PowerDesigner16 用例图使用说明

用例图主要用来描述角色以及角色与用例之间的连接关系。说明的是谁要使用系统，以及他们使用该系统可以做些什么。一个用例图包含了多个模型元素，如系统、参与者和用例，并且显示这些元素之间的各种关系，如泛化、关联和依赖。它展示了一个外部用户能够观察到的系统功能模型图。

用例图中包含 6 个元素，分别是执行者（Actor），用例（Use Case），关联关系（Association），包含关系(Include)，扩展关系(Extend)以及泛化关系(Generalization)。

- 角色（Actor）：即使用本系统的有哪些角色，不同的角色使用的系统功能部分是不同的，在用例图中用小人  表示。其中，角色可能是人，也可能不是人，而是另外的一个系统，本系统与另外一个系统交互的话，可以将另外一个系统画成某某角色。

  分析得到角色的原则，也可以看做是我们在获得角色时，需要思考的内容：

  1）有哪些直接使用系统的人

  2）涉及到哪些维护人员

  3）使用哪些外设

  4）相连的其他系统

  5）还有哪些人和事物对这个系统产生的结果感兴趣。

- 用例（Use Case）：即系统具有的功能，在用例图中用椭圆圈  表示，圈里用文字描述该用例，一般为动宾短语。

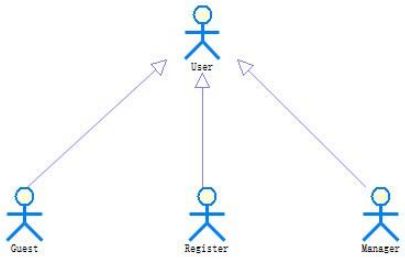
  其中，某个用例不一定是只属于一个角色的，有些用例是同时属于多个角色的，即被多个角色“共享”。

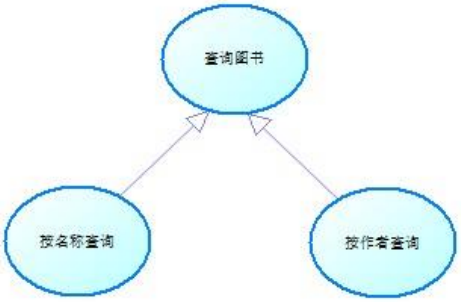
  - 关系：用例图中涉及的关系有：关联、泛化、包含、扩展。

  1）关联（Association）：表示参与者与用例之间的通信，任何一方都可发送或接受消息。【箭头指向】：无箭头或者 Actor 指向 Use Case，将参与者与用例相连接，指向消息接收方 。图标 

或

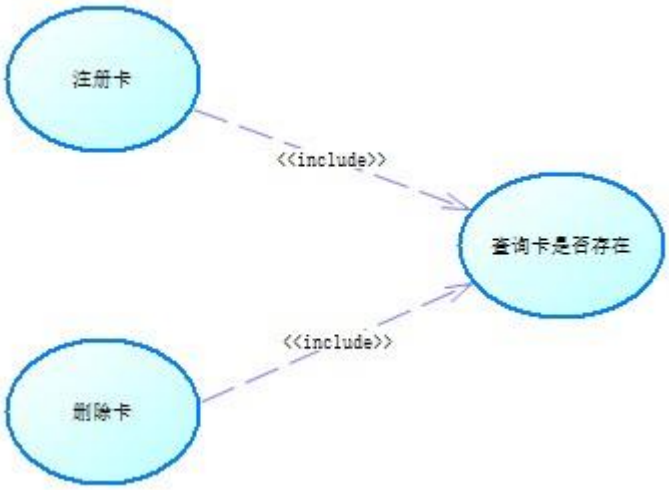
2）泛化（Generalization）：就是通常理解的继承关系，子用例和父用例相似，但表现出更特别的行为；子用例将继承父用例的所有结构、行为和关系。子用例可以使用父用例的一段行为，也可以重载它。父用例通常是抽象的。在实际应用中很少使用泛化关系，子用例中的特殊行为都可以作为父用例中的备选流存在。【箭头指向】：子参与者指向父参与者或者子用例指向父用例



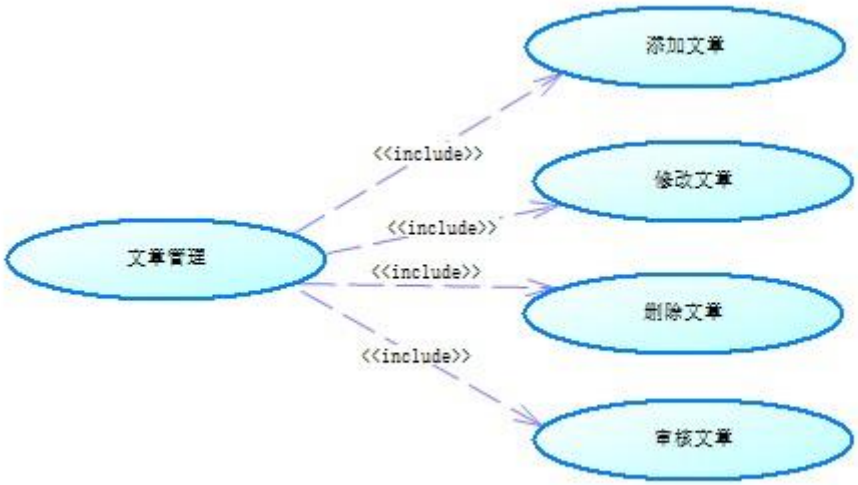


或

3) 包含（Include）：是指用例中的包含关系，通常发生在多个用例中，有可以提取出来的公共部分以便基用例复用.当两个或多个用例中共用一组相同的动作，这时可以将这组相同的动作抽出来作为一个独立的子用例，供多个基用例所共享。因为子用例被抽出，基用例并非一个完整的用例，所以 include 关系中的基用例必须和子用例一起使用才够完整，子用例也必然被执行。include 关系在用例图中使用带箭头的虚线(Link/Traceability Link)  表示(在线上标注<<include>>)，箭头从基用例指向子用例。

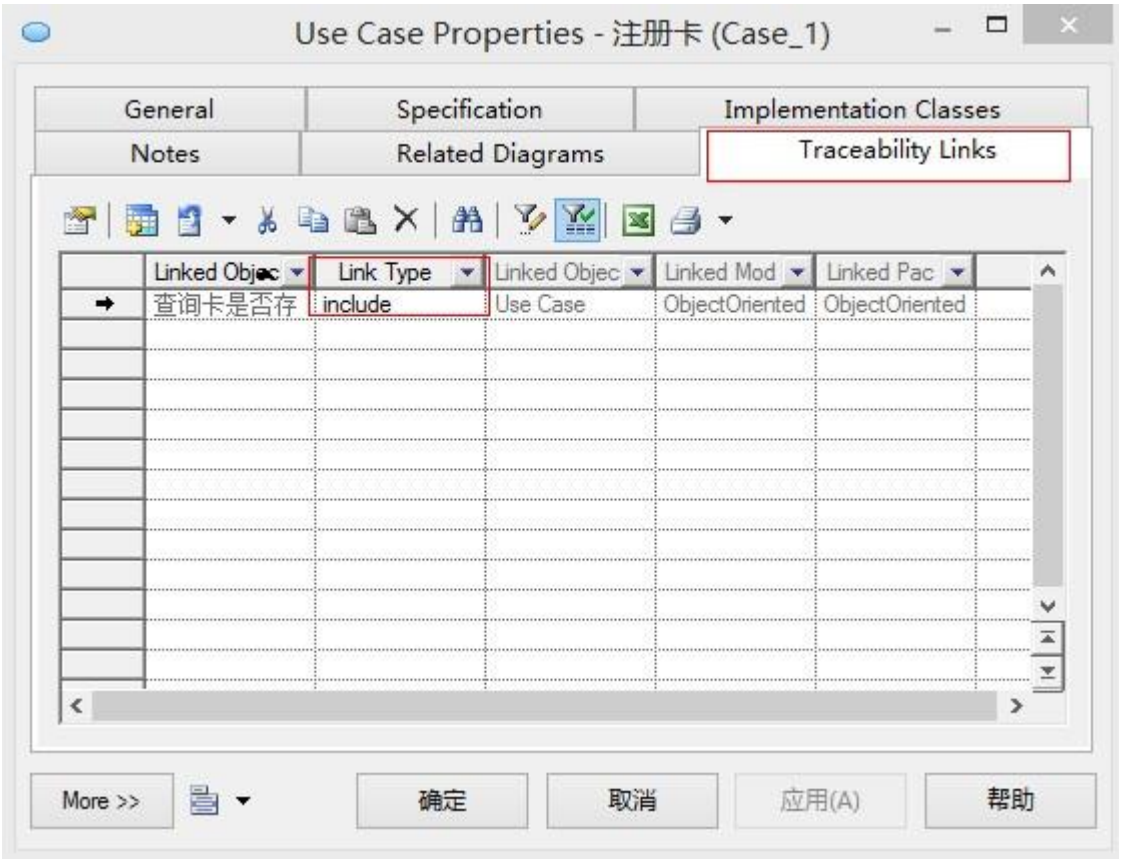


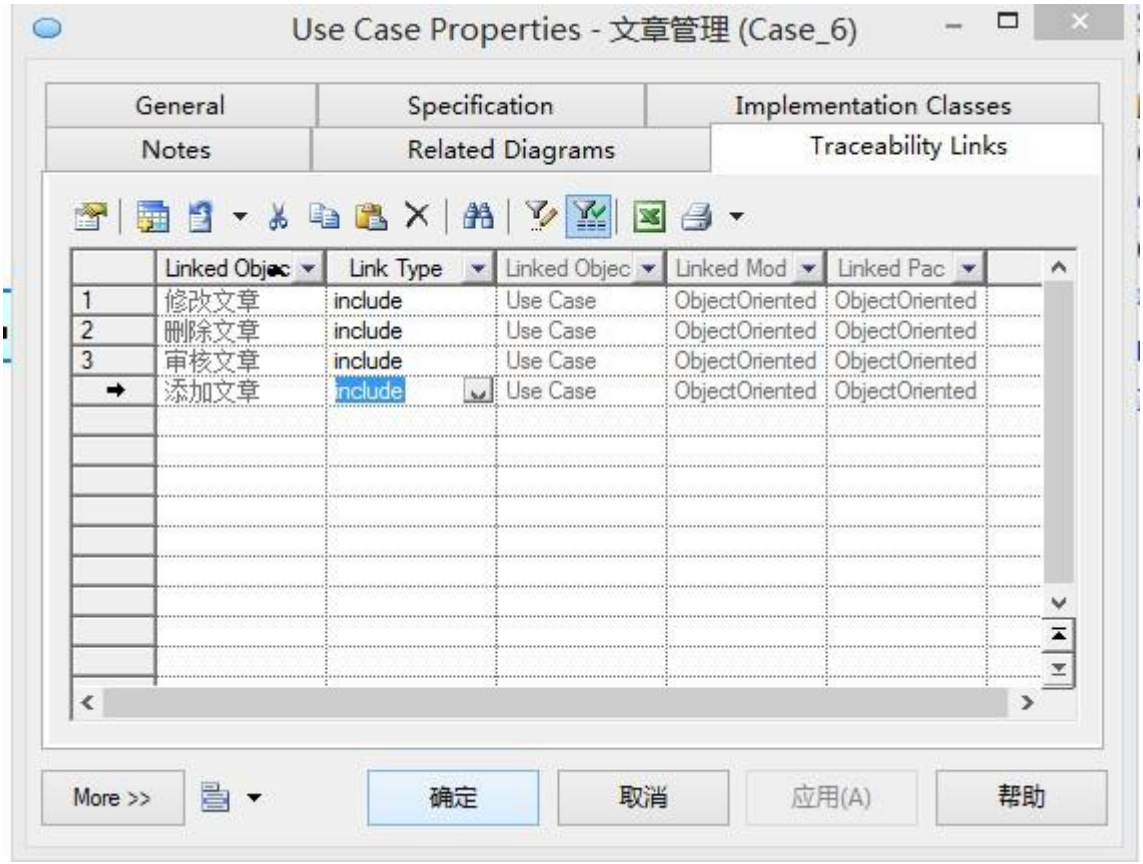
注册卡和删除卡之前都必须检验卡是否存在，注册卡和删除卡着两个用例并不完整，必须和查询卡是否存在这个子用例一起才能完成它的功能。



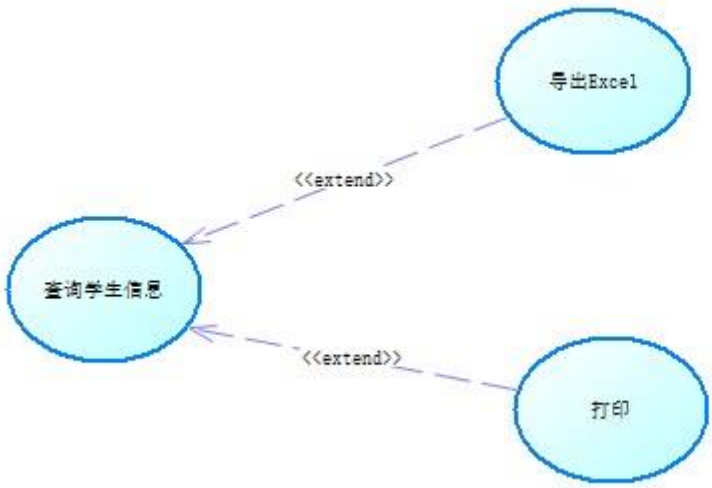
文章管理包含添加文章，修改文章，删除文章，审核文章才是一个完整的功能。

include 的表示方法见如下图所示;：

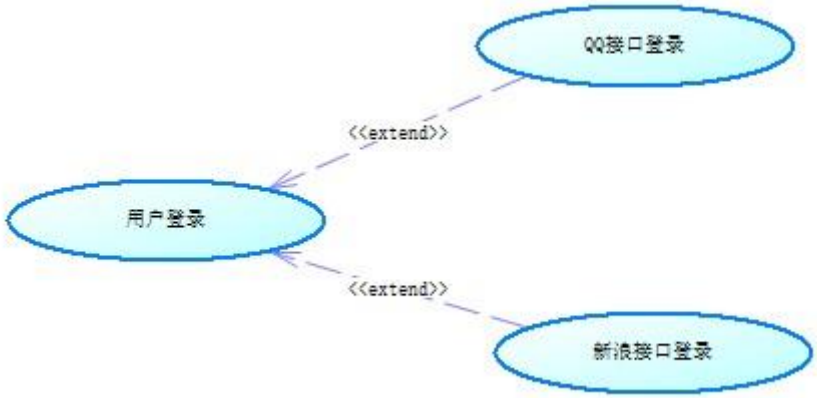




4) 扩展关系(Extend)：extend 关系是对基用例的扩展，基用例是一个完整的用例，即使没有子用例的参与，也可以完成一个完整的功能。extend 的基用例中将存在一个扩展点，只有当扩展点被激活时，子用例才会被执行。 extend 关系在用例图中使用带箭头的虚线(Link/Traceability Link)  *表示(在线上标注<<extend>>)，箭头从子用例指向基用例。*

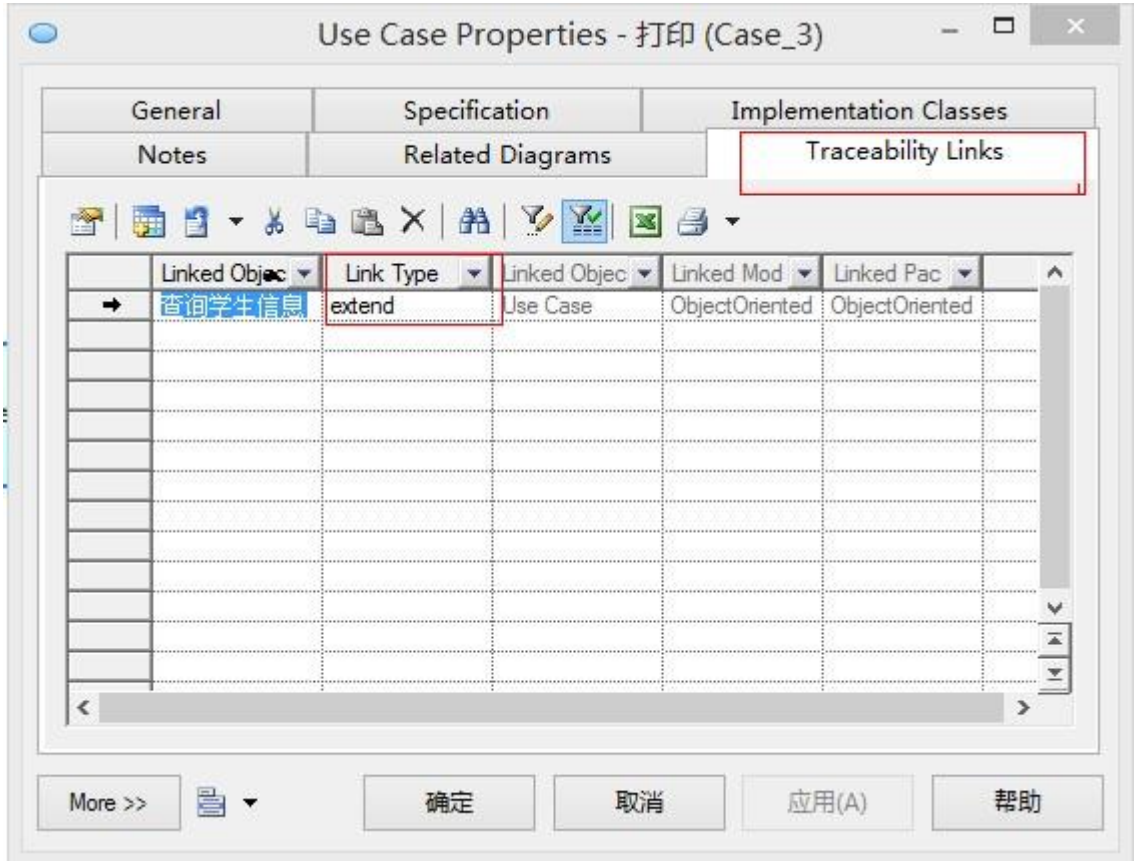


查询学生信息可以独立完成，不需要子用例的参与。只有点击导出为 excel 按钮或打印按钮时才会执行相应的动作。

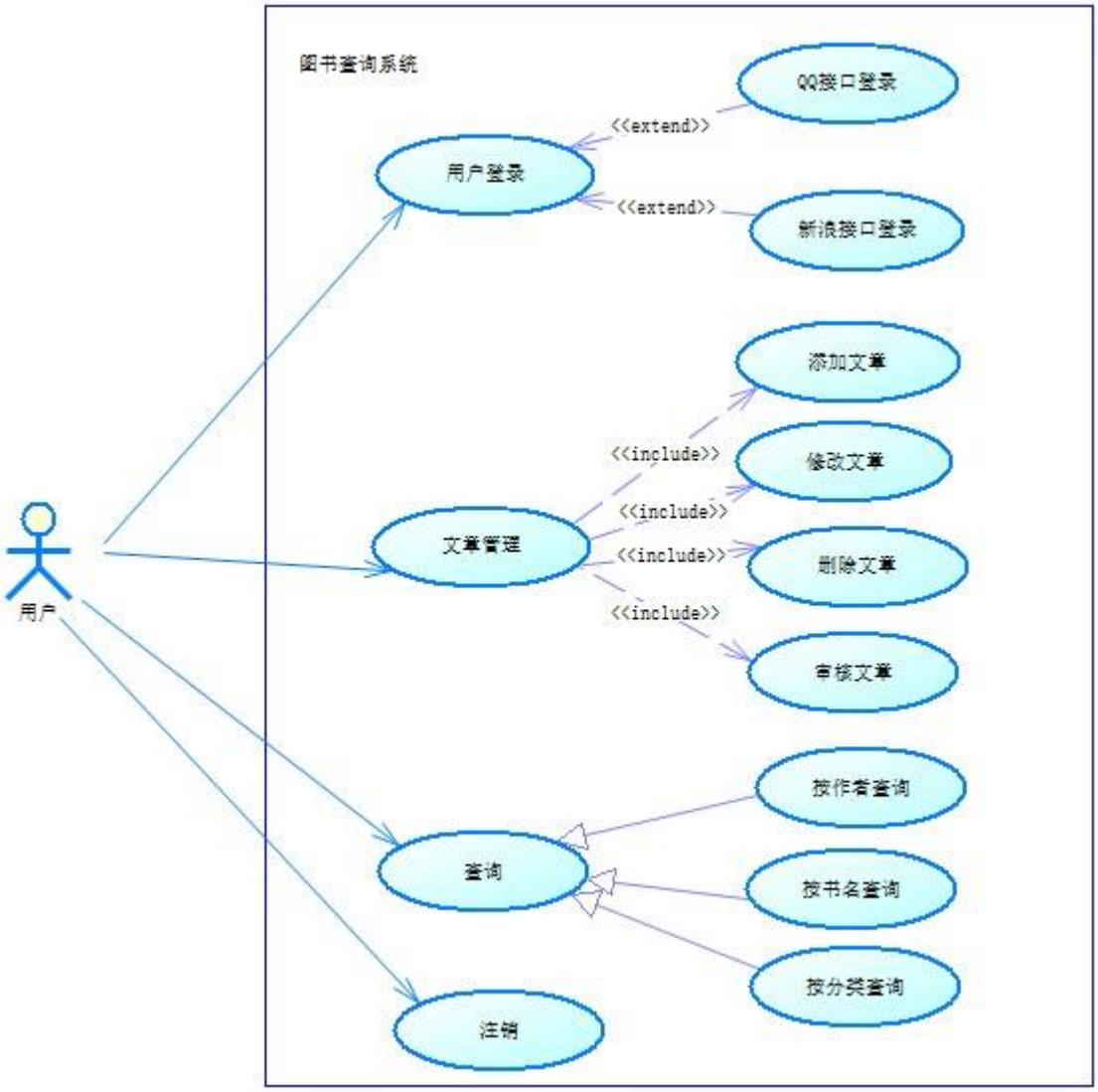


用户登录可以独立完成，不需要子用例的参与。只有点击 QQ 登录或新浪帐号登录才会执行相应的动作。

extend 的表示方法见如下图所示;：



- *一个完整的用例图实例*



- 附：UML 用例 UseCase 的几个理解误区

误区 1：用例就是功能点

这是一个很大的误区，也是技术人员容易犯的一个错误。功能点是站在软件开发的角度来说的，而用例是站在用户的角度来说的。获取用例是领域专家干的活，而最后的功能实现是技术专家干的活，不同的角色。所以获取用例的关键就是要站在用户角度看问题。

怎么获得用例？首先确定位于系统边界之外的主角是谁？他的期望和目的是什么？这个期望和回报要求在系统之内。所以，用例是帮助确定系统边界的一个好方法。用例也是获取需求的一个方法。

误区 2：用例和步骤混淆

举例来说，用户输入密码，要有密码错误提示，并且三次错误自动锁定用户，最后登录成功。“输入密码”是一个步骤，不是用例。整个过程是一个用例：“用户登录”。中间步骤和场景可以有很多。比如输入密码是一个步骤；“要有密码错误提示”这是一个业务需求，不是用例；“并且三次错误自动锁定用户”这是一个业务需求，也不是用

例。

用例的特征：有目的，有用户期望，有回报预期。当结果不可定义或不清晰时不能用 Use Cases，意思是如果目标成功或目标失败不能有一个明确的定义，那就不是一个用例。举例来说，用户输入密码，这是不是一个用例？用户输入密码的目的是什么？是为了输入密码吗？不是的，是为了登录系统，所以，用户登录是一个用例。

误区 3：用例的粒度不明

用例的粒度大小要看情况，因地制宜，因时制宜。

因地制宜：一般系统用例 10-50 个为宜。比较小型系统可以粒度更小一些。

因时制宜：在业务建模阶段，在概念建模阶段，在系统建模阶段都是不同的。在系统建模阶段，用例的粒度是以每个用例能够描述操作者与计算机的一次完整的交互为宜。根据项目的不同阶段，不断缩小边界可以获得更小的粒度用例。一个大的用例还可以 include 一个小的用例，比如网上下订单是一个用例，修改订单是一个子用例，因为除了用户，管理员可以修改订单，这个子用例有意义。

误区 4：用例和场景混淆

一个用例的执行是要有前因和后果的（前提是什么，结果会怎么样）；比如，煮饭和炒菜是用例，他们各自都有步骤，各自有好几个场景。比如煮饭，我可以用电饭锅煮，也可以蒸饭，煮饭前要先淘米，等等，这些都是一个用例的不同场景，但用户的最终目的都是一样的。不要把用例和场景混淆。

误区 5：软件工程是不是用例驱动？

软件工程是不是用例驱动？需求是重要的，用例是构造需求的好方法。但如果你同时要考虑开发的所有因素包括重用，架构，花费，时间，你就无法仅仅从一个方面来驱动你的项目。好的软件工程是被一系列重要因素所驱动的，而且因素也因不同的公司和项目有着不同的重要程度。这些因素包括： 技术上对于设计的考虑，用户需求，重用，可更改性，系统性能，标准化，日程的安排以及其他的商务驱动。每个项目都有着自己不同的考虑。对于每一种情况，可以精确的说项目被域模型和用例共同驱动。

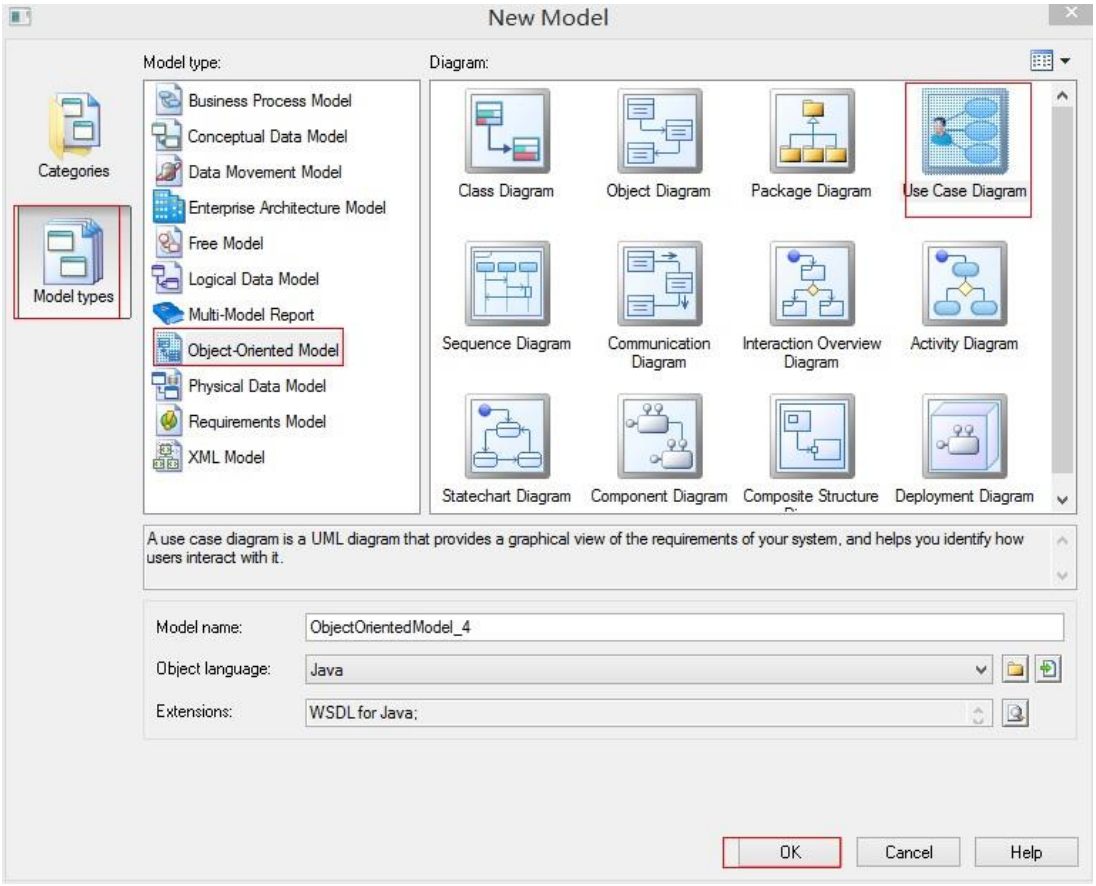
误区 6：用例直接推导出设计

不要从用例直接推论出设计。如果这样做，“用例开发”仅仅成为了功能分解的一个借口。用例止于系统接口的边界！用例应该描述参与者使用系统时所遵循的次序，但用例决不说明系统内部采用什么步骤来响应参与者的刺激。

用例是帮助确定系统边界的一个好方法。用例也是获取需求的一个方法。用例也是产生测试用例的好方法。但是，从系统边界、需求、到详细设计还有很长的路要走。比如说，类图，事实上类图和用例图没有对应关系。换句话来说，用例是需求分析时的产物，类（边界类外）的设计期的产物。

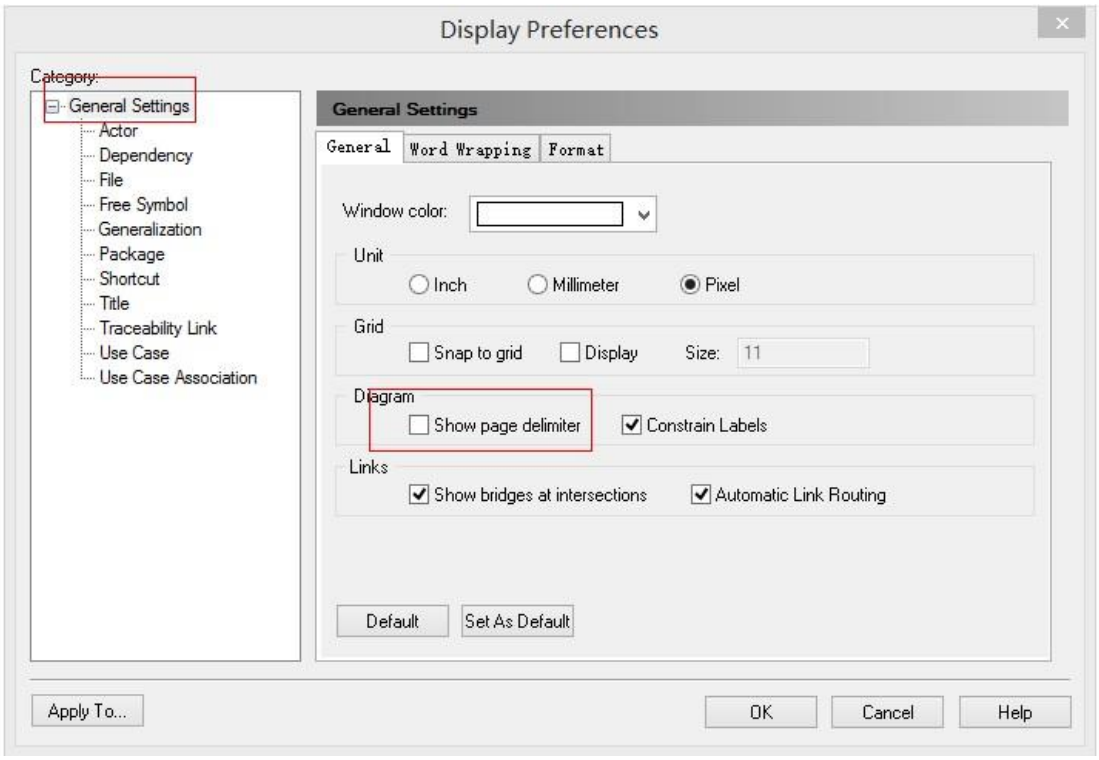
下面是菜单说明：

一：新建一个用例的模型 File->new Model

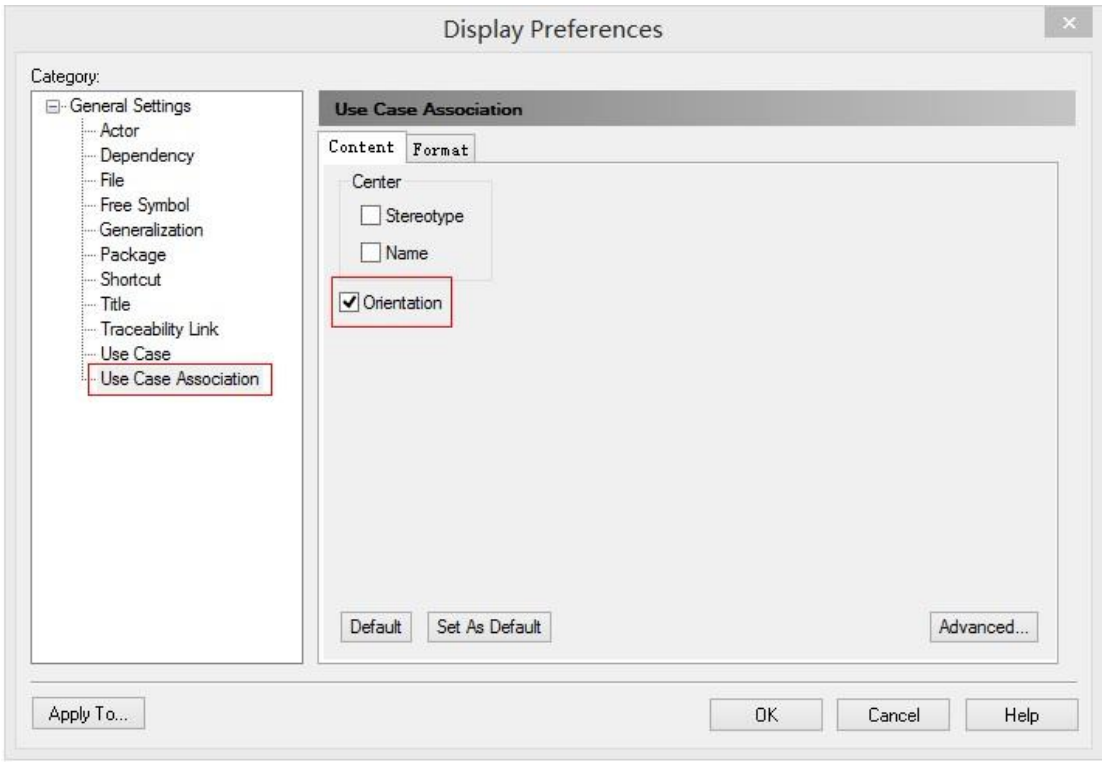


二：设置一下让画图区的那些页面线不显示，这样就不会干扰我们的视线。

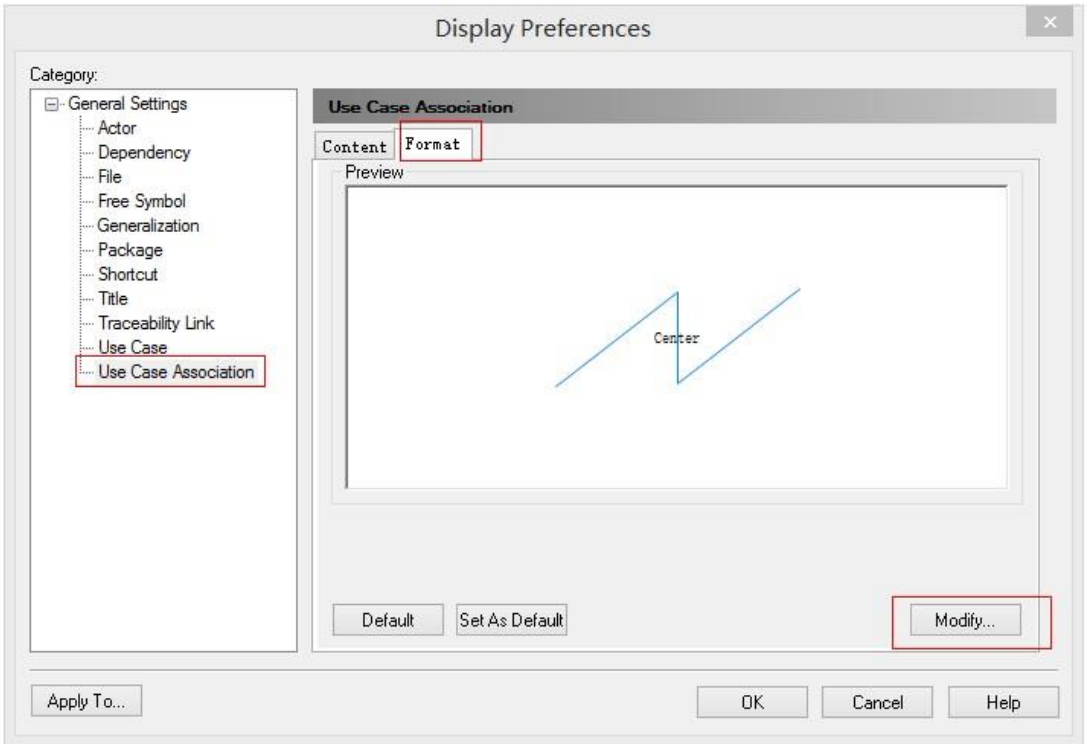
Tools->Display Preferences

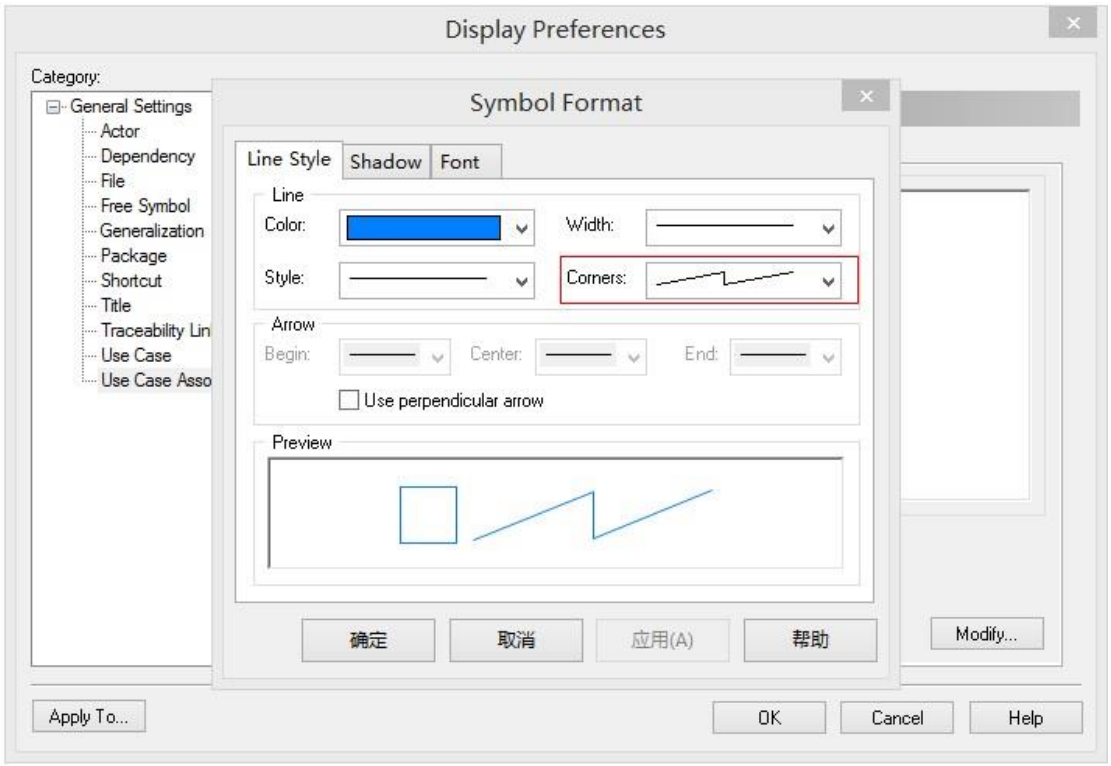


三：设置一下线的箭头这样可以更好的清楚用例的出发者，也更好描写需求。



四：设置一下用例图的线，不设置的话会画成折线，我们一般喜欢用直线，这样可以更好的描写用例。





五：设置一下让名称和密码不一致，因为一致的话很麻烦 Tools->General Options