به نام او
درس سیستم‌های توزیع‌شده‌ی دی‌ماه ۱۴۰۰ - استاد راهنما: دکتر کلباسی - دانشجو: کیارش آذرنیا

گزارش پروژه پیاده‌سازی یک سیستم فراخوانی متد محلی

Remote Method Invocation Implementation Report

مقدمه

فراخوانی متد از راه دور این کمک در معماری سیستم‌های توزیع‌شده نقش بسیار مهمی دارد چرا که امکان استفاده از توان پردازشی ماشین‌های محلی را فراهم می‌آورد.

پیاده‌سازی پروژه به زبان گولنگ انجام شده است یک زبان با کارایی بالا برای توسعه‌ی سیستم‌های توزیع‌شده در گوگل طراحی و ساخته شده است.

طراحی سازوکار فراخوانی متد به شکل محلی

در ادامه پیاده‌سازی فراخوانی متد از راه دور با الگوهای زبان گولنگ که عمدتا عبارت است از Interface و Function و Struct را شرح می‌دهیم.

یک آبجکت ساده که فقط می‌تواند سلام کند را در نظر می‌گیریم، برای این کلاس در گولنگ یک اینترفیس می‌نویسیم. این اینترفیس بین کلاینت و سرور مشترک بوده و شفافیت محلی را برای ما مهیا خواهد کرد. لذا کد آن را در کتابخانه‌ای با عنوان rmi توسعه داده‌ایم.

```
72  type Hello interface {
73      SayHello() string
74  }
```

حالا برای برای پیاده‌سازی این اینترفیس سمت سرور یک استراکت با نام سلام‌کننده‌ی محلی (HelloRemoteObject) می‌سازیم و متد SayHello را برای آن به این ترتیب پیاده‌سازی می‌کنیم.

```go
type HelloRemoteObject struct {
    helloSentence string
}

func (h HelloRemoteObject) SayHello() string {
    return h.helloSentence
}
```

حالا در سمت سرور می‌توانیم این استراکت را فارغ از تایپ اصلی آن، از جنس اینترفیس مشترک ببینیم. مثلا در این قطعه کد می‌بینیم که اشاره‌گر آن از جنس Hello است و نه HelloRemoteObject:

```go
// we instantiate HelloRemoteObject but save it in an Hello type variable
var hello Hello = HelloRemoteObject{
    helloSentence: "Hello World",
}
```

سمت کلاینت نیز مسیر مشابهی طی می‌کنیم. یک استراکت با نام سلام‌کننده‌ی مصنوعی (HelloStub) می‌سازیم و دقیقا مشابه حالت قبل متد SayHello را برای آن پیاده‌سازی می‌کنیم.

```go
type HelloStub struct {
    name          string
    version       int
    remoteAddress string
}

func (h *HelloStub) SayHello() string {
    body, _ := json.Marshal(h)
    requestBody := bytes.NewBuffer(body)
    response, _ := http.Post(h.remoteAddress, "application/json", requestBody)
    defer response.Body.Close()
    responseBody, _ := ioutil.ReadAll(response.Body)
    return string(responseBody)
}
```

سمت کلاینت نیز وقتی آبجکت از رجیستری محلی لوک-آپ می‌شود، آن را در متغیری با تایپ Hello می‌ریزیم و متدش را کال می‌کنیم.

```go
var hello rmi.Hello = lookup("<rmi.Hello Value>", 1).(rmi.Hello)
result := hello.SayHello()
```

به این ترتیب هر دو طرف اینترفیس Hello استفاده می‌شود با پیاده‌سازی‌های متفاوت:
سمت سرور پیاده‌سازی اصلی و ارسال پاسخ (در حال پیاده‌سازی)
سمت کلاینت سریالایز کردن و ارسال درخواست به سرور.

معماری سیستم

پیاده‌سازی انجام شده بر پایه سه پراسس است:

server.out

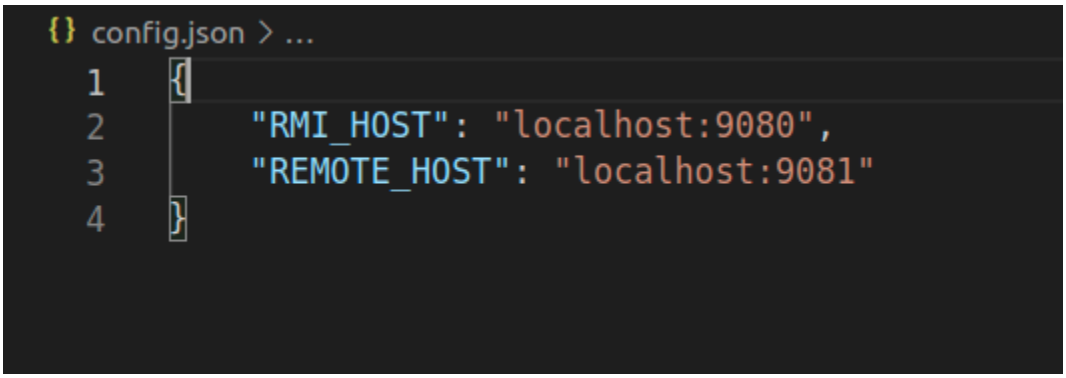
سرور محلی‌ای است که آبجکت‌های اصلی را میزبانی می‌کند. می‌توان چند سرور داشت که روی آی-پی پورت‌های مختلف ورژن‌های متفاوتی از آبجکت‌ها ارائه می‌دهند.

client.out

کلاینتی که از سرویس‌ها استفاده می‌کند. می‌توان بی‌نهایت از این کلاینت‌ها با هر سناریویی از فراخوانی متدها داشت.

rmi.out

سرور رجیستری آبجکت‌ها که سرور محلی میزبان آبجکت‌ها باید به آن اعلام حضور کند. در پیاده‌سازی من تنها یک سرور رجیستری باید حاضر باشد تا کلاینت از او آدرس آبجکت‌ها را جویا شود.



تصویر فایل کانفیگ که برای پیکربندی استفاده شده است.

یک نمونه اجرا

در ادامه نمایی از اجرای پروژه در سه ترمینال مختلف را می‌بینیم. نکته جالب تفاوت زمان اجرای کلاس Fibonacci در ورژن یک و دو است که به ترتیب به روش بازگشتی با زمان حدودا ۵۰ ثانیه و به روش دینامیک با زمان کمتر از یک ثانیه برای جمله‌ی پنجاهم دنباله‌ی فیبوناچی صرف کرده است.

تصویر لاگ‌های سه برنامه کلاینت، سرور میزبان و سرور رجیستری در ادامه آمده است.

```
kiarash@zen:~/Documents/uni/polytech/term1/distributed_systems/minimal-rmi$ ./client.out
2022/01/05 23:36:44 localhost:9080
Server is responsive: localhost:9080
2022/01/05 23:36:44 looking up remote object: {1 Fibonacci}
2022/01/05 23:36:44 looking up result: &{200 OK 200 HTTP/1.1 1 1 map[Content-Length:[66] Content-Type:[text/plain; charset=utf-8] Date:[Wed, 05 Jan 2022
 20:06:44 GMT]] 0xc00020e040 66 [] false false map[] 0xc0001b4000 <nil>} <nil>
2022/01/05 23:36:44 making stub type: Fibonacci:1  object: main.FibonacciClientStub
2022/01/05 23:36:44 object value: <main.FibonacciClientStub Value>
2022/01/05 23:36:44 value interface: main.FibonacciClientStub
2022/01/05 23:36:44 conversion result: &{0 }
2022/01/05 23:36:44 client stub calling method: {Fibonacci 1 Fibonacci 50 true}
2022/01/05 23:36:44 sending request: {"ObjectName":"Fibonacci","Version":1,"MethodName":"Fibonacci","Parameters":"50","HasParameters":true}  address: ht
tp://localhost:9081/remote
2022/01/05 23:37:46 response: &{200 OK 200 HTTP/1.1 1 1 map[Content-Length:[11] Content-Type:[text/plain; charset=utf-8] Date:[Wed, 05 Jan 2022 20:07:46
 GMT]] 0xc00020e240 11 [] false false map[] 0xc000240000 <nil>} <nil>
2022/01/05 23:37:46 response: 12586269025
2022/01/05 23:37:46 RMI result: 12586269025 <nil>
2022/01/05 23:37:46 **** Fibonacci version:1 took 1m1.956864076s ******
2022/01/05 23:37:46 fiboroial 50!=12586269025
2022/01/05 23:37:46 looking up remote object: {2 Fibonacci}
2022/01/05 23:37:46 looking up result: &{200 OK 200 HTTP/1.1 1 1 map[Content-Length:[66] Content-Type:[text/plain; charset=utf-8] Date:[Wed, 05 Jan 2022
 20:07:46 GMT]] 0xc00020e300 66 [] false false map[] 0xc000240200 <nil>} <nil>
2022/01/05 23:37:46 making stub type: Fibonacci:2  object: main.FibonacciClientStub
2022/01/05 23:37:46 object value: <main.FibonacciClientStub Value>
2022/01/05 23:37:46 value interface: main.FibonacciClientStub
2022/01/05 23:37:46 conversion result: &{0 }
2022/01/05 23:37:46 client stub calling method: {Fibonacci 2 Fibonacci 50 true}
2022/01/05 23:37:46 sending request: {"ObjectName":"Fibonacci","Version":2,"MethodName":"Fibonacci","Parameters":"50","HasParameters":true}  address: ht
tp://localhost:9081/remote
2022/01/05 23:37:46 response: &{200 OK 200 HTTP/1.1 1 1 map[Content-Length:[11] Content-Type:[text/plain; charset=utf-8] Date:[Wed, 05 Jan 2022 20:07:46
 GMT]] 0xc0001c41c0 11 [] false false map[] 0xc000240400 <nil>} <nil>
2022/01/05 23:37:46 response: 12586269025
2022/01/05 23:37:46 RMI result: 12586269025 <nil>
2022/01/05 23:37:46 **** Fibonacci version:2 took 966.523µs ******
2022/01/05 23:37:46 fiboroial 50!=12586269025
kiarash@zen:~/Documents/uni/polytech/term1/distributed_systems/minimal-rmi$
```

```
kiarash@zen:~/Documents/uni/polytech/term1/distributed_systems/minimal-rmi$ ./rmi.out
2022/01/05 23:36:36 localhost:9080
running rmi server
2022/01/05 23:36:36 running server on localhost:9080
2022/01/05 23:36:39 {1 Hello localhost:9081}
2022/01/05 23:36:39 {2 Hello localhost:9081}
2022/01/05 23:36:39 {1 Fibonacci localhost:9081}
2022/01/05 23:36:39 {2 Fibonacci localhost:9081}
```

```
kiarash@zen:~/Documents/uni/polytech/term1/distributed_systems/minimal-rmi$ ./server.out
2022/01/05 23:36:39 localhost:9080
Server is responsive: localhost:9080
2022/01/05 23:36:39 registering remote object: {1 Hello localhost:9081}
2022/01/05 23:36:39 registeration result and error: &{200 OK 200 HTTP/1.1 1 1 map[Content-Length:[0] Date:[Wed, 05 Jan 2022 20:06:39 GMT]] {} 0 [] false
 false map[] 0xc00022c000 <nil>} <nil>
2022/01/05 23:36:39 registering remote object: {2 Hello localhost:9081}
2022/01/05 23:36:39 registeration result and error: &{200 OK 200 HTTP/1.1 1 1 map[Content-Length:[0] Date:[Wed, 05 Jan 2022 20:06:39 GMT]] {} 0 [] false
 false map[] 0xc00017a000 <nil>} <nil>
2022/01/05 23:36:39 registering remote object: {1 Fibonacci localhost:9081}
2022/01/05 23:36:39 registeration result and error: &{200 OK 200 HTTP/1.1 1 1 map[Content-Length:[0] Date:[Wed, 05 Jan 2022 20:06:39 GMT]] {} 0 [] false
 false map[] 0xc0001c2000 <nil>} <nil>
2022/01/05 23:36:39 registering remote object: {2 Fibonacci localhost:9081}
2022/01/05 23:36:39 registeration result and error: &{200 OK 200 HTTP/1.1 1 1 map[Content-Length:[0] Date:[Wed, 05 Jan 2022 20:06:39 GMT]] {} 0 [] false
 false map[] 0xc00017a200 <nil>} <nil>
2022/01/05 23:36:39 running remote server on: localhost:9081
2022/01/05 23:36:44 handing remote method invocation
2022/01/05 23:36:44 decoded RMI request: {Fibonacci 1 Fibonacci 50 true}
2022/01/05 23:36:44 invoking method: {Fibonacci 1 Fibonacci 50 true}
2022/01/05 23:36:44 interface: {} name: Fibonacci args: [50] args len: 1
2022/01/05 23:36:44 inputs: [<uint64 Value>] len: 1
2022/01/05 23:37:46 method call result: [<uint64 Value>]
2022/01/05 23:37:46 handing remote method invocation
2022/01/05 23:37:46 decoded RMI request: {Fibonacci 2 Fibonacci 50 true}
2022/01/05 23:37:46 invoking method: {Fibonacci 2 Fibonacci 50 true}
2022/01/05 23:37:46 interface: {} name: Fibonacci args: [50] args len: 1
2022/01/05 23:37:46 inputs: [<uint64 Value>] len: 1
2022/01/05 23:37:46 method call result: [<uint64 Value>]
```

اجرای برنامه

در محیط لینوکس و با نصب کامپایلر گولنگ انجام دستور make build برای کامپایل شدن همه‌ی فایل‌ها کفایت می‌کند.
برای توضیحات تکمیلی به فایل موجود در README.md در ریپازیتوری مراجعه نمایید.

ضمنا ارجاعات انجام شده به منابع در پایان فایل README.md انجام شده است. آدرس ریپازیتوری گیتهاب پروژه که فعلا به صورت پرایوت است در این لینک آمده است.

سپاس و آرزوی سلامتی