**RabbitMq学习笔记**

官网：http://www.rabbitmq.com/documentation.html

实例：http://www.rabbitmq.com/tutorials/tutorial-one-go.html

# 安装rabbitMq

|  |  |
| --- | --- |
| 安装Erlang | rpm -Uvh <http://www.rabbitmq.com/releases/erlang/erlang-18.1-1.el7.centos.x86_64.rpm>  如果无法安装，直接复制网址在浏览器下载rpm包，然后用命令rpm -ivh 包文件  或下载更低版本http://www.rabbitmq.com/releases/erlang/ |
| 安装rabbitMQ | rpm -Uvh <http://www.rabbitmq.com/releases/rabbitmq-server/v3.5.6/rabbitmq-server-3.5.6-1.noarch.rpm>  如果无法安装，直接复制网址在浏览器下载rpm包，然后用命令rpm -ivh 包文件 |

安装完后rabbitMQ服务默认帐号和密码都是guest，而且默认是不允许远程访问的。

## 服务器启动与关闭

|  |  |
| --- | --- |
| 启动 | service rabbitmq-server start |
| 关闭 | service rabbitmq-server stop |
| 重启 | service rabbitmq-server restart |

## 帐号管理

|  |  |
| --- | --- |
| 添加用户 | rabbitmqctl add\_user 新用户名 pwd密码  例如：rabbitmqctl add\_user vison pwd123456 |
| 删除用户 | rabbitmqctl delete\_user 用户名 |
| 修改密码 | rabbitmqctl change\_password 用户名 密码  例如：rabbitmqctl change\_password vison 123456 |
| 设置用户角色 | rabbitmqctl set\_user\_tags 用户名 角色名称  角色名称有administrator,monitoring, management  例如：rabbitmqctl set\_user\_tags vison administrator |

## 插件管理

|  |  |
| --- | --- |
| 开启某个插件 | rabbitmq-plugins enable 插件名称 |
| 关闭某个插件 | rabbitmq-plugins disable 插件名称 |

注：需要重启rabbitMQ服务

## 添加远程访问功能

vim /etc/rabbitmq/rabbitmq.config

添加内容：

|  |
| --- |
| [  {rabbit, [{tcp\_listeners, [5672]},  {loopback\_users, ["guest"]}]}  ]. |

重启rabbitMQ服务：service rabbitmq-server restart

## 开启RabbitMQ web管理工具

rabbitmq-plugins enable rabbitmq\_management

## 恢复初始状态(删除所有的exchange和队列)

在rabbitmq运行的情况下执行下面命令

|  |
| --- |
| rabbitmqctl stop\_app  rabbitmqctl reset  rabbitmqctl start\_app |

## 其他管理

rabbitmqctl -q status //打印了一些rabbitmq服务状态信息，包括内存，硬盘，和使用erlong的版本信息

rabbitmqctl list\_queues //查看所有队列消息

# 设置开机自启动

(1) 在/etc/init.d下新建一个脚本文件rabbitmq-server，内容如下。

(2) 修改为可执行文件：chmod +x rabbitmq-server

(3) 增加服务chkconfig --add rabbitmq-server

注:

|  |
| --- |
| 如何增加一个服务：  1.服务脚本必须存放在/etc/ini.d/目录下；  2.chkconfig --add servicename  在chkconfig工具服务列表中增加此服务，此时服务会被在/etc/rc.d/rcN.d中赋予K/S入口了；  3.chkconfig --level 35 mysqld on  修改服务的默认启动等级。 |

|  |
| --- |
| **#!/bin/sh** *# # rabbitmq-server RabbitMQ broker # # chkconfig: - 80 05 # description: Enable AMQP service provided by RabbitMQ #  ### BEGIN INIT INFO # Provides: rabbitmq-server # Required-Start: $remote\_fs $network # Required-Stop: $remote\_fs $network # Description: RabbitMQ broker # Short-Description: Enable AMQP service provided by RabbitMQ broker ### END INIT INFO  # Source function library.* . /etc/init.d/functions  PATH=/sbin:/usr/sbin:/bin:/usr/bin NAME=rabbitmq-server DAEMON=/usr/sbin/**$**{NAME} CONTROL=/usr/sbin/rabbitmqctl DESC=rabbitmq-server USER=rabbitmq ROTATE\_SUFFIX= INIT\_LOG\_DIR=/var/log/rabbitmq PID\_FILE=/var/run/rabbitmq/pid  START\_PROG=**"daemon"** LOCK\_FILE=/var/lock/subsys/$NAME  test -x $DAEMON **||** exit 0 test -x $CONTROL **||** exit 0  RETVAL=0 set -e  **[ -f** /etc/default/**$**{NAME} **] &&** . /etc/default/**$**{NAME}  **ensure\_pid\_dir** () {  PID\_DIR=*`dirname* **$**{PID\_FILE}*`* **if [** ! **-d $**{PID\_DIR} **]** ; **then** *mkdir* -p **$**{PID\_DIR}  *chown* -R **$**{USER}:**$**{USER} **$**{PID\_DIR}  *chmod* 755 **$**{PID\_DIR}  **fi** }  **remove\_pid** () {  *rm* -f **$**{PID\_FILE}  *rmdir `dirname* **$**{PID\_FILE}*`* **||** : }  **start\_rabbitmq** () {  *status\_rabbitmq* quiet  **if [** $RETVAL **=** 0 **]** ; **then** echo RabbitMQ is currently running  **else** RETVAL=0  *ensure\_pid\_dir* set +e  RABBITMQ\_PID\_FILE=$PID\_FILE *$START\_PROG* $DAEMON \  *> "${INIT\_LOG\_DIR}/startup\_log"* \  *2> "${INIT\_LOG\_DIR}/startup\_err"* \  *0<&-* &  *$CONTROL* wait $PID\_FILE *>/dev/null 2>&1* RETVAL=$?  set -e  **case "$RETVAL" in** 0)  echo SUCCESS  **if [ -n "$LOCK\_FILE" ]** ; **then** *touch* $LOCK\_FILE  **fi  ;;** \*)  *remove\_pid* echo FAILED - check **$**{INIT\_LOG\_DIR}/startup\_\{log, \_err\}  RETVAL=1  **;;  esac  fi** }  **stop\_rabbitmq** () {  *status\_rabbitmq* quiet  **if [** $RETVAL **=** 0 **]** ; **then** set +e  *$CONTROL* stop **$**{PID\_FILE} *> ${INIT\_LOG\_DIR}/shutdown\_log 2> ${INIT\_LOG\_DIR}/shutdown\_err* RETVAL=$?  set -e  **if [** $RETVAL **=** 0 **]** ; **then** *remove\_pid* **if [ -n "$LOCK\_FILE" ]** ; **then** *rm* -f $LOCK\_FILE  **fi  else** echo FAILED - check **$**{INIT\_LOG\_DIR}/shutdown\_log, \_err  **fi  else** echo RabbitMQ is not running  RETVAL=0  **fi** }  **status\_rabbitmq**() {  set +e  **if [ "$1" != "quiet" ]** ; **then** *$CONTROL* status *2>&1* **else** *$CONTROL* status *> /dev/null 2>&1* **fi  if [** $? **!=** 0 **]** ; **then** RETVAL=3  **fi** set -e }  **rotate\_logs\_rabbitmq**() {  set +e  *$CONTROL* rotate\_logs **$**{ROTATE\_SUFFIX}  **if [** $? **!=** 0 **]** ; **then** RETVAL=1  **fi** set -e }  **restart\_running\_rabbitmq** () {  *status\_rabbitmq* quiet  **if [** $RETVAL **=** 0 **]** ; **then** *restart\_rabbitmq* **else** echo RabbitMQ is not runnning  RETVAL=0  **fi** }  **restart\_rabbitmq**() {  *stop\_rabbitmq  start\_rabbitmq* }  **case "$1" in** start)  echo -n **"Starting $DESC: "** *start\_rabbitmq* echo **"$NAME."  ;;** stop)  echo -n **"Stopping $DESC: "** *stop\_rabbitmq* echo **"$NAME."  ;;** status)  *status\_rabbitmq* **;;** rotate-logs)  echo -n **"Rotating log files for $DESC: "** *rotate\_logs\_rabbitmq* **;;** force-reload|reload|restart)  echo -n **"Restarting $DESC: "** *restart\_rabbitmq* echo **"$NAME."  ;;** try-restart)  echo -n **"Restarting $DESC: "** *restart\_running\_rabbitmq* echo **"$NAME."  ;;** \*)  echo **"Usage: $0 {start|stop|status|rotate-logs|restart|condrestart|try-restart|reload|force-reload}"** *>&2* RETVAL=1  **;; esac** exit $RETVAL |

# golang的mq消费者接口

## Dial(url string) 连接rabbitMQ服务器，获取socket

|  |
| --- |
| conn, err := amqp.Dial("amqp://guest:guest@192.168.0.222:5672/") |

## Channel()关联conn和channel

|  |
| --- |
| ch, err := conn.Channel() |

## ExchangeDeclare() 声明Exchange

|  |
| --- |
| 源文件定义：  type exchangeDeclare struct {  reserved1 uint16  Exchange string // exchange名称  Type string // type，"fanout" "direct" "topic"三选一  Passive bool  Durable bool // 是否持久化  AutoDelete bool // 没有consumer时，队列是否自动删除  Internal bool // 当前连接不在时，队列是否自动删除  NoWait bool // no-wait  Arguments Table // arguments  }  实际调用：  err = ch.ExchangeDeclare(  "logs\_direct", // name  "direct", // type  true, // durable  false, // auto-deleted  false, // internal  false, // no-wait  nil, // arguments  ) |

## QueueDeclare() 声明queue

|  |
| --- |
| 源文件定义：  type queueDeclare struct {  reserved1 uint16  Queue string // name  Passive bool  Durable bool // 队列是否持久化  Exclusive bool // 当前连接不在时，队列是否自动删除  AutoDelete bool // 没有consumer时，队列是否自动删除  NoWait bool // no-wait  Arguments Table // arguments  }  实际调用：  q, err := ch.QueueDeclare(  "", // name  false, // durable  false, // delete when usused  true, // exclusive 专用单独通道  false, // no-wait  nil, // arguments  ) |

## QueueBind() 绑定exchange和queue

|  |
| --- |
| 源文件定义：  type queueBind struct {  reserved1 uint16  Queue string // 队列名称  Exchange string // exchange名称  RoutingKey string // 路由key  NoWait bool // no-wait  Arguments Table  }  实际调用：  err = ch.QueueBind(  q.Name, // queue name  s, // routing key  "logs\_direct", // exchange  false,  nil) |

## Consume() 消费消息

|  |
| --- |
| 源文件定义：  type basicConsume struct {  reserved1 uint16  Queue string // 队列名称  ConsumerTag string  NoLocal bool // no local  NoAck bool // 是否需要确认消息后再从队列中删除消息  Exclusive bool // exclusive  NoWait bool // no wait  Arguments Table  }  实际调用：  msgs, err := ch.Consume(  q.Name, // queue  "", // consumer  true, // auto ack  false, // exclusive  false, // no local  false, // no wait  nil, // args  ) |

## consumer样例

|  |
| --- |
| */\* mq消费者例子 (1)新建一个mq实例，设置路由规则 (2)一直等待接收并消费消息 注：每个实例使用一个独立的sock \*/* **package** main  **import** (  **"fmt"   "github.com/streadway/amqp"** )  **const *MqServerAddr*** = **"amqp://guest:guest@192.168.0.233:5672/"** *// exchange属性，和消息生产者的exchange对应* **var** (  *// 注：如果多个路由的ExchangeName和QueueName都相同，RoutingKey无效，会轮流接收接收消息* Info = &Exchange{ExchangeName: **"logs\_direct"**, QueueName: **"queue\_info"**, RoutingKey: **"info"**, ExchangeType: **"direct"**, AutoAck: true}  Warning = &Exchange{ExchangeName: **"logs\_direct"**, QueueName: **"queue\_warning"**, RoutingKey: **"warning"**, ExchangeType: **"direct"**, AutoAck: false}  Erro = &Exchange{ExchangeName: **"logs\_direct"**, QueueName: **"queue\_error"**, RoutingKey: **"error"**, ExchangeType: **"direct"**, AutoAck: false} )  *// 消费消息接口* **type** MqConsumer **interface** {  MqConsume(hf HandleFunc) }  *// rabbitMQ最小管理单元exchange，每个exchange可以绑定多个队列，队列之间由RoutingKey区分* **type** Exchange **struct** {  ExchangeName string *// exchange名称* QueueName string *// 队列名称* RoutingKey string *// 路由key* ExchangeType string *// exchange类型，只有三种"fanout"、"direct"、"topic"* AutoAck bool *// true表示忽略，false表示手动ack* }  *// mq消费对象* **type** mqConsume **struct** {  addr string *// mq服务器地址* conn \*amqp.Connection *// 连接* ch \*amqp.Channel *// 通道* exchange \*Exchange *// mq路由规则* stopChan **chan** bool *// 退出订阅* }  *// 新建对象* **func** NewMqConsume(addr string, exchange \*Exchange) MqConsumer {  **return** &mqConsume{addr: addr, exchange: exchange} }  *// 连接mq服务器* **func** (mq \*mqConsume) connect() error {  conn, err := amqp.Dial(mq.addr)  **if** err != nil {  **return** err  }  mq.conn = conn  **return** nil }  *// 创建一个通道* **func** (mq \*mqConsume) createChannel() error {  ch, err := mq.conn.Channel()  **if** err != nil {  **return** err  }  mq.ch = ch  **return** nil }  *// 创建一个exchange，设置exchange相关属性* **func** (mq \*mqConsume) createExchange() error {  **return** mq.ch.ExchangeDeclare(  mq.exchange.ExchangeName, *// name* mq.exchange.ExchangeType, *// exchange类型"fanout"、"direct"、"topic"* true, *// durable* false, *// auto-deleted* false, *// internal* false, *// no-wait* nil, *// arguments* ) }  *// 创建一个queue，设置queue相关属性* **func** (mq \*mqConsume) createQueue() error {  \_, err := mq.ch.QueueDeclare(  mq.exchange.QueueName, *// name* true, *// durable 持久化* false, *// delete when usused* false, *// exclusive* false, *// no-wait* nil, *// arguments* )  **if** err != nil {  **return** err  }  **return** nil }  *// 绑定exchange和queue* **func** (mq \*mqConsume) bind() error {  **return** mq.ch.QueueBind(  mq.exchange.QueueName, *// queue name* mq.exchange.RoutingKey, *// routing key* mq.exchange.ExchangeName, *// exchange* false,  nil,  ) }  *// 订阅消息* **func** (mq \*mqConsume) subscribe(logic HandleFunc) error {  fmt.Printf(**"RoutingKey=[%s], ExchangeType=[%s], Waiting for receive message.\n"**, mq.exchange.RoutingKey, mq.exchange.ExchangeType)  msgs, err := mq.ch.Consume(  mq.exchange.QueueName, *// queue* **""**, *// consumer* mq.exchange.AutoAck, *// 手动还是自动回复，如果为手动回复，实际没有回复，该消息会mq服务器一直发送，直到有回复才会删除该消息* false, *// exclusive* false, *// no local* false, *// no wait* nil, *// args* )  **if** err != nil {  **return** err  }  **loop**:  **for** {  **select** {  **case** msg := <-msgs: *// 从通道获取消息* **if** !mq.exchange.AutoAck { *// 判断是否需要手动回复* **if** ok := logic(msg.Body); ok { *// 正确逻辑处理，发出通知* **if** err := mq.ch.Ack(msg.DeliveryTag, false); err != nil {  fmt.Printf(**"DeliveryTag: %s"**, err.Error())  } **else** {  fmt.Println(**"已经手动通知RabbitMQ删除消息："**, string(msg.Body))  }  }  } **else** { *// 忽略回复* logic(msg.Body)  }   **case** <-mq.stopChan:  **break loop** }  }   *// 关闭通道和连接* mq.ch.Close()  mq.conn.Close()  **return** nil }  *// 接收消息(通道阻塞)* **func** (mq \*mqConsume) MqConsume(hf HandleFunc) {  **var** err error  **defer func**() {  **if** err != nil {  fmt.Println(**"MqConsume:"**, err.Error())  }  }()   **if** err = mq.connect(); err != nil { *// 连接mq服务器* **return** }  **if** err = mq.createChannel(); err != nil { *// 创建一个通道* **return** }  **if** err = mq.createExchange(); err != nil { *// 创建一个exchange* **return** }  **if** err = mq.createQueue(); err != nil { *// 创建一个队列* **return** }  **if** err = mq.bind(); err != nil { *// 绑定exchange和queue* **return** }  **if** err = mq.subscribe(hf); err != nil { *// 订阅消息* **return** } }  **type** HandleFunc **func**(message []byte) bool  *// 逻辑处理函数* **func** HandleFuncLogic(message []byte) bool {  fmt.Printf(**"Handle Message = %s\n"**, string(message))  **return** true }  **func** main() {  exit := make(**chan** bool)  println(**" To exit press CTRL+C "**)   info := NewMqConsume(***MqServerAddr***, Info)  **go** info.MqConsume(HandleFuncLogic)   warning := NewMqConsume(***MqServerAddr***, Warning)  **go** warning.MqConsume(HandleFuncLogic)   erro := NewMqConsume(***MqServerAddr***, Erro)  **go** erro.MqConsume(HandleFuncLogic)   <-exit } |

# golang的mq生产者接口

## Dial(url string) 连接rabbitMQ服务器，获取socket

|  |
| --- |
| conn, err := amqp.Dial("amqp://guest:guest@192.168.0.222:5672/") |

## Channel()关联conn和channel

|  |
| --- |
| ch, err := conn.Channel() |

## ExchangeDeclare() 声明Exchange

|  |
| --- |
| 源文件定义：  type exchangeDeclare struct {  reserved1 uint16  Exchange string // exchange名称  Type string // type，"fanout" "direct" "topic"三选一  Passive bool  Durable bool // 是否持久化  AutoDelete bool // 没有consumer时，队列是否自动删除  Internal bool // 当前连接不在时，队列是否自动删除  NoWait bool // no-wait  Arguments Table // arguments  }  实际调用：  err = ch.ExchangeDeclare(  "logs\_direct", // name  "direct", // type  true, // durable  false, // auto-deleted  false, // internal  false, // no-wait  nil, // arguments  ) |

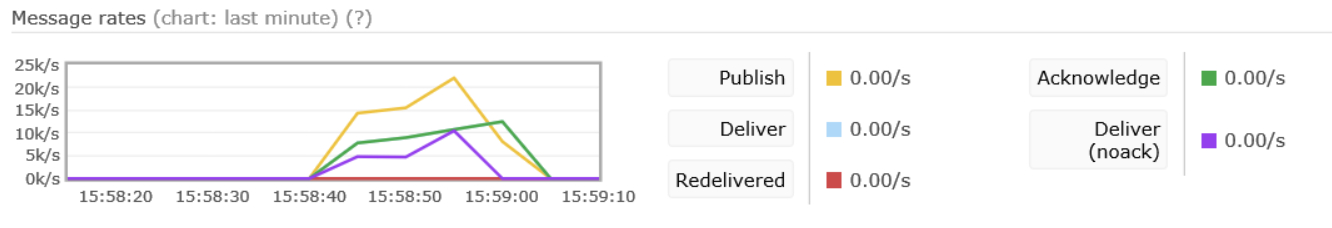
## producer样例

|  |
| --- |
| */\* mq生产者例子 (1)新建一个mq实例，设置路由规则 (2)发送消息 消息生产者不需要声明队列 每个实例独立一个socket比多个实例共用一个socket要快，前提是socket够用情况下 \*/* **package** main  **import** (  **"fmt"  "time"   "github.com/streadway/amqp"** )  **const *MqServerAddr*** = **"amqp://guest:guest@192.168.0.233:5672/"** *// exchange属性，对应消费消息客户端的exchange属性，但不需要指明队列名称* **var** (  Info = &MqExchange{ExchangeName: **"logs\_direct"**, RoutingKey: **"info"**, ExchangeType: **"direct"**}  Warning = &MqExchange{ExchangeName: **"logs\_direct"**, RoutingKey: **"warning"**, ExchangeType: **"direct"**}  Errors = &MqExchange{ExchangeName: **"logs\_direct"**, RoutingKey: **"error"**, ExchangeType: **"direct"**} )  *// 生产者接口* **type** MqProducer **interface** {  MqPublish(message []byte) error  Close() }  *// rabbitMQ最小管理单元exchange* **type** MqExchange **struct** {  ExchangeName string *// exchange名称* RoutingKey string *// 路由key* ExchangeType string *// exchange类型，只有三种"fanout"、"direct"、"topic"* }  *// mq生产对象* **type** mqProduce **struct** {  addr string *// mq服务器地址* conn \*amqp.Connection *// 连接* ch \*amqp.Channel *// 通道* exchange \*MqExchange *// exchange* }  *// 新建对象* **func** NewMqProduce(addr string, exchange \*MqExchange) (MqProducer, error) {  mq := &mqProduce{addr: addr, exchange: exchange}   **var** err error  **if** err = mq.connect(); err != nil { *// 连接mq服务器* **return** nil, err  }  **if** err = mq.createChannel(); err != nil { *// 创建一个通道* **return** nil, err  }  **if** err = mq.createExchange(); err != nil { *// 创建一个exchange* **return** nil, err  }  **return** mq, nil }  *// 连接mq服务器* **func** (mq \*mqProduce) connect() error {  conn, err := amqp.Dial(mq.addr)  **if** err != nil {  **return** err  }  mq.conn = conn  **return** nil }  *// 创建一个通道* **func** (mq \*mqProduce) createChannel() error {  ch, err := mq.conn.Channel()  **if** err != nil {  **return** err  }  mq.ch = ch  **return** nil }  *// 创建一个exchange，设置相关exchange属性* **func** (mq \*mqProduce) createExchange() error {  **return** mq.ch.ExchangeDeclare(  mq.exchange.ExchangeName, *// name* mq.exchange.ExchangeType, *// exchange类型"fanout"、"direct"、"topic"* true, *// durable* false, *// auto-deleted* false, *// internal* false, *// no-wait* nil, *// arguments* ) }  *// 发布消息* **func** (mq \*mqProduce) MqPublish(message []byte) error {  err := mq.ch.Publish( *// 根据路由规则发送消息到相应的队列* mq.exchange.ExchangeName, *// exchange name* mq.exchange.RoutingKey, *// routing key* false, *// mandatory* false, *// immediate* amqp.Publishing{  ContentType: **"text/plain"**,  Body: message,  })  **if** err != nil {  **return** err  }   *// fmt.Printf("Sent: %s\n\n", string(message))* **return** nil }  *// 释放资源* **func** (mq \*mqProduce) Close() {  mq.ch.Close()  mq.conn.Close() }  **func** main() {  *// 新建生产者* info, err := NewMqProduce(***MqServerAddr***, Info)  **if** err != nil {  fmt.Print(err)  **return** }  **defer** info.Close()   warning, err := NewMqProduce(***MqServerAddr***, Warning)  **if** err != nil {  fmt.Print(err)  **return** }  **defer** warning.Close()   erro, err := NewMqProduce(***MqServerAddr***, Errors)  **if** err != nil {  fmt.Print(err)  **return** }  **defer** erro.Close()   *// 待发送的消息* infoMessage := []byte(**"---info---"**)  waringMessage := []byte(**"---warning---"**)  errorMessage := []byte(**"---error---"**)  count := 100000  t := time.Now()  *// 发送消息* **for** i := 0; i < count; i++ {  info.MqPublish(infoMessage)  warning.MqPublish(waringMessage)  erro.MqPublish(errorMessage)  }  fmt.Printf(**"send %d messages spend time: %fs\n"**, count\*3, time.Now().Sub(t).Seconds()) } |

# 测试结果

打开网页<http://192.168.0.233:15672>，启动程序

在虚拟机测试结果如下图，每条消息约10字节，publish峰值达到20000，手动Ack峰值达到12000



其他实例参考http://www.rabbitmq.com/tutorials/tutorial-one-go.html