상시 수신할 수 있다.

```
void close(in optional unsigned long code, in optional DOMString reason);  
void send(in DOMString data);
```



---

## **Go-SocketIO vs gorilla Websocket**

go-socket-io go스럽지 않은 코드  
콘의 좌충우돌 줄다리기 서버에서 생긴  
브로드캐스트 락해제관련 버그이슈

**이제 Websocket 지원 안되는 폰은 고려하지말자**

## Gorilla websocket 라이브러리

The screenshot shows a web browser window with the URL [www.gorillatoolkit.org/pkg/websocket](http://www.gorillatoolkit.org/pkg/websocket) in the address bar. The page content is as follows:

**Gorilla web toolkit**

**package websocket**

```
import "github.com/gorilla/websocket"
```

[Installation](#) | [Overview](#) | [API](#) | [Files](#)

The page features a dark background with yellow and white text. A cartoon gorilla icon is positioned next to the word "Gorilla". The "websocket" package page is currently selected, indicated by a yellow background.

---

## Websocket Upgrader

```
var upgrader = websocket.Upgrader{  
    ReadBufferSize: 1024,  
    WriteBufferSize: 1024,  
}
```

```
http.HandleFunc("/ws", func(w http.ResponseWriter, r *http.Request) {  
    conn, err := upgrader.Upgrade(w, r, nil)  
    if err != nil {  
        log.Println(err)  
        return  
    }  
    client := NewClient(world, conn)  
    client.world.ChanEnter <- client  
})
```

## 클라이언트 구조체

```
type Client struct{
    world *World
    conn *websocket.Conn
    send chan []byte
}

func NewClient (w *World, c *websocket.Conn) (client *Client) {
    client = &Client{
        world: w,
        conn: c,
        send: make(chan []byte, 256),
    }
    go client.readPump()
    go client.writePump()
    return client
}
```

클라이언트 생성과 동시에 readPump, writePump 고루틴 2개 실행

```
func (c *Client) readPump() {
    defer func() {
        c.world.ChanLeave <- c
        c.conn.Close()
    }()

    c.conn.SetReadLimit(maxMessageSize)
    c.conn.SetReadDeadline(time.Now().Add(pongWait))
    c.conn.SetPongHandler(func(string) error { c.conn.SetReadDeadline(time.Now().Add(pongWait)); return nil })

    for {
        _, message, err := c.conn.ReadMessage()
        if err != nil {
            if websocket.IsUnexpectedCloseError(err, websocket.CloseGoingAway, websocket.CloseAbnormalClosure) {
                log.Printf("error: %v", err)
            }
            break
        }
        c.world.broadcast <- message
    }
}
```

conn.ReadMessage()에서 클라이언트의 send 메세지를 기다린다.

```
func (c *Client) writePump() {
    ticker := time.NewTicker(pingPeriod)
    defer func() {
        ticker.Stop()
        c.conn.Close()
    }()
    for {
        select {
        case message, ok := <-c.send:
            c.conn.SetWriteDeadline(time.Now().Add(writeWait))
            if !ok {
                c.conn.WriteMessage(websocket.CloseMessage, []byte{})
                return
            }
            c.conn.WriteMessage(websocket.TextMessage, message)
        case <-ticker.C:
            c.conn.SetWriteDeadline(time.Now().Add(writeWait))
            if err := c.conn.WriteMessage(websocket.PingMessage, []byte{}); err != nil { return }
        }
    }
}
```

핑메세지를 보내는 Ticker의 존재

Client의 send 채널을 통해 들어온 메세지를 WriteMessage로 전달

## 월드 구조체

```
type World struct {
    clientMap map[*Client]bool
    ChanEnter chan *Client
    ChanLeave chan *Client
    broadcast chan []byte
}

func newWorld () *World {
    return &World{
        clientMap: make(map[*Client]bool, 5),
        broadcast: make(chan []byte),
    }
}
```

입장 채널 : ChanEnter

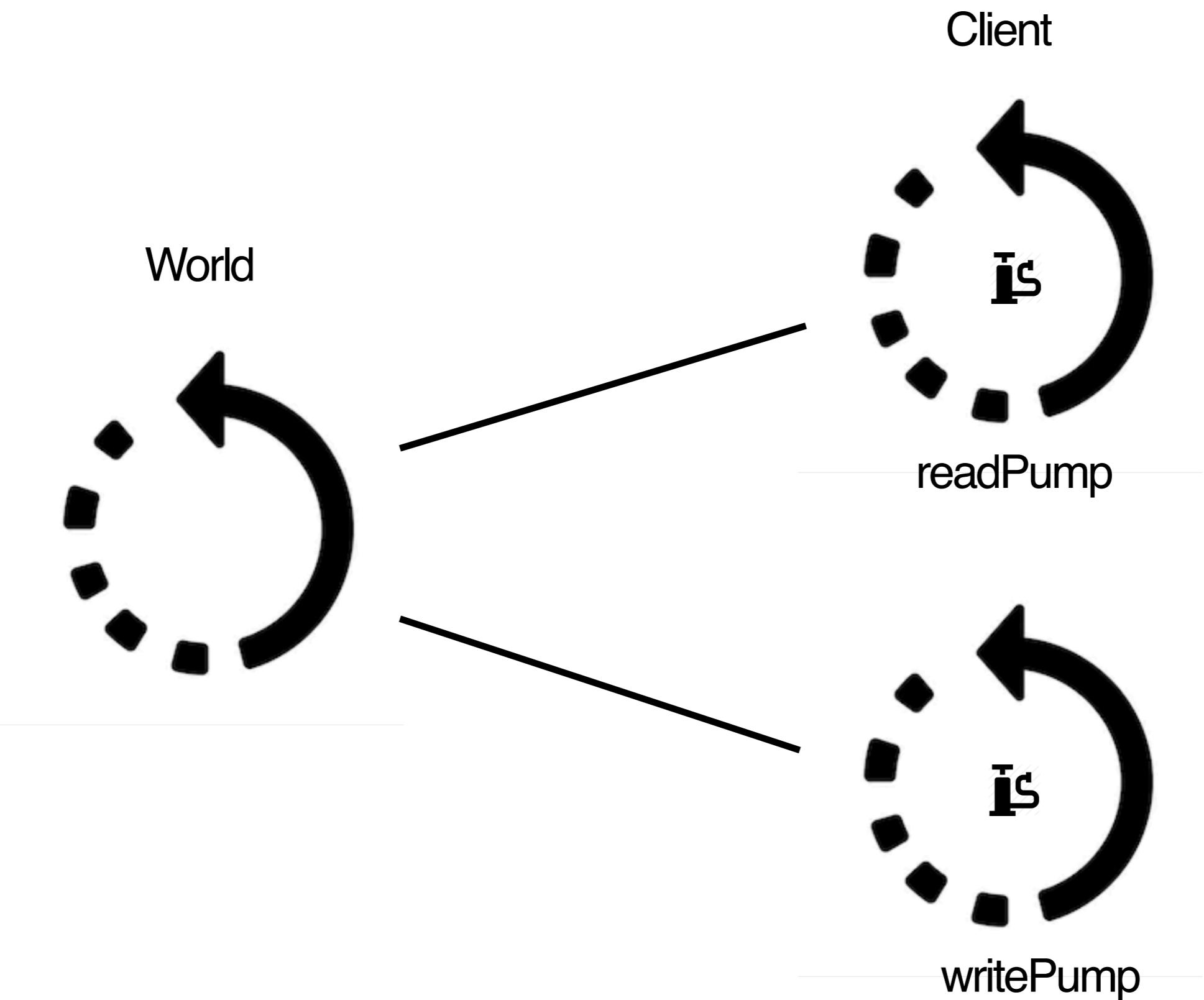
퇴장 채널 : ChanLeave

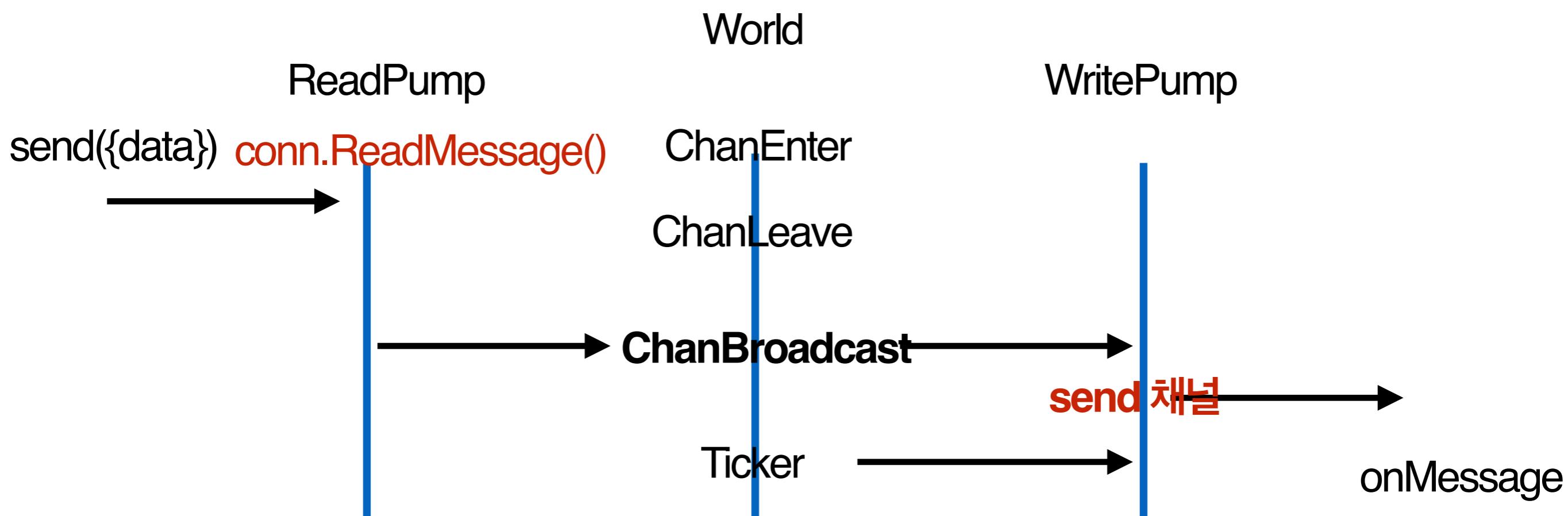
브로드캐스트 채널 : broadcast

```
func (w *World) run() {
    w.ChanEnter = make(chan *Client)
    w.ChanLeave = make(chan *Client)

    ticker := time.NewTicker(1 * time.Second)

    for {
        select {
        case client := <- w.ChanEnter:
            fmt.Println("클라이언트 입장")
            w.clientMap[client] = true
        case client := <- w.ChanLeave:
            if _, ok := w.clientMap[client]; ok {
                delete(w.clientMap, client)
                fmt.Println("클라이언트 퇴장")
                close(client.send)
            }
        case message := <- w.broadcast:
            for client := range w.clientMap {
                client.send <- message
            }
        case tick := <- ticker.C:
            for client := range w.clientMap {
                client.send <- []byte(tick.String())
            }
            fmt.Println(tick.String())
        }
    }
}
```





예제 코드 주소

<https://github.com/aidanbae/websocket-example>

Demo

서버 (컨테이너 인스턴스)당 17000명까지 무리없이!  
2만명부터는 메세징 딜레이가 생기고 (느껴짐) 힘들어했다.  
개발기간 (server 커밋은 3월 2일부터 4월 20일까지)

서버는 안정적으로 10000명 기준으로 배포.

## Golang은 생산성, 퍼포먼스, 안정성 셋 다 만족스럽다.

뮤텍스는 안정성을 포기하고 퍼포먼스를 사는 양날의 검과 같다.  
채널 중심으로 생각을 바꾸어보자.  
테스트를 두려워하지말고, 실수와 결과를 팀원들에게 늘 공유하자

## 참고한 좋은 자료들

Golang Design pattern 스터디

Golang 관련 한국어 블로그 스캔

Golang Korea 댓글



## 가장 큰 도움

Dennis

연차가 부족함에도 자유와 신뢰 제공

Rebecca, Mason

배포시스템 및 Golang 경험 공유



# **Thank You**