常见的设计模式

1. 单例模式

只能创建一个实例的类，这种模式被成为单例模式。

将构造函数设为private，定义一个私有的静态指针变量指向类的唯一实例，通过一个类的公有静态方法获取这个实例。

class Singleton

{

private:

Singleton() {}

static Singleton \*Instance;

public:

static Singleton \* GetInstance()

{

if (Instance == NULL) //判断是否第一次调用

Instance = new Singleton();

return Instance;

}

};

Singleton \* Singleton::instance = NULL;

该方法是线程不安全的，考虑两个线程同时首次调用GetInstance方法且同时检测到Instance是NULL值，则两个线程会同时构造一个实例给Instance。

单例模式实现的两种方式：

**懒汉**：第一次用到类实例时才会去实例化。在访问量较少时采用。

加锁实现：

class Singleton

{

private:

Singleton() {

pthread\_mutex\_init(&mutex);

}

static Singleton \*Instance;

static pthread\_mutex\_t mutex;

public:

static Singleton \* GetInstance()

{

if (Instance == NULL) //判断是否第一次调用

{

pthread\_mutex\_lock(&mutex);

if (Instance == NULL)

Instance = new Singleton();

return Instance;

pthread\_mutex\_unlock(&mutex);

}

}

};

Singleton \* Singleton::instance = NULL;

内部静态变量实现：

class Singleton

{

private:

Singleton() {

pthread\_mutex\_init(&mutex);

}

static pthread\_mutex\_t mutex;

public:

static Singleton \* GetInstance()

{

pthread\_mutex\_lock(&mutex);

static Singleton Instance;

pthread\_mutex\_unlock(&mutex);

return & Instance;

}

};

**饿汉**: 在单例类定义的时候就进行实例化。访问量比较大，或者可能访问的线程比较多时，采用饿汉实现。

class Singleton

{

private:

Singleton() {}

static Singleton \*Instance;

public:

static Singleton \* GetInstance()

{

return Instance;

}

};

//在 GetInstance() 调用之前就已经实例化

Singleton\* Singleton::Instance = new Singleton;

饿汉式是线程安全的，在类创建的同时就已经创建好一个静态的对象供系统使用，以后不再改变。