复数对于旋转的表示非常重要

- 它引入了**旋转算子（rotational operator）**的思想：可以通过复数表示一个旋转变换。
- 它是**四元数**和**多向量**的内在属性。

两个复数的乘积

- 两个复数的乘积就是各项分别相乘并相加$z_1z_2=(a+bi)(c+di)=(ac-bd)+(ad+bc)i$
- 两个复数的加减和乘积都是一个复数。

共轭复数

- 两个复数相乘还有个特殊情况：$(a+bi)(a-bi)=a^2+b^2$

两个复数的除法

- 利用共轭复数的性质，$z_1/z_2=(ac+bd)+(bc-ad)i\ /\ c^2+d^2$

复数与旋转

- **乘以 i^2看成是绕原点 180° 的旋转**、乘以 **i** 表示**90° 的旋转**。旋转 **45°** 究竟是乘以 **0.5i**
- **乘以一个复数，可以同时带来两种变换的效果。**
  长度的缩放（通过改变模长）。旋转（通过改变幅角）。

旋转子

- **乘以一个模为 1 的复数时，不会导致缩放，只会产生旋转。**这样的复数就称为**旋转子（rotor）**
- **旋转子的共轭复数等于顺时针旋转**

自定义复数类

```
class ComplexNum
{
public:
 ComplexNum();
```

```
 ComplexNum(double a,double b);
public:
 //复数的四则运算
 ComplexNum operator +(const ComplexNum& num);
 ComplexNum operator -(const ComplexNum& num);
  ComplexNum operator *(const ComplexNum& num);
  ComplexNum operator /(const ComplexNum& num);

  //其他函数,设置和取模
  void setComplexNumValue(double a,double b);
  double getComplexNumMold();
  double A();
  double B();

private:
  double a,b;
};
```

```
ComplexNum::ComplexNum():a(0),b(0)
{

}

ComplexNum::ComplexNum(double a,double b){
 this->a=a;
 this->b=b;
}

//复数的四则运算
ComplexNum ComplexNum::operator +(const ComplexNum& num){
  return ComplexNum(this->a+num.a,this->b+num.b);
}

ComplexNum ComplexNum::operator -(const ComplexNum& num){
```

```
17  return ComplexNum(this->a-num.a,this->b-num.b);
18 }
19
20 //复数的相乘(a+bi)(c+di)=(ac-bd)+(bc+ad)i两个复数的积仍然是一个复数
21 ComplexNum ComplexNum::operator *(const ComplexNum& num){
22  return ComplexNum(this->a*num.a-this->b*num.b,this->b*num.a+th
is->a*num.b);
23 }
24
25 //复数的除法
26 ComplexNum ComplexNum::operator /(const ComplexNum& num){
27  if (!num.a && !num.b){
28  qDebug()<<"除数不能位0";
29  return ComplexNum(a, b);
30  }else{
31  return ComplexNum((a*num.a + b*num.b) / (num.a*num.a + num.b*n
um.b),
32  (b*num.a - a*num.b) / (num.a*num.a + num.b*num.b));
33  }
34 }
35
36 //其他函数,设置和取模
37 void ComplexNum::setComplexNumValue(double a,double b){
38  this->a=a;
39  this->b=b;
40 }
41
42 double ComplexNum::getComplexNumMold(){
43  return sqrt(a*a+b*b);
44 }
45
46 double ComplexNum::A(){
47  return this->a;
48 }
49 double ComplexNum::B(){
50  return this->b;
```

```
51 }
```