# UML常见类图关系

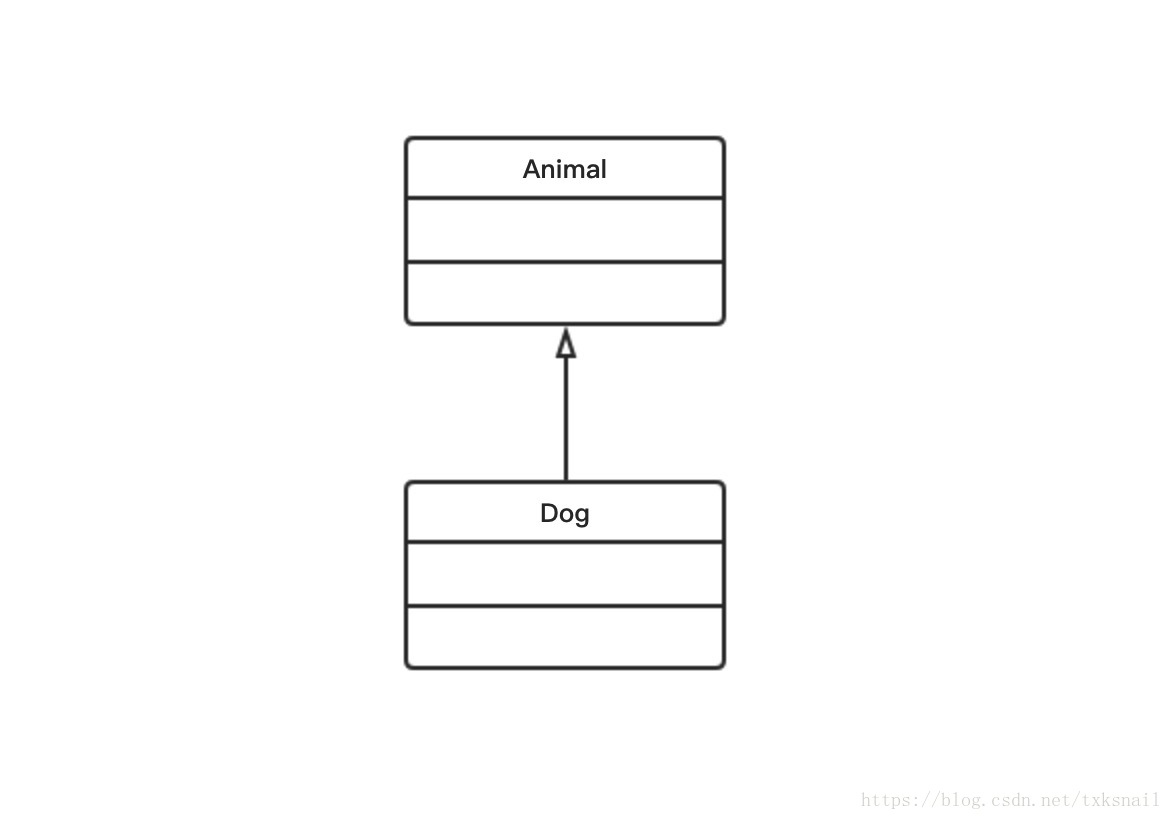
# 本文链接：<https://blog.csdn.net/txksnail/article/details/81609292>

### **文章目录**

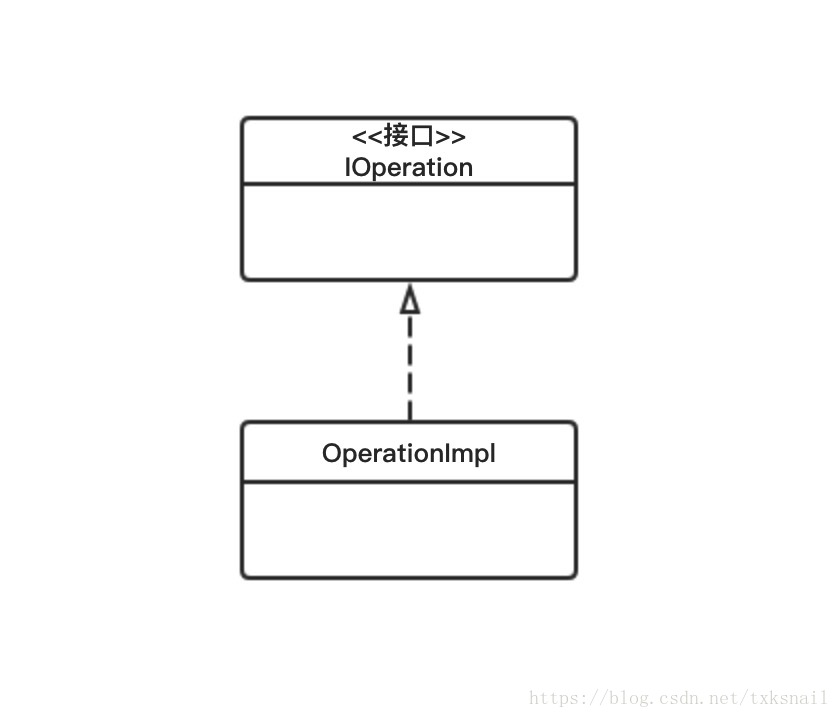
* [1、UML类图关系](https://blog.csdn.net/txksnail/article/details/81609292" \l "1UML_3" \t "https://blog.csdn.net/txksnail/article/details/_self)
  + [1.1、泛化](https://blog.csdn.net/txksnail/article/details/81609292" \l "11_4" \t "https://blog.csdn.net/txksnail/article/details/_self)
  + [1.2、实现](https://blog.csdn.net/txksnail/article/details/81609292" \l "12_10" \t "https://blog.csdn.net/txksnail/article/details/_self)
  + [1.3、关联](https://blog.csdn.net/txksnail/article/details/81609292" \l "13_16" \t "https://blog.csdn.net/txksnail/article/details/_self)
  + [1.4、聚合](https://blog.csdn.net/txksnail/article/details/81609292" \l "14_22" \t "https://blog.csdn.net/txksnail/article/details/_self)
  + [1.5、组合](https://blog.csdn.net/txksnail/article/details/81609292" \l "15_28" \t "https://blog.csdn.net/txksnail/article/details/_self)
  + [1.6、依赖](https://blog.csdn.net/txksnail/article/details/81609292" \l "16_34" \t "https://blog.csdn.net/txksnail/article/details/_self)
* [2、UML类图总结](https://blog.csdn.net/txksnail/article/details/81609292" \l "2UML_39" \t "https://blog.csdn.net/txksnail/article/details/_self)
* [3、UML类图绘制工具](https://blog.csdn.net/txksnail/article/details/81609292" \l "3UML_42" \t "https://blog.csdn.net/txksnail/article/details/_self)
  + [3.1、[ProcessOn](https://www.processon.com/)](https://blog.csdn.net/txksnail/article/details/81609292" \l "31ProcessOnhttpswwwprocessoncom_43" \t "https://blog.csdn.net/txksnail/article/details/_self)
  + [3.2、[StarUML](http://staruml.io/)](https://blog.csdn.net/txksnail/article/details/81609292" \l "32StarUMLhttpstarumlio_47" \t "https://blog.csdn.net/txksnail/article/details/_self)

# **1、UML类图关系**

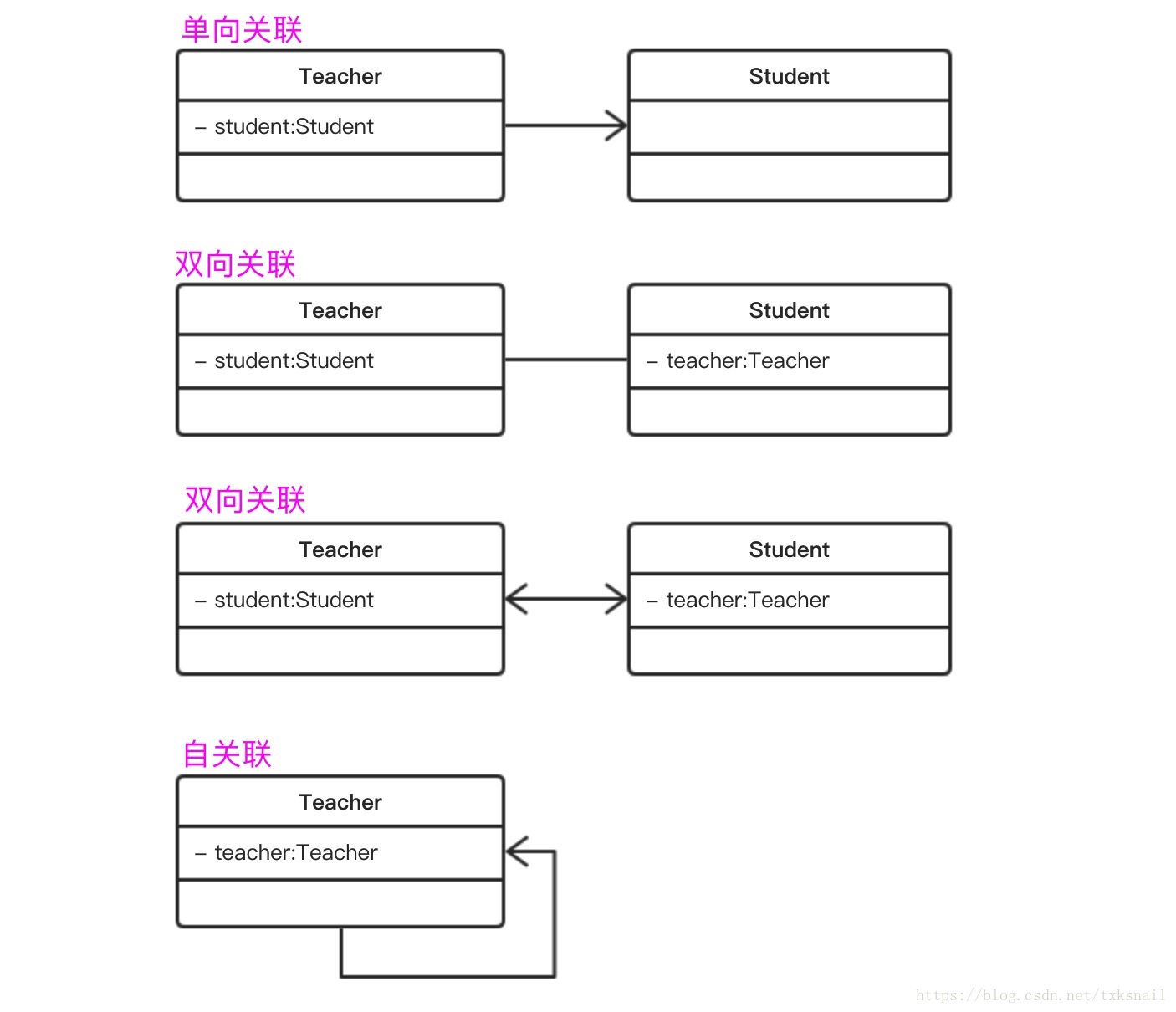
## **1.1、泛化**

【泛化】表示是一般与特殊的关系，是is a 的关系，表示的是一种****继承****关系。  
【比如】狗和动物，狗属于动物，狗和动物是一种继承关系，是泛化关系  
【代码】java中 A extends B,A和B就是泛化关系  
【UML画法示例】（带三角箭头的实线，箭头指向父类）  


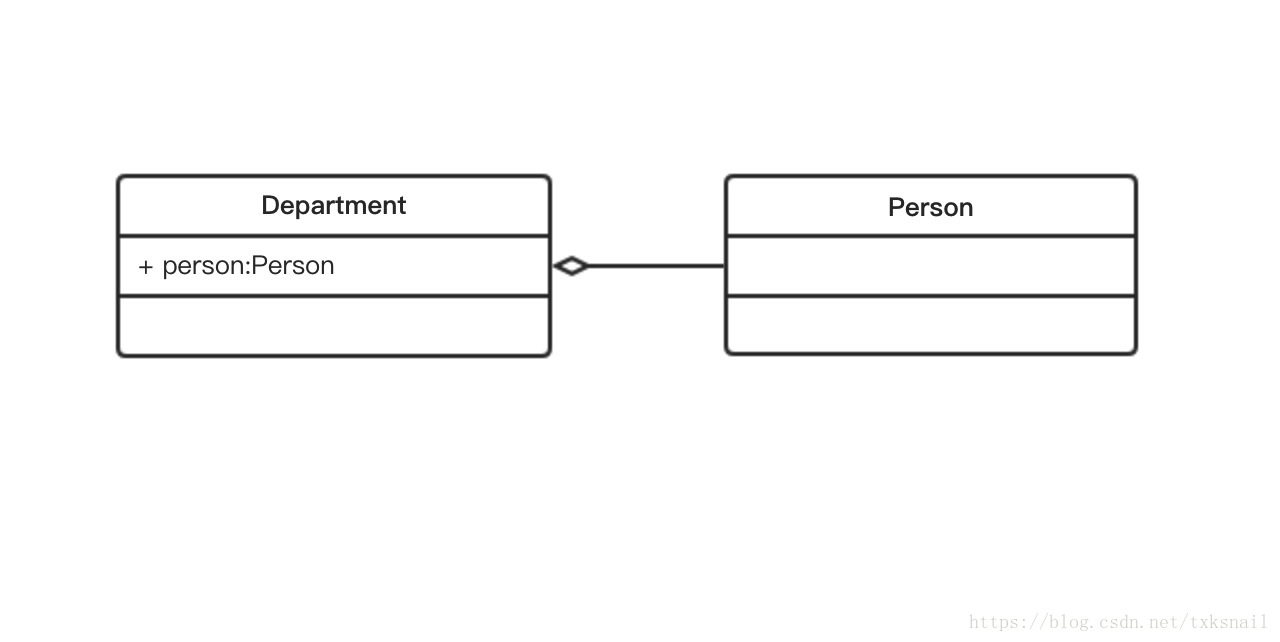
## **1.2、实现**

【实现】是一种类与接口的关系，表示类是接口特征和行为的实现  
【比如】通常我们编写代码的时候都会先写一个接口，然后实现类去实现接口。实现类和接口之间是一种实现关系。  
【代码】java中 A implement B ,A和B是泛化关系  
【UML画法示例】（带三角箭头的虚线，箭头指向接口）  


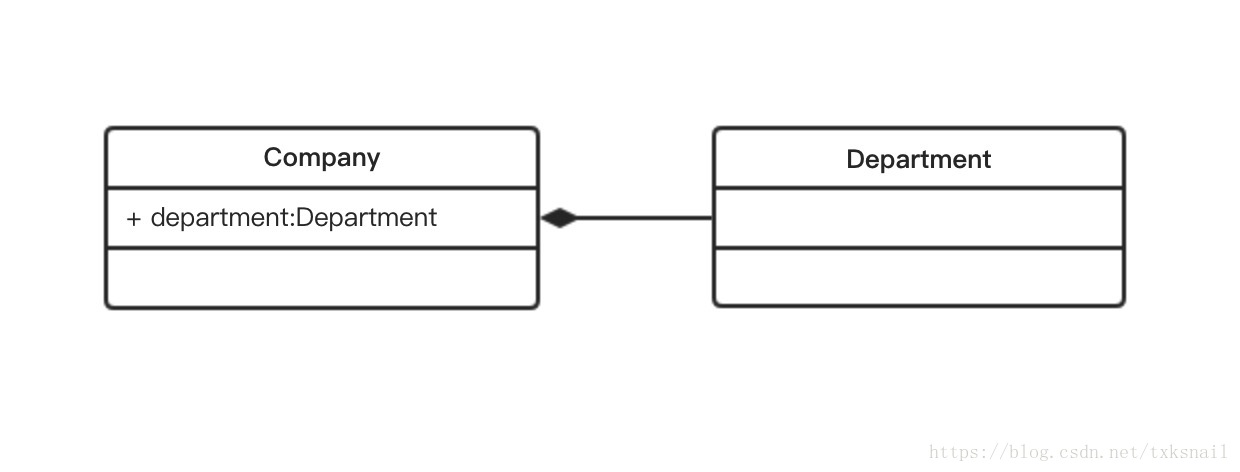
## **1.3、关联**

【关联】它描述了不同类的对象之间的结构关系，使一个类知道另一个类的属性和行为。关联关系有单向关联、双向关联、自关联。  
【比如】老师和学生，老师知道班里的学生信息，学生不知道老师信息，老师和学生是一种单向关联关系；老师知道学生信息，学生也知道老师的信息，老师和学生是双向关联关系；老师知道同行老师的信息，老师和老师是自关联关系。  
【代码】成员变量；类A持有类B的引用作为成员变量，类A和B就是关联关系，箭头指向B  
【UML画法示例】  


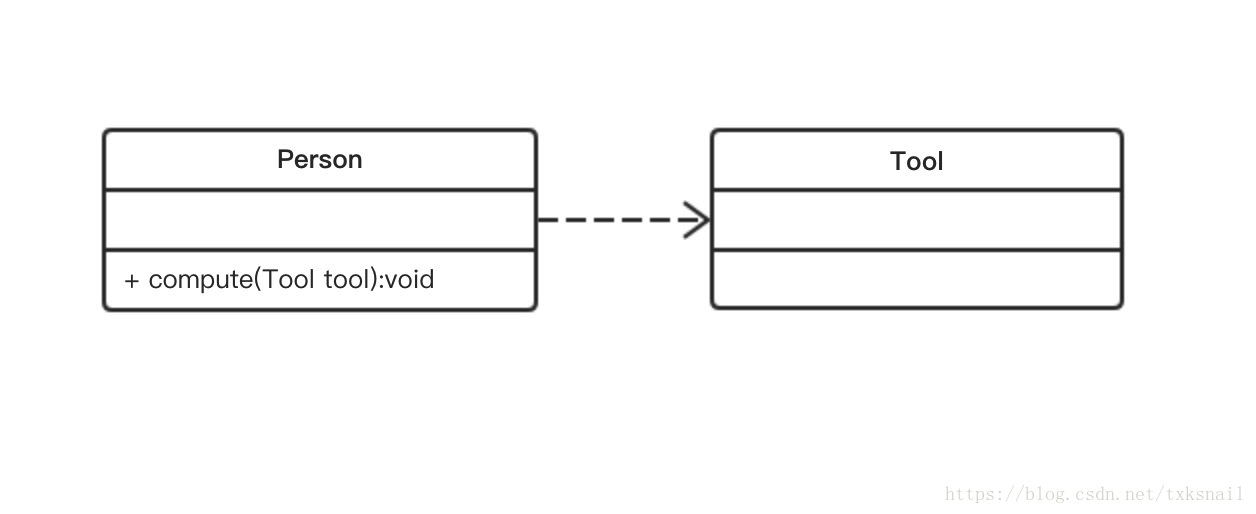
## **1.4、聚合**

【聚合】聚合是一种特殊的关联关系，是一个整体和部分的关系，部分可以离开整体而单独存在。聚合是一种语义关系，需要分析逻辑关系。  
【比如】一个部门和人员的关系，部门不存在了，人员照样可以单独存在。部门和人员在语义上就是聚合关系  
【代码】成员变量；比如部门类(Department)持有人员类（Person）作为成员变量，部门类和人员类是聚合关系  
【UML画法示例】（空心菱形指向整体）  


## **1.5、组合**

【组合】组合是一种特殊的关联关系，是一个整体和部分的关系，但是部分不能离开整体单独存在。组合和聚合一样，也是一种语义关系，需要分析逻辑关系  
【比如】公司和部门的关系，公司都不存在了，部门自然也不存在了。所以公司和部门就是一个组合关系  
【代码】成员变量；公司类（Company）持有部门类(Department)作为成员变量，公司类和部门类是组合关系  
【UML画法示例】（实心菱形指向整体）  


## **1.6、依赖**

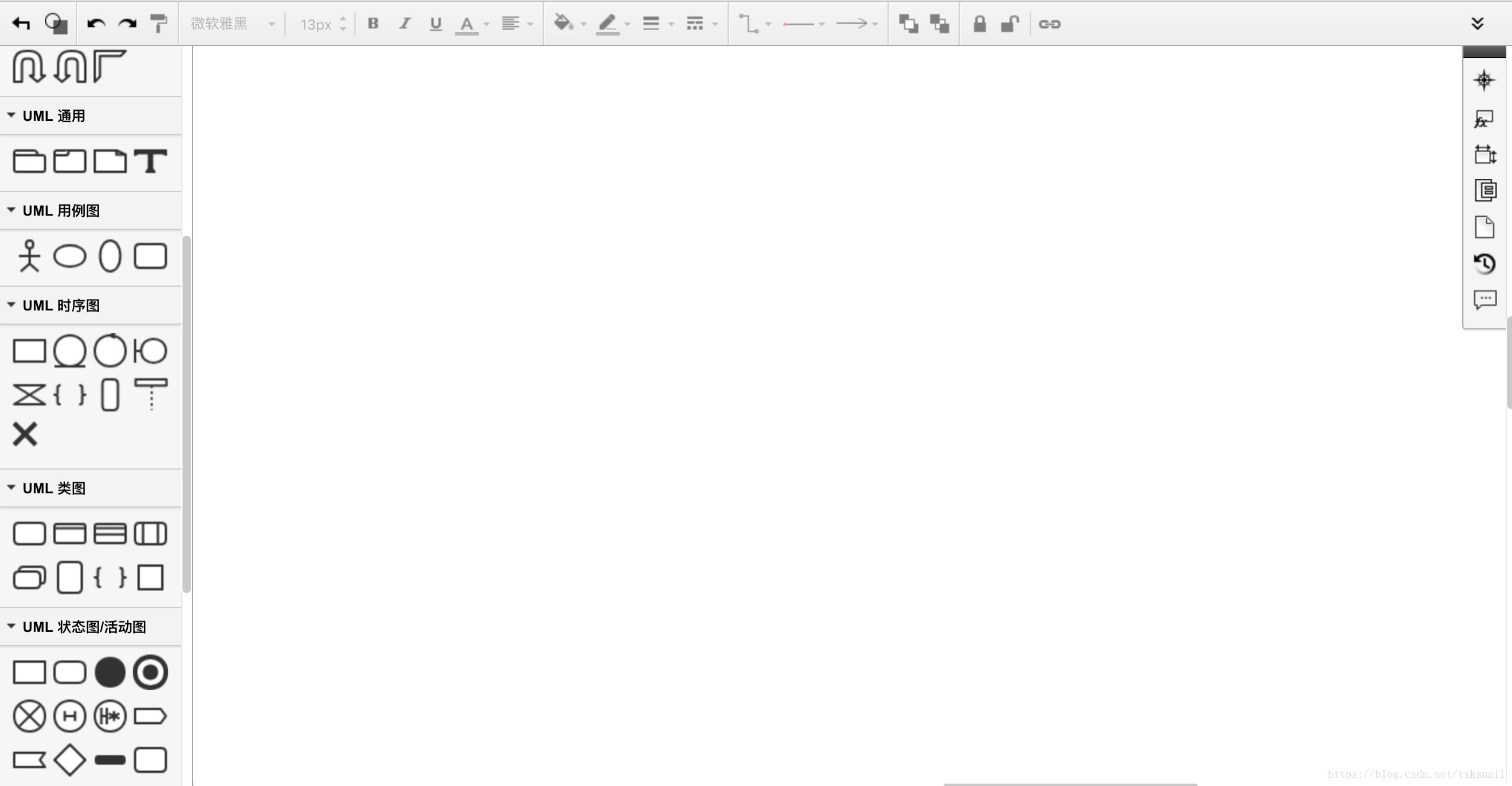
【依赖】一个类A使用到了另一个类B，而这种使用关系是具有偶然性、临时性、非常弱的，但是B类的变化会影响到A,是一种使用关系  
【代码】局部变量、方法的参数  
【UML画法示例】（一条虚线，箭头执行被依赖的类）  


# **2、UML类图总结**

* 聚合和组合是特殊的关联关系
* 关系强弱顺序：泛化=实现>组合>聚合>关联>依赖

# **3、UML类图绘制工具**

## **3.1、[ProcessOn](https://www.processon.com/)**

非常强大的在线工具，可以画流程图、UML、原型图等，完全免费。  


## **3.2、[StarUML](http://staruml.io/)**

UML绘制的客户端软件，Mac、Windows、Linux都有，可一直免费试用。  
