量子计算
—编程篇

Quantum Computer

网址：www.qubits.top

作者：Calvin Tang

邮箱：179209347@qq.com

介绍

本开发教程基于本源量子的Qpanda框架的python版 – **PyQPanda** 编写。

- 一种功能齐全，运行高效的量子软件开发工具包
- QPanda 2是由本源量子开发的开源量子计算框架，它可以用于构建、运行和优化量子算法。
- QPanda 2作为本源量子计算系列软件的基础库，为OriginIR、Qurator、量子计算服务提供核心部件。

QPanda使用文档：
https://pyqpanda-toturial.readthedocs.io/zh/latest/index.html

系统配置和安装

系统配置

pyqpanda是以C++为宿主语言，其对系统的环境要求如下：

software	version
GCC	>= 5.4.0
Python	>= 3.6.0（建议3.8以支持VQNet）

安装 pyqpanda

```
pip install pyqpanda
```

QPanda 常用对象

QPanda 中与量子计算相关的对象类型:

- QGate(量子逻辑门)
- Measure(测量)
- ClassicalProg(经典程序)
- QCircuit(量子线路)、
- Qif(量子条件判断程序)
- QWhile(量子循环程序)
- QProg(量子程序)

QPanda 把所有的量子逻辑门封装为API向用户提供使用，并可获得QGate类型的返回值。比如，您想要使用Hadamard门，就可以通过如下方式获得：

```
from pyqpanda import *
import numpy as np
init(QMachineType.CPU)
qubits = qAlloc_many(4)
h = H(qubits[0])
```

QPanda 常用对象

在QPanda2中，QProg是量子编程的一个容器类，是一个量子程序的最高单位。它也是QNode中的一种，初始化一个QProg对象有以下两种：

```
prog = QProg()
```

或：

```
prog = create_empty_qprog()
```

你可以通过如下方式向QProg尾部填充节点，在这里pyqpanda重载了 << 运算符作为插入量子线路的方法：

```
cir << node
```

QNode的类型有QGate，QPorg，QIf，Measure等等，QProg支持插入所有类型的QNode 。

还可以由已有的QNode节点来构建量子程序，如：

```
qubit = qAlloc()
gate = H(qubit)
prog = QProg(gate)
```

0_classes.py

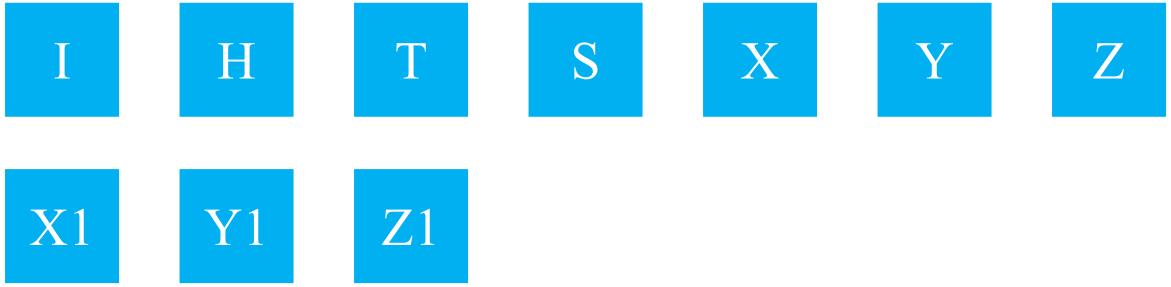
```
from pyqpanda import *

if __name__ == "__main__":

    init(QMachineType.CPU)
    qubits = qAlloc_many(4)
    cbits = cAlloc_many(4)
    prog = QProg()

    # 构建量子程序
        prog << H(qubits[0]) \
        << X(qubits[1]) \
        << iSWAP(qubits[0], qubits[1]) \
        << CNOT(qubits[1], qubits[2]) \
        << H(qubits[3]) \
        << measure_all(qubits, cbits)

    # 量子程序运行1000次，并返回测量结果
        result = run_with_configuration(prog, cbits, 1000)
    # 打印量子态在多次运行结果中出现的次数
        print(result)
        finalize()
```

单比特量子逻辑门

1. 支持的不含角度的单门有：

```
qubits = qAlloc_many(4)
h = H(qubits[0])
```

其中参数为目标比特，返回值为量子逻辑门



2. 支持的单门带有一个旋转角度的逻辑门有：

```
rx = RX(qubits[0]， np.pi/3)
```

第一个参数为目标比特 第二个参数为旋转角度

3. 其它支持的单比特逻辑门有：

```
# U2(qubit, phi, lambda) 有两个角度
  u2 = U2(qubits[0]， np.pi, np.pi/2)
# U3(qubit, theta, phi, lambda) 有三个角度
  u3 = U3(qubits[0]， np.pi, np.pi/2, np.pi/4)
# U4(qubit, alpha, beta, gamma, delta) 有四个角度
  u4 = U4(qubits[0]， np.pi, np.pi/2, np.pi/4, np.pi/2)
```

两比特量子逻辑门

两比特量子逻辑门的使用和单比特量子逻辑门的用法相似，
只不过是输入的参数不同，例如CNOT门：

```
cnot = CNOT(qubits[0]， qubits[1])
```

第一个参数为控制比特 第二个参数为目标比特。

注：两个比特不能相同

1. 支持的双门不含角度的逻辑门有：

CNOT　CZ　SWAP　iSWAP　SqiSWAP

2.支持的双门含旋转角度的逻辑门有：

CR　CU　CP

例如CR门：

```
cr = CR(qubits[0]， qubits[1]， np.pi)
```

第一个参数为控制比特, 第二个参数为目标比特, 第三个参数为旋转角度。

例如CU门：

```
# CU(control, target, alpha, beta, gamma, delta) 有四个角度
  cu = CU(qubits[0]， qubits[1]， np.pi,np.pi/2,np.pi/3,np.pi/4)
```

三量子比特逻辑门

获得三量子逻辑门 Toffoli 的方式：

```
toffoli = Toffoli(qubits[0], qubits[1], qubits[2])
```

Toffoli

三比特量子逻辑门Toffoli实际上是CCNOT门，前两个参数是控制比特，最后一个参数是目标比特。

pyqpanda还支持在量子逻辑门中添加量子比特数组操作，即将该数组中的所有量子比特赋予同一种逻辑门运算，举个使用单门H的例子：

```
# 这里返回的是一个量子线路
circuit = H(Qvec);
```

常用接口介绍

所有的量子逻辑门都是酉矩阵，那么您也可以对量子逻辑门做**转置共轭操作**，获得一个量子逻辑门 dagger 之后的量子逻辑门可以用下面的方法：

```
rx_dagger = RX(qubits[0], np.pi).dagger()
```

或：

```
rx_dagger = RX(qubits[0], np.pi)
rx_dagger.set_dagger(true)
```

可以为量子逻辑门**添加控制比特**，获得一个量子逻辑门 control 之后的量子逻辑门可以用下面的方法：

```
qvec = [qubits[0], qubits[1]]
rx_control = RX(qubits[2], np.pi).control(qvec)
```

或：

```
qvec = [qubits[0], qubits[1]]
rx_control = RX(qubits[2], np.pi)
rx_control.set_control(qvec)
```

1_interface.py

```
from pyqpanda import *
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(3)
    control_qubits = [qubits[0], qubits[1]]
    prog = create_empty_qprog()
    # 构建量子程序
    prog << H(qubits) \
         << H(qubits[0]).dagger() \
         << X(qubits[2]).control(control_qubits)
    # 对量子程序进行概率测量
    result = prob_run_dict(prog, qubits, -1)
    # 打印测量结果
    print(result)
    finalize()
```

量子测量

在量子程序中我们需要对某个量子比特做测量操作，并把测量结果存储到经典寄存器上，可以通过下面的方式获得一个测量对象：

```
measure = Measure(qubit, cbit)
```

第一个是测量比特，第二个是经典寄存器。

如果想测量所有的量子比特并将其存储到对应的经典寄存器上， 可以如下操作：

```
measureprog = measure_all(qubits， cbits);
```

其中qubits的类型是 QVec ， cbits的类型是 ClassicalCondition list。

在得到含有量子测量的程序后，我们可以调用 directly_run 或 run_with_configuration 来得到量子程序的测量结果。
run_with_configuration 的功能是统计量子程序多次运行的测量结果。

2_ measure.py

```
from pyqpanda import *

if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(4)
    cbits = cAlloc_many(4)

    # 构建量子程序
        prog = QProg()
    prog << H(qubits[0])\
        << H(qubits[1])\
        << H(qubits[2])\
        << H(qubits[3])\
        << measure_all(qubits, cbits)

    # 量子程序运行1000次，并返回测量结果
        result = run_with_configuration(prog, cbits, 1000)

    # 打印测量结果
        print(result)
    finalize()
```

概率测量

概率测量是指获得目标量子比特的振幅，目标量子比特可以是一个量子比特也可以是多个量子比特的集合。 在QPanda2 中概率测量又称为PMeasure 。 概率测量和量子测量 是完全不同的过程，Measure是执行了一次测量，并返回一个确定的0/1结果，并且改变了量子态，：

QPanda2提供了三种获得PMeasure结果的方式，其中有 prob_run_list 、 prob_run_tuple_list 、 prob_run_dict。

- prob_run_list ： 获得目标量子比特的概率测量结果列表。
- prob_run_tuple_list： 获得目标量子比特的概率测量结果，为字典类型，其对应的下标为十进制。
- prob_run_dict ： 获得目标量子比特的概率测量结果，为字典类型，其对应的下标为二进制。

第一个参数是量子程序， 第二个参数是 QVec 它指定了我们关注的量子比特。 第三个参的值为-1时，获取所有的概率测量结果，大于0时表示获取最大的前几个数。

3_ pmeasure.py

```
from pyqpanda import *

if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(2)
    cbits = cAlloc_many(2)

    prog = QProg()
    prog << H(qubits[0])\
        << CNOT(qubits[0], qubits[1])

    print("prob_run_dict: ")
    result1 = prob_run_dict(prog, qubits, -1)
    print(result1)

    print("prob_run_tuple_list: ")
    result2 = prob_run_tuple_list(prog, qubits, -1)
    print(result2)

    print("prob_run_list: ")
    result3 = prob_run_list(prog, qubits, -1)
    print(result3)

    finalize()
```

量子线路

在QPanda2中，QCircuit类是一个仅装载量子逻辑门的容器类型，它也是QNode中的一种，初始化一个QCircuit对象有以下两种：

```
cir = QCircuit()
```

或：

```
cir = create_empty_circuit()
```

你可以通过如下方式向QCircuit尾部填充节点，在这里pyqpanda重载了 << 运算符作为插入量子线路的方法：

```
cir << node
```

node的类型可以为QGate或QCircuit。

还可以获得QCircuit的转置共轭之后的量子线路，使用方式为：

```
cir_dagger = cir.dagger()
```

如果想复制当前的量子线路，并给复制的量子线路添加控制比特，可以使用下面的方式：

```
qvec = [qubits[0], qubits[1]]
cir_control = cir.control(qvec)
```

4_circuit.py

```
from pyqpanda import *
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(4)
    cbits = cAlloc_many(4)

    # 构建量子程序
    prog = QProg()
    circuit = create_empty_circuit()
    circuit << H(qubits[0]) \
            << CNOT(qubits[0], qubits[1]) \
            << CNOT(qubits[1], qubits[2]) \
            << CNOT(qubits[2], qubits[3])
    prog << circuit << Measure(qubits[0], cbits[0])
# 量子程序运行1000次，并返回测量结果
    result = run_with_configuration(prog, cbits, 1000)  #
打印量子态在量子程序多次运行结果中出现的次数
    print(result)
    finalize()
```

运行结果：

```
{'0000': 486, '0001': 514}
```

79209347 Mail: 179209347@qq.com

H (Hadamard) 门 - 编程实现

Hadamard 门是一种可以将基态变为叠加态的量子逻辑门，简称H门。

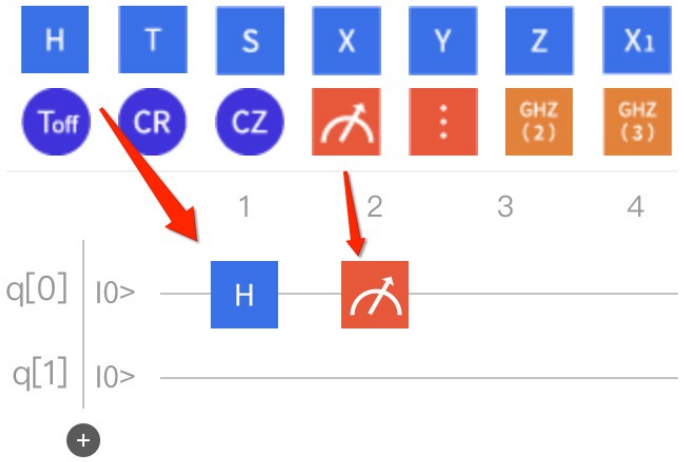
矩阵形式 $H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 1\end{bmatrix}\langle 0| + \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ -1\end{bmatrix}\langle 1|$

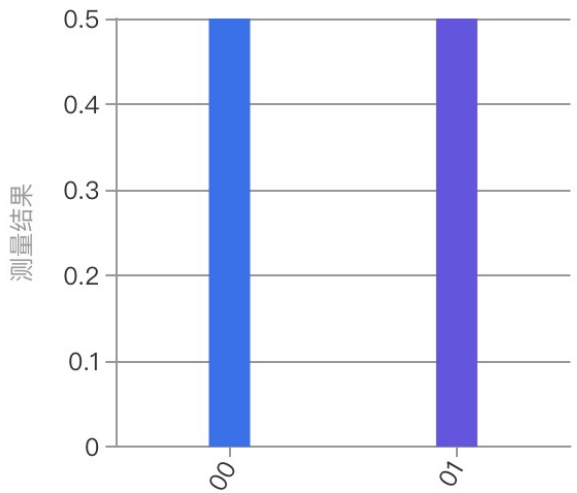
量子线路符号：

测量符号：

H 门作用在基态：

$$H|0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)$$





5_HGate.py

```
from pyqpanda import *
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(1)
    cbits = cAlloc_many(1)
    # 构建量子程序
    prog = QProg()
    prog << H(qubits[0]) \
         << Measure(qubits[0], cbits[0])
    # 量子程序运行1000次，并返回测量结果
    result = run_with_configuration(prog, cbits, 1000)
    print(result)
    finalize()
```

运行结果：

```
{'0': 505, '1': 495}
```

Pauli-X 门 - 编程实现

Pauli-X 作用在单量子比特上，跟经典计算机的NOT门的量子等价，将量子态翻转，量子态变换规律是：

Pauli-X 门矩阵形式为泡利矩阵σ_x，即：

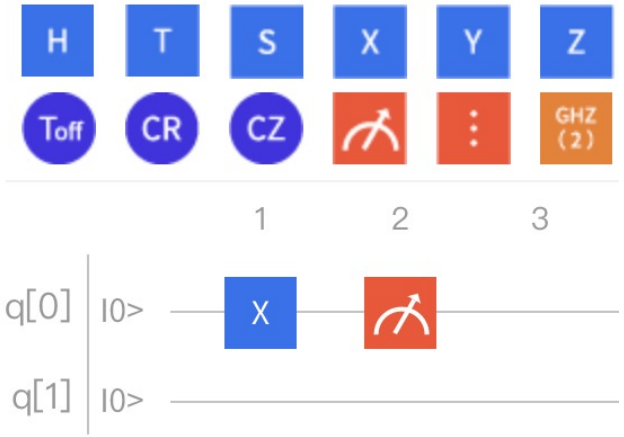
$$X = \sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

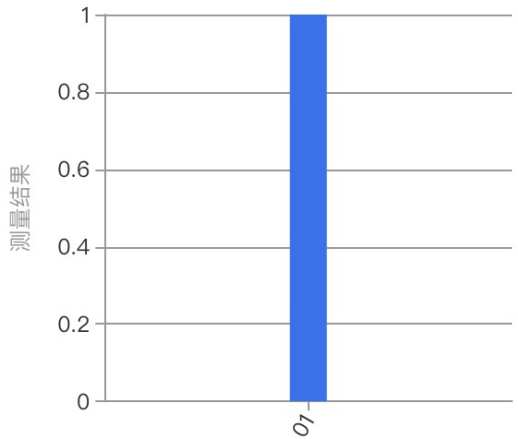
$|0\rangle \rightarrow |1\rangle$

$|1\rangle \rightarrow |0\rangle$

X 门作用在基态：

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$





6_XGate.py

```
from pyqpanda import *
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(1)
    cbits = cAlloc_many(1)
    # 构建量子程序
    prog = QProg()
    prog << X(qubits[0]) \
         << Measure(qubits[0], cbits[0])
    # 量子程序运行1000次，并返回测量结果
    result = run_with_configuration(prog, cbits, 1000)
    print(result)
    finalize()
```

运行结果：

```
{'1': 1000}
```

Pauli-Y 门 - 编程实现

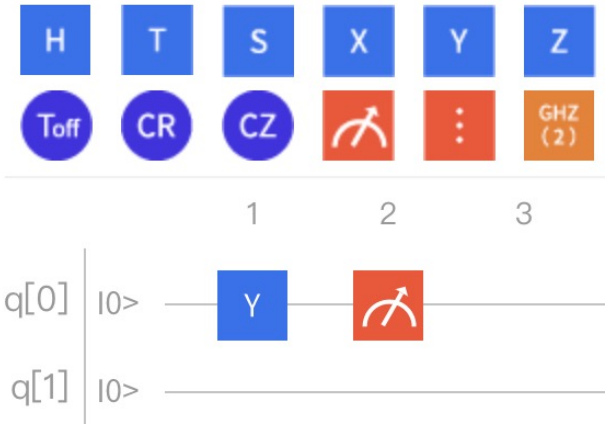
Pauli-Y 作用在单量子比特上，作用相当于绕布洛赫球 Y 轴旋转角度π.

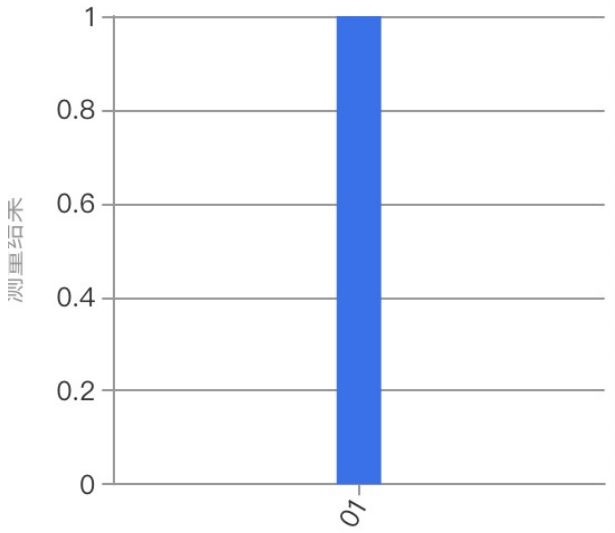
Pauli-Y 门矩阵形式为泡利矩阵σ_y，即：

$$Y = \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Y 门作用在基态：

$$Y|0\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ i \end{bmatrix} = i \begin{bmatrix} 0 \\ 1 \end{bmatrix} = i|1\rangle$$





7_YGate.py

```
from pyqpanda import *
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(1)
    cbits = cAlloc_many(1)
    # 构建量子程序
    prog = QProg()
    prog << X(qubits[0]) \
         << Measure(qubits[0], cbits[0])
    # 量子程序运行1000次，并返回测量结果
    result = run_with_configuration(prog, cbits, 1000)
    print(result)
    finalize()
```

运行结果：

```
{'1': 1000}
```

Pauli-Z 门 - 编程实现

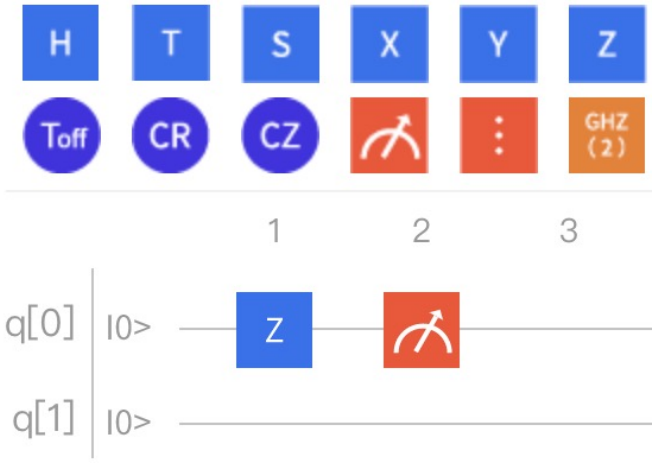
Pauli-Z 作用在单量子比特上，作用相当于绕布洛赫球 Z 轴旋转角度π.

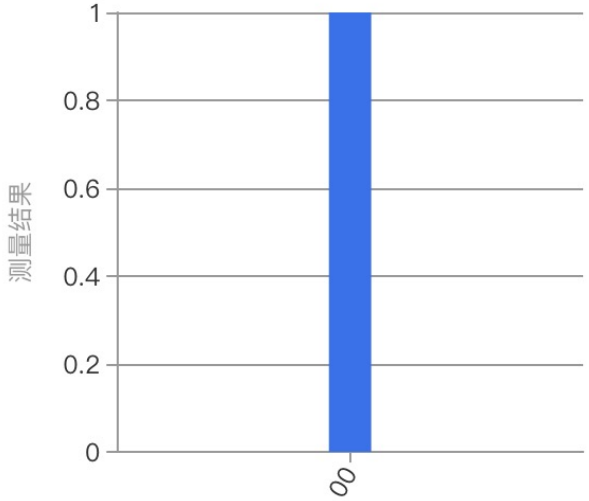
Pauli-Z 门矩阵形式为泡利矩阵σ_z，即：

$$Z = \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Z 门作用在基态：

$$Z|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$





8_ZGate.py

```
from pyqpanda import *
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(1)
    cbits = cAlloc_many(1)
    # 构建量子程序
    prog = QProg()
    prog << Z(qubits[0]) \
         << Measure(qubits[0], cbits[0])
    # 量子程序运行1000次，并返回测量结果
    result = run_with_configuration(prog, cbits, 1000)
    print(result)
    finalize()
```

运行结果：

```
{'0': 1000}
```

RX(θ) 门 - 编程实现

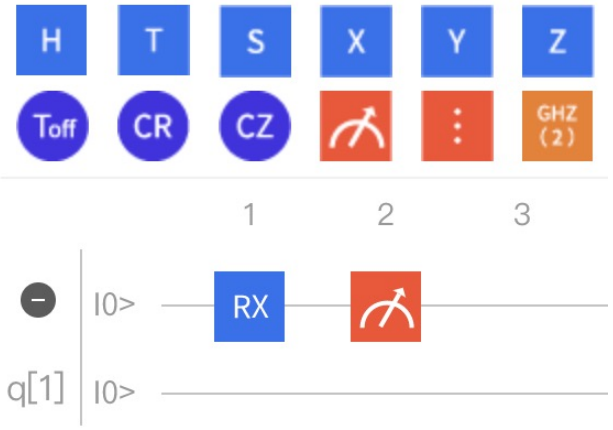
RX(π/2)门作用在基态：

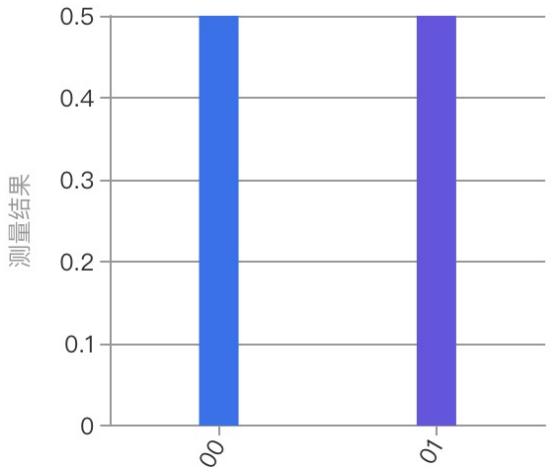
$$R_x(\theta)\,|0\rangle = \begin{bmatrix} \cos(\frac{\pi}{4}) & -i\sin(\frac{\pi}{4}) \\ -i\sin(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\frac{\pi}{4}) \\ -i\sin(\frac{\pi}{4}) \end{bmatrix}$$

$$= \cos\left(\frac{\pi}{4}\right)|0\rangle - \mathrm{i}\sin\left(\frac{\pi}{4}\right)|1\rangle$$

$$= \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}\mathrm{i}\,|1\rangle$$





9_RXGate.py

```
from pyqpanda import *
import numpy as np
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(1)
    cbits = cAlloc_many(1)
    # 构建量子程序
    prog = QProg()
    prog << RX(qubits[0]，np.pi/2) \
         << Measure(qubits[0], cbits[0])
    # 量子程序运行1000次，并返回测量结果
    result = run_with_configuration(prog, cbits, 1000)
    print(result)
    finalize()
```

运行结果：

```
{'0': 473, '1': 527}
```

RY(θ) 门 - 编程实现

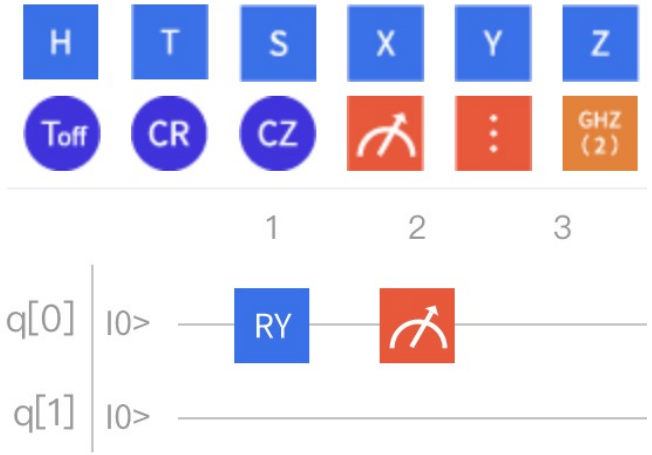
RY(π/2) 门作用在基态：

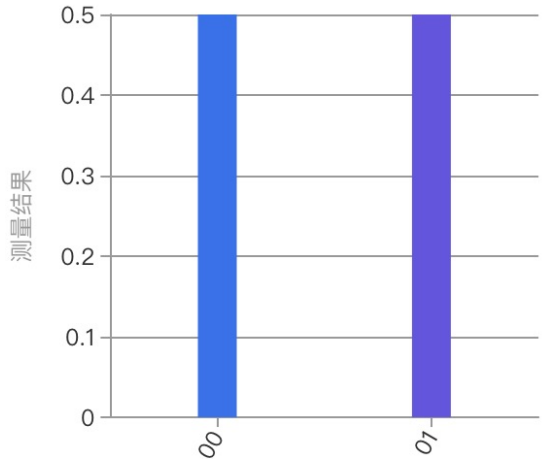
$$R_y(\theta)\,|0\rangle = \begin{bmatrix} \cos(\frac{\pi}{4}) & -sin(\frac{\pi}{4}) \\ \sin(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\frac{\pi}{4}) \\ \sin(\frac{\pi}{4}) \end{bmatrix}$$

$$= \cos\left(\frac{\pi}{4}\right)|0\rangle + \sin\left(\frac{\pi}{4}\right)|1\rangle$$

$$= \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$





10_RYGate.py

```
from pyqpanda import *
import numpy as np
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(1)
    cbits = cAlloc_many(1)
    # 构建量子程序
    prog = QProg()
    prog << RY(qubits[0]，np.pi/2) \
         << Measure(qubits[0], cbits[0])
    # 量子程序运行1000次，并返回测量结果
    result = run_with_configuration(prog, cbits, 1000)
    print(result)
    finalize()
```

运行结果：

```
{'0': 505, '1': 495}
```

Calvin, QQ: 179209347 Mail: 179209347@qq.com

RZ(θ) 门 - 编程实现

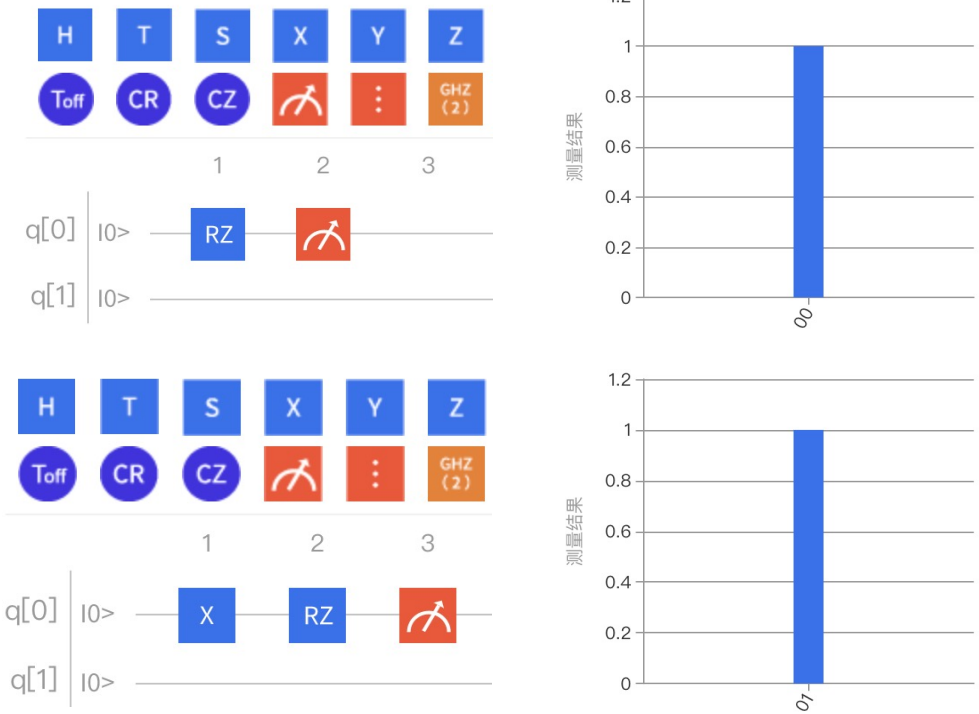
RZ门又称为相位转化门(phase-shift gate)，由Pauli-Z 矩阵作为生成元生成，其矩阵形式为：

RZ门作用在基态：

$$R_z(\theta)\,|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

$$R_z(\theta)\,|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ e^{i\theta} \end{bmatrix} = e^{i\theta}\,|1\rangle$$

$e^{-i\theta/2}$ 并没有对计算基|0⟩ 和 |1⟩ 做任何改变，而只是在原来的态上绕Z轴逆时针旋转θ角。



11_RZGate.py

```
from pyqpanda import *
import numpy as np
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(1)
    cbits = cAlloc_many(1)
    # 构建量子程序
    prog = QProg()
    prog << RZ(qubits[0]， np.pi/2) \
         << Measure(qubits[0], cbits[0])
    # 量子程序运行1000次，并返回测量结果
    result = run_with_configuration(prog, cbits, 1000)
    print(result)
    finalize()
```

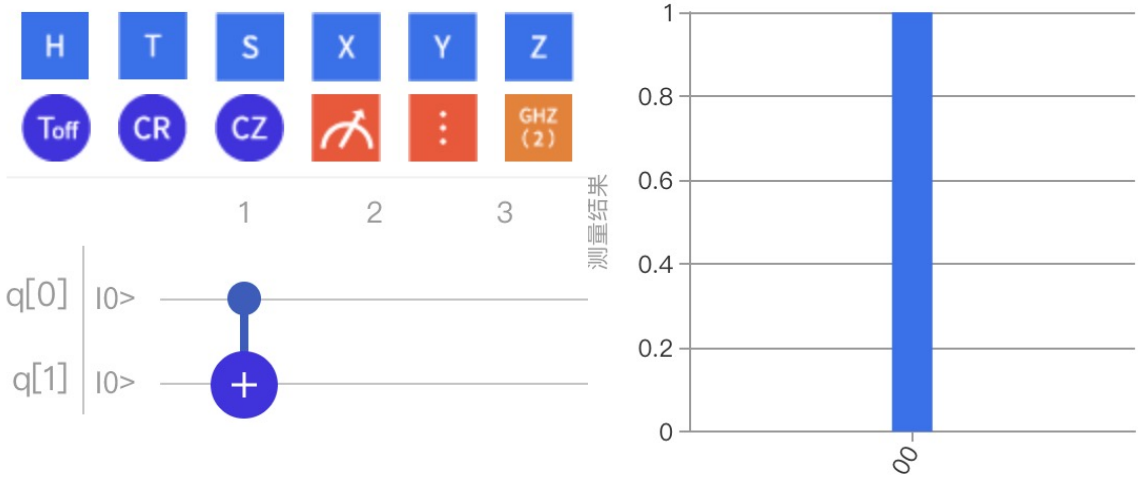
运行结果：{'0': 1000}

Calvin，QQ: 179209347 Mail: 179209347@qq.com

CNOT 门 - 编程实现

控制非门(Control - NOT)，通常用 CNOT 表示，是一种普遍使用的两量子比特门。
如果低位作为控制比特，则它的矩阵形式：

$$\mathrm{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$



12_CNOTGate.py

```
from pyqpanda import *
import numpy as np
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(2)
    cbits = cAlloc_many(2)
    # 构建量子程序
    prog = QProg()
    prog << CNOT(qubits[0], qubits[1]) \
         << Measure(qubits[0], cbits[0]) \
         << Measure(qubits[1], cbits[1])
    # 量子程序运行1000次，并返回测量结果
    result = run_with_configuration(prog, cbits, 1000)
    print(result)
    finalize()
```

运行结果：

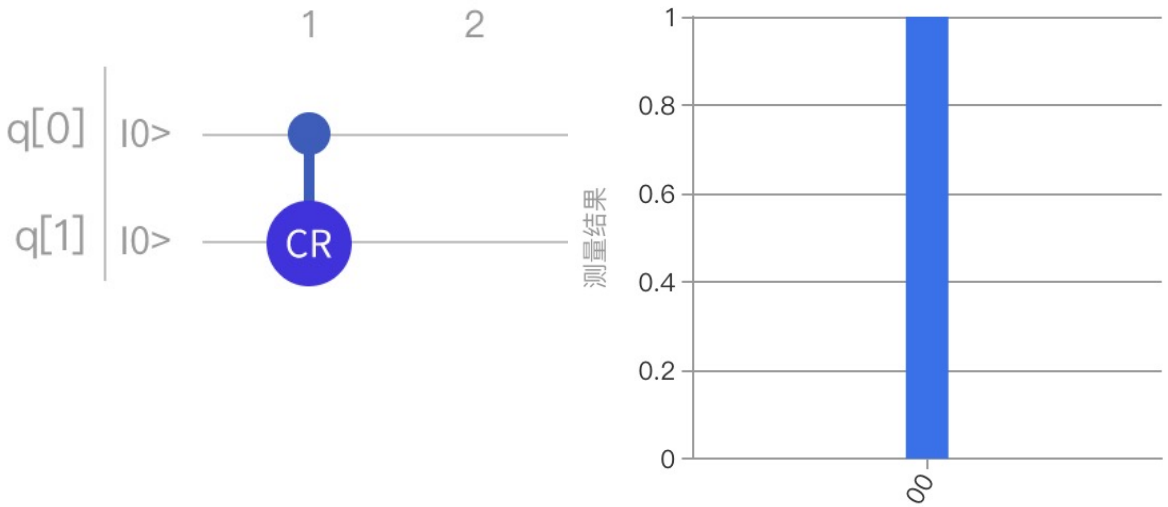
```
{'00': 1000}
```

Calvin, QQ: 179209347 Mail: 179209347@qq.com

CR 门 - 编程实现

控制相位门(Control phase gate) 和控制非门类似，通常用 CR（CPhase）表示，它的矩阵形式：

$$\mathrm{CR}\,(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{bmatrix}$$



13_CRGate.py

```
from pyqpanda import *
import numpy as np
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(2)
    cbits = cAlloc_many(2)
    # 构建量子程序
    prog = QProg()
    prog << CR(qubits[0], qubits[1] , np.pi/2) \
         << Measure(qubits[0], cbits[0]) \
         << Measure(qubits[1], cbits[1])
    # 量子程序运行1000次，并返回测量结果
    result = run_with_configuration(prog, cbits, 1000)
    print(result)
    finalize()
```

运行结果：

```
{'00': 1000}
```

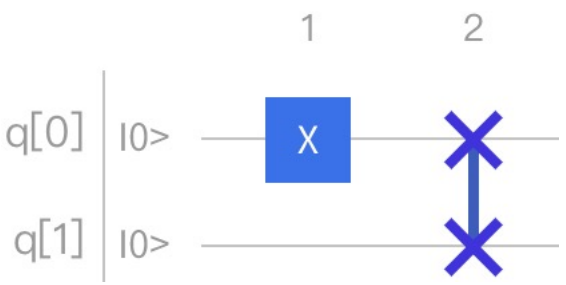
Calvin, QQ: 179209347 Mail: 179209347@qq.com

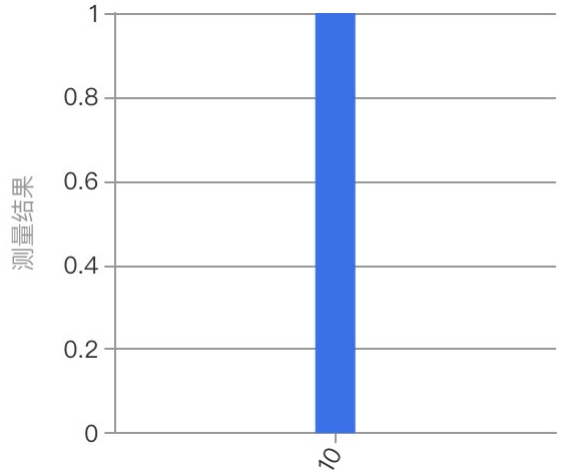
SWAP 门 - 编程实现

SWAP门可以将 |01⟩ 态变为 |10⟩ ， |10⟩ 变为 |01⟩，它的矩阵形式： 14_SWAPGate.py

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$|\psi'\rangle = SWAP\ |01\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |10\rangle$$






```
from pyqpanda import *
import numpy as np
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(2)
    cbits = cAlloc_many(2)
    # 构建量子程序
    prog = QProg()
    prog << X(qubits[0]) \
         << SWAP(qubits[0], qubits[1]) \
         << Measure(qubits[0], cbits[0]) \
         << Measure(qubits[1], cbits[1])
    # 量子程序运行1000次，并返回测量结果
    result = run_with_configuration(prog, cbits, 1000)
    print(result)
    finalize()
```

运行结果：

```
{'10': 1000}
```

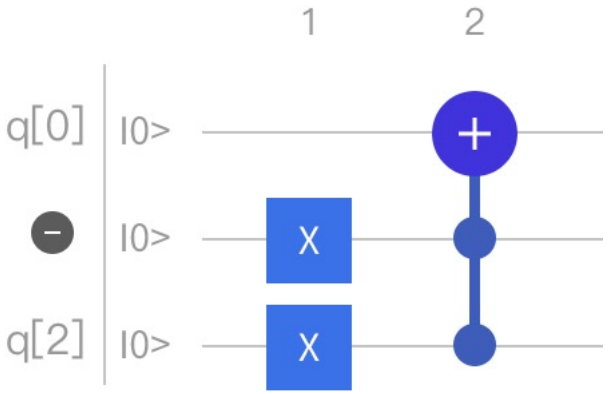
Calvin, QQ: 179209347 Mail: 179209347@qq.com

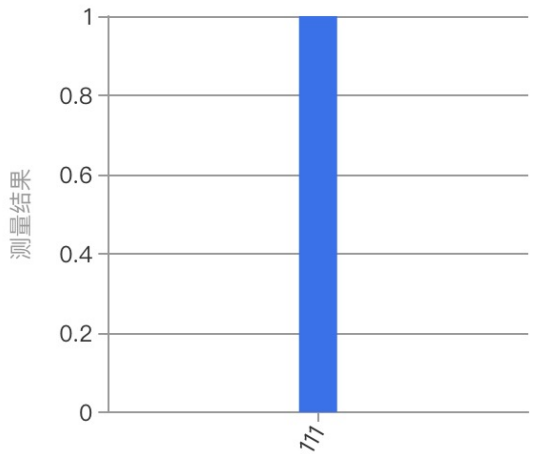
Toffoli（CCNOT）- 编程实现

Toffoli门即CCNOT门，它涉及3个量子比特，两个控制比特，一个目标比特。

Toffoli门作用于|110⟩ ：

$$\text{CCNOT}\,|110\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |111\rangle$$





15_ToffoliGate.py

```
from pyqpanda import *
import numpy as np
if __name__ == "__main__":
    init(QMachineType.CPU)
    qubits = qAlloc_many(2)
    cbits = cAlloc_many(2)
    # 构建量子程序
    prog = QProg()
    prog << X(qubits[1]) \
         << X(qubits[2]) \
         << Toffoli(qubits[1],qubits[2],qubits[0])  \
         << Measure(qubits[0], cbits[0]) \
         << Measure(qubits[1], cbits[1]) \
         << Measure(qubits[2], cbits[2])
    # 量子程序运行1000次，并返回测量结果
    result = run_with_configuration(prog, cbits, 1000)
    print(result)
    finalize()
```

运行结果：

```
{'111': 1000}
```

Calvin，QQ: 179209347 Mail: 179209347@qq.com

Thank You