AlexOne

一、单例模式

目录

1. 单例模式

```
#include <iostream>
#include <string.h>
#include <fstream>

using namespace std;

// 懒汉模式
class Singletion{
public:
    static Singletion * getInstance(){
        if(_ins==NULL) _ins = new Singletion;
        return _ins;                                // 最后，要返回一个不为空的单例
    }
    int getData(){
        return data;
    }
private:                                            // 重写五大默认都私有化将会堵死所有的构造，成为真正的单例
    Singletion();
    ~Singletion();
    static Singletion * _ins;                       // 私有化数据通过公有化方法进行操作
    int data;
};

Singletion * Singletion::_ins=NULL;                 // 静态成员要在类外初始化。static的要放在cpp里面。

Singletion::Singletion(){                           // 构造的时候就装载数据，也能放在其它函数里，用的时候调用
    fstream fs;
    fs.open("aa.txt",ios::in|ios::out);
    if(!fs) cout<<"Error"<<endl;                    // !fs是重载了的
    else
        fs>>data;
    fs.close();
}

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

```
    ~A(){}
};
A::A(){
    Singletion * dt=Singletion::getInstance();
    cout<<dt->getData()<<endl;
}

int main(){
    A a;
    return 0;
}

// 测试用例：建立aa.txt，内容为一行IP一行port数字
```

2. 懒汉模式和饿汉模式

```
// Lazy 需要的时候再初始化，资源加载比较慢
class Singletion{
public:
    static Singletion * getInstance();
protected:
    Singletion(){}
private:
    static Singletion * p;
};
Singletion * Singletion::p = NULL;
Singletion * Singletion::getInstance(){
    if (p == NULL)
        p = new Singletion;                    // 在静态成员函数里获取资源
    return p;
}


// Hungry 生成实例即完成资源初始化
class Singletion{
public:
    static Singletion * getInstance();
protected:
    Singletion(){}
private:
    static Singletion * p;
};

Singletion * Singletion::p = new Singletion;    // 对象初始化的时候获取资源

Singletion * Singletion::getInstance(){
    return p;
}

// 异同：构造的时机不同。需要的时候构造、立即构造两种
```