

## 一、MQ介绍

1、什么是MQ? 为什么要用MQ?

2、MQ的优缺点

3、几大MQ产品特点比较

## 二、Rabbitmq安装

实验环境

版本选择

安装Erlang语言包

安装RabbitMQ:

## 三、RabbitMQ集群搭建

搭建普通集群

搭建镜像集群

RabbitMQ基础使用

# 一、MQ介绍

这一部分理论介绍，其实在每个MQ产品中都是大致相同的，可以参见RocketMQ的部分。

## 1、什么是MQ? 为什么要用MQ?

MQ: MessageQueue, 消息队列。队列，是一种FIFO 先进先出的数据结构。消息由生产者发送到MQ进行排队，然后按原来的顺序交由消息的消费者进行处理。

QQ和微信就是典型的MQ。

MQ的作用主要有以下三个方面:

- 异步

例子：快递员发快递，直接到客户家效率会很低。引入菜鸟驿站后，快递员只需要把快递放到菜鸟驿站，就可以继续发其他快递去了。客户再按自己的时间安排去菜鸟驿站取快递。

作用：异步能提高系统的响应速度、吞吐量。

- 解耦

例子：《Thinking in JAVA》很经典，但是都是英文，我们看不懂，所以需要编辑社，将文章翻译成其他语言，这样就可以完成英语与其他语言的交流。

作用：

1、服务之间进行解耦，才可以减少服务之间的影响。提高系统整体的稳定性以及可扩展性。

2、另外，解耦后可以实现数据分发。生产者发送一个消息后，可以由一个或者多个消费者进行消费，并且消费者的增加或者减少对生产者没有影响。

- 削峰

例子：长江每年都会涨水，但是下游出水口的速度是基本稳定的，所以会涨水。

引入三峡大坝后，可以把水储存起来，下游慢慢排水。

作用：以稳定的系统资源应对突发的流量冲击。

## 2、MQ的优缺点

上面MQ的所用也就是使用MQ的优点。但是引入MQ也是有他的缺点的：

- 系统可用性降低

系统引入的外部依赖增多，系统的稳定性就会变差。一旦MQ宕机，对业务会产生影响。这就需要考虑如何保证MQ的高可用。

- 系统复杂度提高

引入MQ后系统的复杂度会大大提高。以前服务之间可以进行同步的服务调用，引入MQ后，会变为异步调用，数据的链路就会变得更复杂。并且还会带来其他一些问题。比如：如何保证消费不会丢失？不会被重复调用？怎么保证消息的顺序性等问题。

- 消息一致性问题

A系统处理完业务，通过MQ发送消息给B、C系统进行后续的业务处理。如果B系统处理成功，C系统处理失败怎么办？这就需要考虑如何保证消息数据处理的一致性。

## 3、几大MQ产品特点比较

常用的MQ产品包括Kafka、RabbitMQ和RocketMQ。我们对这三个产品做下简单的比较，重点需要理解他们的适用场景。

	优点	缺点	使用场景
kafka	吞吐量非常大，性能非常好，集群高可用。	会丢数据，功能比较单一。	日志分析，大数据采集
Rabbit MQ	消息可靠性高，功能全面。	吞吐量比较低，消息积累会影响性能，erlang语言不好定制。	小规模场景
Rocket MQ	高吞吐，高性能，高可用，功能全面。	开源版功能不如云上版，官方文档比较简单，客户端只支持java。	几乎全场景

另外，关于这三大产品更详细的比较，可以参见《kafka vs rabbitmq vs rocketmq.pdf》

关于RabbitMQ的功能特性，可以在官网(<https://www.rabbitmq.com/>)上看到，包含 Asynchronous Message(异步消息)、Developer Experience(开发体验)、Distributed Deployment(分布式部署)、Enterprise & Cloud Ready(企业云部署)、Tools & Plugins(工具和插件)、Management & Monitoring(管理和监控)六大部分。所以其中的功能是相当丰富的，而我们肯定只能关注重点的部分内容，所以还是要经常到官网上去看看的。

## 二、Rabbitmq安装

### 实验环境

准备了三台虚拟机 192.168.232.128~130，预备搭建三台机器的集群。

三台机器均预装CentOS7 操作系统。分别配置机器名 worker1, worker2, worker3。然后需要关闭防火墙(或者找到RabbitMQ的业务端口全部打开。5672(amqp端口); 15672(http Api端口); 25672(集群通信端口))。

### 版本选择

RabbitMQ版本，通常与他的大的功能是有关系的。3.8.x版本主要是围绕Quorum Queue功能，而3.9.x版本主要是围绕Streams功能。目前还有3.10.x版本，还在rc阶段。我们这次选择3.9.15版本。

RabbitMQ是基于Erlang语言开发，所以安装前需要安装Erlang语言环境。需要注意下的是RabbitMQ与ErLang是有版本对应关系的。3.9.15版本的RabbitMQ只支持23.2以上到24.3版本的Erlang。

| Docker hub上也已经有官方上传的镜像

## 安装Erlang语言包

这个语言包，在windows下的安装比较简单，是一个可执行程序，直接图形化安装就行了。

Linux上的安装稍微复杂，需要有非常多的依赖包。简单起见，可以下载rabbitmq提供的zero dependency版本。下载地址 <https://github.com/rabbitmq/erlang-rpm/releases>

下载完成后，可以尝试使用下面的指令安装

```
1 [root@worker1 tools]# rpm -ivh erlang-23.2.7-1.el7.x86_64.rpm
2 警告: erlang-23.2.7-1.el7.x86_64.rpm: 头V4 RSA/SHA1 Signature, 密钥 ID
   6026dfca: NOKEY
3 准备中...                               ##### [100%]
4 正在升级/安装...
5    1:erlang-23.2.7-1.el7                 #####
   [100%]
```

这样Erlang语言包就安装完成了。安装完后可以使用 `erl -version` 指令检测下erlang是否安装成功。

```
1 [root@worker1 tools]# erl -version
2 Erlang (SMP,ASYNC_THREADS,HIPE) (BEAM) emulator version 11.1.8
```

## 安装RabbitMQ:

RabbitMQ的安装方式有很多，我们采用RPM安装包的方式。安装包可以到github仓库中下载发布包。下载地址: <https://github.com/rabbitmq/rabbitmq-server/releases>

然后使用 `rpm -Uvh` 指令安装RabbitMQ的rpm包时，会报错，需要安装一个socat。

而这个socat我也在网上下载到了rpm安装包。 socat-1.7.3.2-1.1.el7.x86\_64.rpm , 但是安装时, 却提示需要tcp\_wrappers依赖。

```
1 [root@worker2 tools]# rpm -ivh socat-1.7.3.2-1.1.el7.x86_64.rpm
2 警告: socat-1.7.3.2-1.1.el7.x86_64.rpm: 头V4 RSA/SHA1 Signature, 密钥 ID
   87e360b8: NOKEY
3 错误: 依赖检测失败:
4      tcp_wrappers 被 socat-1.7.3.2-1.1.el7.x86_64 需要
```

这时, 当然可以按他的提示去安装依赖包。但是我却没有这么做了。直接用yum安装这个socat依赖。在使用yum时, 可以做一个小配置, 将yum源配置成阿里的yum源, 这样速度会比较快。

```
1 mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-
   Base.repo.backup
2
3 curl -o /etc/yum.repos.d/CentOS-Base.repo
   http://mirrors.aliyun.com/repo/Centos-7.repo
4
5 yum makecache
6
7 然后安装socat
8 yum install socat
```

socat安装完成后, 就可以安装RabbitMQ了。

```
1 [root@worker1 tools]# rpm -Uvh rabbitmq-server-3.9.15-1.el7.noarch.rpm
2 警告: rabbitmq-server-3.9.15-1.el7.noarch.rpm: 头V4 RSA/SHA512 Signature, 密钥
   ID 6026dfca: NOKEY
3 准备中... ##### [100%]
4 正在升级/安装...
5 1:rabbitmq-server-3.9.15-1.el7 #####
   [100%]
```

安装完成后, 可以查看下他的安装情况

```
1 [root@worker1 share]# whereis rabbitmqctl
2 rabbitmqctl: /usr/sbin/rabbitmqctl /usr/share/man/man8/rabbitmqctl.8.gz
3 ##启动RabbitMQ服务
4 [root@worker1 rabbitmq]# service rabbitmq-server start
5 Redirecting to /bin/systemctl start rabbitmq-server.service
6 Job for rabbitmq-server.service failed because the control process exited
   with error code. See "systemctl status rabbitmq-server.service" and
   "journalctl -xe" for details.
```

```

7  ##查看服务状态
8  [root@worker1 rabbitmq]# service rabbitmq-server status
9  Redirecting to /bin/systemctl status rabbitmq-server.service
10 ● rabbitmq-server.service - RabbitMQ broker
11     Loaded: loaded (/usr/lib/systemd/system/rabbitmq-server.service;
disabled; vendor preset: disabled)
12     Active: activating (start) since 五 2022-04-22 17:16:47 CST; 5s ago
13     Main PID: 4327 (beam.smp)
14     Status: "Startup in progress"
15     CGroup: /system.slice/rabbitmq-server.service
16             └─4327 /usr/lib64/erlang/erts-11.1.8/bin/beam.smp -W w -MBas
ageffcbf -MHas ageffcbf -MBlmbcs 512 -MHlmbcs 512 -MMmcs 30 -P 1048576 -t
5000000 -stbt db -zdbbl 128000 -sbwt ...
17             └─4342 erl_child_setup 32768
18             └─4390 inet_gethost 4
19             └─4391 inet_gethost 4
20
21  4月 22 17:16:49 worker1 rabbitmq-server[4327]: 2022-04-22
17:16:49.126900+08:00 [info] <0.229.0> Feature flags: list of feature flags
found:
22  4月 22 17:16:49 worker1 rabbitmq-server[4327]: 2022-04-22
17:16:49.144773+08:00 [info] <0.229.0> Feature flags:    [x]
implicit_default_bindings
23  4月 22 17:16:49 worker1 rabbitmq-server[4327]: 2022-04-22
17:16:49.144856+08:00 [info] <0.229.0> Feature flags:    [x]
maintenance_mode_status
24  4月 22 17:16:49 worker1 rabbitmq-server[4327]: 2022-04-22
17:16:49.144903+08:00 [info] <0.229.0> Feature flags:    [x] quorum_queue
25  4月 22 17:16:49 worker1 rabbitmq-server[4327]: 2022-04-22
17:16:49.145025+08:00 [info] <0.229.0> Feature flags:    [ ] stream_queue
26  4月 22 17:16:49 worker1 rabbitmq-server[4327]: 2022-04-22
17:16:49.145075+08:00 [info] <0.229.0> Feature flags:    [ ] user_limits
27  4月 22 17:16:49 worker1 rabbitmq-server[4327]: 2022-04-22
17:16:49.145110+08:00 [info] <0.229.0> Feature flags:    [x]
virtual_host_metadata
28  4月 22 17:16:49 worker1 rabbitmq-server[4327]: 2022-04-22
17:16:49.145154+08:00 [info] <0.229.0> Feature flags: feature flag states
written to disk: yes
29  4月 22 17:16:49 worker1 rabbitmq-server[4327]: 2022-04-22
17:16:49.605843+08:00 [noti] <0.44.0> Application syslog exited with
reason: stopped
30  4月 22 17:16:49 worker1 rabbitmq-server[4327]: 2022-04-22
17:16:49.605930+08:00 [noti] <0.229.0> Logging: switching to configured
handler(s); following messages may not be... log output
31  Hint: Some lines were ellipsized, use -l to show in full.

```

其他常用的启停操作：

```
rabbitmq-server -deched --后台启动服务
```

```
rabbitmqctl start_app --启动服务
```

```
rabbitmqctl stop_app --关闭服务
```

这样RabbitMQ服务就启动完成了。之后可以配置下打开他的Web管理页面：

```
1 [root@worker1 rabbitmq]# rabbitmq-plugins enable rabbitmq_management
2 Enabling plugins on node rabbit@worker1:
3 rabbitmq_management
4 The following plugins have been configured:
5   rabbitmq_management
6   rabbitmq_management_agent
7   rabbitmq_web_dispatch
8 Applying plugin configuration to rabbit@worker1...
9 The following plugins have been enabled:
10  rabbitmq_management
11  rabbitmq_management_agent
12  rabbitmq_web_dispatch
13
14 set 3 plugins.
15 Offline change; changes will take effect at broker restart.
```

可以看到，这时需要重启RabbitMQ服务才能生效。重启后，就可以访问Web控制台了。访问端口192.168.232.129: 15672。

这时，可以使用默认的guest/guest用户登录。但是注意下，默认情况下，只允许在localhost本地登录，远程访问是无法登录的。这时，可以创建一个管理员账户来登录。

```
1 [root@worker1 ~]# rabbitmqctl add_user admin admin
2 Adding user "admin" ...
3 Done. Don't forget to grant the user permissions to some virtual hosts! See
   'rabbitmqctl help set_permissions' to learn more.
4 [root@worker2 tools]# rabbitmqctl set_permissions -p / admin "." "." ".*"
5 Setting permissions for user "admin" in vhost "/" ...
6 [root@worker2 tools]# rabbitmqctl set_user_tags admin administrator
7 Setting tags for user "admin" to [administrator] ...
```

这样就可以用admin/admin用户登录Web控制台了。

## 三、RabbitMQ集群搭建

在RabbitMQ中，一个节点的服务其实也是作为一个集群来处理的，在web控制台的admin-> cluster 中可以看到集群的名字，并且可以在页面上修改。而多节点的集群有两种方式

- **默认的普通集群模式：**

这种模式使用Erlang语言天生具备的集群方式搭建。这种集群模式下，集群的各个节点之间只会有相同的元数据，即队列结构，而消息不会进行冗余，只存在一个节点中。消费时，如果消费的不是存有数据的节点，RabbitMQ会临时在节点之间进行数据传输，将消息从存有数据的节点传输到消费的节点。

很显然，这种集群模式的消息可靠性不是很高。因为如果其中有个节点服务宕机了，那这个节点上的数据就无法消费了，需要等到这个节点服务恢复后才能消费，而这时，消费者端已经消费过的消息就有可能给不了服务端正确应答，服务起来后，就会再次消费这些消息，造成这部分消息重复消费。另外，如果消息没有做持久化，重启就消息就会丢失。

并且，这种集群模式也不支持高可用，即当某一个节点服务挂了后，需要手动重启服务，才能保证这一部分消息能正常消费。

所以这种集群模式只适合一些对消息安全性不是很高的场景。而在使用这种模式时，消费者应该尽量的连接上每一个节点，减少消息在集群中的传输。

- **镜像模式：**

这种模式是在普通集群模式基础上的一种增强方案，这也就是RabbitMQ的官方HA高可用方案。需要在搭建了普通集群之后再补充搭建。其本质区别在于，这种模式会在镜像节点中间主动进行消息同步，而不是在客户端拉取消息时临时同步。

并且在集群内部有一个算法会选举产生master和slave，当一个master挂了后，也会自动选出一个来。从而给整个集群提供高可用能力。

这种模式的消息可靠性更高，因为每个节点上都存着全量的消息。而他的弊端也是明显的，集群内部的网络带宽会被这种同步通讯大量的消耗，进而降低整个集群的性能。这种模式下，队列数量最好不要过多。

## 搭建普通集群

1：需要同步集群节点中的cookie。

默认会在 /var/lib/rabbitmq/目录下生成一个.erlang.cookie。 里面有一个字符串。我们要做的就是保证集群中三个节点的这个cookie字符串一致。



我们实验中将worker1和worker3加入到worker2的RabbitMQ集群中，所以将worker2的.erlang.cookie文件分发到worker1和worker3。

2：将worker1的服务加入到worker2的集群中。

首先需要保证worker1上的rabbitmq服务是正常启动的。然后执行以下指令：

```
1 [root@worker1 rabbitmq]# rabbitmqctl stop_app
2 Stopping rabbit application on node rabbit@worker1 ...
3 [root@worker1 rabbitmq]# rabbitmqctl join_cluster --ram rabbit@worker2
4 Clustering node rabbit@worker1 with rabbit@worker2
5 [root@worker1 rabbitmq]# rabbitmqctl start_app
6 Starting node rabbit@worker1 ...
```

--ram 表示以Ram节点加入集群。RabbitMQ的集群节点分为disk和ram。disk节点会将元数据保存到硬盘当中，而ram节点只是在内存中保存元数据。

1、由于ram节点减少了很多与硬盘的交互，所以，ram节点的元数据使用性能会比较高。但是，同时，这也意味着元数据的安全性是不如disk节点的。在我们这个集群中，worker1和worker3都以ram节点的身份加入到worker2集群里，因此，是存在单点故障的。如果worker2节点服务崩溃，那么元数据就有可能丢失。在企业进行部署时，性能与安全性需要自己进行平衡。

2、这里说的元数据仅仅只包含交换机、队列等的定义，而不包含具体的消息。因此，ram节点的性能提升，仅仅体现在对元数据进行管理时，比如修改队列queue，交换机exchange，虚拟机vhosts等时，与消息的生产和消费速度无关。

3、如果一个集群中，全部都是ram节点，那么元数据就有可能丢失。这会造成集群停止之后就启动不起来了。RabbitMQ会尽量阻止创建一个全是ram节点的集群，但是并不能彻底阻止。所以，综合考虑，官方其实并不建议使用ram节点，更推荐保证集群中节点的资源投入，使用disk节点。

然后同样把worker3上的rabbitmq加入到worker2的集群中。

加入完成后，可以在worker2的Web管理界面上看到集群的节点情况：

▼ Nodes									
Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats	
rabbit@worker1	98 32768 available	0 29401 available	444 1048576 available	82 MiB 728 MiB high watermark	10 GiB 48 MiB low watermark	9m 4s	basic RAM 1 rss	This node	All nodes
rabbit@worker2	36 32768 available	0 29401 available	446 1048576 available	83 MiB 728 MiB high watermark	11 GiB 48 MiB low watermark	32m 24s	basic disc 1 rss	This node	All nodes
rabbit@worker3	99 32768 available	0 29401 available	445 1048576 available	81 MiB 728 MiB high watermark	11 GiB 48 MiB low watermark	9m 22s	basic disc 1 rss	This node	All nodes

也可以用后台指令查看集群状态 `rabbitmqctl cluster_status`

## 搭建镜像集群

这样就完成了普通集群的搭建。再此基础上，可以继续搭建**镜像集群**。

通常在生产环境中，为了减少RabbitMQ集群之间的数据传输，在配置镜像策略时，会针对固定的虚拟主机virtual host来配置。

RabbitMQ中的virtual host可以类比为MySQL中的库，针对每个虚拟主机，可以配置不同的权限、策略等。并且不同虚拟主机之间的数据是相互隔离的。

我们首先创建一个/mirror的虚拟主机，然后再添加给对应的镜像策略：

```
1 [root@worker2 rabbitmq]# rabbitmqctl add_vhost /mirror
2 Adding vhost "/mirror" ...
3 [root@worker2 rabbitmq]# rabbitmqctl set_policy ha-all --vhost "/mirror" "^"
  '{"ha-mode":"all"}'
4 Setting policy "ha-all" for pattern "^" to '{"ha-mode":"all"}' with priority
  "0" for vhost "/mirror" ...
```

同样，这些配置的策略也可以在Web控制台操作。另外也提供了HTTP API来进行这些操作。

## Policies

▼ User policies

Filter:  ☐ Regex ?

1 item, page size up to

Virtual Host	Name	Pattern	Apply to	Definition	Priority
/mirror	ha-all	^	all	ha-mode: all	0

▼ Add / update a policy

Virtual host:

Name:

Pattern:

Apply to:

Priority:

Definition:  =

Queues [All types]

Queues [Classic]

Queues [Quorum]

Exchanges

Federation

Max length | Max length bytes | Overflow behaviour ?

Dead letter exchange | Dead letter routing key

HA mode ? | HA params ? | HA sync mode ?

HA mirror promotion on shutdown ? | HA mirror promotion on failure ?

Message TTL | Auto expire | Lazy mode | Master Locator

Max in memory length ? | Max in memory bytes ? | Delivery limit ?

Alternate exchange ?

Federation upstream set ? | Federation upstream ?

Users

Virtual Hosts

Feature Flags

Policies

Limits

Cluster

这些参数需要大致了解下。其中，pattern是队列的匹配规则，^表示全部匹配。^ha\ 这样的配置表示以ha开头。通常就用虚拟主机来区分就够了，这个队列匹配规则就配置成全匹配。

然后几个关键的参数：

HA mode: 可选值 all , exactly, nodes。生产上通常为了保证高可用，就配all

- all : 队列镜像到集群中的所有节点。当新节点加入集群时，队列也会被镜像到这个节点。
- exactly : 需要搭配一个数字类型的参数(ha-params)。队列镜像到集群中指定数量的节点。如果集群内节点数少于这个数字，则队列镜像到集群内的所有节点。如果集群内节点少于这个数，当一个包含镜像的节点停止服务后，新的镜像就不会去另外找节点进行镜像备份了。
- nodes: 需要搭配一个字符串类型的参数。将队列镜像到指定的节点上。如果指定的队列不在集群中，不会报错。当声明队列时，如果指定的所有镜像节点都不在线，那队列会被创建在发起声明的客户端节点上。

还有其他很多参数，可以后面慢慢再了解。

通常镜像模式的集群已经足够满足大部分的生产场景了。虽然他对系统资源消耗比较高，但是在生产环境中，系统的资源都是会做预留的，所以正常的使用是没有问题的。但是在做业务集成时，还是需要注意队列数量不宜过多，并且尽量不要让RabbitMQ产生大量的消息堆积。

这样搭建起来的RabbitMQ已经具备了集群特性，往任何一个节点上发送消息，消息都会及时同步到各个节点中。而在实际企业部署时，往往会以RabbitMQ的镜像队列作为基础，再增加一些运维手段，进一步提高集群的安全性和实用性。

例如，增加keepalived保证每个RabbitMQ的稳定性，当某一个节点上的RabbitMQ服务崩溃时，可以及时重新启动起来。另外，也可以增加HA-proxy来做前端的负载均衡，通过HA-proxy增加一个前端转发的虚拟节点，应用可以像使用一个单点服务一样使用一个RabbitMQ集群。这些运维方案我们就不做过多介绍了，有兴趣可以自己了解下。

## RabbitMQ基础使用

RabbitMQ搭建完成后，可以在Web控制台上选择Exchange或者Queue来发送消息了，我们可以简单体验下，也可以留到下一部分编程模型时再深入体验。

例如，先在Admin菜单，配置admin用户可以操作/mirror虚拟机。

OverviewConnectionsChannelsExchangesQueuesAdmin

User: admin

Overview

Tags

administrator

Can log in with password

Permissions

Current permissions

Virtual host	Configure regexp	Write regexp	Read regexp	
/	.*	.*	.*	Clear
/mirror	.*	.*	.*	Clear

Set permission

Virtual Host:

/mirror

Configure regexp:

.\*

Write regexp:

.\*

Read regexp:

.\*

Set permission

然后，创建一个经典队列。

OverviewConnectionsChannelsExchangesQueuesAdmin

Queues

▼ All queues (0)

Pagination

Page ▼ of 0 - Filter:  ☐ Regex ?

... no queues ...

▼ Add a new queue

Virtual host: /mirror ▼

Type: Classic ▼

Name: test1 \*

Durability: Transient ▼

Node: rabbit@worker2 ▼

Auto delete: ? No ▼

Arguments:  =  String ▼

Add Message TTL ? | Auto expire ? | Overflow behaviour ? | Single active consumer ? | Dead letter exchange ? | Dead letter routing key ? | Max length ? | Max length bytes ? | Maximum priority ? | Lazy mode ? | Master locator ?

Add queue

创建完成后，选择这个test1队列，就可以在页面上直接发送消息以及消费消息了。

OverviewConnectionsChannelsExchangesQueuesAdmin

Queue test1 in virtual host /mirror

► Overview

► Consumers (0)

► Bindings (1)

► Publish message

► Get messages

► Move messages

► Delete

► Purge

在整体使用过程中你会发现，对于队列，有Classic、Quorum、Stream三种类型，其中，Classic和Quorum两种类型，使用上几乎是没有什么区别的。但是Stream队列就无法直接消费消息了。这种区别也会带到后面的使用过程中。

然后，RabbitMQ的各种管理功能，整理上还是非常简单的，几乎所有的配置都可以在Web管理页面上直观的看到，并直接完成操作。同时，这些管理功能，也都可以通过后端的命令行工具进行。在进行体验的过程中，也可以自行尝试了解后端配置的各种指令。每个指令都有help帮助文档，大家可以自行尝试了解。

有道云笔记链接

文档：RabbitMQ1集群搭建及快速上手.md

链接：<http://note.youdao.com/noteshare?id=f2a7001c83d43549dd4bfc76c3b9fa4c&sub=98A54B367DC6456A915C21031A5DD281>