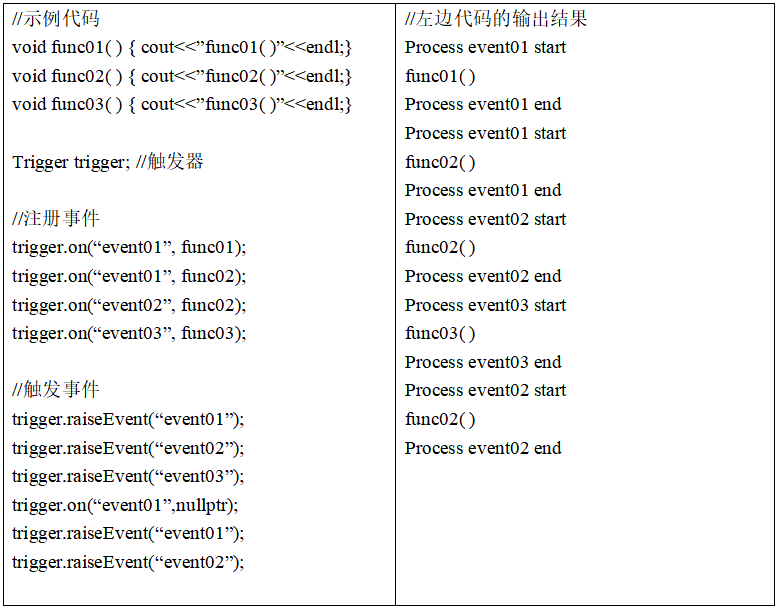
**题目：**

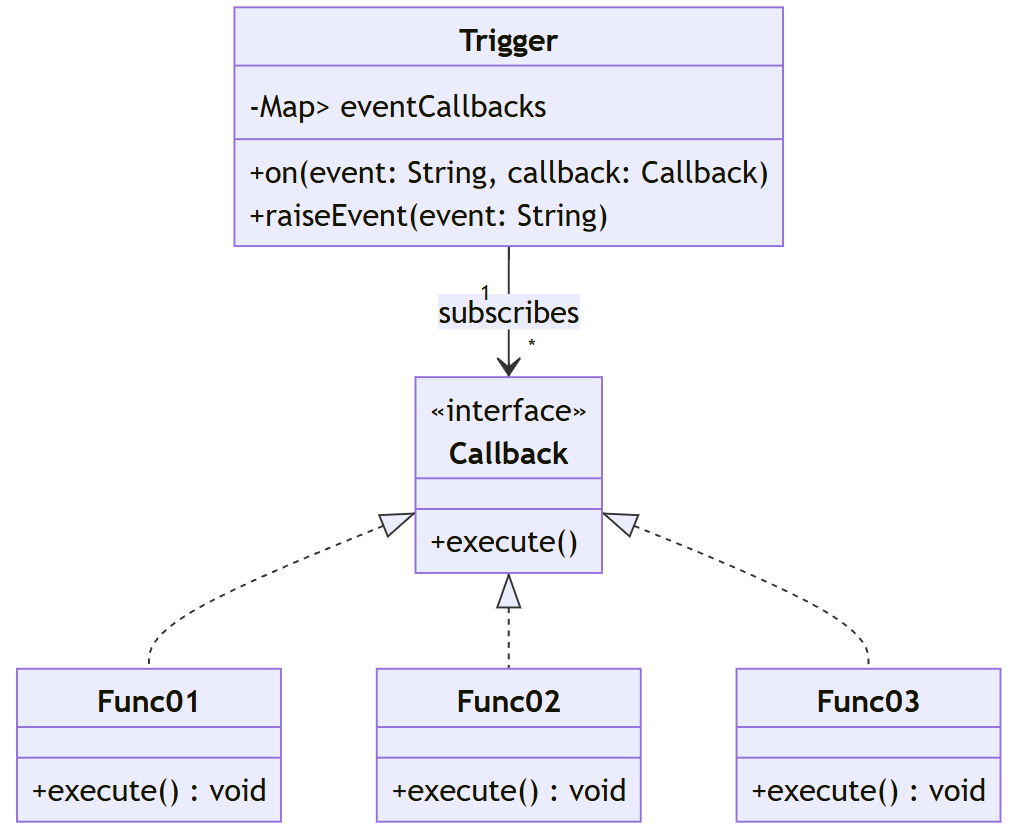
当前开发一个新语言、新中间件或程序框架时，基于充分利用多核CPU能力等方面的考虑，多采用事件循环处理模型+回调函数的设计方式，使用该模型的示例代码和预期输出详见下表。针对该模型，请写出你所选择的设计模式，画出类图，并给出Trigger类及相关类的设计和实现，使得输出能够符合预期结果。（为简便起见，假设所有事件处理函数的原型均为void xxxx(void);）



**模式：**

观察者模式

**类图：**



|  |
| --- |
| **代码** |
| #include <iostream>  #include <string>  #include <vector>  #include <map>  using namespace std;  *// 回调接口类*  class Callback {  public:      virtual void execute() = 0;      virtual ~Callback() {}  };  *// 具体回调函数类 func01*  class Func01 : public Callback {  public:      void execute() override {          cout << "func01()" << endl;      }  };  *// 具体回调函数类 func02*  class Func02 : public Callback {  public:      void execute() override {          cout << "func02()" << endl;      }  };  *// 具体回调函数类 func03*  class Func03 : public Callback {  public:      void execute() override {          cout << "func03()" << endl;      }  };  *// 触发器类*  class Trigger {  private:      map<string, vector<Callback\*>> eventCallbacks;  public:  *// 注册事件及对应的回调函数*      void on(const string& event, Callback\* callback) {          eventCallbacks[event].push\_back(callback);      }  *// 触发事件，调用所有注册的回调函数*      void raiseEvent(const string& event) {          cout << "Process " << event << " start" << endl;          if (eventCallbacks.find(event) != eventCallbacks.end()) {              for (Callback\* callback : eventCallbacks[event]) {                  if (callback) {                      callback->execute();                  }              }          }          cout << "Process " << event << " end" << endl;      }  };  *// 主函数测试代码*  int main() {      Trigger trigger;  *// 创建回调函数对象*      Func01 func01;      Func02 func02;      Func03 func03;  *// 注册事件和对应的回调函数*      trigger.on("event01", &func01);      trigger.on("event01", &func02);      trigger.on("event02", &func02);      trigger.on("event03", &func03);  *// 触发事件*      trigger.raiseEvent("event01");      trigger.raiseEvent("event02");      trigger.raiseEvent("event03");      trigger.raiseEvent("event01");      trigger.raiseEvent("event02");      return 0;  } |