# 系统分析与建模

## UML可视化建模

* 1. UML基本概念
     1. 什么是UML

Unified Modeling Language，统一建模语言是一种开放的方法，用于说明、可视化、构建和编写一个正在开发的、面向对象的、软件密集系统的制品的开放方法。

* + 1. UML中的模型

有三个主要的模型：

* + - 1. 功能模型

从用户的角度展示系统的功能，包括用例图等。

* + - 1. 对象模型

采用对象、属性、操作、关联等概念展示系统的结构和基础，包括类图、对象图等。

* + - 1. 动态模型

展现系统的内部行为，包括序列图、活动图、状态图等。

* 1. UML diagrams

UML核心视图包括静态视图和动态视图，其中静态视图包括用例图、类图、包图等，动态视图包括活动图、状态图、时序图、通信图、部署图等。

* + 1. 用例图（use case diagram）

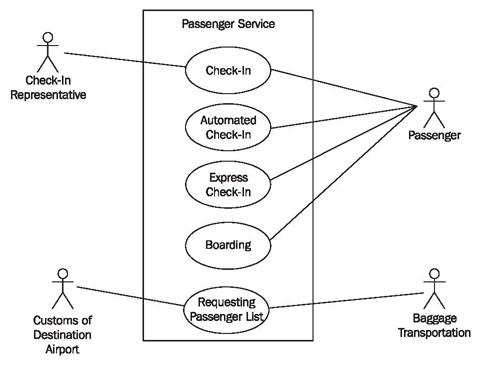
A use case is a list of actions or events steps typically defining the interactions between an actor and a system to achieve a goal.

用例是一系列动作或事件的步骤，通常定义角色和系统之间的交互以实现目标。

A use case diagram at its simplest is a representation of an actor’s interaction with the system that shows the relationship between the user and the different use cases the actor is involved.

用例图是用户与系统交互的最简表达形式，展现了参与者和与之相关的用例之间的关系。

e.g.

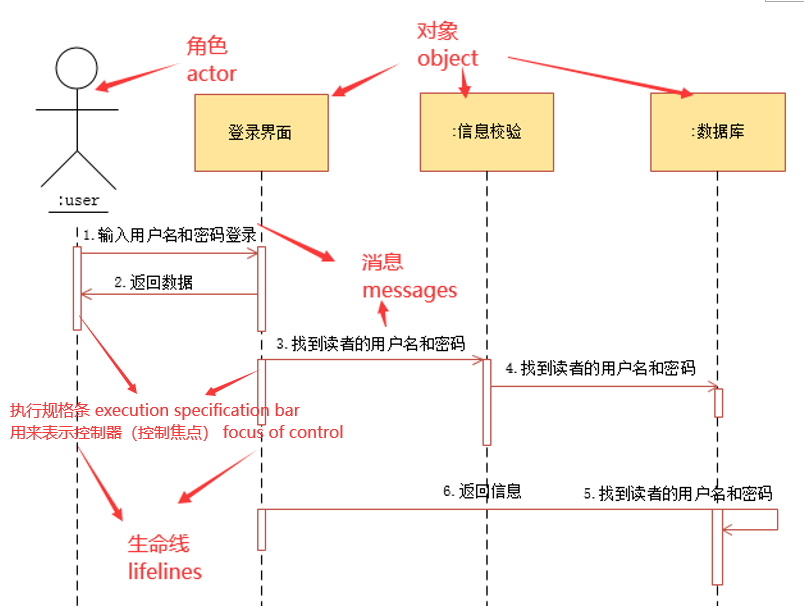


* + 1. 顺序图/时序图（sequence diagram）

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

时序图描述消息在生命线上按照约定顺序执行一种交互行为，可以表示用例的行为顺序。

e.g.

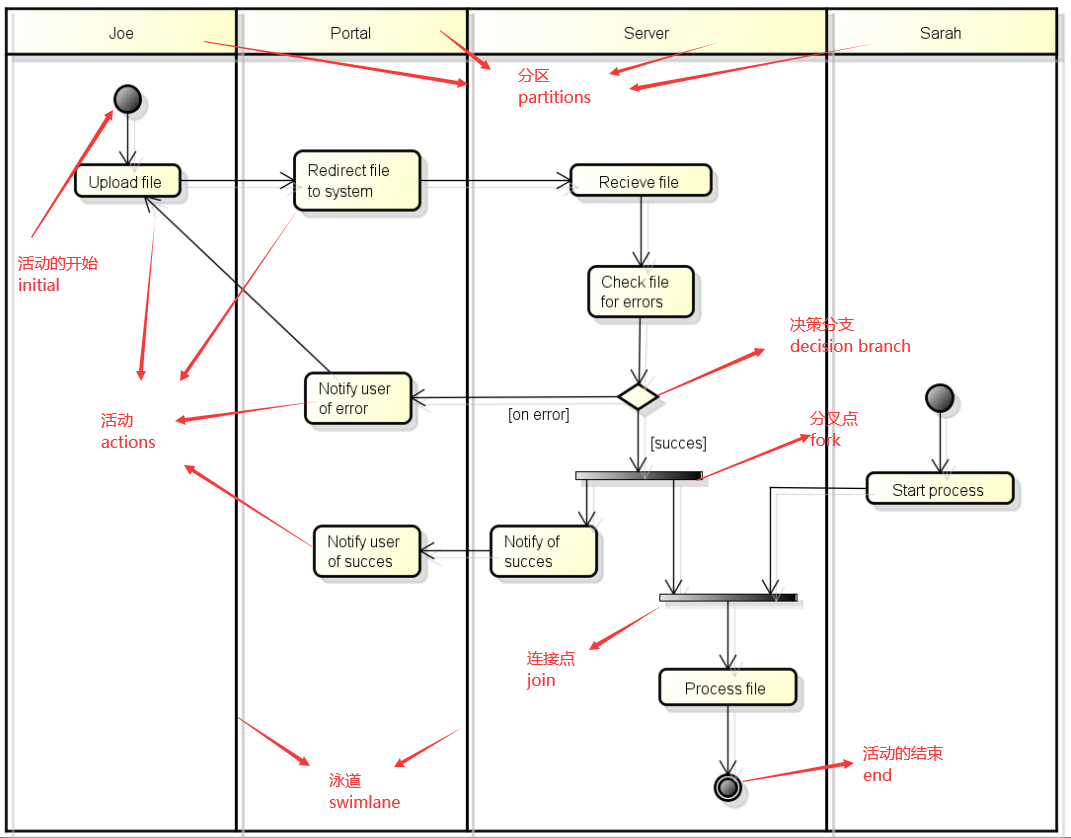


* + 1. 活动图（activity diagram）

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.

活动图表示一个过程中的多个顺序活动和并行活动，有助于对业务过程、工作流、数据流和复杂算法进行建模，用以辅助编写用例文本的UML工具图。

e.g.



* + 1. 交互图（interaction diagram）

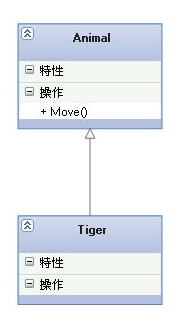
交互图是顺序图（sequence diagram）和通信图（communication）的统称。

* + 1. 类图（class diagram）

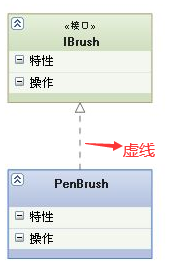
A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

类图用于展示系统中的类的内部结构及其相互之间的关系，是一种静态结构图。其中类与类之间的关系有以下几种：泛化（Generalization）、实现（Realization）、关联（Association）、聚合（Aggregation）、组合（Composition）、依赖（Dependency）。

* + - 1. 泛化。是一种继承关系，表示一般和特殊的关系。



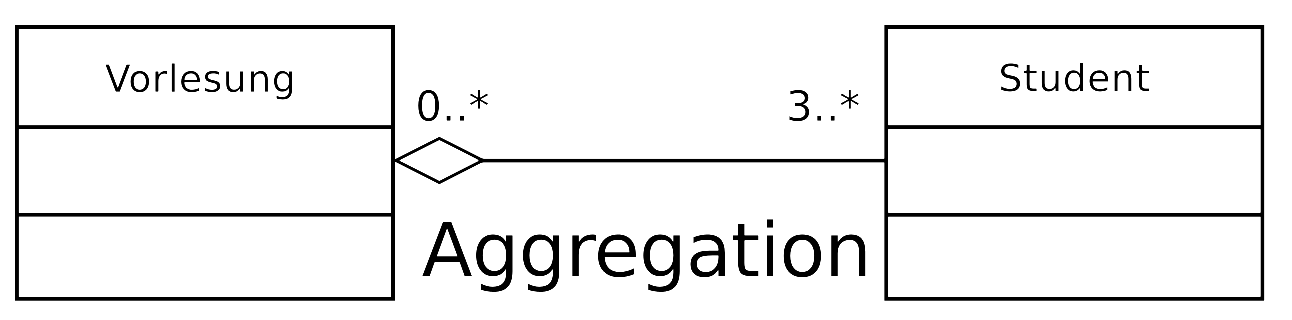
* + - 1. 实现。是一种类与接口的关系，也是继承。



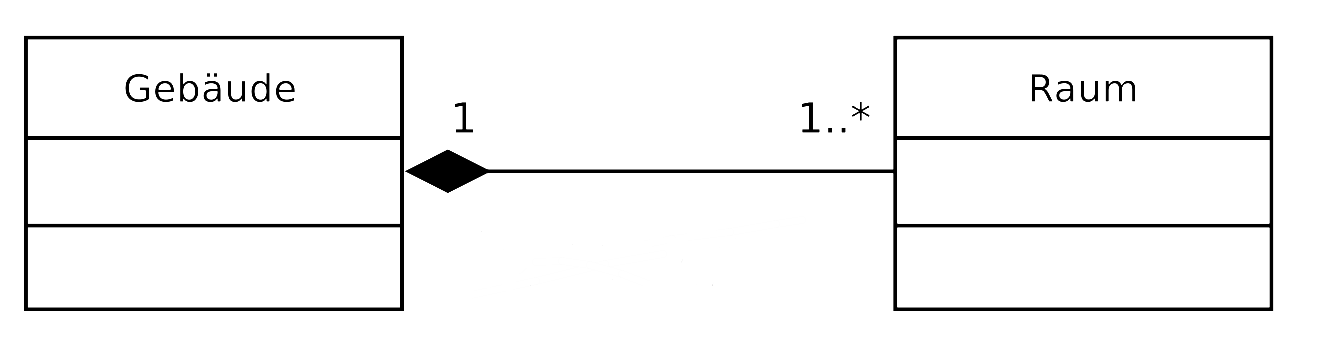
* + - 1. 关联。是一种拥有的关系，在代码中表现为一个类是另一个类的成员变量。



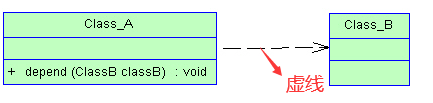
* + - 1. 聚合。是整体和部分的关系，且部分可以独立于整体而存在。在UML图中用空心菱形表示聚合。



* + - 1. 组合。也是整体与部分的关系，但部分不能离开整体而单独存在，比如公司和部门的关系。在UML图中用实心菱形表示组合。



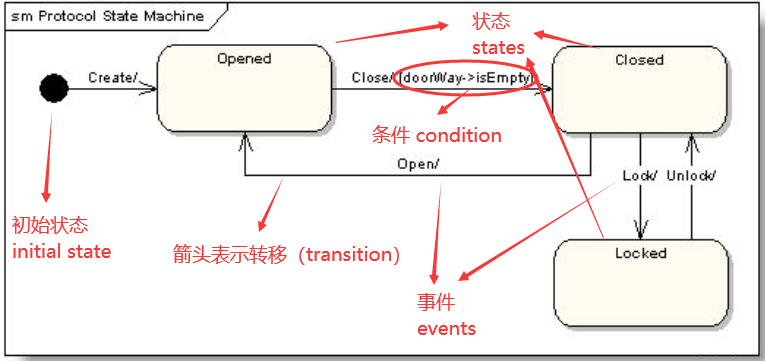
* + - 1. 依赖。是一种使用的关系，且这种使用关系具有偶然性、临时性，不具有关联关系那种较强的、长期的关系。在代码实现中表现为一个类作为参数在另外一个类中的方法中使用。应尽量避免双向依赖。



* + 1. 状态图/状态机图（state diagram）

A state diagram used in computer science describes the behavior of systems using (finite) state machine.

状态图用于对模型元素的动态行为通过状态机进行建模，即对系统行为中受事件驱动的方面进行建模，主要用于描述对象的状态变化以确定何种行为改变了对象状态。



* + 1. 包图（package diagram）

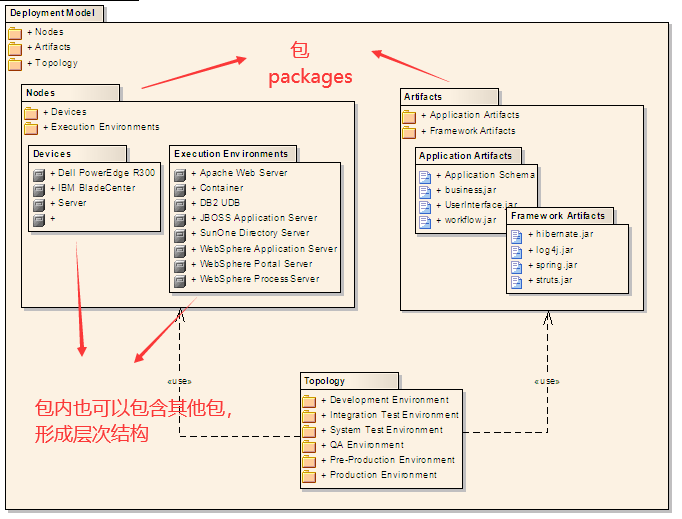
A package is used "to group elements, and to provide a namespace for the grouped elements".

包的主要作用是容纳并为其他元素分类，用以表示命名空间，一般可以容纳任何UML元素。

A package diagram depicts the dependencies between the packages that make up a model.

包图是一种粗粒度、更高层次的图，用来表示包和包之间的关系。

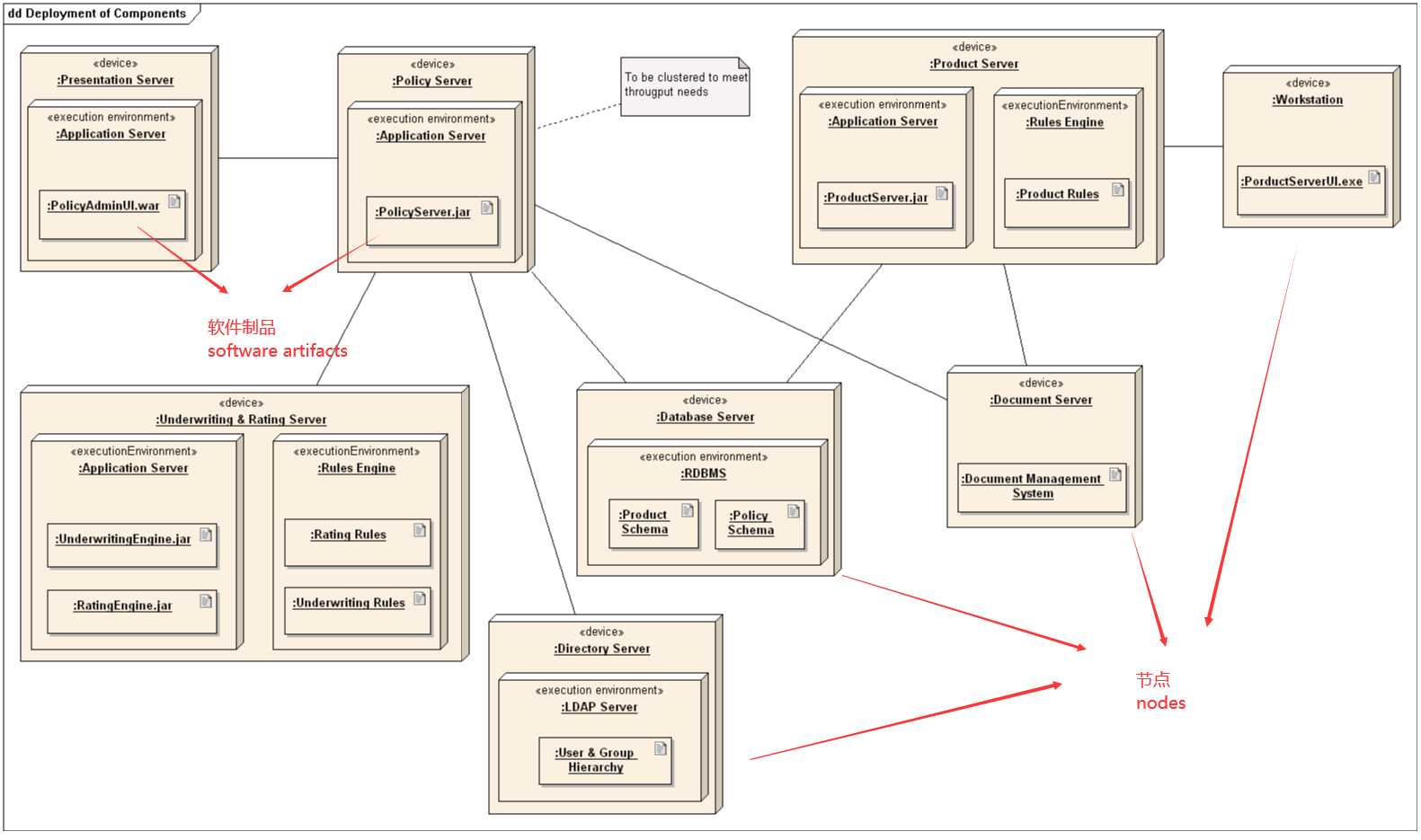
e.g.



* + 1. 部署图（deployment diagram）

A deployment diagram in the Unified Modeling Language models the physical deployment of artifacts on nodes.

部署图表示的是如何将具体的软件制品（比如可执行文件）分配到计算节点上，表示了软件元素在物理架构上的部署以及物理元素之间的通信，有助于沟通物理或者架构部署。



## OOA/D

* 1. 面向对象的四大特性
     1. 抽象（Abstraction）

提取出一个实体中区分其他类型实体的本质特征，抽象成为对象（类）。

* + 1. 封装（Encapsulation）

对用户隐藏内部执行过程和数据等。

* + 1. 继承（Inheritance）

一种可以基于已有对象创建新的对象的机制：子对象继承父对象的操作和属性。

* + 1. 多态（Polymorphism）

指由继承而产生的相关的不同的类，其对象对同意消息会做出不同的响应。

* 1. 建模
     1. 什么是建模？

建模是指通过对客观事物建立一种抽象的方法用以表征事物并获得对事物本身的理解，同时把这种理解概念化，将这些逻辑概念组织起来，构成一种对所观察的对象的内部结构和工作原理的便于理解的表达。

* + 1. 为什么要建模？

为了更好的理解业务模型和软件系统：帮助可视化软件系统（可视化建模）、为构造软件系统提供模板（可重用性）、将决策文档化（建模文档）。

* 1. RUP
     1. 什么是RUP（统一过程）

Rational Unified Process，统一过程，是一种软件工程方法，采用面向对象思想，是一种用例驱动的迭代式软件开发流程。统一过程归纳和集成了软件开发活动中的最佳实践，定义了软件开发过程中最重要的阶段和工作（四个阶段），定义了参加软件开发过程的各种角色和他们的职责等。

* + 1. RUP的四个阶段
       1. 初始阶段（Inception）

包括用户沟通和计划活动两个方面，强调定义和细化用例。该阶段结束之后达到第一个重要的里程碑（milestone）：生命周期目标（Lifecycle Objective, LCO）里程碑。

* + - 1. 细化阶段（Elaboration）

包括用户沟通和建模活动，重点是创建分析和设计模型，强调类的定义和体系结构的表示。该阶段结束之后达到第二个里程碑：生命周期结构（Lifecycle Architecture, LCA）里程碑。

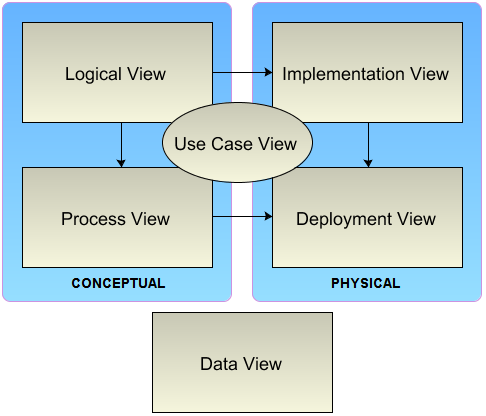
* + - 1. 构造阶段（Construction）

将设计转化为实现，并进行集成和测试。该阶段结束后达到第三个里程碑：初步可执行能力（Initial Operational Capability, IOC）里程碑。

* + - 1. 移交阶段（Transition）

将产品发布给用户进行测试评价，并收集用户的意见，之后再次进行迭代修改产品使之完善。该阶段完成之后达到第四个里程碑：产品发布（Product Release）里程碑。

* + 1. RUP的九个工作流
       1. 业务建模（Business Modeling）
       2. 需求（Requirements）
       3. 分析和设计（Analysis and design）
       4. 实现（Implementation）
       5. 测试（Test）
       6. 部署（Deployment）
       7. 配置和变更管理（Configuration and change management）
       8. 项目管理（Project management）
       9. 环境（Environment）
    2. RUP使用的“4+1”视图



* + 1. RUP中的6个最佳实践（six best practice）
       1. 迭代开发（Develop iteratively）
       2. 管理需求（Manage requirements）
       3. 使用基于构件的架构（Employ a component-based architecture）
       4. 可视化建模（Model software visually）
       5. 持续验证质量（Continuously verify quality）
       6. 管理变更（Control changes）
  1. 需求
     1. 用例分析的步骤
        1. 补充用例说明。
        2. 找出用例中的行为，把行为合理分配给各个类。
        3. 对每一个分析出来的类，描述其职责、属性以及类与类之间的关联，使其满足分析机制。
        4. 统一分析类。
        5. 查验分析过程和结果。
  2. 分析设计
     1. 分析和设计的区别是什么？

|  |  |
| --- | --- |
| 分析 | 设计 |
| 专注于理解问题； | 专注于理解解决方案； |
| 是理想化的设计； | 是具体的设计，包括相关操作和属性。 |
| 考虑行为层面的功能需求； | 还需考虑非行为层面（非功能性）的需求； |
| 需要分析系统架构； | 更接近真实代码； |
| 功能性需求； | 模型规模较大； |
| 模型规模较小； |  |

* + 1. 什么是分析类？简述三种分析类的区别。

分析类代表问题域中的简洁抽象，应该映射到真实世界的业务概念，包括边界类、控制类和实体类。

* + - 1. 边界类（Boundary class）

负责处理用户和系统之间交互协作的类。

* + - 1. 控制类（Control class）

负责处理用例行为操作的类。

* + - 1. 实体类（Entity class）

负责处理建模事物的持久化数据的类。

* 1. 架构
     1. 什么是系统架构？

系统的架构是开发过程中最重要的一项制品，它定义了系统的概念、结构，是管理开发过程、展开系统的重要指导。

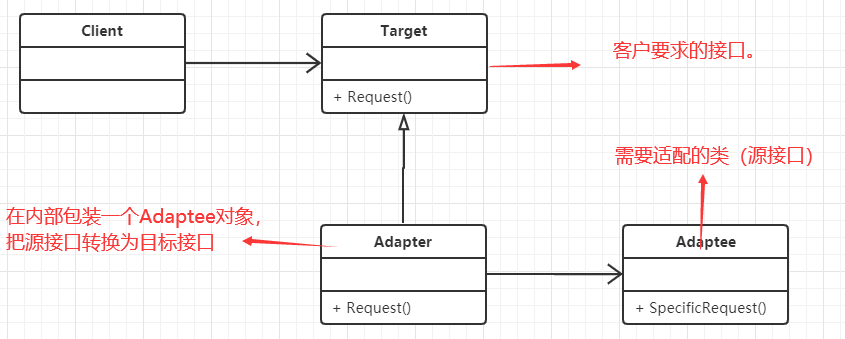
* + 1. 什么是以架构为中心的开发过程？

统一开发过程强调架构是项目小组给系统定形的中心，因为一个模型难以反映系统的各个方面，统一开发过程多个模型和视图。

## 设计模式

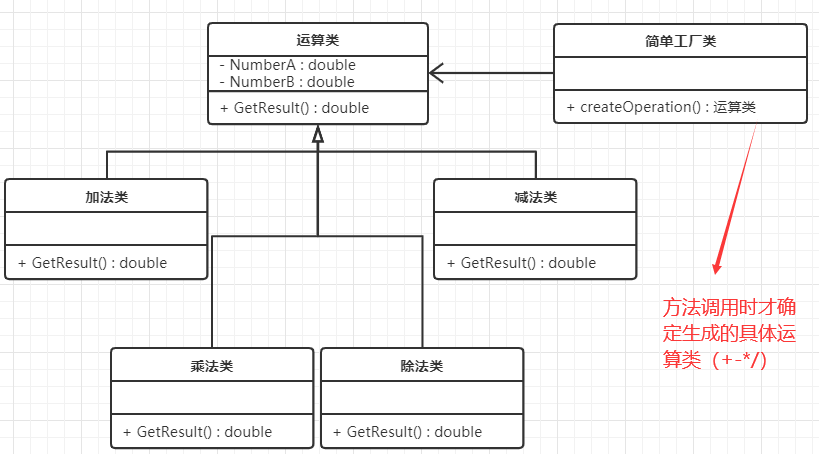
* 1. 适配器模式（Adapter）

适配器模式将一个类的接口转换成客户希望的另外一个接口，使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。



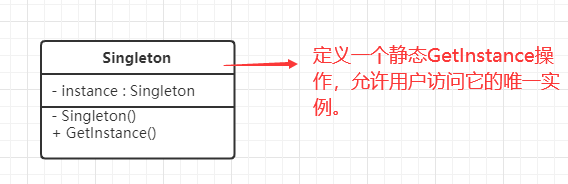
* 1. 工厂模式（Factory）

定义一个创建对象的接口，让其子类自己决定实例化哪一个工厂类。工厂模式使其创建过程延迟到子类进行。



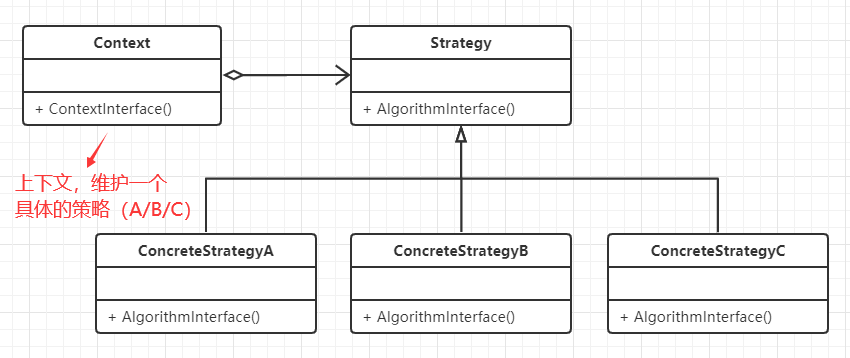
* 1. 单例模式（Singleton）

保证一个类仅有一个实例，并提供一个访问它的全局访问点。



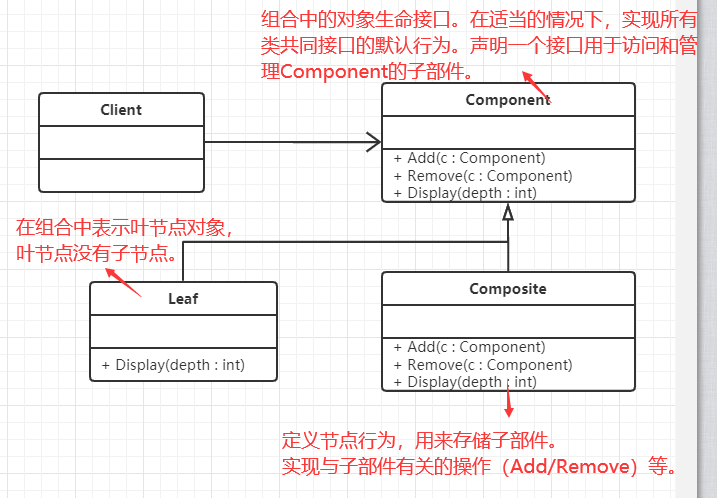
* 1. 策略模式（Strategy）

策略模式时一种定义一系列算法的方法，从概念上来看，所有这些算法完成的都是相同的工作，只是实现不同。它可以以相同的方式调用所有的算法，减少了各种算法类与算法类之间的耦合。



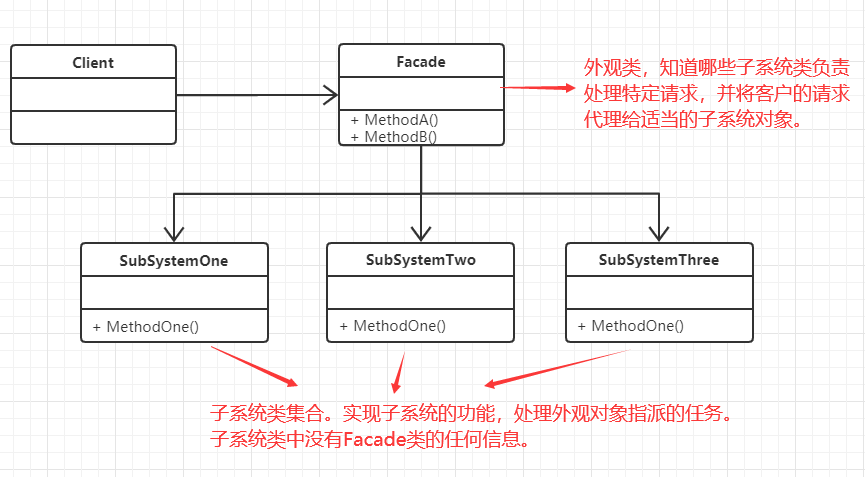
* 1. 组合模式（Composite）

组合模式将对象组成树形结构以表示“部分-整体”的层次结构，使得用户对单个对象和组合对象的使用具有一致性。



* 1. 外观模式（Facade）

外观模式为子系统中的一组接口提供一个一致的界面。该模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。



* 1. 观察者模式/发布-订阅模式(Observer/Publish-Subscribe)

观察者模式定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题对象。这个主题对象在状态发生变化时，会通知所有观察者对象，使它们能够自动更新自己。

