Class Bit:

|  |  |
| --- | --- |
| 属性 | 说明 |
| value | 当前bit的值，只能0或者1 |
|  |  |

|  |  |
| --- | --- |
| 方法 | 说明 |
| \_\_init\_\_ | 对value进行初始化，需要的参数为0或者1 |
| \_\_add\_\_ | 重写加法 bit+bit=str,0+1=01 |

Class baseQubit

|  |  |
| --- | --- |
| 属性 | 说明 |
| matrix |  |
| amplitude |  |

|  |  |
| --- | --- |
| 方法 | 说明 |
| setAmp |  |
| setMatrix | 将新计算出来的matrix复制给当前qubit或者qubits，执行结果后调用setAmp，将计算结果同样赋值给amplitude |
| getMatrix |  |
| getAmp |  |
| \_\_mul\_\_ |  |
| normalize | 在decideProb函数中调用。对该量子比特或纠缠量子比特进行归一化处理，计算结束后调用setMatrix和setAmp |

Class Qubit

|  |  |
| --- | --- |
| 属性 | 说明 |
| mode | 类属性，执行过程中所有qubit共享，当前执行的模式：theory or simulator |
| matrix | Qubit的矩阵表示2\*1 |
| amplitude | 对应的概率幅，1\*2 |
| assignmentError | 制备的误差，在errorRate的配置文件中根据当前ids读取 |
| singleGateError | 单量子比特逻辑门作用在该实例上导致的误差，在errorRate的配置文件中根据当前ids读取 |
| ids | Qubit的id，不可重复 |
| idList | 类属性，存放执行过程中所有的id |
| entanglement | 表示当前比特是否存在于某个qs中，初始化为false |

|  |  |
| --- | --- |
| 方法 | 说明 |
| setMatrix | 参数是一个2\*1的list，将这个list赋给当前matrix |
| setAmp | 根据matrix来计算amplitude |
| \_\_mul\_\_ | Qubit\*Qubit，张量积 |
| \_\_del\_\_（存疑） | 析构函数，在del qubit的时候要删掉在idList里面对应的内容 |
| getMatrix | 只能通过这个函数来访问qubit的矩阵表示，因为可以比特在CNOT的时候被修改而不知道 |
| decideProb | 计算出该qubit的0态和1态各自对应的概率。该函数只在circuit的execute函数中执行，因为在量子比特一旦纠缠起来，只考虑其整体，不再分割开来考虑；除非在测量的时候后仅测量纠缠比特其中的一个 |
| recordQubit | 将当前qubit存入circuit的实例中：   1. 在qubitExecuteList中新建一个当前qubit的字典 2. 对qubitNum加一 |

Class Qubits

|  |  |
| --- | --- |
| 属性 | 说明 |
| number | 存放的qubit的个数 |
| matrix | Qubits的矩阵表示 n\*1 |
| amplitude | 对应的概率幅，1\*n |
| qubitList | 存放qubit实例的列表 |

|  |  |
| --- | --- |
| 方法 | 说明 |
| \_\_getitem\_\_ | 使得qs[1]可以直接获取对应的qubitlist里面的元素 |
| setMatrix | 这三个与上面的相同 |
| setAmp |  |
| \_\_mul\_\_ |  |
| addNewItem | 向原Qubits里面添加新的qubits或者qubit，使用两者的乘积作为新的矩阵 |
| \_\_init\_\_ | q1,q2这两个参数都是qubit类型，但是不能是已经纠缠的qubit。如果想对应纠缠的比特添加新的，使用addNewItem函数 |
| \_\_mulMatrix | 参数是两个matrix，计算两个矩阵的张量积，并返回这个张量积 |
| getIndex | 参数是一个qubit，返回这个qubit在qubitList里面的索引；如果不在返回-1 |

Class Circuit

|  |  |
| --- | --- |
| 属性 | 说明 |
| name | 该实验的名字，可以传递成参数，如果该参数为空那么会自己生成（EXP+时间） |
| ids | 实验的唯一标志，name可以重复但是这个不会重复 |
| qubitExecuteList | 量子比特的执行情况；  每一维度记录一个量子比特。  应该是字典里面套列表的数据结构，即：  {q.ids: [“X q.ids;…”],  q.ids:[“ Y q.ids,…”]} |
| currentIDList | 这个属性是类属性，共有的；存放当前所有Circuit的实例的id，只有当该变量的len为1且ids与当前执行函数的实例的ids相同时，才为允许运行的环境 |
| qubitNum | 线路中包含的量子比特的个数 |
| Urls | 该实验所在的路径 |
| instance | 类属性，共有的，存放最新的circuit实例，新声明的实例会覆盖掉旧实例，这个属性在gate等函数中会大量使用 |
| measureList | 实例属性，将当前线路中需要测量的量子比特的实例存放在这个list中，初始化是在gate文件中的M函数，只在self.execute()函数中对其进行访问，且一旦初始化就不能修改 |
| beginTime | Datetime.datetime.now，记录该线路执行的开始时间 |
| endTime | 记录该线路执行的终止时间 |
| beginMemory | Psutil库，记录该线路之前开始时占用的内存 |
| endMemory | 记录该线路执行终止的内存 |
| mode | 记录当前线路的执行模式：simulator; theory |

|  |  |
| --- | --- |
| 方法 | 说明 |
| \_\_exportCircuit | 绘制线路图 |
| \_\_del\_\_ | 在currentIDList里面删去当前销毁的实例的ids |
| \_\_QASM | 将代码转化为QASM格式并导出为txt文件，但这个代码并没有考虑到实际的CNOT联通情况，IBMQX类中会对代码进行优化 |
| execute | 用实例调用这个函数的时候才开始执行测量；此外这个函数还调用了\_\_exportCircuit,\_\_QASM和\_\_expoerChart  该函数还有一个参数为executeTimes，数据类型为int。表示执行多少次，根据最终态各自对应的概率来决定测量结果满足预期的概率，执行次数越多，理论上与实际结果越接近 |
| checkEnvironment | 检查执行的instance的id和类属性中存储的instance的id是否相同 |
| \_\_exportChart | 绘制柱状图和扇形图 |
| \_\_orderTheID | 因为有待测量的量子比特可能存在纠缠，所以测量的时候不一定能保证是按照顺序测量的。该函数用快排的方式将测量后的量子比特按照ids递增的顺序排列。将调整之后量子比特对应的原来的位置返回。 |
| countGate | 统计当前量子逻辑门的数量：  1.单量子比特逻辑门  2.双量子比特逻辑门  3.测量 |

Class IBMQX

|  |  |
| --- | --- |
| 属性 | 说明 |
| \_\_config | 用于初始化api的参数，包含了url |
| api | 用于调用api接口 |
| device | 可用设备的名称，可用设备列表通过IBM提供的api接口available\_backends得到 |
| shot | 执行次数 |
| connectivity | 当前设备的CNOT的联通情况：  {‘0’:[‘1’,’2’]}即可以以q0作为控制比特控制q1和q2 |

|  |  |
| --- | --- |
| 方法 | 说明 |
| \_\_getAvailalbeBak | 获取可用的设备列表 |
| executeQASM | 执行QASM代码（circuit.QASM（）生成的txt文件），可选的参数是实验名称experimentName |
| \_\_canExecute | 调整QASM代码，使其满足CNOT连通性的要求，如果能够运行，返回执行代码，如果不能运行，返回None；当前并非只有一个circuit，返回None；qasm代码还尚未生成，返回None（这个版本不提供自动修改功能） |

File Gate

|  |  |
| --- | --- |
| 方法 | 说明 |
| handleQubits | 当逻辑门作用的比特处于纠缠态时调用；假设Qubits中有n个量子比特，那个构成的态可以由一个包含2^n个元素的列向量构成，循环访问这个向量中的每个元素，判断这个元素对应的概率是否为0，为0则跳过该循环；不为0则判断该元素对应的二进制表示法中，执行单量子比特逻辑门的那一位是否为0，为零则执行gate\*|0>，然后将这个计算的结果乘以该元素对应的概率；为1则执行gate\*|1>；将计算的结果存储为一个list：tmpResult，再将该list存在endResult中。最后将endResult中各个list的对应位置相加，即得出最终的结果，将该结果赋值为qs.matrix即可 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |