北京圣思园科技有限公司
http://www.shengsiyuan.com

主讲人：张龙

装饰模式(Decorator)

- 装饰模式又名包装（Wrapper）模式
- 装饰模式以对客户端透明的方式扩展对象的功能，是继承关系的一个替代方案
- 装饰模式以对客户透明的方式动态的给一个对象附加上更多的责任。换言之，客户端并不会觉得对象在装饰前和装饰后有什么不同。
- 装饰模式可以在不创造更多子类的情况下，将对象的功能加以扩展。

装饰模式(Decorator)

- 装饰模式**把客户端的调用委派到被装饰类**。装饰模式的关键在于这种扩展完全是透明的。
- 装饰模式是在不必改变原类文件和使用继承的情况下，动态的扩展一个对象的功能。它是通过创建一个包装对象，也就是装饰来包裹真实的对象。

装饰模式(Decorator)

- 装饰模式的角色：
  - 抽象构件角色（Component）：给出一个抽象接口，以规范准备接收附加责任的对象。
  - 具体构件角色（Concrete Component）：定义一个将要接收附加责任的类。
  - 装饰角色（Decorator）：持有一个构件（Component）对象的引用，并定义一个与抽象构件接口一致的接口
  - 具体装饰角色（Concrete Decorator）：负责给构件对象“贴上”附加的责任。

装饰模式(Decorator)

- 装饰模式的特点：
  - 装饰对象和真实对象有相同的接口。这样客户端对象就可以以和真实对象相同的方式和装饰对象交互。
  - 装饰对象包含一个真实对象的引用（reference）
  - 装饰对象接收所有来自客户端的请求。它把这些请求转发给真实的对象。
  - 装饰对象可以在转发这些请求以前或以后增加一些附加功能。这样就确保了在运行时，不用修改给定对象的结构就可以在外部增加附加的功能。在面向对象的设计中，通常是通过继承来实现对给定类的功能扩展。

装饰模式 VS 继承

- 装饰模式
  - 用来扩展特定对象的功能
  - 不需要子类
  - 动态
  - 运行时分配职责
  - 防止由于子类而导致的复杂和混乱
  - 更多的灵活性
  - 对于一个给定的对象，同时可能有不同的装饰对象，客户端可以通过它的需要选择合适的装饰对象发送消息。

装饰模式 VS 继承

- 继承
  - 用来扩展一类对象的功能
  - 需要子类
  - 静态
  - 编译时分派职责
  - 导致很多子类产生
  - 缺乏灵活性

装饰模式(Decorator)

- 实现自己的装饰模式