RPC in Go

CS 4740: Cloud Computing

Fall 2024

Lecture 6

Yue Cheng

Some material taken/derived from:

- Princeton COS-418 materials created by Michael Freedman and Wyatt Lloyd.
- MIT 6.824 by Robert Morris, Frans Kaashoek, and Nickolai Zeldovich.

Go RPC

- Implementation in built-in library net/rpc

Go RPC

- Implementation in built-in library net/rpc
- Write stub receiver methods of the form
  - **`func (t *T) MethodName(args T1, reply *T2) error`**

Go RPC

- Implementation in built-in library net/rpc
- Write stub receiver methods of the form
  - **`func (t *T) MethodName(args T1, reply *T2) error`**
- Register receiver methods

Go RPC

- Implementation in built-in library net/rpc
- Write stub receiver methods of the form
  - **`func (t *T) MethodName(args T1, reply *T2) error`**
- Register receiver methods
- Create a listener (i.e., server) that accepts requests

Writing a WordCount RPC server in Go

```
type WordCountServer struct {
      addr string
}

type WordCountRequest struct {
      Input string
}

type WordCountReply struct {
      Counts map[string]int
}
```

Writing a WordCount RPC server in Go

```
type WordCountServer struct {
      addr string
}

type WordCountRequest struct {
      Input string
}

type WordCountReply struct {
      Counts map[string]int
}
```

```
func (*WordCountServer) Compute(
            request WordCountRequest,
            reply *WordCountReply) error {
      counts := make(map[string]int)
      input := request.Input
      tokens := strings.Fields(input)
      for _, t := range tokens {
            counts[t] += 1
      }
      reply.Counts = counts
      return nil
}
```

Writing a WordCount RPC server in Go

```
type WordCountServer struct {
      addr string
}

type WordCountRequest struct {
      Input string
}

type WordCountReply struct {
      Counts map[string]int
}
```

```
func (*WordCountServer) Compute(
            request WordCountRequest,
            reply *WordCountReply) error {
      counts := make(map[string]int)
      input := request.Input
      tokens := strings.Fields(input)
      for _, t := range tokens {
            counts[t] += 1
      }
      reply.Counts = counts
      return nil
}
```

Writing a WordCount RPC server in Go

```
func (server *WordCountServer) Listen() {
	rpc.Register(server)
	listener, err := net.Listen("tcp", server.addr)
	checkError(err)
	go func() {
		rpc.Accept(listener)
	}()
}
```

Writing a WordCount RPC server in Go

```
func (server *WordCountServer) Listen() {
	rpc.Register(server)
	listener, err := net.Listen("tcp", server.addr)
	checkError(err)
	go func() {
		rpc.Accept(listener)
	}()
}
```

Writing a WordCount RPC server in Go

```
func (server *WordCountServer) Listen() {
	rpc.Register(server)
	listener, err := net.Listen("tcp", server.addr)
	checkError(err)
	go func() {
		rpc.Accept(listener)
	}()
}
```

Writing a WordCount RPC server in Go

```
func (server *WordCountServer) Listen() {
      rpc.Register(server)
      listener, err := net.Listen("tcp", server.addr)
      checkError(err)
      go func() {
            rpc.Accept(listener)
      }()
}
```

WordCount client

```
func makeRequest(input string, serverAddr string) (map[string]int, error) {
	client, err := rpc.Dial("tcp", serverAddr)
	checkError(err)
	args := WordCountRequest{input}
	reply := WordCountReply{make(map[string]int)}
	err = client.Call("WordCountServer.Compute", args, &reply)
	if err != nil {
		return nil, err
	}
	return reply.Counts, nil
}
```

WordCount client

```
func makeRequest(input string, serverAddr string) (map[string]int, error) {
	client, err := rpc.Dial("tcp", serverAddr)
	checkError(err)
	args := WordCountRequest{input}
	reply := WordCountReply{make(map[string]int)}
	err = client.Call("WordCountServer.Compute", args, &reply)
	if err != nil {
		return nil, err
	}
	return reply.Counts, nil
}
```

WordCount client

```
func makeRequest(input string, serverAddr string) (map[string]int, error) {
	client, err := rpc.Dial("tcp", serverAddr)
	checkError(err)
	args := WordCountRequest{input}
	reply := WordCountReply{make(map[string]int)}
	err = client.Call("WordCountServer.Compute", args, &reply)
	if err != nil {
		return nil, err
	}
	return reply.Counts, nil
}
```

WordCount client

```
func makeRequest(input string, serverAddr string) (map[string]int, error) {
	client, err := rpc.Dial("tcp", serverAddr)
	checkError(err)
	args := WordCountRequest{input}
	reply := WordCountReply{make(map[string]int)}
	err = client.Call("WordCountServer.Compute", args, &reply)
	if err != nil {
		return nil, err
	}
	return reply.Counts, nil
}
```

WordCount client-server

```
func main() {
	serverAddr := "localhost:8888"
	server := WordCountServer{serverAddr}
	server.Listen()
	input1 := "hello I am good hello bye bye bye bye good night hello"
	wordcount, err := makeRequest(input1, serverAddr)
	checkError(err)
	fmt.Printf("Result: %v\n", wordcount)
}
```

WordCount client-server

```
func main() {
	serverAddr := "localhost:8888"
	server := WordCountServer{serverAddr}
	server.Listen()
	input1 := "hello I am good hello bye bye bye bye good night hello"
	wordcount, err := makeRequest(input1, serverAddr)
	checkError(err)
	fmt.Printf("Result: %v\n", wordcount)
}
```

```
Result: map[hello:3 I:1 am:1 good:2 bye:4 night:1]
```

Is this synchronous or asynchronous?

```
func makeRequest(input string, serverAddr string) (map[string]int, error)
{
	client, err := rpc.Dial("tcp", serverAddr)
	checkError(err)
	args := WordCountRequest{input}
	reply := WordCountReply{make(map[string]int)}
	err = client.Call("WordCountServer.Compute", args, &reply)
	if err != nil {
		return nil, err
	}
	return reply.Counts, nil
}
```

Making client asynchronous

```
func makeRequest(input string, serverAddr string) chan Result {
      client, err := rpc.Dial("tcp", serverAddr)
      checkError(err)
      args := WordCountRequest{input}
      reply := WordCountReply{make(map[string]int)}









      return ch
}
```

Making client asynchronous

```
func makeRequest(input string, serverAddr string) chan Result {
	client, err := rpc.Dial("tcp", serverAddr)
	checkError(err)
	args := WordCountRequest{input}
	reply := WordCountReply{make(map[string]int)}
	ch := make(chan Result)
	go func() {
		err := client.Call("WordCountServer.Compute", args, &reply)
		if err != nil {
			ch <- Result{nil, err} // something went wrong
		} else {
			ch <- Result{reply.Counts, nil} // success
		}
	}()
	return ch
}
```

Lab 1 tutorial