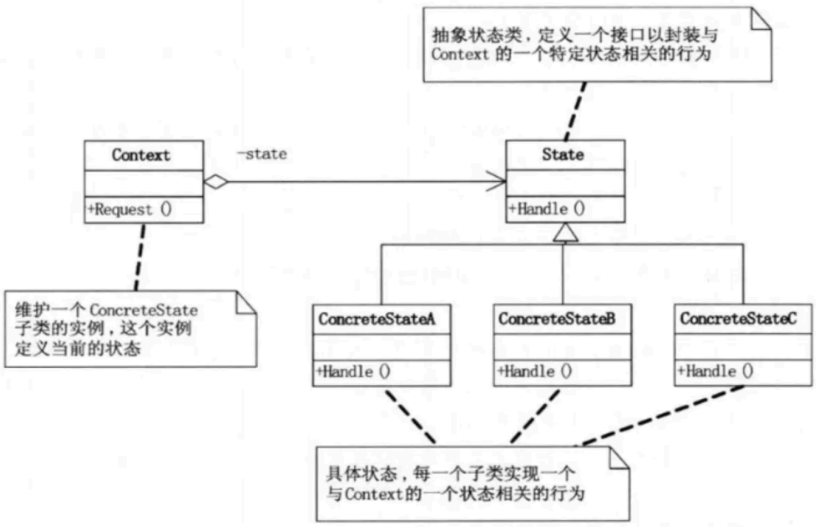
动机:在软件构建过程中, 某些对象的状态如果改变, 其行为也地随之而发生改变,比如文档处于只读状态,其支持的行为和读写状态支持的行为就可能完全不同. 如何在运行时根据对象的状态来透明地更改对象的行为? 而不会为对象操作和状态转化之间引入紧耦合?

定义:允许一个对象在其内部状态改变时改变它的行为. 从而使对象看起来似乎修改了其行为. --<设计模式> GoF

State类：抽象状态类，定义一个接口以封装与Context的一个特定状态相关的行为。

ConcreteState类：具体状态，每一个子类实现一个与Context的一个状态相关的行为。

Context类：维护一个ConcreteState子类的实例，这个实例定义当前的状态。



状态模式的主要解决的是，当控制一个对象状态转换的条件表达式过于复杂时的情况。把状态的判断逻辑转移到表示不同状态的一系列类当中，可以把复杂的判断逻辑简化。

```
#include <iostream>
using namespace std;
class Context;
class Status{
public:
    virtual ~Status() {}
    virtual void Handle(Context*) = 0;
};
class Context{
private:
    Status *m_pStatus;
public:
    Context(Status *pS){
        m_pStatus = pS;
    }
    void Request(){
        m_pStatus->Handle(this);
    }
    void SetStatus(Status *s){
        m_pStatus  = s;
    }
};
class ConcretStatusA : public Status{
public:
    void Handle(Context* c);
};
class ConcretStatusB : public Status{
```

```
public:
    void Handle(Context* c);
};
class ConcretStatusC : public Status{
public:
    void Handle(Context* c);
};
void ConcretStatusA::Handle(Context *c){
    cout << "Handle A " << endl;
    c->SetStatus(new ConcretStatusB());
}
void ConcretStatusB::Handle(Context *c){
    cout << "Handle B " << endl;
    c->SetStatus(new ConcretStatusC());
}
void ConcretStatusC::Handle(Context *c){
    cout << "Handle C " << endl;
    c->SetStatus(new ConcretStatusC());
}
int main(){
    Status * s = new ConcretStatusA();
    Context *c =  new Context(s);
    c->Request();
    c->Request();
    c->Request();
    delete s;
    delete c;
    return 0;
};
```

```
192:DesignPattnsStudy weishichun$ g++ -o State.out Status_1.cpp
192:DesignPattnsStudy weishichun$ ./State.out
Handle A
Handle B
Handle C
192:DesignPattnsStudy weishichun$
```