StarUML使用说明-指导手册

# 综述

UML（Unified Modeling Language，统一建模语言）是一个支持模型化和软件系统开发的图形化语言，为软件开发的所有阶段提供模型化和可视化支持，包括由需求分析到规格，到构造和配置。常见的几种UML图有

用例图：从用户角度描述系统功能，并指各功能的操作者。

类图：描述系统中类的静态结构。

顺序图：对象之间的动态合作关系，强调对象发送消息的顺序，同时显示对象之间的交互。

协作图：描述对象之间的协助关系。

状态图：是描述状态到状态控制流，常用于动态特性建模。

活动图：描述了业务实现用例的工作流程等。

StarUML是一种生成类图和其他类型的统一建模语言(UML)图表的工具。这是一个用Java语言描述的创建类图的简明手册

下载

[下载](http://staruml.io/download)StarUML-v2.8.1.msi（本文环境），可以支持mdj和uml格式

破解:

## 破解

https://blog.csdn.net/qq\_23370223/article/details/78069728

​1.使用Editplus或者Notepad++等特殊的文本编辑器打开 安装位置下/www/license/node/LicenseManagerDomain.js文件

２.修改validate方法，在如下指定的位置上添加指定的代

function validate(PK, name, product, licenseKey) {

var pk, decrypted;

**//添加代码开始**

**return {**

**name: "hancher",**

**product: "StarUML",**

**licenseType: "vip",**

**quantity: "www.hancher.com",**

**licenseKey: "hello,hancher!"**

**}**

**//添加代码结束**

try {

pk = new NodeRSA(PK);

decrypted = pk.decrypt(licenseKey, 'utf8');

} catch (err) {

return false;

}

var terms = decrypted.trim().split("\n");

if (terms[0] === name && terms[1] === product) {

return {

name: name,

product: product,

licenseType: terms[2],

quantity: terms[3],

licenseKey: licenseKey

};

} else {

return false;

}

}

在如上位置，插入如上代码。

3.验证打开StarUML，打开菜单Help->Enter License，输入上面指定的name和license信息，分别为hancher和hello,hancher!，成功！

**不行的话，重启程序就好**

然后就有注册成功的提示信息了，help->about startuml，可以看到​提示信息已注册

## 软件基本概念

### 模型、视与图（Model, View and Diagram）

模型、视与图（Model, View andDiagram）的概念。模型是包含软件模式信息的元素。视则是模型中信息的可视表达法，图则是表示用户特定设计思想的可视元素的集合。

### 项目

项目是基本的管理单位。一个项目可以管理一个或多个软件模型，它是在任何软件模型中都存在的顶级的包。一般地说，一个项目保存在一个文件中。

一个项目包含并管理下列子元素：

项目子元素 描述

模型（Model） 管理一软件模型的元素。

子系统（Subsystem） 管理表示子系统的模型的元素。

包（Package） 管理元素所需的最一般的元素

项目文件中包含下信息：

项目中所用的UML轮廓（profiles） 

项目所引用单元文件 

项目中包含的所有模块的信息 

项目中包含的所有视与图信息

### 单元

一般地说，一个项目保存在一个文件中。但有时需要团队协作，划分为更小的单元俩管理。一个单元可以按等级来划分，还可以包含多个字单元，格式为.umll,可以为项目文件.uml或其他单元unt所引用。

### 图分类

### 用UseCase图

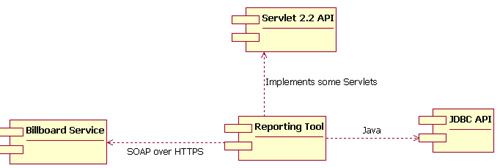
# 图分类

## **构件图** （组件图）

  描述代码构件的物理结构以及各种构建之间的依赖关系。用来建模软件的组件及其相互之间的关系，这些图由构件标记符和构件之间的关系构成。在组件图中，构件时软件单个组成部分，它可以是一个文件，产品、可执行文件和脚本等。

组件图提供系统的物理视图。它的用途是显示系统中的软件对其他软件组件（例如，库函数）的依赖关系。组件图可以在一个非常高的层次上显示，从而仅显示粗粒度的组件，也可以在组件包层次2上显示。

　　组件图的建模最适合通过例子来描述。图7显示了4个组件：Reporting Tool、Billboard Service、Servlet 2.2 API和JDBC API。从Reporting Tool组件指向Billboard Service、Servlet 2.2 API和JDBC API组件的带箭头的线段，表示Reporting Tool依赖于那三个组件。



组件图显示了系统中各种软件组件的依赖关系\fhgsdfgdfgfhgsdfgdfgsdfgsdfgsdfgsdfgsdfgsdfgsdfgsdfgsdfgsdfgsdfgsdfgsdfggfgfsdfggsdfgsdfggsdfgfgfgfgfgfgfgfgfgfgfgfgfgfgfgfgfg

[UML图详解（五）组件图](https://blog.csdn.net/fanxiaobin577328725/article/details/51647248)

# 第五章 用图建立模型

## 用UseCase图建模

下列元素在用例图中可用：

Actor 角色

UseCase 用例

Association 联系

Derected Association 直接联系

Generalization 泛化

Dependency 依赖

Include 包含

Extend 扩充

System Boundary 系统边界

Package 包

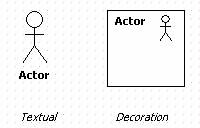
### 参与者（Actor）

语义

    参与者定义了在与实体交互时该实体的用户可以发挥作用的一套清楚的角色。参与者可以被认为是对于每个用来交流的每个用例而言的独立的角色。

#### 创建参与者的过程

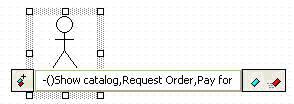
    要创建参与者，点击[工具条Toolbox]-> [用例UseCase]-> [参与者Actor] 按钮，然后在要放置参与者的地方单击。参与者以人轮廓形式或带方框的图标记形式显示，那是个装饰视图。要在装饰视图中显示参与者，在工具条上的组合框中选择[Decoration]项或[格式Format]-> [构造型显示StereotypeDisplay] -> [Decoration] [Decoration]菜单。



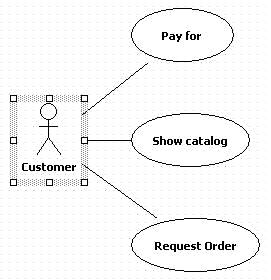
#### 用角色一次创建多个用例

    要一次创建多个关联到参与者的用例，用参与者创建句法的快捷方式。

   1.在参与者快捷对话框，在"-()"后输入用例名。要创建多个用例，输入方法相同，用 ","隔开用例名。



   2.按[Enter] 键. 几个用例就创建了，并按垂直方向排列。



### 用例UseCase

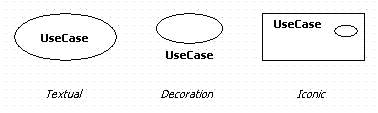
语义

    用例构造用于定义系统行为或者气压的语义实体而不展示其内部结构。每个用例指定一系列行为，包括变体，可执行的实体，与参与者实体交互。

#### 创建用例的过程

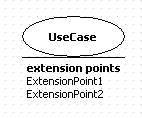
    要创建用例，点击[Toolbox]-> [UseCase]按钮，然后在主窗口上点击要放置用例的地方。

    用例可以用文本、装饰及图标的方式表示。要改变用例的可视风格，选择[Format] -> [Stereotype Display]下菜单项，或者选择组合框中的[ http://img.blog.163.com/photo/NhT52CoeI0EDQYWtkAiJdg==/290763650942707605.jpg]按钮。

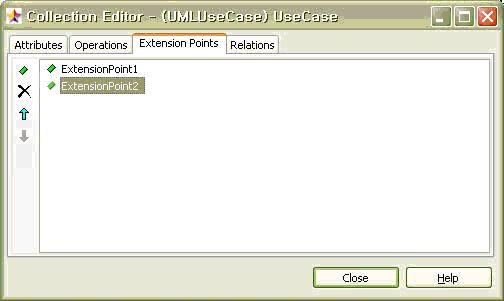


#### 添加扩展（Extension）的过程

    在用例可以扩展的地方，一个扩展点引用一个或一个位置集合。

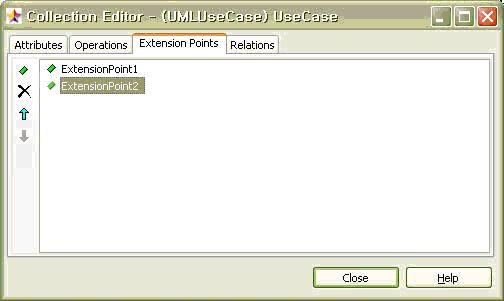


要编辑用例的扩展点，点击用例弹出菜单上的[Collection Editor...]，或者点击集合属性的[ExtensionPoints]按钮。



#### 输入用例证规格说明的过程

    要输入用例的基本流（flow）,可选流，选择弹出[Tagged Values...] 菜单或者按[Ctrl+F7]。 在标记值编辑器，选择[UseCaseSpecification]项，输入属性。.



#### 由用例创建参与者的过程

    为了一次创建多个与用例相关的参与者，可用快捷创建句法。

    1．  双击用例，或者选择这个用例，按[Enter]键。在快捷对话框"()-"后输入参与者名，名与名之间用","隔开。

   2． 按[Enter]键。几个与该用例相关的参与者就创建了，并垂直排列。

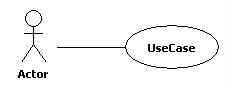
### 关联/直接关联

语义

    关联是两个类元之间（包括一个类元到它自身的）的关系。

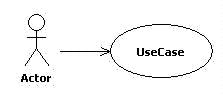
#### 创建关联的过程

    要创建关联，点击[Toolbox] -> [UseCase] -> [Association]按钮，在处窗口中从第一个元素拖动，到第二个元素放下。

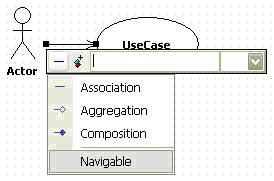


    创建直接关联的过程

    过程与创建关联一样，只是按箭头方向拖放。



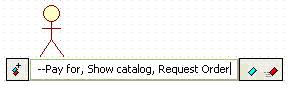
    或者创建关联，点击关联的参数者一侧端点。在快捷对话框，取消可导航复选框，关联就变成了直接的。



#### 创建与关联/直接关联相关的元素的过程

    要创建与当前元素相关的元素，可用快捷创建句法。

    1.双击元素，在快捷对话框，在"--" 或 "->输入相关元素名。



    2.按[Enter]键，几个相关元素就创建了，并垂直排列。

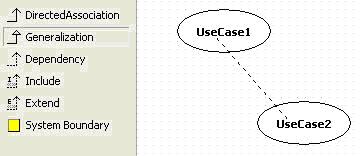
### 泛化Generalization

语义

    泛化是一中分类学关系。是一个较广泛的元素（父类）和一个较特殊的元素（子类）之间的关系。较特殊的元素（子类）和第一个元素完全一致的，只是有些额外的信息。

创建泛化的过程

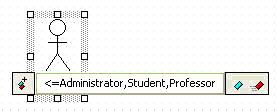
    要创建泛化，点击[工具条Toolbox]-> [用例UseCase]->[泛化Generalization]按钮。在主窗口中，从子元素起拖动鼠标，到父元素处放下。



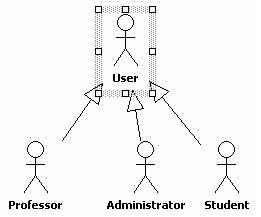
### 创建多个继承自参与者的子参与者

    要创建继承自某个元素的多个元素，

    1.在快捷对话框，"<="后输入元素名，继承自所选元素的几个元素就一次创建出来了。



    2.子元素在所选元素下生成，并自动排列。



    如果你要一次创建多个父元素，在快捷对话框中的"=>"字符串后而不是在"<="后输入元素名。

### 依赖Dependency

语义

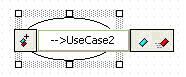
    依赖是一种类型的关系。一个（或一组）元素，作为客户，依赖于另一个（或一组）元素，作为提供者。它是一种弱关系（relationship），这意味着提供者的改变，客户会受到影响。它是一种非直接的关系。

#### 创建依赖的过程

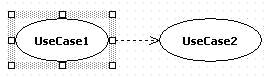
    要创建依赖，点击[工具条Toolbox] -> [用例UseCase] -> [依赖Dependency]按钮，拖动元素到依赖的元素放开。

#### 创建当前用例所依赖的其他用例的过程

    在快捷对话框，用字符串输入依赖名，如下。



    这样就在两个元素之间创建了依赖关系。



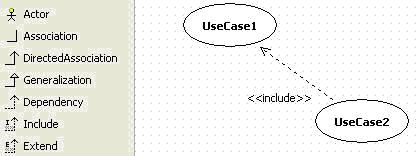
### 包含Include

语义

    包含关系定义了一个用例包含了另一用例所定义的行为。

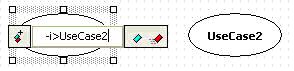
#### 创建包含的过程

    要创建包含关系，点击[工具条Toolbox]-> [用例UseCase]-> [包含Include]按钮。在主窗口拖动包含元素到被包含元素。

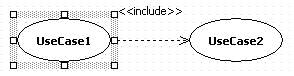


#### 创建当前用例所包含的其他用例的过程

    在快捷对话框用"-i>"字符串按如下方式输入。



    这样包含关系就在两个元素之间创建了。



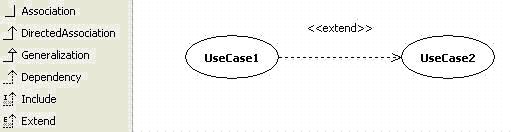
### 扩展Extend

语义

    扩展关系定义是，用例实例可以被扩充，以增加扩充的用例中所定义的附加的行为。

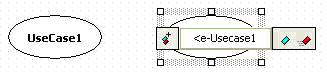
#### 创建扩充的过程

要创建扩展，点击[工具条Toolbox]-> [用例UseCase]-> [扩展Extend]按钮，在主窗口中拖动扩展元素到被扩展的元素。

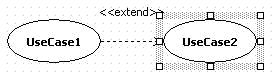


#### 创建扩展当前用例的其他用例

    在快捷对话框，用"<e-"字符串按如下方式输入。



    这样扩展关系就在两个元素之间创建了。



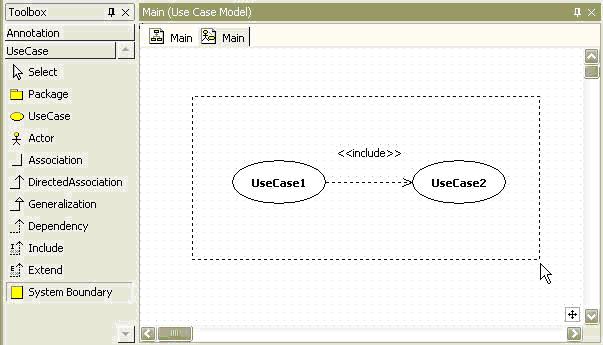
### 系统边界System Boundary

语义

    系统边界是表示用例（边界内）与参数者（边界之外）一种类型的划分。它最典型的用法是这个系统的边界。用例可以用来表示子系统和类，因而边界比这个系统更明确。构造型为顶层（topLevel）的包可以作为系统边界。用例模型内的名称空间也同样地表示用例的边界。

#### 创建系统边界的过程

    要创建系统边界，点击[工具条Toolbox]-> [用例UseCase]-> [系统边界SystemBoundary]按钮，从系统边界的起点拖动鼠标，到系统边界的右下放开。



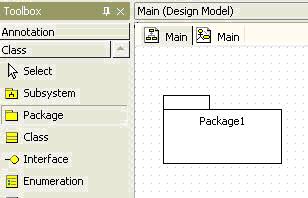
### 包Package

语义

    包是一组模型元素。包可以嵌套在另外的包内。一个包也许包含下级包以及其他种类的模型元素。所有种类的模型元素都可以组织到包里。

#### 创建包的过程

    要创建包，点击[工具条Toolbox]-> [用例UseCase]-> [包Package]按钮，然后点击主窗口中要放置包的地方。

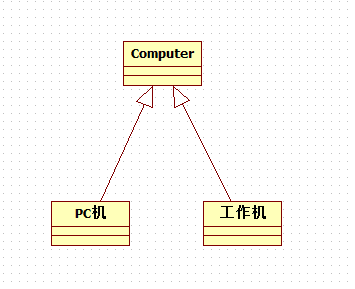


## 概念

### 类与类间的关系

#### 泛化（Generalization）

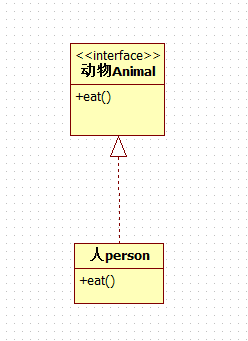
泛化（也有人直接称为继承）是类与类间或者接口与接口间最常见的的关系指的是一个类（子类、子接口）继承另一个类（父类、父接口）的属性与方法，并且可以扩展它自己的新属性与方法。二者是is-a的关系，通俗地讲就是“这个东西是那个东西的一种”，再比如PC机是计算机，工作站也是计算机。PC机和工作站是两种不同类型的计算机，但都继承了计算机的共同特性。在Java中用“extend”来表示此关系。在类图中使用带空心三角箭头的实线表示，箭头从子类、子接口指向父类、父接口。



泛化is-a（空心三角箭头的实线）

#### 实现（Realization）

指的是一个类实现接口的功能。在类图中使用带空心三角箭头的虚线表示，箭头从实现类指向接口。例如动物都有eat吃的这个行为，人是动物的一个具体实例，实现具体的eat动作

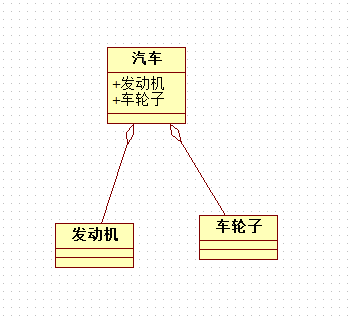


实现（空心三角箭头的虚线）

两个相对独立的对象，当一个对象的实例与另外一个对象的特定实例存在固定关系时，这两个对象之间就存在关联关系

#### 聚合（Aggregation）

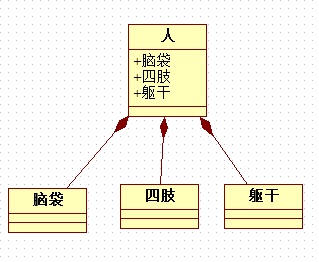
指的是整体和部分的弱关系，是has-a的关系，此时整体与部分之间是可分离的，即没有了整体，局部也可单独存在。比如公司和员工的关系，公司包含员工，但如果公司倒闭，员工依然可以换公司。处于聚合关系的两个类生命周期不同步，即他们可以具有各自的生命周期——当A创建的时候，B不一定创建；当A消亡时，B不一定消亡。在类图使用空心菱形的实线表示，菱形从局部指向整体。



聚合has-a（空心菱形的实线）

#### 组合（Composite）

指的是整体和部分的强关系，是一种contains-a的关系，部分不能脱离整体存在。整体的生命周期结束也就意味着部分的生命周期结束，比如“国破家亡”。在类图使用实心菱形的实线表示，菱形从局部指向整体。处。

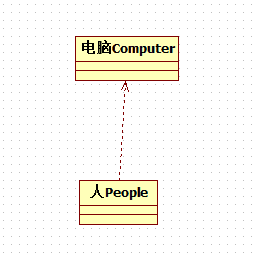


组合contains-a（实心菱形的实线）

#### 依赖（Dependency）

类A要完成某个功能必须引用类B，但这种使用关系是具有偶然性的、临时性的、非常弱的，而B类的变化会影响到A，则A与B存在依赖关系，依赖关系是弱的关联关系。比如本来人与电脑没有关系的，但由于偶然的机会，人需要用电脑写程序，这时候人就依赖于电脑。在java中表现为局域变量、方法的形参，或者对静态方法的调用。在类图使用带箭头的虚线表示，箭头从使用类指向被依赖的类。

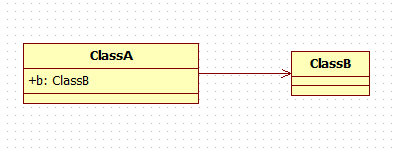
依赖（箭头的虚线）



依赖（箭头的虚线）

#### 6、关联（Association）

关联是指一个类用到了另外的一个类，但不依赖。一般是长期性的，而且双方的关系一般是平等的。关联可以是自身关联、单向关联、双向关联。在java中一般使用成员变量来实现。在类图使用带箭头的实线表示，箭头指向被拥有者。



关联（箭头的实线）

关联和依赖的区别——

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。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。

从关系的生命周期来看：

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。

代码演示聚合、组合间的区别

就结合上面的几个例子来讲，比如人拥有脑袋，人死了，脑袋也废了，这是个组合关系。人可以有一台电脑，但二者不是共生关系，这个是聚合关系。

public class Computer {

}

public class Brain {

}

public class Person {

private Computer computer;

public Person(Computer computer){

this.computer = computer; //聚合

}

private Brain brain;

public Person(){

this.brain=new Brain(); //组合

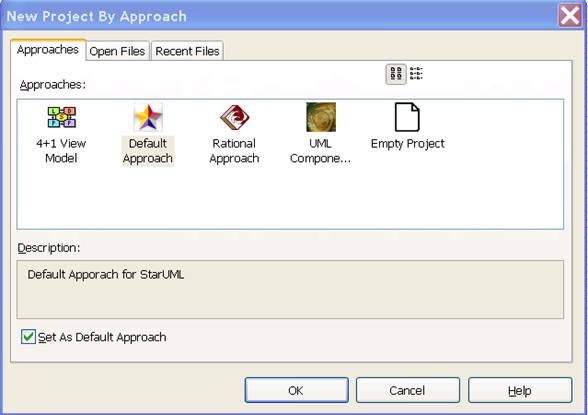
}

}

非商业转载请注明出处。明出处。

# 基本流程

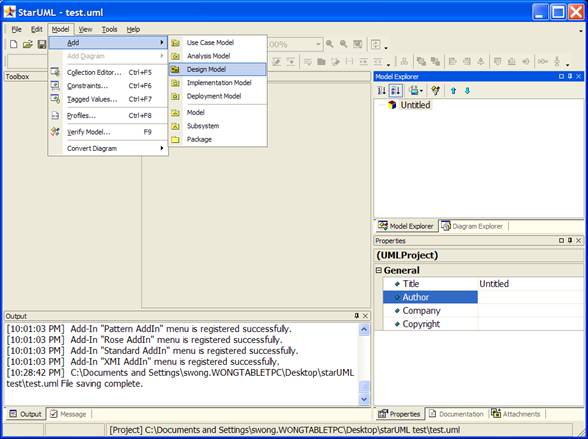
New Project By Approach的对话框会弹出。选择“Empty Project”并且按下"确定"。这里建议你不要勾选"设置为默认的做法" 复选框。



## 模块管理

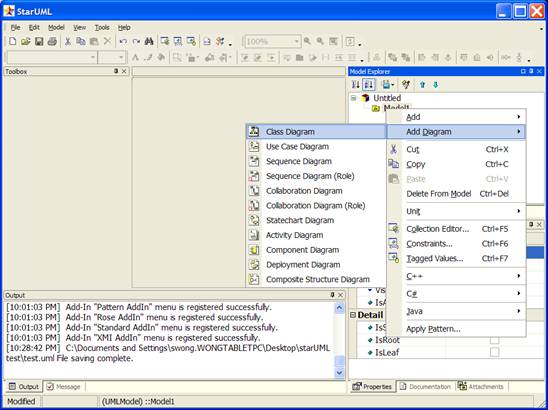
选择模块：在右边的“Model Explorer”框中选定“Untitled”模块。

添加模块：通过“Model”主菜单，或右击选定的模型，可以" Add/Design Model "



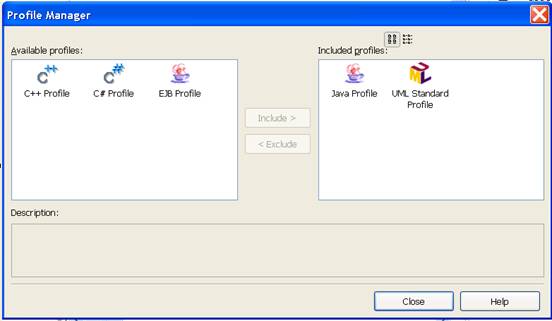
## 类图管理

通过“Model”主菜单，或右击选定模型，可以“Add Diagram/Class Diagram”：



## 设置profile

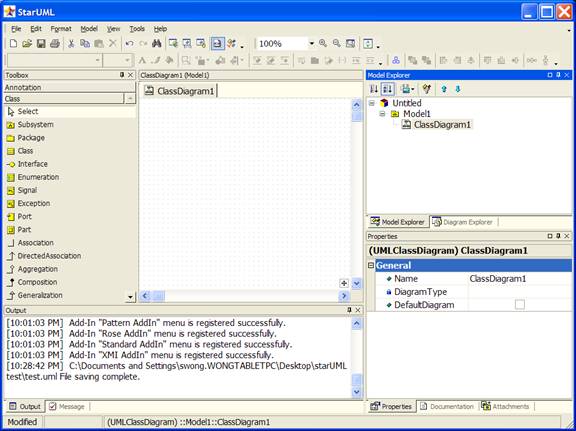
通过“Model/Profile...”菜单去设置工程所需的profile。这决定了工程所使用的规则和约定。一定要包含"JAVA Porfile"这一项目。



## 保存工程

立即就保存工程，这样在出现问题的时候，您就不会丢失信息。

从“File ”菜单，选择“Save” ，并选择一个地方以保存工程。你的StarUML项目现在应该看起来的是这样的：



## .创造图表

现在，开始真正创造图表

### 类管理

从默认就在屏幕的左边的“Toolbox”选择“类”图标，然后左键单击diagram窗口的某处。这样就使用通用名字创造了一个新的类。双击，将类改名为Circle。

#### 添加属性

右击图中的目标，在弹出菜单中选择“Add”中的“Attribute”(被标示为绿色)，为其添加一个属性(或者域)，填入期望的名字“\_radius”。

l 具体的数据类型，在属性面板（右下侧的窗口） ，由双打字，在"类型"时段。 在窗体右下边的Properties面板中，找到“Type”输入框，输入double作为\_radius属性的类型。

l 类的内部数据（域/属性）都是私有的，因为他们是严格由类内部使用的。所以，在Properties面板中将\_radius设置为“私有”。

继续进行设计

重复同样的过程，添加所谓的名字叫做Rectangle的类和double型的私有成员\_width和\_height。

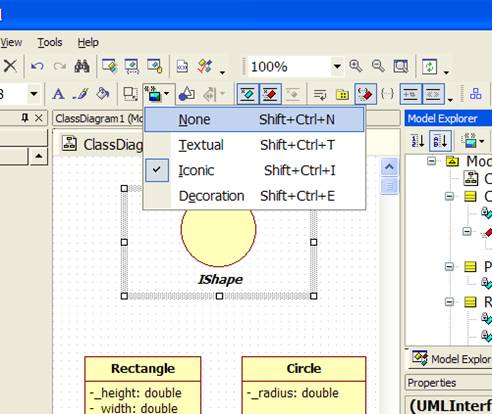
#### 创造 IShape interface

Sdaf

从toolbox中，选择“Interface”，并点击图表的某处。将其改名为IShape。创建以后，选中它。

            在顶部工具栏，选择 “Stereotype Display” 下拉按钮，将值改变为“None”。这将改变以往的圆形形状，使其变为成长方形。

还是在顶部工具栏，取消选中" Suppress Operations "。这将使我们能够看到接口所拥有的方法。



  向IShape 接口添加返回值为double的getArea方法。

      可以通过右击interface的图标，在弹出菜单中点击红色的“Operation”按钮，然后输入getArea。

      设定返回值类型。在“Model Explorer”中展开IShape节点，右击你刚刚创建的getArea方法，在弹出的dlg并选择“Add Parameter”。在“Properties”框中，将参数的名子变为空，将“DirectionKind”变为“RETURN”，将“Type”变为dboule。

            将IShape和getArea的IsAbstract属性框打上勾，他们在图标上的名字将变为斜体。这是UML的标准，表示这是接口或者其他纯虚实体。

#### 添加类和接口的关系

     可以通过从toolbox中选择表示“Realization”的箭头，并从Circle拖拽向IShape，使Circle实现接口IShape。重复同样的过程，为Rectangle添加实现关系。这是添加了Circle 和 Rectangle对于IShape接口的实现关系。

      如果想使连接线表现为直角的方式，右击连接线，并选择" Format/Line Style/Rectilinear"菜单。你通过这种方式，使箭头重叠在一起，可以使你的图看起来更整洁。

#### 添加类基于接口的行为

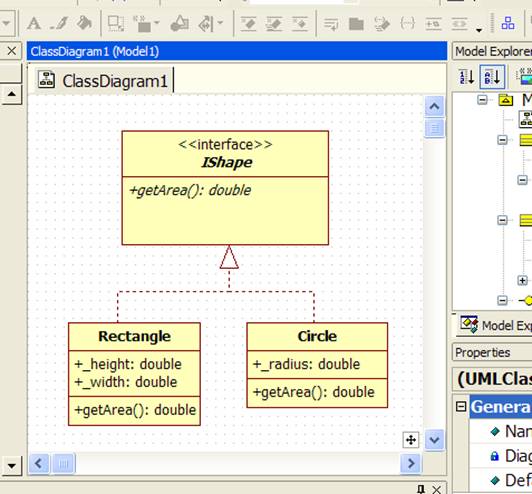
Sdfg

    由于Circle和Rectangle类都实现了IShape接口，就必须有同样的行为(方法)。

        在“Model Explorer”面板中，复制getArea法(按Ctrl-C或者右键点击并选择Copy菜单），并粘贴到Circle和Rectangle类。

        这些实现了的方法在Circle和Rectangle类中都不是抽象的，而是具体的。这是因为他们实际上是执行一些特定行为（例如，为一个圆形和长方形分别计算面积） ，所以不要勾选IsAbstract框。

你的图现在应该是这样的：



### 添加Pizza类

 向Pizza添加double型的私有域\_price。

        添加返回double类型的共有操作getPrice。

#### 为Pizza类添加IShape 的引用

        从toolbox中选择" DirectedAssociation "箭头，点击Pizza类，并向IShape拖拽 。

        选中箭头，在右边的“Properties”框上，将name一栏改为“has-a”，“End1.Aggregation” 一栏改为“AGGREGATE”(这个图示说明Pizza和shape对象是“聚合“的关系)。

        将“End2.Name”一栏改为\_shape 。这样就自动为Pizza添加一个名字为\_shape，使用IShape接口的私有域，的所谓\_shape型ishape以pizza饼。

        将“End2.Visibility”改为私有。

        为\_shape创建一个“获得者”方法，名字叫做getShape，返回IShape 。这就是创建一个行为，名字是getShape，返回IShape 。

#### 为pizza类添加构造函数

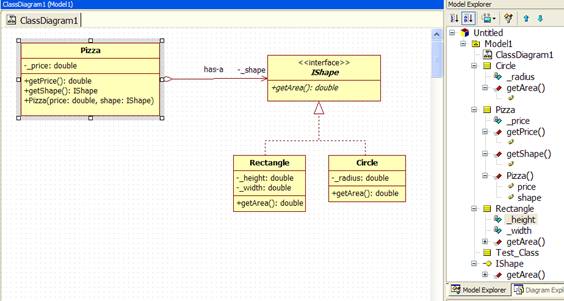
        为Pizza添加构造函数，右击，在弹出的“Add” 菜单中选择“Operation”。从这里，增加一个普通的带有dboule型price参数和IShape类型shape参数的操作

        增加一个输入参数，就像之前增加了一个返回型的输出参数一样，你指定的参数的名称，如价格和形状等，以及适当的数据类型。

        为Circle增加一个带有double型的radius参数的构造函数。

        为Rectangle增加一个带有double型width和height参数的构造函数。

你的图现在应该是这样的：



### 添加Test\_Pizza类

为了说明UML类图更多的功能，又增加了一个叫做“Test\_Pizza”的类，它用作测试目的，并使用到Pizza和IShape类。

        两个类之间的关系有多种形式。举例来说，一个类可以实例化另一个类，而不是将其作为一个成员。又或，一类的方法可能需要另一个类作为输入参数，保留一个引用仅仅是为这个方法的执行。

        通过从toolbox中选择“Dependency”箭头，从一个类拖向他所以来的类，来添加不通类之间的依赖关系。在这个例子中， Test\_Pizza 依赖于Pizza，Circle和Rectangle类，因为它实例化了它们。

        从Properties box选择name属性，或者双击图表上的“依赖线”，可以为依赖关系添加标签。特别的是，当一类实例化另一个类，我们会把依赖线叫做“instantiates”。

        你可以选中并拖动依赖线的标签，以达到更美观的效果。

        依赖关系不会影响代码生成。

## 24.保存项目

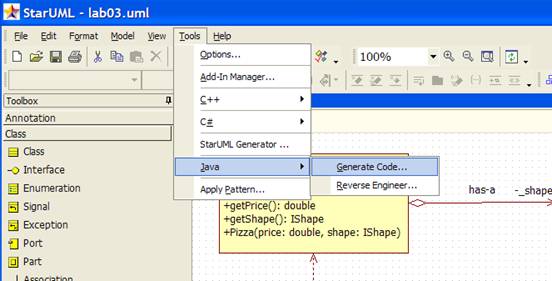
在“File”菜单中，选择“Save”。 SU的所有资料只有一个单一的项目文件，所以你目前应该只有一个文件生成。

25.导出

将图表导出为其他格式，例如图片等，是非常有用的。您可以通过选择“File”菜单的“Export Diagram” ，并且选择合适的文件类型来执行改操作。

生成Java stub代码：

l 点击主菜单的“Tools->Java”菜单，选择“Generate Code”。



从对话框中选择你的模块(这里可能Model1),点击“Next”。

l 为了使你的模块或者图标的所有类都生成stub code，选择“Select All”然后按“Next”。

l 选择一个有效的输出目录，“Next”。

l In the "Options Setup", be sure to check both "Generate the Documentation by JavaDoc" and "Generate empty JavaDoc". All other checkboxes should be unchecked. Then press "Next".在“Options Setup” ，请务必选中“Generate the Documentation by JavaDoc”，“Generate empty JavaDoc”，所有其他复选框不选中，“Next”。

l 现在StarUML将从你的图产生代码，点击“Finish”退出对话框。

l 现在，您可以编辑生成的代码，以增加应用。

# 高级使用

select ：选中某个元素

object：对象

Stimulus：对象与对象间消息

SelfStimulus：对象间自调用的消息

Combined Fragment：结合片段，比如循环片段等，分支片段

Interaction Operand： 交互操作

Frame ： 框架，主题（图最外面的边界）

# REF

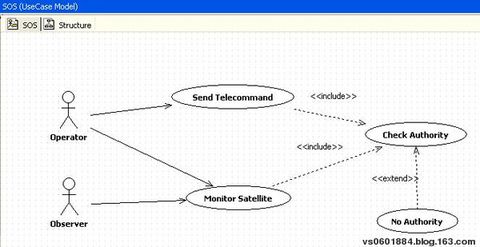
2010年11月08日：[StarUML使用说明-指导手册](https://blog.csdn.net/monkey_d_meng/article/details/5995610)

# TASK

[如何敏捷开发](https://wenku.baidu.com/view/f32c8e69011ca300a6c39025.html)

[UML的9种图例解析](https://www.cnblogs.com/firstcsharp/p/5327659.html)

# 口诀

1. UML纲要  
     
       看了几年的 UML有关文档，但是有时想想连九种类型的图也记不全，于是想到编几句歌诀来助记。这就是“UML 9图歌诀”。后来想干脆编全了，看看助记效果如何。这就是下面的全部内容的由来。现在这里公布出来，望专家及有兴趣的同好批评指正。  
     
       零、UML 9图歌诀  
     
       类与对象加用例，  
       状态顺序活动矣，  
       协作构件再部署，  
       统一建模9图齐。  
     
     
       一、用例  
     
   什么是用例  
   [](http://img.bimg.126.net/photo/h6LeiowoxJWO1y2yUuEVWQ==/4799148352917777982.jpg)  
       系列事件谁发起？  
       人机系统都可以。  
       参与事件将如何？  
       使用场景即用例。  
     
   用例的包含  
        用例之中重复的，  
       抽取出来使独立。  
       包含进去几合一，  
       思路清晰好处理。  
     
   用例的扩充  
       已有扩充为新例，  
       额外步骤加进去。  
       这是重用又一法，  
       扩展派生出新例。  
     
   用例的泛化  
       子到父类为泛化，  
       参与行为都可以。  
       空心箭头加连线，  
       继承关系很明晰。  
     
   用例的分组  
       用例多了要分组，  
       层次类别才明晰。  
       相关打包包一起，  
       父子系统成体系。  
   用例分析  
       开始交谈进领域，  
       初步类图要获取。  
       注意名词新术语，      
       相关动词也要记。  
         
       询问如何用系统？  
       侯选用例可获取。  
       都谁参与应列表，  
       看清用例谁发起。  
     
       逐步深入问下去，  
       不断发掘新用例。  
       有助界面之设计，  
       编程决策也得益。  
     
   用例的用途  
       预期行为来收集，  
       图形工具强有力。  
       其他类别相结合，  
       明确用户心中疑。  
     
     
       理解用户和领域，  
       用例高层先注意。  
       只重行为非实施，  
       系统边界可明晰。  
     
   用例图与文档  
     
       文档之中用例图，  
       每例N页来描述。  
       场景步骤要清晰，  
       上层注释不相符。  
     
   用例的细节追踪  
       发起与者加用例，  
       场景步骤写清晰。  
       前后条件莫忘记，  
       参与者里谁受益。  
     
   二、状态图  
   状态图  
       对象时序改状态，  
       展示变化状态图。  
       变化序列起终点，  
       对象单一莫疏忽。  
     
       圆角矩形表状态，  
       箭头实线表迁移。  
       实心圆点为起点，  
       牛眼圆圈为终点。  
     
   状态的转移细节：事件和动作  
     
       状态变化之行为，  
       引发变化之事件，  
       二者‘/’来分开，  
       都可加到转移线。  
     
       还有事件无触发，  
       活动结束无转移，  
       此类都叫无触发。  
       特殊情形要牢记。  
     
     
   状态的转移细节：保护条件  
     
       保护条件另细节，  
       满足条件才转移。  
       可以写进状态图，  
       写成布尔表达式。  
     
   子状态  
     
       状态之中有状态，  
       其中就叫子状态。  
       顺序并发两形式，  
       单一状态为母体。  
     
   顺序子状态  
       顺序子态较简单，  
       依次逐个来出现。  
     
   并发子状态  
       并发子态也不难，  
       两个状态同出现。  
       并发子态虚线分，  
       母子组成彼此间。  
     
   历史状态  
       历史状态也需知，  
       ‘H’加圈做标记。  
       实线连回记忆态，  
       深浅故态可复忆。  
     
     
   消息与信号  
       对象之间要通信，  
       消息概念必须知。  
       触发也是发消息，  
       对象之间来传递。  
       能触发的叫信号，  
       信号为类可承继。  
     
     
       三、顺序图  
     
   什么是顺序图  
       对象之间有交互，  
       发生起讫顺序图。  
       时间维度加进去，  
       时序通信靠此图。  
     
   对象  
       方框对象名下线，  
       实线箭头表消息。  
       垂直虚线表时间，  
       激活生命重顺序。  
     
     
   消息  
       同步、异步简单的，  
       三个类别皆消息。  
       实心半边与两边，  
       三类箭头三消息。  
     
   时间  
       垂直方向时间维，  
       自顶向下时间序。  
       对象下面生命线，  
       激活长度表时序。  
   ————————  
     
       四、协作图  
     
       协作也是表交互，  
       语义等价顺序图。  
       交互对象显整体，  
       空间组织布置图。  
     
     
       对象(图)扩展协作图，  
       消息传递是为主。  
       箭头表示传消息，  
       发送指向接收的。  
     
       名称序号靠箭头，  
       接收消息操作的。  
       消息名称加序号，  
       冒号中间要隔起。  
     
     
   消息多发：  
     
       对象消息可多发，  
       有时消息重顺序。  
     
   返回结果  
     
       返回结果可表示，  
       变量操作赋值系。  
     
   主动对象  
     
       主动对象有权利，  
       面象其他法消息。  
       两个以上为并发，  
       主动边框给加黑，  
       强调作用请注意。  
     
   消息同步  
     
       等到回信才再发,  
       这种机制叫同步。  
       需要同步消息前，  
       再加消息要传递。  
       序号之间逗号分，  
       ‘/’与消息相隔离。  
     
   ————————————  
   五、活动图  
     
       过程步骤要表述，  
       可以使用活动图。  
       状态图形之扩展，  
       此类活动更突出。  
     
       圆角矩形表活动，  
       实心圆是起点图，  
       公牛眼形为终结，  
       依次活动最突出。  
     
   判定  
     
       条件引发活动列，  
       不同条件各自行。  
       表示方法有两种，  
       用或不用小菱形。  
       分支各自注条件，  
       不同行为分的清。  
     
   并发路径  
     
     
     
   信号  
     
       活动可以发信号，  
       信号接收活动兴。  
       发送凸角五边形，  
       接收凹角五边形。  
       二者输出与输入，  
       信号相关记得清。  
     
   泳道  
     
       可视维度能增加，  
       分别角色可视化。  
       把图分为平行段，  
       称为泳道形象化。  
     
   混合图  
     
       图符来自不同类，  
       有其长处无是非。  
       各种方式合一起，  
       综合长处非新类。  
     
   ——————————————  
     
   六、构件图  
     
       软件系统一单元，  
       驻留系统叫构件。  
       数据文件程序库，  
       潜在重用仔细算。  
     
       建模构件有好处，  
       利于开发与用户。  
       系统结构可清晰，  
       目标清楚都好办。  
       编制文档能明确，  
       便于重用才周全。  
     
   构件和接口  
     
       对象窗口即接口，      
       信息隐蔽或封装  
       类的行为它体现，  
       成组操作记端详。  
     
     
   构件类型  
     
     
       部署构件是一类，  
       工业构件也一种。  
       执行构件另一类，  
       几个类别要分清。  
     
     
   构件图  
       大框是个大矩形，  
       左边2小套其中，  
       包含成员加上名，  
       细节信息更分明。  
     
   接口表示法  
     
       矩形框里注信息，  
       空心箭头来联系。  
       连接虚线到实现，  
       两类构件系一起。  
     
       还有一种表示法，  
       实线构件来联系。  
       表示接口小圆圈，  
       这个表示也牢记。  
     
       除了实现有依赖，  
       导入接口来联系。  
       构件与其用虚线，  
       加上箭头来表示。  
     
   ————————————  
     
   七、部署图  
     
       部署图里有节点，  
       能否执行得分清。  
       节点表示立方体，  
       构件接口在框中。  
       节点之间可连线，  
       连接信息构造型。  
       关联类型几大类，  
       聚集依赖等等等。  
     
   ————————————  
       八、类  
     
   类图  
         
       类的表示用矩形，  
       属性操作在其中。  
     
       其间使用分隔线，  
       动静两类才分明。（属性和方法）  
     
   属性  
       附加信息可加入，  
       数据类型注清楚。  
     
   约束  
       花括号内为约束，  
       满足条件来标注。  
     
   附加注释  
       类的旁边可加注，  
       必要说明更清楚。  
     
     
   如何获取类  
         
       领域名词和术语，  
       分析知识来获取。  
       领域模型化为类，  
       名词动词都注意。  
     
   关系  
       类类之间有关联，  
       表示关联一实线。  
       实心箭头来指明，  
       角色方向线上边。  
     
   关联上的约束  
     
       关联之后有规则，  
       约束条件注清楚。  
       与或条件都可注，  
       加上标记才清楚。  
     
       关联实际也成类，      
       也有属性和操作。  
       联系诸类用虚线，  
       彼此之间都关联。  
     
     
   链  
       关联之类有实例，  
       两者关系就叫链。  
       连线连接两相通，  
       链名下线加下边。  
     
   多重性  
       联系可以一对多，  
       多重关联要琢磨。  
       数字标记两边类，  
       再加星号还有或。  
     
   限定关联  
       关联多重要限定，  
       查找才能分的清。  
       限定ID小矩形，  
       多重之中一对一。  
     
   自身关联  
       类的关联和自己，  
       自身关联也关系。      
       角色关联加方向，  
       关联线上做标记。  
     
   继承和泛化  
             
       继承泛化也好懂，  
       父子特征有相同。  
       子类特征比父多，  
       父类泛泛更普通。  
     
   抽象类  
       抽象类别无实例，  
       子类实例才实际。  
     
   依赖  
       类中定义其他类，  
       依赖关系要知微。  
         
   聚集  
       一类分成几部分，  
       此种类型是聚集。  
       总分关系为特征，  
       整体部分关系体。  
     
   聚集上的约束  
       多个部分成一体，  
       约束关系就成立。  
       并列就是或关系，  
       空心箭头来表示。  
       彼此同级虚线连，  
       再加or字来标记。  
     
   组成  
       组成聚集强类型，  
       实心箭头也菱形。  
       正如桌子和桌腿，  
       此种类型记心中。  
     
   语境  
       类簇含类很常见，  
       其中局部放大看。  
       这是类的语境图，  
       层次关联好体现。  
     
   接口和实现  
       接口为了可重用，  
       虽然类别可不同。  
       相同操作不同类，  
       代码仍然能重用。  
     
       接口只是有操作，  
       没有属性是特征。  
       假如类也省属性，  
       二者区分构造型。  
     
       类与接口有关系，  
       关系名就叫实现。  
       图形表示很简单，  
       空心箭头实线连。  
     
   可见性  
       接口操作都可见，  
       他类调用才简单。  
         
   —————————————  
     
   九、对象图  
     
       类的实例为对象，  
       名有下线外有框。  
       冒号隔开名列表，  
       匿名冒号省对象。  
     
     
     
     
   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
     
   UML基础知识  
     
       体系结构有四层，  
       用户对象为第一。  
       再一层次模型层，  
       早期分析必处理。  
       描述语言第三层，  
       名字叫做元模型。  
       元元模型为第四，  
       逐步抽象建体系。  
     
   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
     
   开发应用  
     
       问题小组理解好，  
       各类角色都不少。  
       良好通讯要畅通，  
       阶段交流不能少。  
     
   领域分析  
       客户领域各实体，  
       交谈分析要笔记。  
       名词建模成为类，  
       后降属性也或许。  
       动词也要都注意，  
       用例操作都或许。  
     
   识别协作系统  
         
       软件系统不孤立，  
       协作系统须注意。  
       节点通讯加新老，  
       各方配合部署齐。  
     
   获取需求  
     
       联合会议来开始，  
       各方人员提建议。  
       会议产品一包图，  
       高层领域之用例。  
       结果提交用户议，  
       得到认可再继续。  
       成本估算不能少，  
       具体情况自算计。  
     
   分析  
       初步需求获取后，  
       类图逐步要加细。  
       系统用法要理解，  
       高层用例先分析。  
       需要画出用例图，  
       依赖构造都要记。  
     
       充实用例再继续，  
       分出每个步骤序。  
         
       细化类图再充实，  
       关联对象和聚集。  
         
       展示对象状态变，  
       状态图出为目的。  
     
       对象交互要定义，  
       顺序协作图表意。  
     
       协作集成诸细节，  
       通信网络都注意。  
     
       数据库类要考虑，  
       包括物理和逻辑。  
       产品详细部署图，  
       分析好了转设计。  
   ————————————————  
   设计  
         
     
   细化对象图  
         
       对象图形要细化，  
       程序员来完成它。  
       检查操作活动图，  
       编码就要根据它。  
     
   开发构件图  
     
       程序员们是主力，  
       画出构件之关系。  
     
   制定部署计划  
     
       部署图在构件后，  
       系统协作与集成。  
       节点驻留何构件，  
       图里处处注分明。  
     
   开发用户界面  
     
       全部用例都完成，  
       考虑界面如何弄。  
       纸上原型先设计，  
       用户满意再换屏。  
       屏幕原型出快照，  
       这个步骤算完成。  
     
   测试设计  
     
       测试依据是用例，  
       测试专家来参与。  
       测试脚本写出来，  
       看看功能是否齐。  
     
   开始编制文档  
     
       文档编制有专人，  
       高层文档先认真。  
       高层结构先指定，  
       结构就是好成品。  
     
   开发  
       类图对象活动图，  
       再加构件是基础。  
       程序代码开始写，  
       阶段产品代码出。  
     
   代码测试  
     
       运行脚本来测试，  
       预期任务完成否。  
       产生信息前反馈，  
       各层测试不用愁。  
     
   界面及其连接代码测试  
     
       界面原型更靠近，  
       连接代码更类真。  
       界面功能要有效，  
       功能系统为产品。  
     
   完成文档  
     
       文档专家程序员，  
       并行工作写文档。  
       确保完成要及时，  
       交付用户不影响。  
     
     
   部署  
     
   编制备份和恢复计划      
     
       周密计划事先有  
       系统崩溃要预防。  
     
   安装最终系统  
     
       最终系统到装机，  
       接近最后之目的。  
       安装完成再测试，  
       看看精化需不需。  
       测试结果为产品，  
       没有其他那么的。