1. rank 为 2；rank 应该译为秩，这个秩与矩阵的秩意义不同，tensor（张量）的 rank 指的是 tensor 的维度，和 rank 意义相同的表达方式还有 order、degree、n-dimension。只有一个实数的 tensor，其 shape 为（,），rank 为 0；一个向量，其 shape 为（n,），rank 为 1；一个矩阵，其 shape 为（n，m），rank 为 2，依次类推。
2. tf.matmul(A,B)是矩阵乘法运算，返回值就是两个矩阵进行矩阵乘法运算的结果。
3. tf.multiply(A,B)是 tensor 的 element-wise product，即对应元素相加。要求 A 和 B 的 shape 相同。
4. tf.square(A)是 element-wise square，即各个元素的平方。更多的 Tensor 运算可以查看 tf.math 包。
5. A，B 不能进行矩阵乘法
6. A，B 矩阵乘法结果的 shape 为（2,3,4,4），即除了最后两维要满足矩阵乘法的规则外，其他的维度的大小要相同。
7. （4,5）；Tensorflow 的 broadcasting 机制。
8. （2,3,4,5）；Tensorflow 的 broadcasting 机制。
9.
```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess=tf.Session(config=config)
```
这样得到的 session 会动态分配显存。
```
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.4
session = tf.Session(config=config)
```
这样得到的 session 会一次占用 GPU 40%的显存。
10. C=tf.where(A>B,A,B)
11. 不能使用 for i in range(tf.shape(A)[2]): 或者 for i in range(numpy.shape(A)[2]):.
前者的 tf.shape()没有与之相关联的节点被运算，所以不会返回 A 的第二个维度的大小。
而定义 placeholder A 时 shape 为[2,3,None]，所以使用 numpy.shape()仍然无法得到 A 的动态维度的大小。
解决这种根据 Tensor 的参数循环的问题，应该使用 tf.while_loop(cond,body,(…)).
```
def cond(n, i, other_param):
  return i < n

def body(n, i, other_param):
  i += 1
  other_param=other_param+"???"
  return (n,i,other_param)

print(sess.run(tf.while_loop(cond, body, (tf.shape(A)[2], 0, ''))))
```
或者
```
print(sess.run(tf.while_loop(lambda n, i, param: i < n,
                             lambda n, i, param:(n,i+1,param+'???'),
                            (tf.shape(A)[2], 0, ''))))
```
如果传入(feed)的 A 的 shape 是[2,3,4]，则输出都是（4,4,'????????????'）

12. 与上题同理，同样不能直接使用 if tf.shape(A)[2]==25: 和 if numpy.shape(A)[2]==25:
应当使用 tf.cond(cond,fun1,fun2)来进行条件控制。代码如下：

```
print(sess.run(tf.cond(tf.equal(tf.shape(A)[2],10),
                       lambda: 'yesyes',
                       lambda: 'nono'))
```

如果 A 的第二维大小是 10，则应该输出 yesyes。

13.

namescope/var_1:0
var_2:0
variable_scope/var_3:0
variable_scope/var_4:0

14. tf_reuse_node_mem_test()的时间效率和空间效率都比另一个高。
tf_add_node_mem_test()在每次 run 时都会新建一个 op 计算节点，这样会不停的往 session 的 graph 中添加节点，使 graph 不断增大，所以空间效率低；而不停的申请内存和添加节点也拖慢了运行速度。
15. 2
16. [1,2,3]
17. [2,2]
18. [2,2]
19. 2
20. (3,4)
21. (2,4)
22. (2,3)
23. (4,)
24. (4,)
25. (2,3)
26. (5,7,4)　(5,7,4)
27. 无法分割，因为维度 1 的大小 7 不能被分割块数 2 整除
28. (5,7,2)　(5,7,2)　(5,7,4)
29. （2,8）
30. 无法合并
31. 使用 tf.add_to_collection() 添加的集合是添加到默认图中，可以使用 tf.Graph.clear_collection(tf.get_default_graph(),key)来清除。