资源编排服务

模板参考

文档版本 01

发布日期 2025-02-10

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 模板简介

RFS服务主要包含模板和资源栈两部分，

其中资源栈是指用户通过RFS服务创建的华为云资源的集合，而模板是用来创建、更新资源栈的脚本。

2 语法指南

2.1 基本语法

RFS配置语言兼容HCL语法，具有配置简单，可读性强等特点，并且兼容JSON语法。本文主要介绍HCL语言的基本语法及常见函数。

RFS配置语言主要由参数(Argument)，块(Block)，表达式(Experssion)和函数(Functions)组成。

参数 (Argument)

使用等号将一个值或表达式赋值给指定的参数名称，参数名称可以使用字母、数字、下划线(_)和连接符(-)表示，且首字母不能是数字，例如：

```
image_id = "ad091b52-742f-469e-8f3c-fd81cadf0743"
```

块 (Block)

块将多个参数聚合在一起，并支持嵌套。块由块类型、块标签和块主体构成，格式如下：

```
resource "huaweicloud_compute_instance" "myinstance" {
  name   = "myinstance"
   ......
   network {
      uuid = "55534eaa-533a-419d-9b40-ec427ea7195a"
   }
}
```

在使用块时必须先声明其对应的类型，样例中resource和network 均为块类型，其中resource为顶层块类型，network为嵌套块类型。HCL语言支持的顶层块类型包括：provider，resource，data，variable，output，module，locals等关键字。

块标签在块类型之后定义，且数量由块类型决定，样例中resource块类型包含两个标签：huaweicloud_compute_instance和myinstance，嵌套的network类型没有块标签。块主体定义在块最后，由 { 和 } 字符进行封装，在块主体内可以嵌套其他类型以实现不同的层级结构。

参数类型

HCL支持以下参数类型：

基本类型

- **string**：字符串类型，由一个或多个Unicode字符组成，例如 "hello"。
- **number**：数字类型，可以表示整数和浮点数。
- **bool**：布尔类型，只能是 true 或 false。

HCL能够根据参数类型自动将number和bool类型转换为string类型。如果一个字符串能够表示为一个数字或布尔类型的值，也可以进行反向转换。字符串、数字和布尔类型的参数可以直接赋值，例如：

```
disk_type = "SSD"
disk_size = 40
enable    = true

# 支持使用字符串表示数字和布尔类型
disk_size = "40"
enable    = "true"
```

集合类型

- **map(...)**：映射类型，以键值对(key-value pair) 的方式组合起来的数据元素集合，其中key为string类型，对应的值可以是string，number，bool等类型，且所有元素的值必须是同一类型。
- **list(...)**：列表类型，具有同类型的数据元素集合，元素可以是基本类型和块类型，列表索引从0开始。
- **set(...)**：集合类型，类似于列表类型，但是集合中的元素是没有任何辅助标识符或顺序，且元素具有唯一性。

映射类型使用 {} 封装，其表示形式非常灵活：键值对可以使用等号"="或冒号":"连接；如果key不以数字开头，可以不加双引号；对于多行映射，键值对之间可以使用换行符或者逗号进行分隔。推荐使用等号连接键值对并用换行符进行分隔，例如：

```
# 推荐格式
tags = {
  foo = "bar"
  key = "value"
}

# 其他格式
tags = {"foo" = "bar", "key" = "value"}
tags = {"foo" : "bar", "key" : "value"}
tags = {foo = "bar", key = "value"}
tags = {foo : "bar", key : "value"}
tags = {
  foo : "bar"
  key : "value"
}
```

列表类型和集合类型的表示方式相同，其中元素为基本类型的列表/集合使用 [] 封装，元素为块类型的列表/集合使用重复块的形式表示，例如：

```
# 基本类型的列表
security_groups = ["default", "internal"]

# 块类型的列表
network {
  uuid = "55534eaa-533a-419d-9b40-ec427ea7195a"
}
network {
  uuid = "ad091b52-742f-469e-8f3c-fd81cadf0743"
}
```


特殊类型

- **null：**空类型，如果将一个参数设置为null，表示这个参数未填写，HCL会自动忽略该参数，并使用默认值。null在条件表达式中较为常见，如 var.test=="" ? null : var.test，表示当var.test的值为""时，就将其忽略。

其他语法

- 单行注释以#或//开头；
- 多行注释以/*开始，以*/结束，不支持嵌套块注释。
- Terraform配置文件使用UTF-8编码，对于标识符、注释和字符串都支持非ASCII字符。
- 多行字符串以<<EOF开头，中间是字符串内容，最后以EOF结尾。EOF也可以替换为其他字符。例如：

```
resource "huaweicloud_obs_bucket" "web_bucket" {
  ...
  website {
    ...
    routing_rules = <<EOF
[{
    "Condition": {
        "KeyPrefixEquals": "docs/"
    },
    "Redirect": {
        "ReplaceKeyPrefixWith": "documents/"
    }
}]
EOF
  }
}
```

2.2 样式约定

样式约定

HCL约定了一些惯用的风格样式，以确保不同团队编写的文件和模块的风格一致性。建议用户遵循这些约定，推荐的样式约定如下：

- 对于每个嵌套级别，缩进两个空格。
- 当多个单行的参数在同一嵌套级别连续出现时，建议将等号对齐。

```
name            = "myinstance"
security_groups = ["default", "internal"]
```

- 使用空行分隔块中的逻辑参数组。
- 当块主体同时包含参数和块时，建议将所有参数放在顶部，嵌套块放在参数的下方并使用空行隔开。
- 将元参数(meta-arguments) 放在块主体的顶部，并使用空行与其它参数隔开；将元参数块(meta-argument blocks) 放在块主体的最后，并用空行与其他块隔开。

```
resource "huaweicloud_obs_bucket" "demo" {
  count = 1

  bucket = "bucket_demo"
  acl    = "public-read"

  tags = {
    foo = "bar"
    env = "test"
```

```
  }

  lifecycle {
    create_before_destroy = true
  }
}
```

- 顶层块之间使用空行将彼此隔开。
- 建议将相同类型的嵌套块放在一起，不同类型的嵌套块使用空行隔开。

参考资料

https://www.terraform.io/docs/configuration/style.html

2.3 表达式

表达式用于引用或计算配置中的值，最简单的表达式是文字表达式，如 "hello world" 或5。Terraform支持多种表达式，包括运算符、条件表达式以及丰富的内置函数。

通过 "terraform console" 命令可以打开一个交互式的控制台，您可以使用该控制台进行表达式及内置函数的体验和测试。

运算符

运算符是执行特定的数学或逻辑操作的服务，Terraform支持以下类型的运算符：

- 算术运算符：操作数和结果都为数字类型，包括：+，-（减法），*，/，%，-（负数）。
- 关系运算符：操作数为任意类型，结果为布尔值，包括：==，！=。
- 比较运算符：操作数为数字类型，结果为布尔值，包括：>，>=，<，<=。
- 逻辑运算符：操作数和结果都为布尔类型，包括：||，&&，！。

在表达式中使用多个运算符时，将按照以下优先级进行求解：

1. !, - (负数)
2. *, /, %
3. +, - (减法)
4. >, >=, <, <=
5. ==, !=
6. &&
7. ||

条件表达式

条件表达式采用布尔表达式的值进行二选一，其语法可以表示为：

```
condition ? true_value : false_value
```

该语句表示：如果condition为true，结果为true_value，否则为false_value。条件表达式的结果可以是任意类型，但true_value和false_value的类型必须保持一致。条件表达式的常见用法是使用默认值替换无效值，如下：

```
var.a != "" ? var.a : "default-a"
```

该语句表示：如果var.a的值不为空，则返回var.a的值，否则返回一个默认值。

for 表达式

for表达式用于遍历集合类型 (map、list、set) 中的每个元素，并对元素进行处理，最后将结果输出为一个新的集合类型。for表达式的输出结果取决于所使用的括号类型：

- 使用 '[' 和 ']' 将生成一个列表
- 使用 '{' 和 '}' 将生成一个映射/对象

假设列表 mylist 的值为 ["AA", "BBB", "CCCC"]，您可以使用for表达式对 mylist 中的每个字符串元素转换为小写，并输出一个列表：

```
> [for str in var.mylist : lower(str)]
[
  "aa",
  "bbb",
  "cccc",
]
```

您也可以将结果输出为一个映射，映射关系通过 "=>" 确定：

```
> {for str in var.mylist : str => lower(str)}
{
  "AA" = "aa"
  "BBB" = "bbb"
  "CCCC" = "cccc"
}
```

映射类型也可以通过for表达式转换进行处理，假设 mymap 的值为 {element1="aaa", element2="bbb", element3="ccc"}，您可以将映射中的每个键值转换为大写：

```
> {for key, value in var.mymap : key => upper(value)}
{
  "element1 = "AAA"
  "element2 = "BBB"
  "element3 = "CCC"
}
```

此外，for表达式还可以使用if语句对元素进行过滤：

```
> [for str in var.list : upper(str) if length(str) >= 3]
[
  "bbb",
  "cccc",
]
```

参考资料

https://www.terraform.io/docs/configuration/expressions.html

2.4 常见函数

HCL支持丰富的内置函数，用于处理字符串、数值计算、加密，类型转换等操作，您可以通过函数名称进行调用，其语法如下：

```
<函数名称>(<参数1>, <参数2> ...)
```

本文主要对HCL中常见的函数进行总结并通过样例说明其用法。您可以通过Terraform **官方文档**查看完整的函数支持列表。

字符串函数

表 2-1 字符串函数

函数名称	函数描述	样例	运行结果
format	字符串格式化	format("Hello, %s!", "cloud")	Hello, cloud!
lower	将字符串中的字母转换为小写	lower("HELLO")	hello
upper	将字符串中的字母转换为大写	upper("hello")	HELLO
join	使用自定义字符将列表拼接成字符串	join(", ", ["One", "Two", "Three"])	One, Two, Three
split	根据分隔符拆分字符串	split(", ", "One, Two, Three")	["One", "Two", "Three"]
substr	通过偏移量和长度从给定的字符串中提取一个子串	substr("hello world!", 1, 4)	ello
replace	把字符串中的str1替换成str2	replace("hello, cloud!", "h", "H")	Hello, cloud!

数值计算函数

表 2-2 数值计算函数

函数名称	函数描述	样例	运行结果
abs	计算绝对值	abs(-12.4)	12.4
max	计算最大值	max(12, 54, 6) max([12, 54, 6]...)	54 54
min	计算最小值	min(12, 54, 6) min([12, 54, 6]...)	6 6
log	计算对数	log(16, 2)	4
power	计算x的y次幂	power(3, 2)	9

集合函数

表 2-3 集合函数

函数名称	函数描述	样例	运行结果
element	通过下标从列表中检索对应元素值	element(["One", "Two", "Three"], 1)	Two
index	返回给定值在列表中的索引，如果该值不存在将报错。	index(["a", "b", "c"], "b")	1
lookup	使用给定的键从映射表中检索对应的值。如果给定的键不存在，则返回默认值。	lookup({IT="A", CT="B"}, "IT", "G") lookup({IT="A", CT="B"}, "IE", "G")	A G
flatten	展开列表中的嵌套元素	flatten([["a", "b"], [], ["c"]])	["a", "b", "c"]
keys	返回map中的所有key	keys({a=1, b=2, c=3})	["a", "b", "c"]
length	获取列表、映射或是字符串的长度	length(["One", "Two", "Three"]) length({IT="A", CT="B"}) length("Hello, cloud!")	3 2 13

类型转化函数

表 2-4 类型转化函数

函数名称	函数描述	样例	运行结果
toset	将列表类型转换为集合类型	toset(["One", "Two", "One"])	["One", "Two"]
tolist	将集合类型转换为列表类型	toset(["One", "Two", "Three"])	["One", "Two", "Three"]
tonumber	将字符串类型转换为数字类型	tonumber("33")	33
tostring	将数字类型转换为字符串类型	tostring(33)	"33"

编码函数

表 2-5 编码函数

函数名称	函数描述	样例	运行结果
base64encode	将UTF-8字符串转换为base64编码	base64encode("Hello, cloud!")	SGVsbG8sIGNsb3VkIQ==
base64decode	将base64编码解码为UTF-8字符串(结果非UTF-8格式会报错)	base64decode("SGVsbG8sIGNsb3VkIQ==")	Hello, cloud!
base64gzip	将UTF-8字符串压缩并转换为base64编码	base64gzip("Hello, cloud!")	H4sIAAAAAAAA//JIzcnJ11FIzskvTVEEAAAA//8BAAD//wbrhYUNAAAA

哈希和加密函数

表 2-6 哈希和加密函数

函数名称	函数描述	样例	运行结果
sha256	计算字符串的SHA256值（16进制）	sha256("Hello, cloud!")	0ad167d1e3ac8e9f4e4f7ba83e92d0e3838177e959858631c770caaed8cc5e3a
sha512	计算字符串的SHA512值（16进制）	sha512("Hello, cloud!")	6eb6ed9fc4edffaf90e742e7697f6cc7d8548e98aa4d5aa74982e5cdf78359e84a3ae9f226313b2dec765bf1ea4c83922dbfe4a61636d585da44ffbd7e900f56
base64sha256	计算字符串的SHA256值，并转换为base64编码	base64sha256("Hello, cloud!")	CtFn0eOsjp9OT3uoPpLQ44OBd+lZhYYxx3DKrtjMXjo=
base64sha512	计算字符串的SHA512值，并转换为base64编码	base64sha512("Hello, cloud!")	brbtn8Tt/6+Q50LnaX9sx9hUjpiqTVqnSYLlzfeDWehKOunyJjE7Lex2W/HqTIOSLb/kphY21YXaRP+9fpAPVg==
md5	计算MD5值	md5("hello world")	5eb63bbbe01eeed093cb22bb8f5acdc3

说明

base64sha512("Hello, cloud!")不等于base64encode(sha512("Hello, cloud!"))，因为sha512计算的十六进制值结果在Terraform中是Unicode编码格式，并没指定UTF-8实现。

文件操作函数

表 2-7 文件操作函数

函数名称	函数描述	样例	运行结果
abspath	计算文件的绝对路径	abspath("./hello.txt")	/home/demo/test/terraform/hello.txt
dirname	计算字符串中包含的路径	dirname("foo/bar/baz.txt")	foo/bar
basename	计算字符串中的文件名	basename("foo/bar/baz.txt")	baz.txt
file	读取文件并返回文件内容	file("./hello.txt")	Hello, cloud!
filebase64	读取文件并返回文件内容的base64编码	filebase64("./hello.txt")	SGVsbG8sIGNsb3VkIQ==

3 配置指南

3.1 Provider

Provider

Terraform的配置文件以 ".tf" 或".tf.json"结尾，主要由**provider**，**resource**，**data source**和**变量**组成。

每个 Provider 代表一个服务提供商，Terraform 通过插件机制与Provider进行交互。Provider通过关键字 "provider" 进行声明，Provider的配置参数请参考**这里**。

执行 terraform init 命令时会下载使用的插件，默认将从Terraform官方仓库下载最新版本的插件。对于Terraform 0.13之后的版本，可以使用 "required_providers" 指定Provider的 registry 源和版本。

```
terraform {
  required_providers {
    huaweicloud = {
      source = "huaweicloud/huaweicloud"
      version = ">= 1.20.0"
    }
  }
  required_version = ">= 0.13"
}
```

在Terraform中，您可以使用 provider块创建多个配置，其中一个 provider块为默认配置，其它块使用 "alias" 标识为非默认配置。在资源中使用元参数 provider 可以选择非默认的 provider块。例如需要在不同的地区管理资源，首先需要声明多个 provider块：

```
provider "huaweicloud" {
  region = "cn-north-1"
  ...
}

provider "huaweicloud" {
  alias  = "guangzhou"
  region = "cn-south-1"
  ...
}
```

示例中声明了北京和广州的华为云provider，并对广州地区的provider增加了别名。在资源中使用元参数 provider 来选择非默认的 provider块，其格式为：<provider名称>.<别名>。


```
resource "huaweicloud_vpc" "example" {
  provider = huaweicloud.guangzhou
  name     = "terraform_vpc"
  cidr     = "192.168.0.0/16"
}
```

华为云Provider 支持在Resource中指定region参数，可以在不同的地区创建资源。相比 alias + provider 的方式，这种方式更加灵活简单。

```
provider "huaweicloud" {
  region = "cn-north-1"
  ...
}

resource "huaweicloud_vpc" "example" {
  region = "cn-south-1"
  name   = "terraform_vpc"
  cidr   = "192.168.0.0/16"
}
```

3.2 Resource

Resource 是Terraform中最重要的元素，通过关键字 "resource" 进行声明。Provider中支持的云服务都有一个或多个资源与之对应，如huaweicloud_compute_instance表示ECS，huaweicloud_vpc表示VPC等。资源之间的关系可以通过关系型资源进行关联，如可以使用 huaweicloud_compute_eip_associate 给ECS分配EIP。

```
resource "huaweicloud_compute_instance" "myinstance" {
  ...
}

resource "huaweicloud_vpc_eip" "myeip" {
  ...
}

resource "huaweicloud_compute_eip_associate" "associated" {
  public_ip   = huaweicloud_vpc_eip.myeip.address
  instance_id = huaweicloud_compute_instance.myinstance.id
}
```

资源引用

可以通过表达式引用资源的属性，格式为：<资源类型>.<名称>.<属性>。假设您已经创建了名称为 myinstance 的 huaweicloud_compute_instance资源，举例如下：

```
# 实例ID
> huaweicloud_compute_instance.myinstance.id
55534eaa-533a-419d-9b40-ec427ea7195a

# 实例安全组
> huaweicloud_compute_instance.myinstance.security_groups
["default", "internet"]

# 实例第一个网卡的IP地址
> huaweicloud_compute_instance.myinstance.network[0].fixed_ip_v4
192.168.0.245

# 实例所有网卡的IP地址
huaweicloud_compute_instance.myinstance.network[*].fixed_ip_v4
["192.168.0.24", "192.168.10.24"]

# 标签key的值
> huaweicloud_compute_instance.myinstance.tags["key"]
value
```

3.3 Data Source

Data Source

Data Source 可以认为是特殊的Resource，通过关键字 "data" 进行声明。Data Source 用于查询已经存在资源的属性和信息，例如可以通过 "Ubuntu 18.04 server 64bit" 的镜像名称查询得到对应镜像的ID及其他属性：

```
data "huaweicloud_images_image" "myimage" {
  name = "Ubuntu 18.04 server 64bit"
}
```

查询到镜像之后，您可以引用该镜像的属性供其他资源使用，引用格式为：data.<数据类型>.<名称>.<属性>

```
resource "huaweicloud_compute_instance""flexibleengine_compute_instance_v2" "demo" {
  name     = "ecs-demo"
  image_id = data.huaweicloud_images_image.myimage.id
  ...
}
```

3.4 变量

3.4.1 输入变量

输入变量可以理解为模块的参数，通过关键字 "variable" 进行声明。通过定义输入变量，您可以无需变更模块的源代码就能灵活修改配置。输入变量的值可以使用默认值，CLI 选项，环境变量等方式来设置。

定义输入变量

按照约定，输入变量通常在名为 variables.tf 的文件中定义。输入变量通过关键字 "variable" 进行声明：

```
variable "image_id" {
  type        = string
  description = "image id of Ubuntu 1804"
}

variable "availability_zone_name" {
  type    = string
  default = "cn-north-1a"
}
```

variable 关键字后的标签是输入变量的名称，该名称在同一模块中的所有变量之间必须唯一。变量的名称可以是除保留关键字以外的任何有效的标识符。保留关键字包括：

```
source    version    providers    count    for_each    lifecycle    depends_on    locals
```

variable块中主要包括如下参数：

- type：指定变量的类型，默认为 string。
- description ： 指定变量的描述信息，用于描述变量的用途。
- default：指定变量的默认值，存在默认值的变量可视为可选变量。

- validation块：指定变量的自定义验证规则。

如果未明确指定变量类型，则默认为 string。建议开发者显式指定变量类型，这样可以方便地提醒用户如何使用该模块，并允许Terraform在使用错误的类型后返回有用的错误信息。Terraform 输入变量支持的类型有：

- 基本类型：string，number，bool
- 复合类型：list(<TYPE>)，set(<TYPE>)，map(<TYPE>)

复合类型的变量定义如下：

```
variable "availability_zone_names" {
  type    = list(string)
  default = ["cn-north-1a"]
}

variable "docker_ports" {
  type = list(object({
    internal = number
    external = number
    protocol = string
  }))
  default = [{
    internal = 8300
    external = 8300
    protocol = "tcp"
  }]
}
```

自定义验证规则

您可以使用 validation嵌套块为输入变量指定自定义验证规则，该特性在 Terraform 0.13.0之后的版本支持，例如：

```
variable "iam_user_password" {
   type        = string
   description = "The password for iam user to log in."

   validation {
     condition     = length(var.iam_user_password)>=8
     error_message = "The password is too short."
   }
}
```

其中，condition 参数是一个布尔表达式，您可以使用 can 函数来检测表达式是否会产生错误，例如：

```
variable "iam_user_name" {
   type        = string
   description = "This name is used for iam user to log in."

   validation {
     # regex(...) 如果匹配失败将返回错误
     condition     = can(regex("([a-zA-Z])", var.iam_user_name))
     error_message = "Incorrect user name. Please check whether it contains upper and lower case letters."
   }
}
```

如果condition 的结果为false，Terraform 将产生一条错误消息，其内容为 error_message 所定义的字符串。error_message 应该至少是一个完整的句子，以大写字母开头，以 "." 或者 "?" 结尾。

引用输入变量

输入变量可以通过 var.<变量名称> 的形式访问，且只能在声明该变量的模块内访问：

```
# variables.tf
variable "vpc_cidr" {
  type        = string
  description = "the CIDR of VPC"
}

# main.tf
resource "huaweicloud_vpc" "vpc_example" {
```

设置变量

通过如下方式可以设置输入变量：

- 通过命令行中 -var 选项指定
- 通过变量定义文件 (.tfvars)，在命令行中指定或自动加载
- 设置环境变量

变量定义 (.tfvars) 文件

如果配置中使用了很多变量，建议使用变量定义文件来设置这些变量，然后通过 -var-file 选项指定该文件：

```
terraform apply -var-file="testing.tfvars"
```

变量定义文件的扩展名为 ".tfvars"，变量定义文件的语法与配置文件的语法相同，但仅用于指定变量名称：

```
vpc_name = "my_vpc"
vpc_cidr = "192.168.0.0/16"
availability_zone_names = [
    "cn-north-1a","cn-north-1c",
]
```

Terraform 还会自动加载特殊命名的变量定义文件：

- 文件名为 terraform.tfvars 或 terraform.tfvars.json 的文件
- 文件名称以 .auto.tfvars 或 .auto.tfvars.json 结尾的文件

对于以 .json 结尾的文件，需要使用 JSON对象表示：

```
{
    "vpc_name": "my_vpc"
    "availability_zone_names": ["cn-north-1a", "cn-north-1c"]
}
```

变量定义优先级

您可以自由组合使用上述设置变量的方式。对于复合类型的变量，为了提高可读性并避免转义带来的问题，建议使用变量定义文件来设置。如果为同一个变量分配了多个值，Terraform 将使用最后一个值进行覆盖。Terraform 根据以下顺序加载变量 (根据顺序，后面的源优于前面的源)：

1. 环境变量

2. terraform.tfvars 或 terraform.tfvars.json 文件
3. *.auto.tfvars 或 *.auto.tfvars.json 文件
4. 命令行中的 -var 和 -var-file 选项

注：不能在单个源中为同一个变量分配多个值。

有关变量的更多信息，请参见Terraform的 **输入变量**文档。

3.4.2 输出变量

输出变量可以理解为模块的返回值，通过关键字 "output" 进行声明。输出变量是一种对外公开某些信息的方法，既可以在根模块中运行 terraform apply/output 命令输出特定的值，又可以在子模块中将资源的属性值提供给父模块。

声明输出变量

按照约定，输出变量通常在名为 variables.tf 的文件中定义。输出变量通过“output”关键字进行声明：

```
output "ecs_address" {
  value       = huaweicloud_compute_instance.myinstance.network[0].fixed_ip_v4
  description = "The private IP address of my ECS"
}
```

output 关键字后的标签为输出变量的名称，该名称必须是有效的标识符。output块中主要包括以下参数：

- value：必选项，输出变量的值，任何有效的表达式都可作为输出使用。
- description：输出变量的描述信息，用于描述输出变量的用途。

  ```
  output "vpc_id" {
    value       = huaweicloud_vpc.myvpc.id
    description = "Check out the VPC ID"
  }
  ```

- sensitive：将输出变量标记为敏感项，在 CLI 中将隐藏输出变量值的显示。

  ```
  output "vpc_id" {
    value       = huaweicloud_vpc.myvpc.id
    description = "Check out the VPC ID"
    sensitive   = true
  }

  $ terraform output
  vpc_id = <sensitive>
  ```

  注意：标记为敏感项的输出变量在输出时会自动被隐藏，但其输出值仍然可以通过以下方式可见：

  - 输出变量的值记录在 state 文件中，其值对所有能够访问state 文件的用户均可见。
  - 子模块中敏感输出变量值被父模块调用，通过父模块的相关输出和资源引用后可以在CLI中显示。

- depends_on：指定输出变量的依赖关系。由于输出变量只是导出数据的一种手段，因此通常不需要设置与其他资源、数据的依赖关系。

3.4.3 本地变量

本地变量可以理解为模块中的临时变量，其作用范围在所声明的模块内，通过关键字 "locals" 进行声明。本地变量适用于配置中有重复定义相同值或表达式的场景，可以减

少代码冗余，并且易于修改。同时过度使用本地变量会导致变量的实际值被隐藏，代码晦涩，不利于维护，因此建议合理使用本地变量。

声明本地变量

本地变量通过“locals”关键字进行声明：

```
locals {
  service_name = "forum"
  owner        = "Community"
}
```

本地变量的表达式不仅限于字符和数值常量，还可以使用输入变量、资源属性和其他本地变量的引用和表达式结果：

```
locals {
  dns_list = concat(huaweicloud_vpc_subnet.subnet_1.dns_list, huaweicloud_vpc_subnet.subnet_2.dns_list)
}

locals {
  common_tags = {
    Service = local.service_name
    Owner   = local.owner
  }
}
```

引用本地变量

在声明本地变量后，可以通过 local.<变量名称> 对其进行引用。

```
resource "huaweicloud_obs_bucket" "bucket_demo" {
  ...
  tags = local.common_tags
}
```

3.5 Metadata

3.5.1 Metadata 说明

Metadata是Terraform支持的内置元参数，可以在 provider，resource，data块中使用。本章节主要介绍 resource块支持的元参数，主要包括：

- depends_on：用于指定资源的依赖项
- count：用于创建多个相同配置的资源
- for_each：用于根据映射、字符串集合创建多个资源
- provider：用于选择非默认的 provider
- lifecycle：用于定制资源的生命周期

3.5.2 depends_on

在同一个 Terraform 配置文件中可以包含多个资源。

通过在资源中引用其他资源的属性值，Terraform可以自动推断出资源的依赖关系。然而，某些资源的依赖关系对于Terraform是不可见的，这就需要使用 depends_on 来创建显式依赖。您可以使用 depends_on 来更改资源的创建顺序或执行顺序，使其在所依赖资源之后处理。

depends_on 的表达式是依赖资源的地址列表。例如在远程操作一台ECS服务器之前，需要为其绑定EIP或配置NAT规则。

```
resource "huaweicloud_compute_instance" "myinstance" {
  ...
}

resource "huaweicloud_vpc_eip" "myeip" {
  ...
}

resource "huaweicloud_compute_eip_associate" "associated" {
  public_ip   = huaweicloud_vpc_eip.myeip.address
  instance_id = huaweicloud_compute_instance.myinstance.id
}

resource "null_resource" "provision" {
  depends_on = [huaweicloud_compute_eip_associate.associated]

  provisioner "remote-exec" {
    connection {
      # 通过公网地址访问 ECS
      host = huaweicloud_vpc_eip.myeip.address
      ...
    }
    inline = [
      ...
    ]
  }
}
```

3.5.3 count

默认情况下，Terraform的 resource块只配置一个资源。当需要创建多个相同的资源时，如果配置多个独立的 resource块就显得很冗余，且不利于维护。可以使用 count 或 for_each 参数在同一个 resource块中管理多个相同的资源。在同一个 resource块中不能同时使用count 和 for_each 参数。示例如下：

```
resource "huaweicloud_evs_volume" "volumes" {
  count = 3

  size              = 20
  volume_type       = "SSD"
  availability_zone = "cn-north-4a"
}
```

通过如上配置创建了3个相同的云硬盘（EVS）。在很多情况下，Provider 要求创建资源的某些参数具有唯一性，这时可以使用 "count.index" 属性来进行区分，这是一个从0开始计数的索引值。

```
resource "huaweicloud_vpc" "vpcs" {
}
```

通过如上配置创建了两个VPC，名字分别为 myvpc_0 和 myvpc_1，它们具有相同的CIDR值。如果进一步修改CIDR值，可以声明一个string列表用于存储不同VPC的CIDR值，然后通过 count.index 去访问列表元素。

```
variable "name_list" {
  type    = list(string)
  default = ["vpc_demo1", "vpc_demo2"]
}
variable "cidr_list" {
  type    = list(string)
  default = ["192.168.0.0/16", "172.16.0.0/16"]
}
```



```
resource "huaweicloud_vpc" "vpcs" {
  count = 2
  name  = var.name_list[count.index]
  cidr  = var.cidr_list[count.index]
}
```

使用 count 创建的资源需要通过索引值进行访问，格式为：<资源类型>.<名称>[索引值]

```
# 访问第一个VPC
> huaweicloud_vpc.vpcs[0]

# 访问第一个VPC的ID
> huaweicloud_vpc.vpcs[0].id

# 访问所有VPC的ID
> huaweicloud_vpc.vpcs[*].id
```

3.5.4 for_each

for_each 在功能上与 count 相似，for_each 使用键值对或字符串集合的形式快速地将值填入到对应的属性中，不仅可以优化脚本结构也有利于理解多实例间的关系。

在使用映射类型表达时，您可以使用 "each.key" 和 "each.value" 来访问映射的键和值。以创建VPC为例，通过 for_each 中的键值对，您可以灵活配置VPC的名称和CIDR。

```
resource "huaweicloud_vpc" "vpcs" {
  for_each = {
    vpc_demo1 = "192.168.0.0/16"
    vpc_demo2 = "172.16.0.0/16"
}

  name = each.key
  cidr = each.value
}
```

在使用字符串集合类型表达时，"each.key" 等同于 "each.value"，一般使用 each.key 表示，另外，可以通过 toset() 函数将定义的 list 类型进行转化：

```
resource "huaweicloud_networking_secgroup" "mysecgroup" {
  for_each = toset(["secgroup_demo1", "secgroup_demo2"])
  name     = each.key
}

# 通过变量表示 for _each
variable "secgroup_name" {
  type = set(string)
}
resource "huaweicloud_networking_secgroup" "mysecgroup" {
  for_each = var.secgroup_name
  name     = each.key
}
```

使用 for_each 创建的资源需要通过键名进行访问，格式为：<资源类型>.<名称>[键名]

```
# 访问 vpc_demo1
> huaweicloud_vpc.vpcs["vpc_demo1"]
# 访问 vpc_demo1 的ID
> huaweicloud_vpc.vpcs["vpc_demo1"].id
```

由于 count 和 for_each 都可用于创建多个资源，建议参考以下规则进行选择：

1、如果资源实例的参数完全或者大部分一致，建议使用count；

2、如果资源的某些参数需要使用不同的值并且这些值不能由整数派生，建议使用for_each；

3.5.5 provider

在Terraform中，您可以使用 provider块创建多个配置，其中一个 provider块为默认配置，其它块使用 "alias" 标识为非默认配置。在资源中使用元参数 provider 可以选择非默认的 provider块。例如需要在不同的地区管理资源，首先需要声明多个 provider块：

```
provider "huaweicloud" {
  region = "cn-north-1"
  ...
}

provider "huaweicloud" {
  alias  = "guangzhou"
  region = "cn-south-1"
  ...
}
```

示例中声明了北京和广州的华为云provider，并对广州地区的provider增加了别名。在资源中使用元参数 provider 来选择非默认的 provider块，其格式为：<provider名称>.<别名>。

```
resource "huaweicloud_networking_secgroup" "mysecgroup" {
  # 使用非默认 provider块名，对应非默认provider块的别名(alias)
  provider = huaweicloud.guangzhou
  ...
}
```

华为云Provider 支持在Resource中指定region参数，可以在不同的地区创建资源。相比 alias + provider 的方式，这种方式更加灵活简单。

```
provider "huaweicloud" {
  region = "cn-north-1"
  ...
}

resource "huaweicloud_vpc" "example" {
  region = "cn-south-1"
  name   = "terraform_vpc"
  cidr   = "192.168.0.0/16"
}
```

3.5.6 lifecycle

每个资源实例都具有创建 、更新和销毁三个阶段，在一个资源实例的生命周期过程中都会经历其中的2至3个阶段。通过元参数 lifecycle 可以对资源实例的生命周期过程进行改变，lifecycle 支持以下参数：

- **create_before_destroy**

  默认情况下，当需要改变资源中不支持更新的参数时，Terraform会先销毁已有实例，再使用新配置的参数创建新的对象进行替换。当您将 create_before_destroy 参数设置为 true 时，Terraform将先创建新的实例，再销毁之前的实例。这个参数可以适用于保持业务连续的场景，由于新旧实例会同时存在，需要提前确认资源实例是否有唯一的名称要求或其他约束。

  ```
  lifecycle {
    create_before_destroy = true
  }
  ```

- **prevent_destroy**

  当您将 prevent_destroy 参数设置为true时，Terraform将会阻止对此资源的删除操作并返回错误。这个元参数可以作为一种防止因意外操作而重新创建成本较高实例的安全措施，例如数据库实例。如果要删除此资源，需要将这个配置删除后再执行 destroy 操作。

  ```
  lifecycle {
    prevent_destroy = true
  }
  ```

- **ignore_changes**

  默认情况下，Terraform plan/apply 操作将检测云上资源的属性和本地资源块中的差异，如果不一致将会调用更新或者重建操作来匹配配置。您可以用 ignore_changes 来忽略某些参数不进行更新或重建。ignore_changes 的值可以是属性的相对地址列表，对于 Map 和 List 类型，可以使用索引表示法引用，如 tags["Name"]，list[0] 等。

  ```
  resource "huaweicloud_rds_instance" "myinstance" {
    ...
    lifecycle {
      ignore_changes = [
        name,
      ]
    }
  }
  ```

  此时，Terraform 将会忽略对 name 参数的修改。除了列表之外，您也可以使用关键字 all 忽略所有属性的更新。

  ```
  resource "huaweicloud_rds_instance" "myinstance" {
    ...
    lifecycle {
      ignore_changes = all
    }
  }
  ```

3.6 Extension

3.6.1 Extension 简介

Extension用于承载RFS系统的扩展能力，这些扩展文件用户可以直接操作和编写。

Extension是RFS支持的内置元参数，可以在模板中配置使用，方便用户更好的配置参数。本章节主要介绍Extension相关参数 ，主要包括：

- i18n：代表要使用多语义化功能
- variables：代表要使用参数预加载功能

3.6.2 Extension 使用

使用Extension注意：

1.在zip压缩包根目录下新建.extension文件夹

2."".extension"文件夹下新建后缀名为.rfs.json的扩展文件，当前只支持扩展功能写在单个文件中

3.6.3 多语义化

参数多语言支持对variable的参数名字、描述和类型进行词条翻译，当前支持zh_cn、en_us两种语义

i18n中示例如下：

```
"i18n": {
      "zh_cn": [
        {
            "variable_name": "参数name",
            "localization": {
                "variable_name": "参数name 中文翻译",
                "description": "描述 中文翻译",
                "type": "类型 中文翻译"
            }
        }
      ],
      "en_us": [
        {
            "variable_name": "参数name",
            "localization": {
                "variable_name": "参数name 英文翻译",
                "description": "描述 英文翻译",
                "type": "类型 英文翻译"
            }
        }
    ]
  }
```

3.6.4 参数预加载

参数预加载支持输入参数时请求云服务api获取预加载选项，当前支持以下六种api接口：

表 3-1 支持的 api 接口说明

属性	描述
hwc:ecs:flavors	ECS的Flavor
hwc:vpc:myVpcs	VPC列表
hwc:vpc:mySubnets	Subnet子网列表
hwc:vpc:mySecurityGroups	securityGropus安全组列表
hwc:evs:types	EVS 类型
hwc:ecs:availabilityZones	AZ 可用区

variables使用样例：

```
"variables":[
      {
          "name": "参数name",
          "prompt": "hwc:vpc:mySubnets"
      },
      {
          "name": "参数name",
          "prompt": "hwc:evs:types"
```

```
    },
    {
        "name": "参数name",
        "prompt": "hwc:ecs:flavors"
    },
    {
        "name": "参数name",
        "prompt": "hwc:vpc:myVpcs"
    },
    {
        "name": "参数name",
        "prompt": "hwc:ecs:availabilityZones"
    },
    {
        "name": "参数name",
        "prompt": "hwc:vpc:mySecurityGroups"
    }
]
```

4 模板约束与限制

使用RFS服务进行模板部署，模板文件内容具有如下约束与限制：

- 模块的数量限制为25，模块嵌套深度限制为3。
- 不能使用**Provisioners功能**、**Backend Configuration功能**和**Cloud 功能**。
- 可以使用**Module Sources功能**，但仅支持Local Modules。
- 可以使用**HuaweiCloud Provider**，但Provider内部分资源禁止使用。具体清单列举如下：
  - huaweicloud_vod_watermark_template
  - huaweicloud_compute_keypair
  - huaweicloud_identity_access_key
  - huaweicloud_images_image_v2
  - huaweicloud_kps_keypair
  - huaweicloud_obs_bucket_object
  - huaweicloud_iotda_batchtask
  - huaweicloud_cce_chart
  - huaweicloud_iotda_batchtask_file
  - huaweicloud_cse_microservice
- 部分函数禁止使用。具体清单列举如下：
  - **abspath**
  - **basename**
  - **dirname**
  - **file**
  - **filebase64**
  - **filebase64sha256**
  - **filebase64sha512**
  - **fileexists**
  - **fileset**
  - **filemd5**
  - **filesha1**

- **filesha256**
- **filesha512**
- **pathexpand**
- **templatefile**

⚠ 警告

不建议使用nonsensitive方法输出敏感信息。随意使用此方法可能会导致本该被隐藏的敏感信息被服务明文打印出来从而导致敏感信息泄露

如果必须进行输出，建议优先考虑编码后再输出（如nonsensitive(sha256(var.sensitive_value))）