aWangz的专栏

个人资料

aWangz

访问： 71256次
积分： 700
等级：
排名： 千里之外

原创： 2篇 转载： 54篇
译文： 1篇 评论： 4条

文章分类

生活杂记 (0)
JAVA (40)
滴水藏海 (0)
Oracle (2)
JavaScript (7)
软件工具 (2)
UML (2)
W3C (8)
算法 (1)
Java Profile (0)

文章存档

2015年04月 (3)
2014年10月 (1)
2014年03月 (3)
2014年02月 (1)
2014年01月 (1)
展开

阅读排行

使用Spring CommonsM (27743)
Redis 缓存数据库 (9336)
【说说JSON和JSONP的 (5973)
十大排序算法总结 (4435)
对于Spring JoinPoint Pc (3349)
Spring3核心技术之Jdbc (1952)
Socket IO 三种方式 (1189)
jQuery ajax回调函数设置 (1142)
Spring + Hessian 实现轻 (699)
java BIO NIO AIO 理论篇 (563)

评论排行

jQuery ajax回调函数设置 (2)
Java ExecutorService (1)
对于Spring JoinPoint Pc (1)
初识EJB 2.0 (0)
Hibernate 执行步骤 (0)
Hibernate工作原理概述 (0)
UML中的几种关系(依赖， (0)
SQL基础：where,group (0)
java 集合框架详解 (0)
tomcat架构分析 (connec (0)

推荐文章

* 2016 年最受欢迎的编程语言是什么？
* Chromium扩展（Extension）的页面（Page）加载过程分析
* Android Studio 2.2 来啦
* 手把手教你做音乐播放器（二）技术原理与框架设计
* JVM 性能调优实战之：使用阿里开源工具 TProfiler 在海量业务代码中精确定位性能代码

UML中的几种关系(依赖，关联，泛化，实现)

标签： uml 设计模式

2012-10-10 10:56 460人阅读 评论(0) 收藏 举报

分类： UML（1）

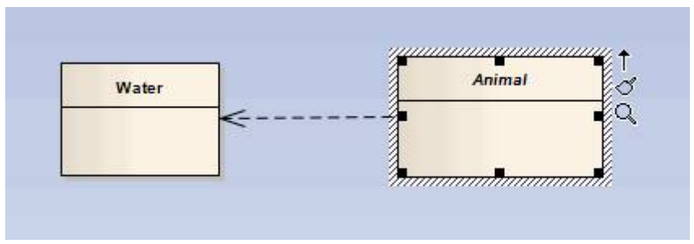
目录(?) [+]

关于设计模式的总结没有落实到代码上，而且设计模式自己确实动手实现的非常少。所以在这一周里，除了看网站开发的视频，着手开始对设计模式进行实现以下。设计模式非常经典，每次看都有不同的收获，写一下自己的收获吧，请大家指正。

在实现设计模式之前，首先来复习以下UML中的五种关系图

<1>依赖

依赖关系用虚线加箭头表示,如图所示:



上图表示:Animal类依赖于Water类（动物依赖于水）。

依赖是类的五种关系中耦合最小的一种关系。因为依赖关系在生成代码的时候，这两个关系类都不会增加属性。这种微弱的关系可以用类之间的相互了解的程度来说明。(下图为代码生成图)

```
public class Water {
    public Water(){
    }
}
```

```
public class Animal {
    public Animal() {
    }
}
```

由上图可见生成的代码中Animal类的属性中没有增加Water类。那么Animal类如何使用Water类呢，有三种方式：

依赖关系的三种表现形式：

① Water类是全局的，则Animal类可以调用它

② Water类是Animal类的某个方法中的变量，则Animal类可以调用它。代码演示如下：

```
public class Animal {

    public void GrownUp() {
        Water water=null ;
    }
}
```

PS：Animal有一个长大（GrownUp）方法，Water类作为该方法的变量来使用。

请注意Water类的生命期，它是当Animal类的GrounUp方法被调用的时候，才被实例化。

持有Water类的是Animal的一个方法而不是Animal类，这点是最重要的。

③ Water类是作为Animal类中某个方法的参数或者返回值时。代码演示如下

```
public class Animal {

    public Water  GrownUp(Water water) {
        return null;
    }
}
```

无用多说，Water类被Animal类的一个方法持有。生命期随着方法的执行结束而结束。

在依赖关系中，必须采用这三种方法之一。

<2>关联

关联是实线加箭头表示。表示类之间的耦合度比依赖要强。***has a***

例：水与气候是关联的，表示图如下

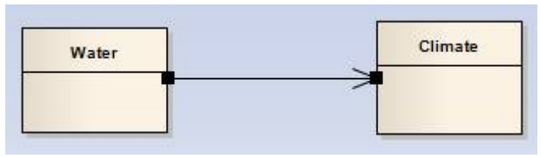
生成代码如下：

```
public class Water {
    public Climate m_Climate;


    public Water(){

    }
}
```

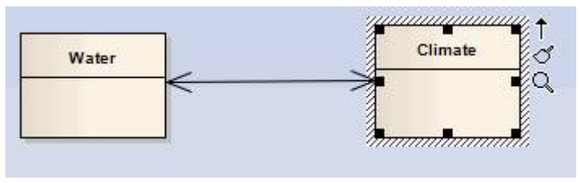
可见生成的代码中，Water类的属性中增加了Climate类。

关联既有单向关联又有双向关联。

单向关联：Water类和Climate类单向关联（如下图），则Water类称为源类，Climate类称为目标类。源类了解目标类的所有的属性和方法，但目标类并不了解源类的信息。



双向关联：源类和目标类相互了解彼此的信息。如将Water类和Climate类之间改为双向关联，如下图




```
public class Climate {
    public Water m_Water;

    public Climate(){

    }
}
```

```
public class Water {
    public Climate m_Climate;


    public Water(){

    }
}
```

依赖和关联的区别：

①　从类的属性是否增加的角度看：

发生依赖关系的两个类都不会增加属性。其中的一个类作为另一个类的方法的参数或者返回值，或者是某个方法的变量而已。

发生关联关系的两个类，其中的一个类成为另一个类的属性，而属性是一种更为紧密的耦合，更为长久的持有关系。

②　从关系的生命期角度看：

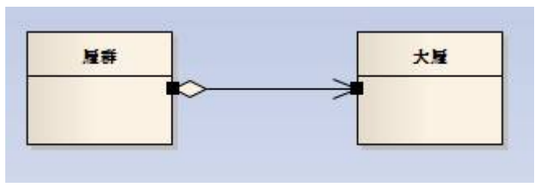
依赖关系是仅当类的方法被调用时而产生，伴随着方法的结束而结束了。

关联关系是当类实例化的时候即产生，当类销毁的时候，关系结束。相比依赖讲，关联关系的生存期更长。

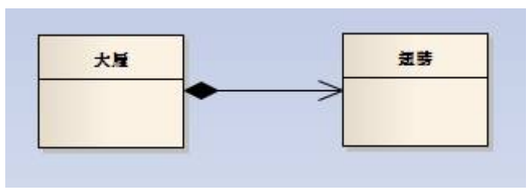
关联关系的细化

关联关系又可以细化为聚合关系和组合关系

聚合关系图：



组合关系图：



l　聚合和组合的区别：

由于聚合和组合都是特殊的关联关系，在生成的代码上看不出区别，都是关联的形式。那到底这两种关系如何来区分呢。

区分的关键有两点：

①　构造函数不同

聚合关系下：雁群类（GooseGroup）和大雁类（Goose）代码如下：

```
public class GooseGroup {

    public Goose goose;

    public GooseGroup(Goose goose){
        this.goose = goose;
    }
}
```

组合关系下：大雁类（Goose）和翅膀类（Wings）代码如下：

```
public class Goose {

    public Wings wings;

    public Goose(){
        wings = new Wings();
    }
}
```

这两种关系的区别在于：

①构造函数不同

聚合类的构造函数中包含了另一个类作为参数。

雁群类（GooseGroup）的构造函数中要用到大雁（Goose）作为参数传递进来。大雁类（Goose）可以脱离雁群类而独立存在。

组合类的构造函数中包含了另一个类的实例化。

表明大雁类在实例化之前，一定要先实例化翅膀类（Wings），这两个类紧密的耦合在一起，同生共灭。翅膀类（Wings）是不可以脱离大雁类（Goose）而独立存在

②　信息的封装性不同

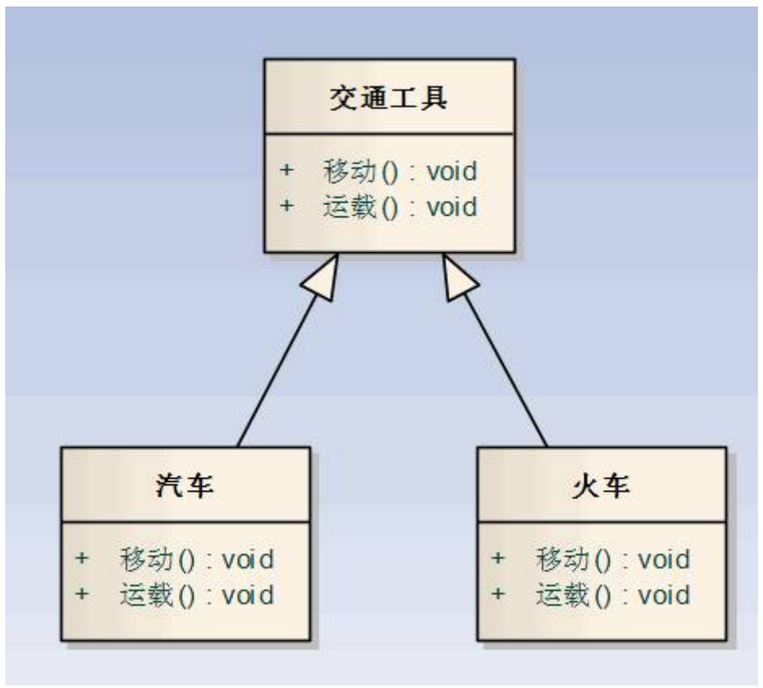
在聚合关系中，客户端可以同时了解雁群类和大雁类，因为他们都是独立的

而在组合关系中，客户端只认识大雁类，根本就不知道翅膀类的存在，因为翅膀类被严密的封装在大雁类中。

<3>泛化

泛化就是一个类继承另一个类所有的描述，并且可以根据需要对父类进行拓展，是面向对象的重要特征之一。

泛化使用一根实线加箭头，泛化关系图 *is a*



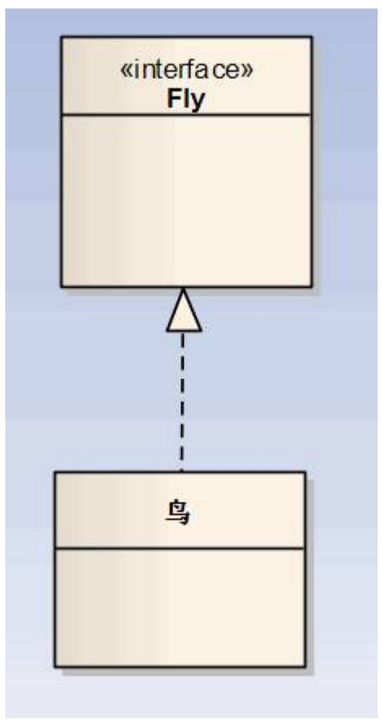
泛化的用处：①实现了代码的复用

②实现了多态

<4>实现

主要针对接口和抽象类而言，实现接口和抽象类的类必须要实现他们的方法。

实现关系表示为：虚线加箭头，关系图如下：



接口只包含方法、委托或事件的签名。方法的实现是在实现接口的类中完成的。

关于泛化关系和实现关系比较简单，这里就不一一展开了。了解清楚了这几种关系以及他们的代码特点，才能更好的学习设计模式！