Session 会话

TensorFlow运行模型 - 会话

会话（session）

会话拥有并管理TensorFlow程序运行时的所有**资源**

当所有计算完成之后需要**关闭会话**帮助系统**回收资源**

会话的模式 1

In [17]:
```
# 定义计算图
tens1 = tf.constant([1,2,3])

# 创建一个会话
sess = tf.Session()

#使用这个创建好的会话来得到关心的运算的结果。比如可以调用 sess.run(result)
#来得到张量result的取值
print(sess.run(tens1))

#关闭会话使得本次运行中使用到的资源可以被释放
sess.close()
```

```
[1 2 3]
```

需要明确调用 Session.close()函数来关闭会话并释放资源

当程序因为异常退出时，关闭会话函数可能就不会被执行从而导致资源泄漏

会话的模式 1

In [17]:
```
# 定义计算图
tens1 = tf.constant([1,2,3])

# 创建一个会话
sess = tf.Session()

try:
    print(sess.run(tens1))
except:
    print("Exception!")
finally:
    #确保能关闭会话使得本次运行中使用到的资源可以被释放
    sess.close()
```

```
[1 2 3]
```

try … except … finally…

会话的模式 2

In [18]:
```
node1 = tf.constant(3.0,tf.float32,name="node1")
node2 = tf.constant(4.0,tf.float32,name="node2")
result = tf.add(node1, node2)

#创建一个会话，并通过Python中的上下文管理器来管理这个会话
with tf.Session() as sess:
    #使用这创建好的会话来计算关心的结果
    print(sess.run(result))

# 不需要再调用 Session.close() 函数来关闭会话
# 当上下文退出时会话关闭和资源释放也自动完成了
```

```
7.0
```

指定默认的会话

TensorFlow不会自动生成默认的会话，需要手动指定

当默认的会话被指定之后可以通过 **tf.Tensor.eval** 函数来计算一个张量的取值

In [19]:
```
node1 = tf.constant(3.0,tf.float32,name="node1")
node2 = tf.constant(4.0,tf.float32,name="node2")
result = tf.add(node1, node2)

sess = tf.Session()
with sess.as_default():
    print(result.eval())
```

```
7.0
```

右边代码也可以完成相同的功能

In [20]:
```
sess = tf.Session()

#下面两个命令有相同的功能
print(sess.run(result))

print(result.eval(session=sess))
```

```
7.0
7.0
```

交互式环境下设置默认会话

在交互式环境下，Python脚本或者Jupyter编辑器下，通过设置默认会话来获取张量的取值更加方便

tf.InteractiveSession 使用这个函数会自动将生成的会话注册为默认会话

In [23]:
```
node1 = tf.constant(3.0,tf.float32,name="node1")
node2 = tf.constant(4.0,tf.float32,name="node2")
result = tf.add(node1, node2)

sess = tf.InteractiveSession()

print(result.eval())
sess.close()
```

```
7.0
```

常量与变量

常量 constant

在运行过程中值不会改变的单元，在TensorFlow中无须进行初始化操作

创建语句：

```
constant_ name = tf.constant(value)
```

In [2]:
```
a = tf.constant(1.0, name='a')
b = tf.constant(2.5, name='b')
c = tf.add(a, b, name='c')

sess = tf.Session()
c_value = sess.run(c)
print(c_value)
sess.close()
```

```
3.5
```

变量 Variable

在运行过程中值会改变的单元，在TensorFlow中须进行初始化操作

创建语句：

```
name_variable = tf.Variable(value, name)
```

注意V是大写字母

个别变量初始化：

```
init_op = name_variable.initializer()
```

所有变量初始化：

```
init_op = tf.global_variables_initializer()
```

变量 Variable

In [3]:
```
node1 = tf.Variable(3.0,tf.float32,name="node1")
node2 = tf.Variable(4.0,tf.float32,name="node2")
result = tf.add(node1, node2, name='add')

sess = tf.Session()

#变量初始化
init = tf.global_variables_initializer()
sess.run(init)

print(sess.run(result))
```

```
7.0
```

增加了一个init初始化变量，并调用会话的run命令对参数进行初始化

变量 Variable

浙江大学城市学院 ZHEJIANG UNIVERSITY CITY COLLEGE

In [4]:
```
node1 = tf.Variable(3.0,tf.float32,name="node1")
node2 = tf.Variable(4.0,tf.float32,name="node2")
result = tf.add(node1, node2, name='add')

sess = tf.Session()

#变量初始化
init = tf.global_variables_initializer()

print(sess.run(result))
```

```
---------------------------------------------------------------------------
FailedPreconditionError                   Traceback (most recent call last)
C:\Users\mingh\Anaconda3\lib\site-packages\tensorflow\python\client\session.py in _do_call(self, fn, *args)
   1138     try:
-> 1139       return fn(*args)
   1140     except errors.OpError as e:
```

使用了Variable变量类型，不进行初始化数值会出现运行错误

变量的赋值

变量赋值

- 与传统编程语言不同，TensorFlow中的变量定义后，一般**无需**人工赋值，系统会根据算法模型，训练优化过程中**自动调整变量对应的数值**
- 后面在将机器学习模型训练时会更能体会，比如权重Weight变量w，经过多次迭代，会自动调

```
epoch = tf.Variable(0,name='epoch',trainable=False)
```

- 特殊情况需要人工更新的，可用变量赋值语句

  变量更新语句：

  update_op = tf.assign(variable_to_be_updated, new_value)

变量赋值案例

In [7]:

```
# 通过变量赋值输出1、2、3...10

import tensorflow as tf

value = tf.Variable(0, name="value")
one = tf.constant(1)
new_value = tf.add(value, one)
update_value = tf.assign(value, new_value)

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    for _ in range(10):
        sess.run(update_value)
        print(sess.run(value))
```

```
1
2
3
4
5
6
7
8
9
10
```

思考题

如何通过TensorFlow的变量赋值计算：1+2+3+…+10 ？

占位符 placeholder

占位符 placeholder

- TensorFlow中的**Variable变量类型**，在定义时需要初始化，但有些变量**定义时并不知道其数值**，只有当真正开始运行程序时，才由外部输入，比如训练数据，这时候需要用到**占位符**
- tf.placeholder**占位符**，是TensorFlow中特有的一种数据结构，类似动态变量，函数的参数、或者C语言或者Python语言中格式化输出时的“%”占位符

占位符 placeholder

- TensorFlow占位符Placeholder，先定义一种数据，其参数为数据的Type和Shape

  占位符Placeholder的函数接口如下：

tf.placeholder(dtype, shape=None, name=None)

```
x = tf.placeholder(tf.float32, [2, 3], name='tx')

# 此代码生成一个2x3的二维数组，矩阵中每个元素的类型都是tf.float32，内部对应的符号名称是tx
```

Feed提交数据和Fetch提取数据

Feed提交数据

如果构建了一个包含placeholder操作的计算图，当在session中调用run方法时，placeholder占用的变量必须通过feed_dict参数传递进去，否则报

In [1]:
```
import tensorflow as tf

a = tf.placeholder(tf.float32, name='a')
b = tf.placeholder(tf.float32, name='b')
c = tf.multiply(a, b, name='c')

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    # 通过feed_dict的参数传值，按字典格式
    result = sess.run(c, feed_dict={a:8.0, b:3.5})

    print(result)
```

```
28.0
```

Feed提交数据

多个操作可以通过一次Feed完成执行

In [2]:
```
import tensorflow as tf

a = tf.placeholder(tf.float32, name='a')
b = tf.placeholder(tf.float32, name='b')
c = tf.multiply(a, b, name='c')
d = tf.subtract(a, b, name='d')

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)

    result = sess.run([c,d], feed_dict={a:[8.0,2.0,3.5], b:[1.5,2.0,4.]})

    print(result)
    # 取结果中的第一个
    print(result[0])
```

```
[array([ 12.,   4.,  14.], dtype=float32), array([ 6.5,  0. , -0.5], dtype=float32)]
[ 12.   4.  14.]
```

Feed和Fetch

一次返回多个值分别赋给多个变量

In [6]:
```
import tensorflow as tf

a = tf.placeholder(tf.float32, name='a')
b = tf.placeholder(tf.float32, name='b')
c = tf.multiply(a, b, name='c')
d = tf.subtract(a, b, name='d')

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)

    #返回的两个值分别赋给两个变量
    rc,rd = sess.run([c,d], feed_dict={a:[8.0,2.0,3.5], b:[1.5,2.0,4.]})

    print("value of c=",rc,"value of d=",rd)
```

```
value of c= [ 12.   4.  14.] value of d= [ 6.5  0.  -0.5]
```