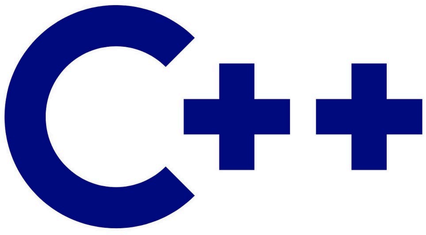
C++类循环依赖破解：前向声明与智能指针的妙用

作者：微信公众号：【架构师老卢】
12-11 15:34



概述： 在C++中，通过前向声明和指针或智能指针，可以解决类之间的循环依赖问题。这种技术确保了代码结构清晰，有效规避了生成错误。通过示例演示了如何使用这些方法处理类之间相互引用的情况，提升了代码的可维护性和可读性。

在C++中，类之间的循环依赖关系可能导致编译错误。为了解决这个问题，可以使用前向声明（Forward Declaration）和指针的方法。以下是详细的解释和示例。

基础功能：

示例源代码：

```
// 文件 A.h
#pragma once

#include "B.h"  // 包含对 B 类的引用

class B;  // 前向声明 B 类

class A {
public:
    A();
    void SetB(B* b);  // 使用 B 类的指针
    void DoSomething();
private:
    B* b_;  // 成员变量使用 B 类的指针
};
```

```
// 文件 B.h
#pragma once

#include "A.h"  // 包含对 A 类的引用

class A;  // 前向声明 A 类

class B {
public:
    B();
    void SetA(A* a);  // 使用 A 类的指针
    void DoSomething();
private:
    A* a_;  // 成员变量使用 A 类的指针
};
```

```
// 文件 A.cpp
#include "A.h"
#include "B.h"

A::A() : b_(nullptr) {}

void A::SetB(B* b) {
    b_ = b;
}

void A::DoSomething() {
    if (b_) {
        b_->DoSomething();
    }
}
```

```
// 文件 B.cpp
#include "B.h"
#include "A.h"

B::B() : a_(nullptr) {}

void B::SetA(A* a) {
    a_ = a;
}

void B::DoSomething() {
    if (a_) {
        a_->DoSomething();
    }
}
```

在这个示例中，A 类和 B 类相互引用，但通过前向声明和使用指针的方法，解决了循环依赖的问题。

高级功能：

示例源代码：

```
// 文件 A.h
#pragma once

#include <memory>

class B;  // 前向声明 B 类

class A {
public:
    A();
    void SetB(std::shared_ptr<B> b);  // 使用 B 类的智能指针
    void DoSomething();
private:
    std::shared_ptr<B> b_;  // 成员变量使用 B 类的智能指针
};
```

```
// 文件 B.h
#pragma once

#include <memory>

class A;  // 前向声明 A 类

class B {
public:
    B();
    void SetA(std::shared_ptr<A>

```cpp
// 文件 B.h
#pragma once

#include <memory>

class A;  // 前向声明 A 类

class B {
public:
    B();
    void SetA(std::shared_ptr<A> a);  // 使用 A 类的智能指针
    void DoSomething();
private:
    std::shared_ptr<A> a_;  // 成员变量使用 A 类的智能指针
};
```

```
// 文件 A.cpp
#include "A.h"
#include "B.h"

A::A() {}

void A::SetB(std::shared_ptr<B> b) {
    b_ = b;
}

void A::DoSomething() {
    if (b_) {
        b_->DoSomething();
    }
}
```

```
// 文件 B.cpp
#include "B.h"
#include "A.h"

B::B() {}

void B::SetA(std::shared_ptr<A> a) {
    a_ = a;
}

void B::DoSomething() {
    if (a_) {
        a_->DoSomething();
    }
}
```

在这个示例中，使用了 `std::shared_ptr` 作为成员变量，这是一种现代 C++ 中更安全、便利的智能指针。智能指针能够在对象不再需要时自动释放资源，避免了内存泄漏的风险。

通过这两个示例，展示了使用前向声明和指针（或智能指针）的方法来解决类之间循环依赖的问题。这种技术在大型项目中尤其有用，确保代码结构清晰而没有不必要的编译错误。
```