抽象数据类型

2019年10月10日 15:37

【抽象的作用】

1. 隔离复杂度，缩小影响范围
2. 隐藏全局数据
3. 内部传参方便 -- C语言很繁琐
4. 控制中心/重构 -- 设计模式
5. 封装相关操作 -- 数学类

【抽象概念简介】

抽象：**层次性 - 将与实体无关方法移除类外。严格遵从is-a原则不要胡乱继承搞乱抽象的层次。**

封装：**将实现细节，数据成员隐藏起来，常用方法：放在private, protected下。**

语义：语义和语法上的封装一样重要。

面向过程语言抽象数据的方式【c语言】：

1. 传参
2. 内部绑定
3. 全局索引

【抽象注意事项】

1. 你有一个派生类，但这个派生类只有一个实例：派生类中的差异能否用数据而非一个新类去解决呢。
2. 只有一个派生类： 可能过度设计
3. **派生的过程中覆盖父类的函数，由于覆盖不当丢失了父类的特性**
4. 避免使用友元类
5. 类内部尽量少实列化其他对象，调用其他对像的内嵌对象函数（避免形成调用链引入更多依赖）
6. 避免只有行为没有数据的类 -- 考虑是否能转为其他类的数据属性
7. 避免只有数据没有行为的类 -- 考虑是否能转为其他类的行为
8. 避免万能的类

【不同语言的抽象类的差异】

1. 继承层次中被覆盖的构造和析构函数的行为
2. 在异常处理时构造函数和析构函数的行为
3. 默认构造函数的重要性
4. 析构函数或终结器调用的时机
5. 重载运算符的知识
6. 对象创建和销毁，内存处理的方式

【抽象的常用技巧】

包裹类：一个类伪装成某个抽象实体。

1. 使用private继承 【可能会破坏继承关系】
2. 使用组合

如何更好的隐藏细节？这里的细节指的是什么？ Effect c++ 34

1. **将具体的实现封装起来，为了做到灵活的替换可以用指针替代**

继承的内容分类：

1. 不能覆盖的函数 -- 普通成员函数
2. 可以覆盖的函数 -- virtual
3. 未提供实现可以覆盖的函数 -- virtual xx()=0;

如何使用 as a 和 has a 关系?

1. 复用实现 -- has a 【接口实现有自己控制 -- 指针/组合 -- 你自己可以换指向/成员】
2. 赋予接口 -- as a 【接口实现由基类控制 -- 一继承就有了基类的接口--不自在】
3. 共享数据使用has a; 共享行为使用 as a
4. Example:

```
class ArrayQueue: public Queue<T>  // is a [行为]
{
    public:
        int getSize();
        bool isEmpty();
        bool enqueue(const T&);
        T dequeue();
        T getFront();
    private:
        Array<T, N> m_data; // has a [数据]
};

 // 使用组合不要使用多继承的方式，更合理
 // class ArrayQueue: Array<T, N>, public Queue<T>  -- not so good
```