

# 如雷贯耳

喜欢听敲代码时的键盘声，清脆悦耳

[首页](#)[关于本人](#)

## 《RabbitMQ入门之Go语言教程》(1) Hello World

作者: [Ray Lei](#) | 时间: May 30, 2018 | 分类: [消息队列](#) | (UV)访问: 35,618 次

本系列摘自[RabbitMQ官方教程](#)，边学习边翻译的中文的版本，水平有限，不妥之处，欢迎交流。

RabbitMQ是一个开源的、使用最广的消息转发器。提供匿名消息类型，可使用多种消息协议、消息队列等，支持集群部署，

它的轻量级能够让你很轻易地在私有云或公有云环境中部署，同时提供丰富的插件工具，能够让你轻易的进行扩展，且可使用HTTP-API、命令行工具和UI界面以便于管理和监控。

RabbitMQ是一个消息转发器，它对消息进行接收并转发。你可以把它想象成邮局，当你把信件投入邮筒之后，邮递员会根据信笺上的地址进行投递。RabbitMQ这里就扮演中邮筒、邮局和邮递员的角色。RabbitMQ跟现实中的邮局所不同的是它不会对真实的信件进行投递，而是对以二进制数据-消息进行存储、转发。

我们先看下几个术语：

- 生产者(**producer**)：产生并发送消息的程序；
- 队列(**queue**)：存在RabbitMQ中的邮筒，虽然消息是在应用程序和RabbitMQ中进行传递，但队列才是唯一能够存储消息的地方。队列的大小取决于宿主机器的内存和磁盘容量，它本质上是一个巨大的消息缓存池。多个生产者可以发送消息给同一个队列，多个消费者也可以从同一个队列中读取消息；
- 消费者(**consuming**)：等待接收消息的程序；

生产者、队列及消费者可以不在同一个宿主机上，通常在实际应用中他们确都是分开部署的。

### Hello World

我们将实现一个使用RabbitMQ的Hello World示例，其中包含两个小程序，一个用于发送消息的send.go，一个用于接收消息的receive.go。

在下面的图示中："P"表示生产者，"C"表示消费者，中间的表格表示队列，即为RabbitMQ中的消息池。



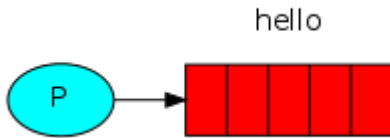
关于Go语言使用RabbitMQ的API，可参考[GO amqp](#)。

我们在这里使用amqp库，因此我们需要先安装GO amqp客户端。

首先，使用go get安装amqp：

```
go get github.com/streadway/amqp
```

### Sending



我们使用send.go称把消息生产者，receive.go成为消费者。send.go连接到RabbitMQ后，发送一条消息后便退出。

首先，需要导入amqp:

```
package main

import (
    "fmt"
    "log"

    "github.com/streadway/amqp"
)
```

然后，我们编写一个通用的辅助方法，用来检查每一步amqp调用的结果。另外，在Go语言中经常需要使用if语句来检查操作结果，为了避免在代码中到处散落if(err != nil)语句，可以使用下列方法：

```
func failOnError(err error, msg string){
    if err != nil {
        log.Fatalf("%s:%s", msg, err)
        panic(fmt.Sprintf("%s:%s", msg, err))
    }
}
```

下面，我们来实现main函数：

连接RabbitMQ服务器：

```
conn, err := amqp.Dial("amqp://guest:guest@localhost:5672")
failOnError(err, "Failed to connect to RabbitMQ")
defer conn.Close()
```

上面代码会建立一个socket连接，处理一些协议转换及版本对接和登录授权的问题。建立连接之后，我们需要创建一个通道channel，之后我们的大多数API操作都是围绕通道来实现的：

```
ch, err := conn.Channel()
failOnError(err, "Failed to open a channel")
defer ch.Close()
```

最后，我们需要定义一个队列用来存储、转发消息，然后我们的sender只需要将消息发送到这个队列中，就完成了消息的publish操作：

```
q, err := ch.QueueDeclare(
    "hello", //name
    false,   //durable
    false,   //delete when unused
    false,   //exclusive
    false,   //no wait
    nil,     //arguments
)
failOnError(err, "Failed to declare q queue")
```

```

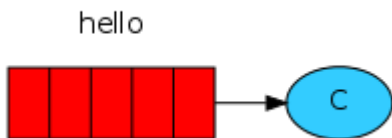
body := "Hello"
err = ch.Publish(
    "",          //exchange
    q.Name,      // routing key
    false,      //mandatory
    false,      //immediate
    amqp.Publishing{
        ContentType: "text/plain",
        Body :      []byte(body),
    }
)

```

定义队列操作具备幂等性，也就是说多次重复定义，相同名称的队列只会创建一个。发送给队列的内容是byte数组，将任意格式数据转换成byte数组是一件很简单的事情，因此对于任何格式的数据，要将其发送到队列中是很容易的。

## Receiving

以上完成了发送消息的程序，现在来实现从RabbitMQ队列中接收消息的消费者程序。不同于消息发送程序只需要将单一的消息推送至队列后推出，消息接收者需要保持一个监听程序从队列中不断的接收消息。



首先，同样的导入包和实现辅助函数：

```

package main

import {
    "fmt"
    "log"

    "github.com/streadway/amqp"
}

func failOnError(err error, msg string){
    if err != nil {
        log.Fatalf("%s:%s", msg, err)
        panic(fmt.Sprintf("%s:%s", msg, err))
    }
}

```

接着，与生产者一样，打开连接并创建通道，注意这里的参数必须与send中的queue name相一致，这样才能实现发送/接受的配对。

```

conn, err := amqp.Dial("amqp://guest:guest@localhost:5672")
failOnError(err, "Failed to connect to server")
defer conn.Close();

ch, err := conn.Channel()
failOnError(err, "Failed to connect to channel")
defer ch.Close()

q, err := ch.QueueDeclare(

```

```
    "hello",    //name
    false,     //durable
    false,     //delete when unused
    false,     // exclusive
    false,     //no-wait
    nil,       // arguments
)

failOnError(err, "Failed to declare a queue")
```

一般来说，接收消息的程序会先于发送者运行，因此在这里我们先定义一个queue，确保后面发送者连接到这个queue时，当前接收消息程序以运行。

接下来，需要RabbitMQ服务器让它将消息分发到我们的消费者程序中，消息转发操作是异步执行的，这里使用goroutine来完成从队列中的读取消息操作：

```
msgs, err := ch.Consume(
    q.Name,      // queue
    "",         // consumer
    true,       // auto-ack
    false,      // exclusive
    false,      // no-local
    false,      // no-wait
    nil,        // arguments
)
failOnError(err, "Failed to register a consumer")

forever := make(chan bool)

go func(){
    for d:= range msgs{
        log.Printf("Received a message : %s", d.Body)
    }
}()

log.Printf(" [*] Waiting for messages, To exit press CTRL+C")
<-forever
```

## Running

首先，在命令行中先运行消费者：

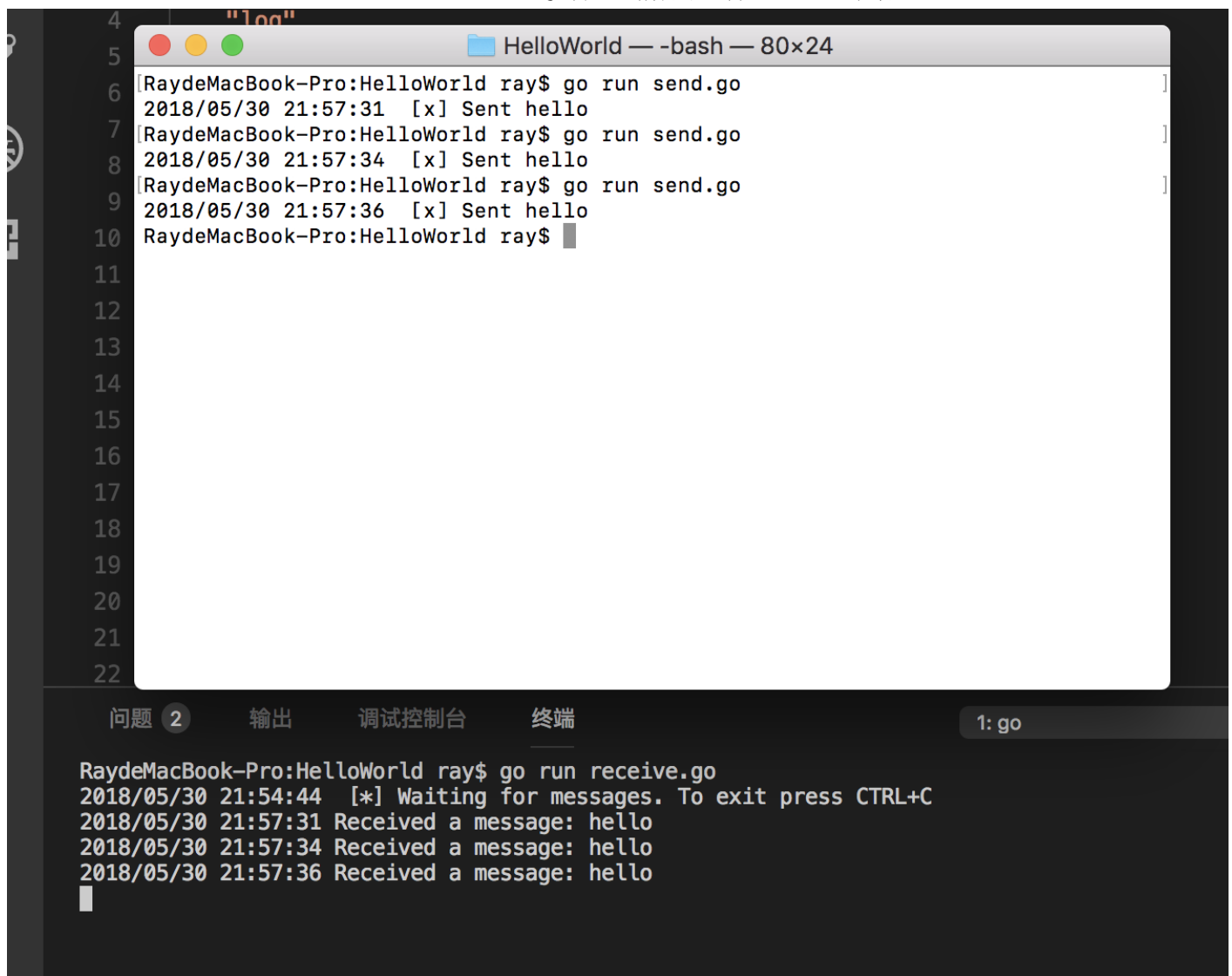
```
go run receive.go
```

当前程序会一直监听RabbitMQ的队列消息，一旦接收到消息后会直接打印出来，使用Ctrl+C可以终止程序；

接着，在另一个命令终端中运行生产者：

```
go run send.go
```

可以看到receive.go程序接收到了当前信息：



The screenshot shows a terminal window titled "HelloWorld — -bash — 80x24". The terminal output is as follows:

```
4 | "log"
5 |
6 | [RaydeMacBook-Pro:HelloWorld ray$ go run send.go
7 | 2018/05/30 21:57:31 [x] Sent hello
8 | [RaydeMacBook-Pro:HelloWorld ray$ go run send.go
9 | 2018/05/30 21:57:34 [x] Sent hello
10 | [RaydeMacBook-Pro:HelloWorld ray$ go run send.go
11 | 2018/05/30 21:57:36 [x] Sent hello
12 | RaydeMacBook-Pro:HelloWorld ray$
13 |
14 |
15 |
16 |
17 |
18 |
19 |
20 |
21 |
22 |
```

Below the terminal window, there are tabs for "问题 2", "输出", "调试控制台", and "终端". The "终端" tab is selected, showing the following output:

```
RaydeMacBook-Pro:HelloWorld ray$ go run receive.go
2018/05/30 21:54:44 [*] Waiting for messages. To exit press CTRL+C
2018/05/30 21:57:31 Received a message: hello
2018/05/30 21:57:34 Received a message: hello
2018/05/30 21:57:36 Received a message: hello
```

如上就是本系列的RabbitMQ的第一个例子。你可以在这里下载完整的[send.go](#), [receive.go](#)文件。

最后，是Go语言实现RabbitMQ入门系列的目录，有兴趣的可以参考一下：

- [RabbitMQ入门 \(1\) Hello world](#)
- [RabbitMQ入门 \(2\) 工作队列](#)
- [RabbitMQ入门 \(3\) 发布/订阅模式](#)
- [RabbitMQ入门 \(4\) 路由](#)
- [RabbitMQ入门 \(5\) 主题交换器](#)
- [RabbitMQ入门 \(6\) 远程过程调用\(RPC\)](#)

原文地址提供了其他语言的实现版本，可参考[RabbitMQ GetStarted](#).

标签: [RabbitMQ](#), [Go](#), [消息队列](#)

## 添加新评论

称呼 \*

Email \*