事件机制实现套利

策略描述:
持有一份期权Q其市场价格为M,而利用BSM模型计算的理论价格为C
我们有

$$C(t,S(t))=S(t)N(d_1)-Ke^{-rT}N(d_2)$$

其中

$$d_1=\frac{\log\frac{S(t)}{K}+(r+\frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}$$

$$d_2=d_1-\sigma\sqrt{T}$$

$$N(x)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x}e^{-t^2/2}\mathrm{d}t$$

采用套利策略，如果此时市场价格M低于理论价格C，则我们可以考虑买入期权，如果市场价格M高于理论价格C，则我们可以考虑卖出期权；等待市场价格回归理论价格，再卖出或买入，从而赚取差价。
利用C++中的事件机制，可以写下伪代码如下

定义一个 Event 类，用于管理事件的监听者（也就是响应者）和触发事件。这个类有两个主要的方法：addListener

定义一个 EventResponsor 类，用于将事件和响应者的行为（也就是处理函数）进行配对。这个类在创建时会将处理函

定义一个 OptionsMarket 类，用于存储期权市场的价格数据。

定义一个函数 blackScholes，用于计算期权的理论价格。

定义一个函数 handlePriceDifference，用于处理市场价格和理论价格的差异。这个函数会根据价格差异给出买入或卖

在 main 函数中，首先计算期权的理论价格，然后创建一个 OptionsMarket 对象来存储市场价格和理论价格。

创建一个 Event 对象，用于管理价格事件。

创建一个 EventResponsor 对象，将价格事件和处理函数进行配对。这个处理函数会调用 handlePriceDifference

最后，模拟市场价格变化并触发价格事件。这会导致所有注册到这个事件的处理函数被调用，也就是调用 handlePrice

```cpp
#include <iostream>
#include <vector>
#include <functional>
#include <cmath>
#include <memory>

// Event 类模板：管理事件响应者对象，实现事件多播
template<typename... Args>
class Event {
public:
    using Callback = std::function<void(Args...)>;

    void addListener(const Callback& callback) {
        listeners.push_back(callback);
    }

    void trigger(Args... args) const {
        for (const auto& listener : listeners) {
            listener(args...);
        }
    }

private:
    std::vector<Callback> listeners;
};

// EventResponsor 类模板：响应者对象与响应者行为配对
template<typename... Args>
class EventResponsor {
public:
    EventResponsor(Event<Args...>& event, std::function<void(Args...)> handler)
        : event(event), handler(handler) {
        event.addListener(handler);
    }

private:
    Event<Args...>& event;
    std::function<void(Args...)> handler;
};

// Empty 类：用于指针转换和作为委托模型
class Empty {};
```

```cpp
template<typename T>
class EmptyWrapper : public Empty {
public:
    EmptyWrapper(std::shared_ptr<T> ptr) : ptr(ptr) {}

private:
    std::shared_ptr<T> ptr;
};

// OptionsMarket 类：用于存储期权市场的价格数据
class OptionsMarket {
public:
    OptionsMarket(double marketPrice, double theoreticalPrice)
        : marketPrice(marketPrice), theoreticalPrice(theoreticalPrice) {}

    double getMarketPrice() const { return marketPrice; }
    double getTheoreticalPrice() const { return theoreticalPrice; }

private:
    double marketPrice;
    double theoreticalPrice;
};

// Black-Scholes 期权定价计算
double blackScholes(double S, double K, double T, double r, double sigma, bool callOption) {
    double d1 = (std::log(S / K) + (r + 0.5 * sigma * sigma) * T) / (sigma * std::sqrt(T));
    double d2 = d1 - sigma * std::sqrt(T);

    double Nd1 = 0.5 * (1 + std::erf(d1 / std::sqrt(2)));
    double Nd2 = 0.5 * (1 + std::erf(d2 / std::sqrt(2)));

    if (callOption) {
        return S * Nd1 - K * std::exp(-r * T) * Nd2;
    } else {
        return K * std::exp(-r * T) * (1 - Nd2) - S * (1 - Nd1);
    }
}

// 处理价格差异
void handlePriceDifference(const OptionsMarket& market) {
    double marketPrice = market.getMarketPrice();
    double theoreticalPrice = market.getTheoreticalPrice();
    std::cout << "市场价格: " << marketPrice << std::endl;
```

```
    std::cout << "理论价格: " << theoreticalPrice << std::endl;
    if (marketPrice < theoreticalPrice) {
        std::cout << "期权被低估，建议买入期权" << std::endl;
        // 执行买入期权的代码
        // 执行对冲的代码
    } else if (marketPrice > theoreticalPrice) {
        std::cout << "期权被高估，建议卖出期权" << std::endl;
        // 执行卖出期权的代码
        // 执行对冲的代码
    } else {
        std::cout << "期权价格与理论价格相符" << std::endl;
    }
}

int main() {
    // 示例参数：标的资产价格(S)，行权价(K)，到期时间(T)，无风险利率(r)，波动率(sigma)
    double S = 100.0; // 标的资产价格
    double K = 100.0; // 行权价
    double T = 1.0;   // 到期时间（1年）
    double r = 0.05;  // 无风险利率（5%）
    double sigma = 0.2; // 波动率（20%）
    bool callOption = true; // 计算看涨期权价格

    double theoreticalPrice = blackScholes(S, K, T, r, sigma, callOption);

    // 创建市场数据
    OptionsMarket market(98.0, theoreticalPrice);  // 示例市场价格和计算出的理论价格

    // 创建事件
    Event<const OptionsMarket&> priceEvent;

    // 创建响应者对象并注册处理函数
    auto responder = std::make_shared<EmptyWrapper<OptionsMarket>>(std::make_shared<OptionsMarke
    EventResponsor<const OptionsMarket&> eventResponsor(priceEvent, [responder](const OptionsMar
        handlePriceDifference(m);
    });

    // 模拟市场价格变化并发射事件
    std::cout << "模拟市场价格检查: " << std::endl;
    priceEvent.trigger(market);

    return 0;
}
```