پاییز ۱۳۹۶

سیستم‌های توزیعی

گزارش پروژه اول

پرهام الوانی

۹۶۱۳۱۱۱۲

مقدمه

پروژه به صورت آزمون-محور [1]و در ۳ فاز انجام شد، در فاز اول پیاده‌سازی به صورتی انجام شد که تست‌های مربوط به یک کلاینت به درستی صحت‌سنجی شوند و در ادامه چند کلاینت و کلاینت‌ها آهسته-خوان [2]پیاده‌سازی شدند. در ادامه به شرح هر یک از فازها و آنچه برای آن‌ها پیاده‌سازی شد می‌پردازیم.

فاز اول

در این فاز برنامه از دو تابع اصلی تشکیل می‌شد، تابع handler که تقاضاها را دریافت کرده و پس از پردازش آن‌ها، تقاضاهای get و set را در channel مربوط به Key-Value-Server قرار می‌دهد و تابع dispatcher که این تقاضاها را خوانده و پس از پردازش آن‌ها پاسخ را به کلاینتی که تقاضا را ارسال کرده است می‌فرستد. آنچه در این فاز پیاده‌سازی شد در کد 1 پیوست شده است.

```
package p1


import (
	"bufio"
	"fmt"
	"net"
)


type request struct {
	command string
	key     string
	value   []byte
	writer  *bufio.Writer
}


type keyValueServer struct {
	ln      net.Listener
	ch      chan *request
	clients int
}


// New creates and returns (but does not start) a new KeyValueServer.
```

[1] Test-Driven
[2] Slow-Read

```
func New() KeyValueServer {
	return &keyValueServer{
		clients: 0,
		ch:      make(chan *request),
	}
}


func (kvs *keyValueServer) Start(port int) error {
	ln, err := net.Listen("tcp", fmt.Sprintf(":%d", port))
	if err != nil {
		return err
	}
	kvs.ln = ln
	createDB()
	go kvs.dispatch()
	go kvs.listen()
	return nil
}


func (kvs *keyValueServer) Close() {
	// TODO: implement this!
}


func (kvs *keyValueServer) Count() int {
	return kvs.clients
}


func (kvs *keyValueServer) listen() {
	for {
		conn, err := kvs.ln.Accept()
		if err != nil {
			return
		}
		kvs.clients++
		go kvs.handle(conn)
	}


}
```

```
func (kvs *keyValueServer) handle(conn net.Conn) {
	cr := bufio.NewReader(conn)
	cw := bufio.NewWriter(conn)
	for {
		var command, key string
		var value []byte


		buf, err := cr.ReadBytes(',')
		if err != nil {
			kvs.clients--
			return
		}
		command = string(buf[:len(buf)-1])
		if command == "get" {
			buf, err := cr.ReadBytes('\n')
			if err != nil {
				kvs.clients--
				return
			}
			key = string(buf[:len(buf)-1])
			kvs.ch <- &request{
				command: "get",
				key:     key,
				writer:  cw,
			}
		} else if command == "set" {
			buf, err := cr.ReadBytes(',')
			if err != nil {
				kvs.clients--
				return
			}
			key = string(buf[:len(buf)-1])
			buf, err = cr.ReadBytes('\n')
			if err != nil {
				kvs.clients--
				return
			}
			value = buf[:len(buf)-1]
			kvs.ch <- &request{
				command: "set",
```

```
                                key:     key,
                                value:   value,
                                writer:  cw,
                        }
                }
        }
}


func (kvs *keyValueServer) dispatch() {
        for {
                select {
                case r := <-kvs.ch:
                        if r.command == "get" {
                                r.writer.WriteString(fmt.Sprintf("%s,%s\n", r.key,
get(r.key)))
                                r.writer.Flush()
                        }
                        if r.command == "set" {
                                set(r.key, r.value)
                        }
                }
        }
}
```

کد 1 نسخه‌ی نهایی فاز اول

فاز دوم

در این فاز به جای یک کلاینت از چند کلاینت پشتیبانی شد، به این منظور نیاز بود لیستی از کلاینت‌ها به صورت اشتراکی بین goroutineها وجود داشته باشد، به این منظور در ابتدا یک concurrent list با همان روش تست-محور پیاده‌سازی شد (در پوشه clist پیوست شده است) و در ادامه با استفاده از آن پاسخ یک تقاضای get از یک user به تمام کلاینت‌ها ارسال می‌شد. در این قسمت کد فاز قبل با اضافه شدن یک تابع sender که پاسخ را به همه‌ی کلاینت‌ها ارسال می‌کرد به صورت نهایی پیاده‌سازی شد. پیاده‌سازی در کد2 پیوست شده است.

```
package p1

import (
        "bufio"
```

```
	"fmt"
	"net"


	"../clist"
)


type keyValueServer struct {
	ln      net.Listener
	res     chan *request
	req     chan *request
	clients *clist.ConcurrentList
}


// New creates and returns (but does not start) a new KeyValueServer.
func New() KeyValueServer {
	return &keyValueServer{
		clients: clist.New(),
		req:     make(chan *request, 500),
		res:     make(chan *request, 500),
	}
}


func (kvs *keyValueServer) Start(port int) error {
	ln, err := net.Listen("tcp", fmt.Sprintf(":%d", port))
	if err != nil {
		return err
	}
	kvs.ln = ln
	createDB()
	go kvs.dispatch()
	go kvs.listen()
	go kvs.sender()
	return nil
}


func (kvs *keyValueServer) Close() {
	// TODO: implement this!
}
```

```
func (kvs *keyValueServer) Count() int {
	return kvs.clients.Len()
}


func (kvs *keyValueServer) listen() {
	for {
		conn, err := kvs.ln.Accept()
		if err != nil {
			return
		}


		c := &client{
			conn:   conn,
			writer: bufio.NewWriter(conn),
			reader: bufio.NewReader(conn),
		}
		kvs.clients.PushBack(c)
		go kvs.receiver(c)
	}


}


func (kvs *keyValueServer) sender() {
	for {
		select {
		case r := <-kvs.res:
			cs := kvs.clients.Iter()
			for c := range cs {

	c.Value.(*client).writer.WriteString(fmt.Sprintf("%s,%s\n", r.key, r.value))
				c.Value.(*client).writer.Flush()


			}
		default:
			continue
		}
```

```
	}
}


func (kvs *keyValueServer) receiver(c *client) {


	defer func() {
		kvs.clients.Remove(c)
	}()
	for {
		var command, key string
		var value []byte


		buf, err := c.reader.ReadBytes(',')
		if err != nil {
			return
		}
		command = string(buf[:len(buf)-1])
		if command == "get" {
			buf, err := c.reader.ReadBytes('\n')
			if err != nil {
				return
			}
			key = string(buf[:len(buf)-1])
			kvs.req <- &request{
				command: "get",
				key:     key,
			}
		} else if command == "set" {
			buf, err := c.reader.ReadBytes(',')
			if err != nil {
				return
			}
			key = string(buf[:len(buf)-1])
			buf, err = c.reader.ReadBytes('\n')
			if err != nil {
				return
			}
			value = buf[:len(buf)-1]
			kvs.req <- &request{
				command: "set",
```

```
                        key:     key,
                        value:   value,
                    }
                }
            }
        }


        func (kvs *keyValueServer) dispatch() {
            for {
                select {
                case r := <-kvs.req:
                    if r.command == "get" {
                        kvs.res <- &request{
                            key:   r.key,
                            value: get(r.key),
                        }
                    }
                    if r.command == "set" {
                        set(r.key, r.value)
                    }
                }
            }
        }
```

کد 2 نسخه‌ی نهایی فاز دوم

فاز سوم

در فاز نهایی برای بدست آوردن کارآیی لازم و جلوگیری از ازیاد پیام‌های کلاینت‌های آهسته-خوان تغییر اساسی در کد داده شد. همانطور که اشاره شده بود پیش از این دو channel برای تقاضاها و پاسخ‌ها برای Key-Value-Server وجود داشت، در این فاز برای هر کلاینت دو channel برای تقاضاها و پاسخ‌ها در نظر گرفته شد و به این ترتیب سرعت پردازش پیام‌ها افزایش یافت زیرا هر کلاینت goroutine مختص خود را برای ارسال پاسخ داشت و از سوی دیگر در این مدل امکان محدودسازی تعداد تقاضاهای پردازش نشده نیز برای هر کلاینت وجود داشت، به این ترتیب با این معماری در این فاز پروژه به صورت کامل به پایان رسیده و تمامی تست‌ها به درستی صحت‌سنجی شدند.

```go
package p1
import (
	"bufio"
	"fmt"
	"net"
	"runtime"


	"../clist"
)


type keyValueServer struct {
	ln      net.Listener
	clients *clist.ConcurrentList
}


// New creates and returns (but does not start) a new KeyValueServer.
func New() KeyValueServer {
	return &keyValueServer{
		clients: clist.New(),
	}
}


func (kvs *keyValueServer) Start(port int) error {
	ln, err := net.Listen("tcp", fmt.Sprintf(":%d", port))
	if err != nil {
		return err
	}
	kvs.ln = ln
	createDB()
	go kvs.dispatch()
	go kvs.listen()
	return nil
}


func (kvs *keyValueServer) Close() {
	kvs.ln.Close()
}
```

```
func (kvs *keyValueServer) Count() int {
	return kvs.clients.Len()
}


func (kvs *keyValueServer) listen() {
	for {
		conn, err := kvs.ln.Accept()
		if err != nil {
			return
		}


		c := &client{
			conn:   conn,
			writer: bufio.NewWriter(conn),
			reader: bufio.NewReader(conn),
			req:    make(chan *request, 500),
			res:    make(chan *request, 500),
		}
		kvs.clients.PushBack(c)
		go kvs.sender(c)
		go kvs.receiver(c)
	}


}


func (kvs *keyValueServer) sender(c *client) {
	for {
		r := <-c.res
		c.writer.WriteString(fmt.Sprintf("%s,%s\n", r.key, r.value))
		c.writer.Flush()
	}
}


func (kvs *keyValueServer) receiver(c *client) {
```

```
		defer func() {
			kvs.clients.Remove(c)
		}()
		for {
			var command, key string
			var value []byte


			buf, err := c.reader.ReadBytes(',')
			if err != nil {
				return
			}
			command = string(buf[:len(buf)-1])
			if command == "get" {
				buf, err := c.reader.ReadBytes('\n')
				if err != nil {
					return
				}
				key = string(buf[:len(buf)-1])
				c.req <- &request{
					command: "get",
					key:     key,
				}
			} else if command == "set" {
				buf, err := c.reader.ReadBytes(',')
				if err != nil {
					return
				}
				key = string(buf[:len(buf)-1])
				buf, err = c.reader.ReadBytes('\n')
				if err != nil {
					return
				}
				value = buf[:len(buf)-1]
				c.req <- &request{
					command: "set",
					key:     key,
					value:   value,
				}
			}
		}
	}
```

```
func (kvs *keyValueServer) dispatch() {
	for {
		var res []*request
		for c := range kvs.clients.Iter() {
			select {
			case r := <-c.Value.(*client).req:
				if r.command == "get" {
					res = append(res, &request{
						key:   r.key,
						value: get(r.key),
					})
				}
				if r.command == "set" {
					set(r.key, r.value)
				}
			default:
				continue
			}
		}
		runtime.Gosched()
		if len(res) != 0 {
			for c := range kvs.clients.Iter() {
				for _, r := range res {
					select {
					case c.Value.(*client).res <- r:
					default:
						break
					}
				}
			}


		}
	}
}
```

کد 3 نسخه‌ی نهایی فاز سوم