架构建模

用户指南

文档版本	01
发布日期	2023-10-19

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

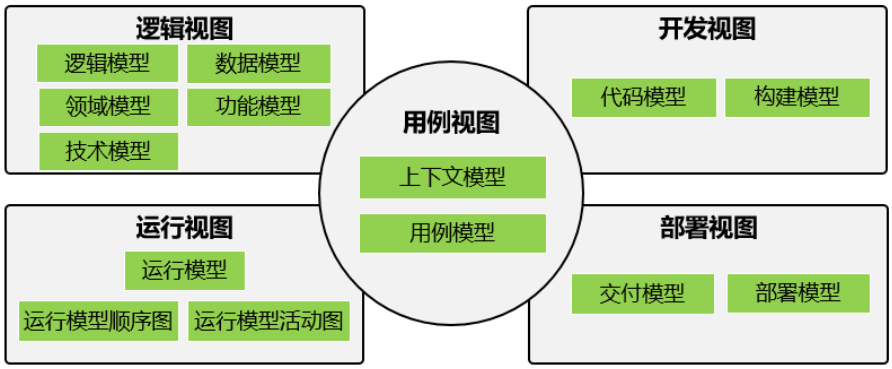
华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 4+1 视图建模

1.1 概述

架构设计4+1视图模型结构如下所示：



4+1视图是一组相关联模型的集合，从不同的视角，反映不同利益干系人的关注点。通过逻辑、开发、部署、运行4个典型视角描述系统的各个切面，以用例串接和验证各切面设计。

视图类型	描述
逻辑视图	逻辑视图面向系统逻辑分析和设计，描述系统逻辑结构的视图，主要解决系统分析和设计的问题，它描述系统的业务上下文、系统的逻辑分解，以及分解出的逻辑元素间的关系。
开发视图	开发视图面向系统开发及软件管理，描述系统代码结构，构建结构的视图，主要解决系统技术实现和开发的问题，它依托逻辑视图，描述代码、构建结构。
运行视图	运行视图面向系统运行，描述系统启动过程、运行期交互的视图，主要解决系统运行期交互，描述各可执行交付件在运行期的交互关系。

视图类型	描述
部署视图	部署视图面向系统部署，描述系统的交付、安装、部署的视图，主要解决系统安装部署的问题，描述系统的交付、安装、部署关系。
用例视图	用例视图以用例作为驱动元素，驱动和验证其他四个视图的设计，用例视图不增加设计元素，仅增加用例作为输入，因此作为+1视图。

1.2 用例视图

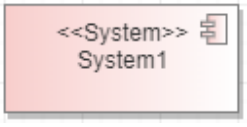
1.2.1 概述

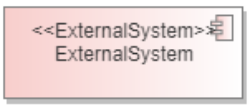
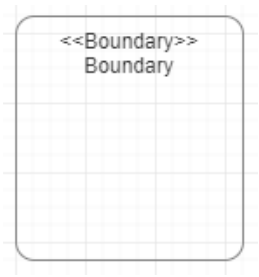
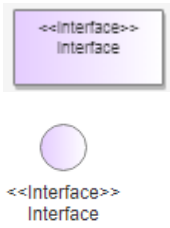
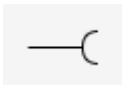
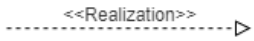
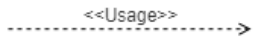
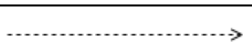
用例视图以用例作为驱动元素，驱动和验证其他四个视图的设计，用例视图不增加设计元素，仅增加用例作为输入，因此作为+1视图。

模型类别	描述
上下文模型（必选）	上下文模型描述系统和外部环境（包括人、系统及外部实体）之间的关系，依赖和交互。通过上下文模型可以显示定义系统的范围、职责、边界。
用例模型（必选）	用例模型描述系统的关键用例和交互场景，用于描述系统与外界的交互关系。其中关键用例部分主要描述系统基本的业务用例模型，以及增量版本中影响架构的用例模型；而交互场景描述系统与外部实体这间的复杂的交互关系图，采用UML的顺序图进行描述绘制，以帮助描述隐含的需求和约束，以及系统的验证。

1.2.2 上下文模型

元素介绍

元素名	图标	含义
Actor		角色，是与系统交互的人或事物。
System		系统。

元素名	图标	含义
ExternalSystem		外部系统、设备或者其它系统。
Boundary		边界。
Interface		接口，可以是单个接口，也可以是抽象的一组接口的组合。
Provided Interface		提供的接口。
Required Interface		使用的接口。 Required Interface和Provided Interface一般是配套使用，一方提供接口，另一方使用，使用Association连线连接两边后，会自动合并。
Realization		实现，是一种类与接口的关系，表示类是接口所有特征和行为的实现。
Usage		使用，是一种使用的关系。表明一个模块在运行的时候，需要使用另外一个模块。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。
Association		关联，是一种拥有的关系，它使一个类知道另一个类的属性和方法。

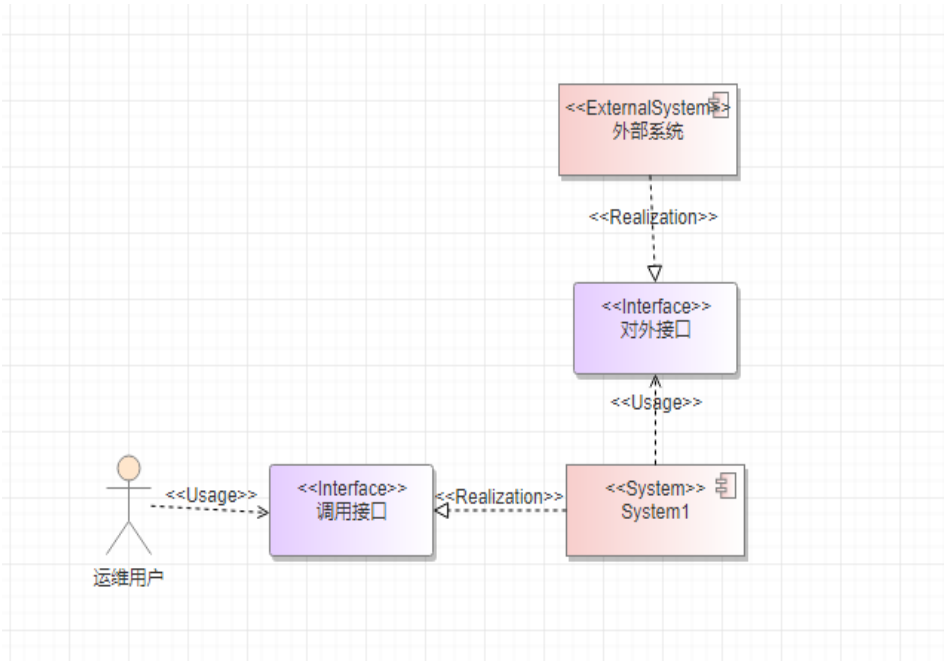
建模步骤

步骤1 创建上下文模型。

可使用初始化创建的上下文模型或者创建新的上下文模型，在目录节点右键“新增图”，如果一个系统的交互的外部角色过多时，不适合在一张上下文模型图中建模时，用户可根据外部角色的分类或者产品的应用场景创建不同的上下文模型。

步骤2 建立系统与外部角色的关系。

在上下文模型中描述系统与外部角色的关系通过接口体现，不直接使用连线表示；在上下文模型中需要定义外部角色、交互接口、外部系统、系统，其中系统如果在逻辑模型中已经定义过，则在上下文模型中不能再重复定义，从逻辑模型中引用至上下文模型中即可。

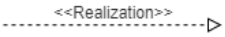
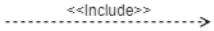


----结束

1.2.3 用例模型

元素介绍

元素名	图标	含义
UserCase		用例，代表的是一个完整的功能。

元素名	图标	含义
Actor		角色，是与系统交互的人或事物。
Boundary		边界。
Realization		实现，是一种类与接口的关系，表示类是接口所有特征和行为的实现。
Use		使用关系，指示一个元素需要另一个元素执行一些交互。在用例图中，表示建模参与者如何使用系统功能。
Association		关联，是一种拥有的关系，它使一个类知道另一个类的属性和方法。
Generalization		泛化，是一种继承关系，一个类（通用元素）的所有信息（属性或操作）能被另一个类（具体元素）继承，继承某个类的类中不仅可以有属于自己的信息，而且还拥有了被继承类中的信息。
Include		包含，包含关系描述的是一个用例需要某种功能，而该功能被另外一个用例定义， 那么在用例的执行过程中，就可以调用已经定义好的用例。
Extend		扩展，用例之间的关系， 是指用例功能的延伸，相当于为基础用例提供一个附加功能。

前提条件

用例模型中的Actor需要在上下文模型中定义，再引用至用例模型中，不能在用例模型上重新定义Actor。

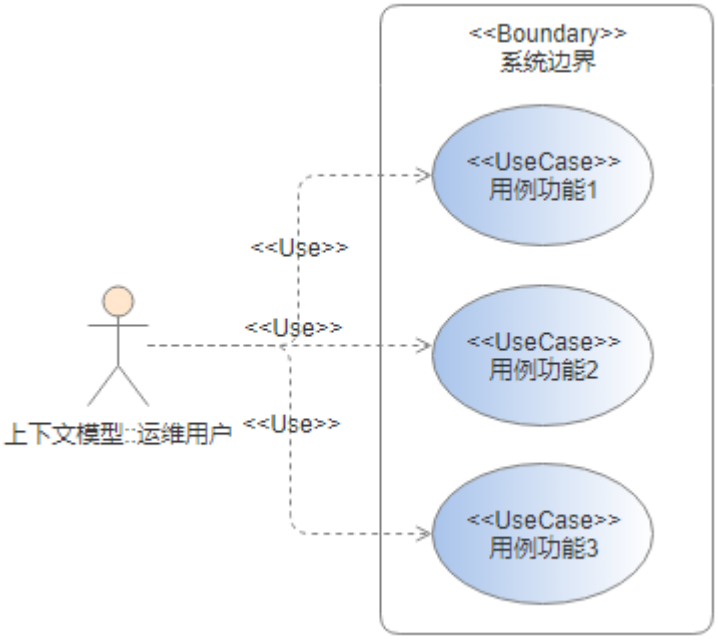
建模步骤

步骤1 创建用例模型。

可以使用工程初始化建好的用例模型或者在其它目录节点右键菜单中“新增图”，创建新的用例模型，如果用例场景较多，可以创建多个用例模型。

步骤2 画用例模型。

用例模型包含系统基本业务的用例模型、以及增量版本中影响架构的用例模型，从上下文模型中将要用到的Actor角色插入到用例模型图中，再从工具箱中拖入要定义的Use Case元素，和系统边界元素，再建立关系，Actor与用例用的是Use连线关系。



----结束

1.3 逻辑视图

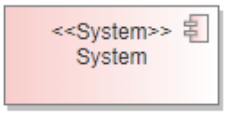
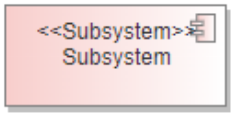
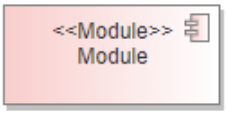
1.3.1 概述

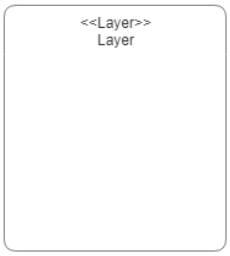
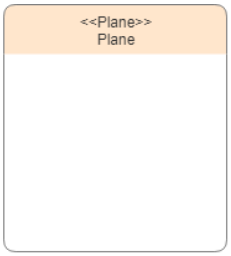
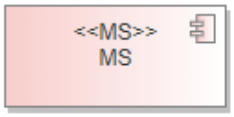
逻辑视图面向系统逻辑分析和设计，描述系统逻辑结构的视图，主要解决系统分析和设计的问题，它描述系统的业务上下文、系统的逻辑分解，以及分解出的逻辑元素间的关系。

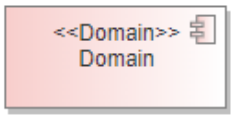
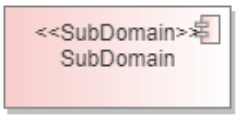
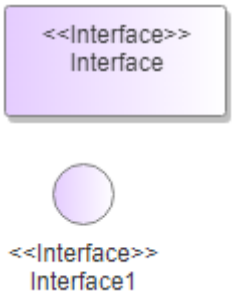
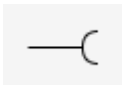
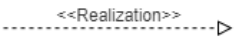
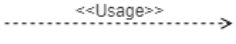
模型类别	描述
逻辑模型（必选）	逻辑模型描述系统的逻辑功能模块分解，将系统分解为相应的逻辑功能元素，并描述各逻辑功能元素之间的关系。
数据模型（强数据场景必选）	数据模型定义系统的关键数据设计，包括关键数据结构设计、数据流，以及数据所有权等。
领域模型（可选）	领域模型描述业务域的概念及其关系，是立足于于业务域的分析模型，它通过业务问题域的分析和建模，抽象出领域概念，建立统一的业务语言，从而指导后续的架构设计工作。
功能模型（可选）	功能模型描述按功能分解出特性、功能组、功能元素，以及它们之间的依赖关系。
技术模型（必选）	技术模型定义系统采用的关键技术部件和技术栈，包括整体框架技术，公共机制，基础设施，公共服务/组件，以及各逻辑功能元素的技术方案等。

1.3.2 逻辑模型

元素介绍

元素名	图标	含义
System		广义上，系统是指提供给市场，被客户注意、获取、使用或者消费，并能满足客户某种需求的载体，包括各种有形的物品、无形的电子产品、服务及观念。 狭义上，系统指能独立满足客户某种需求、并符合客户的理解及业界划分习惯的实体。
SubSystem		子系统是一个独立的能够满足特定功能的组合，通过一个或多个它所实现的接口来提供行为。 1. 完全封装自己的内容，通过接口提供行为。 2. 可由组件/模块或更小的子系统组成。 3. 是平台或更大的子系统的组成部分，与其他子系统相对独立。 4. 必须是交付件（能够支持异步开发和外包）。
Module		（IEEE 610.12-1990）系统中一个逻辑上可分离的部分。系统设计中模块特指系统设计阶段输出的系统最小分解部件，系统设计阶段将模块当作黑盒、不涉及模块的内部结构，但要明确给出模块的功能、模块之间的接口。

元素名	图标	含义
Service		服务，是指具备明确的业务特征，由一个或多个关联紧密的微服务组成，可直接面向客户/用户进行打包、发布、部署、运维的软件单元。用户可以从业务特征、安装部署、监控运维的角度感知到服务的存在。规模上介于Subsystem与FM（Function Module功能模块）之间的逻辑架构模型元素。Service的功能更加内聚，对外依赖少，接口稳定。
Component		组件，可独立加载、部署和运行的二进制代码，采用轻量级通讯机制、松耦合高内聚的软件架构构建单元，部署时不能跨节点类型部署（计算机百科全书：组件是软件系统中具有相对独立功能、接口由契约指定、和语境有明显依赖关系、可独立部署、可组装的软件实体）。
SDK		Software Development Kit，软件开发工具包。
Layer		层，辅助图形，不属于架构元素，一般是分层设计，例如网络层、应用层，常见的7层网络模型。
Plane		面，同Layer属于辅助图形，不属于架构元素，在嵌入式系统里常见用户面、管理面等。
MS		MicroService，微服务，是指可独立设计开发部署测试、粒度较小、采用轻量级通讯机制、松耦合高内聚的软件单元。一般来说，用户感知不到微服务的存在。

元素名	图标	含义
Domain		域，用于在架构表达、开发管理、对外介绍的过程中，表达系统的层次关系或内部分组，一般由多个服务组成，可以是一级（域）或多级（域/子域，或者域/1级子域/2级子域…）。域和子域不对应实际的设计开发实体，可以根据需要灵活调整。域这个概念，来自于云化产品。以前逻辑架构的实体，这个实体一般指的是子系统，但这个架构实体，带有比较严重嵌入式情节。为此云化产品，习惯用域来表示逻辑架构的实体。
SubDomain		子域，用于在架构表达、开发管理、对外介绍的过程中，表达系统的层次关系或内部分组，域和子域交互使用。
Interface		接口，可以是单个接口，也可以是抽象的一组接口的组合。 **圆形接口与矩形接口意义相同，仅形状不同。**
Provided Interface		暴露接口。提供接口动作，和Required Interface之间建立Association，表明一个组件提供另外一个组件需要的接口。
Required Interface		请求接口。和Provided Interface之间建立Association，表明一个组件需要的接口是由另外一个组件提供的。
Composition		组合，是整体与部分的关系，但部分不能离开整体而单独存在。
Aggregation		聚合，是整体与部分的关系，且部分可以离开整体而单独存在。
Realization		实现，是一种类与接口的关系，表示类是接口所有特征和行为的实现。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。
Usage		使用，是一种使用的关系。表明一个模块在运行的时候，需要使用另外一个模块。
Association		关联，是一种拥有的关系，它使一个类知道另一个类的属性和方法。

建模步骤

参考建议步骤是按逐层分解的方式画图设计，示例步骤分解顺序为：System->Susbsystem->Component->Module，其它的分解顺序结构可参考该方式调整即可，如果产品线有统一的分解规范要求，以产品线要求规范步骤为准。

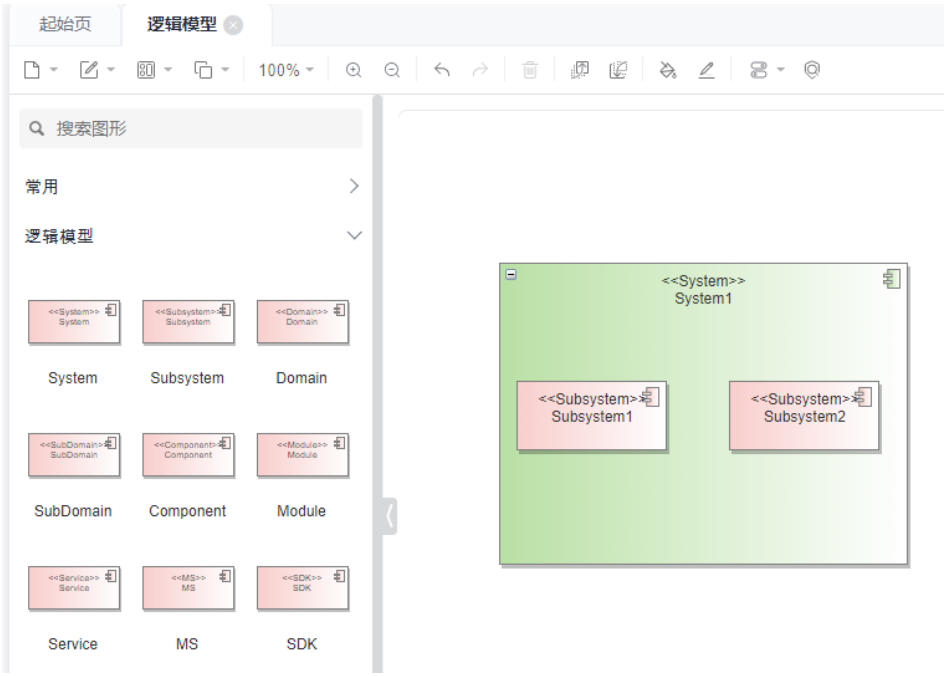
步骤1 创建0层逻辑模型图。

工程初始化创建时会在“逻辑视图>逻辑模型”包目录下默认创建一个逻辑模型图，可当作0层逻辑模型，如果是非初始化结构建目录，则选择要创建图的包节点，单击包后的菜单，选择“新建图”。

图类型选择“4+1视图>逻辑视图>逻辑模型”，输入图名称，单击保存即可。

步骤2 创建0层模型逻辑元素。

在0层模型图创建完后，从工具箱中拖入System、Subsystem元素到0层逻辑模型图中。

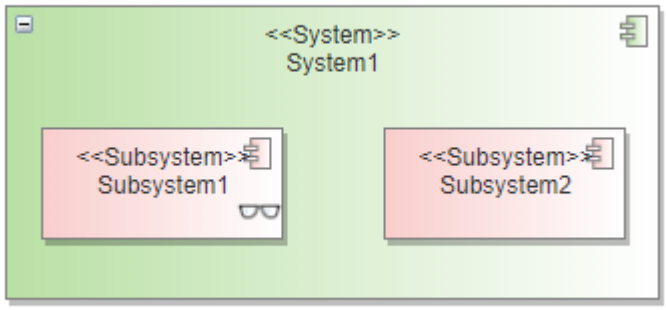


步骤3 创建1层逻辑模型和逻辑元素。

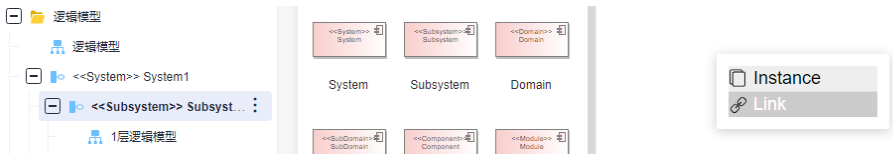
在Subsystem元素下创建子图，子图即为1层逻辑模型，从工程树上将Subsystem元素拖入到1层逻辑模型图中，选择link方式，然后在该Subsystem元素下添加Component元素，建立逻辑关系。

在0层模型上选中Subsystem元素右键“新增图”或者在工程树上的Subsystem元素节点右键“新增图”，在新增图界面图类型仍为逻辑模型。

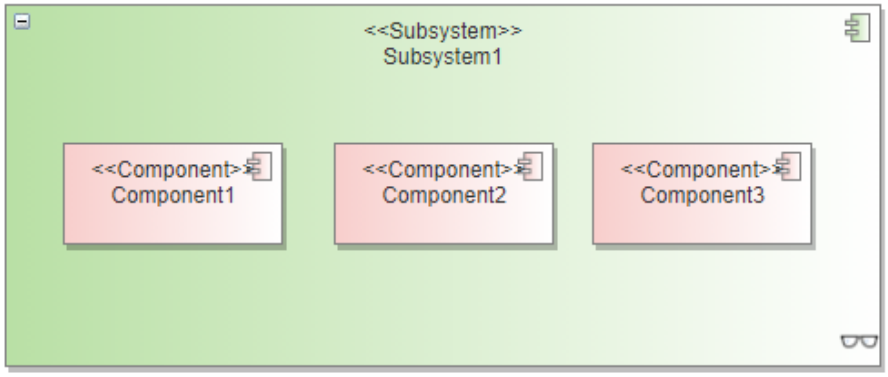
元素创建完子图后，在所有图中元素图形右下角有一个眼镜图标，双击该图标可快速打开这个元素的子图。



将Subsystem1以Link的形式拖入1层逻辑模型。

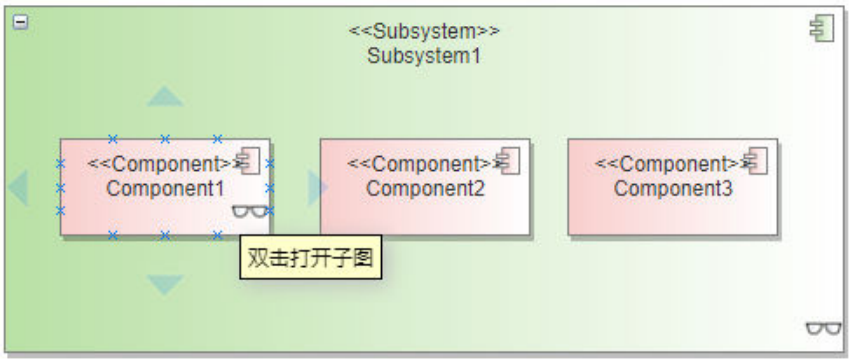


新增的Component组件元素再从工具箱中拖入到图中的Subsystem元素内部，构成包含的父子关系。



步骤4 创建2层逻辑模型。

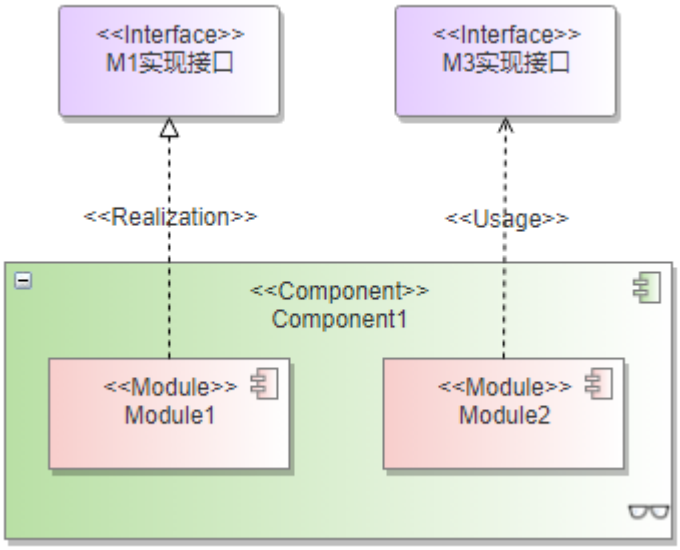
参考1层模型创建方式，2层逻辑模型是基于Component1创建子图。



在Component1的子图中引用该父Component1元素到图中，再到图中的从工具箱中创建Module元素到组件下，构成包含的父子关系，在2层模型中会对Module定义对外接口，和使用的接口。

M1暴露实现的接口，提供给外模块调用，连线关系使用Realization。

Module2使用其他模块实现提供的接口，以引用的方式拖入图中，连线关系为Usage。

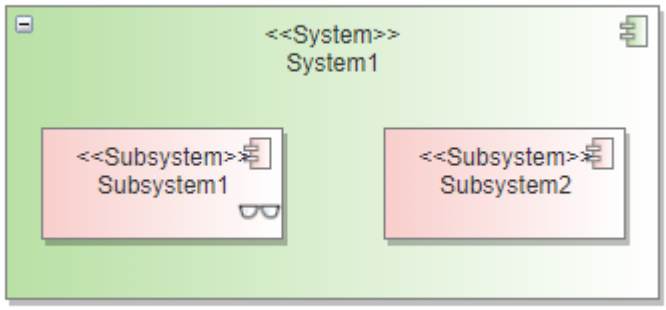


----结束

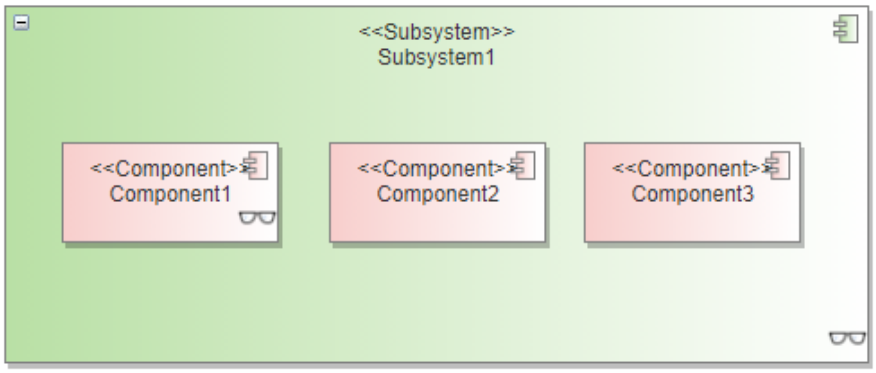
模型示例

最终实现的一个由0层到1层、2层依次展开各层级结构的一个分解逻辑模型图。

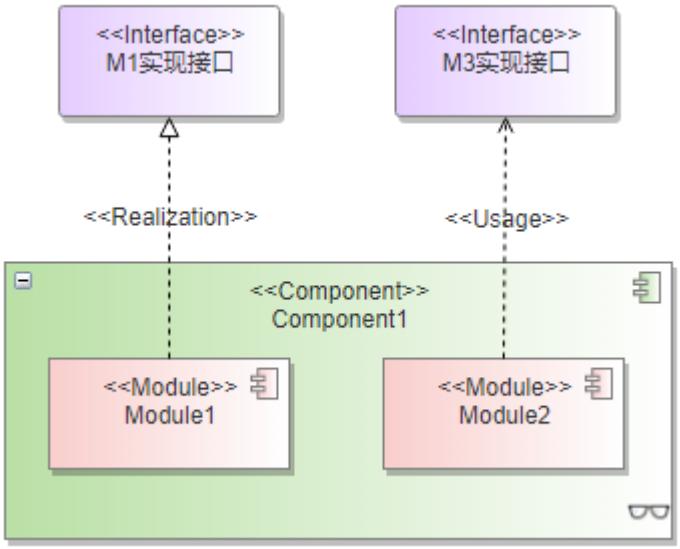
0层逻辑模型。



1层逻辑模型。



2层逻辑模型。

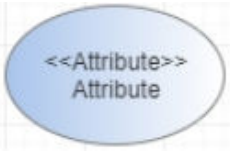
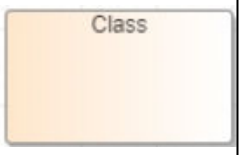
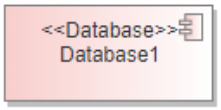
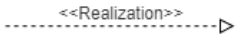
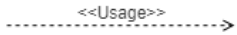


1.3.3 数据模型

数据模型定义系统的关键数据设计，包括关键数据结构设计、数据流，以及数据所有权等。

元素介绍

元素名	图标	含义
Entity		实体，该实体建立了一种和数据库表的映射关系。

元素名	图标	含义
Attribute		属性。
Class		类。
Database		数据库。
Composition		组合，是整体与部分的关系，但部分不能离开整体而单独存在。
Aggregation		聚合，是整体与部分的关系，且部分可以离开整体而单独存在。
Realization		实现，是一种类与接口的关系，表示类是接口所有特征和行为的实现。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。
Usage		使用，是一种使用的关系，表明一个模块在运行的时候，需要使月另外一个模块。
Association		关联，是一种拥有的关系，它使一个类知道另一个类的属性和方法。
Generalization		泛化，是一种继承关系，一个类（通用元素）的所有信息（属性或操作）能被另一个类（具体元素）继承，继承某个类的类中不仅可以有属于自己的信息，而且还拥有了被继承类中的信息。

1.3.4 领域模型

领域模型描述业务域的概念及其关系，是立足于业务域的分析模型，它通过业务问题域的分析和建模，抽象出领域概念，建立统一的业务语言，从而指导后续的架构设计工作。

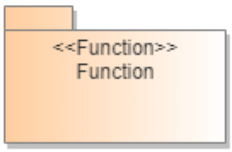
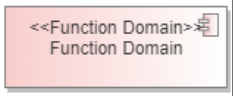
元素介绍

元素名	图标	含义
Domain		域，用于在架构表达、开发管理、对外介绍的过程中，表达系统的层次关系或内部分组，一般由多个服务组成，可以是一级（域）或多级（域/子域，或者域/1级子域/2级子域…）。 域和子域不对应实际的设计开发实体，可以根据需要灵活调整。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。
Association		关联，是一种拥有的关系，它使一个类知道另一个类的属性和方法。

1.3.5 功能模型

功能模型描述按功能分解出特性、功能组、功能元素，以及它们之间的依赖关系。

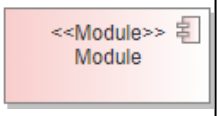
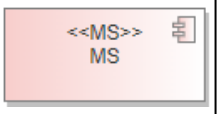
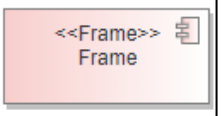
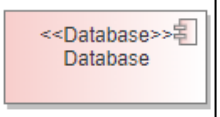
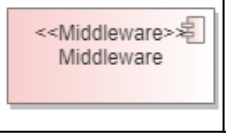
元素介绍

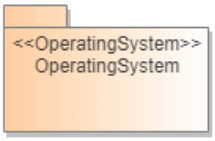
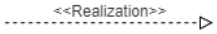
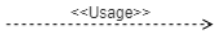
元素名	图标	含义
Function		功能。
Feature		特性。
Function Domain		功能域。
Composition		组合，是整体与部分的关系，但部分不能离开整体而单独存在。
Aggregation		聚合，是整体与部分的关系，且部分可以离开整体而单独存在。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。

1.3.6 技术模型

技术模型定义系统采用的关键技术部件和技术栈，包括整体框架技术，公共机制，基础设施，公共服务/组件，以及各逻辑功能元素的技术方案等。

元素介绍

元素名	图标	含义
Module		（IEEE 610.12-1990）系统中一个逻辑上可分离的部分。系统设十中模块特指系统设计阶段输出的系统最小分解部件，系统设计阶段将模块当作黑盒，不涉及模块的内部结构，但要明确给出模块的功能、模块之间的接口。
Service		服务，是指具备明确的业务特征，由一个或多个关联紧密的微服务组成，可直接面向客户/用户进行打包、发布、部署、运维的软件单元。用户从业务特征安装部署、监控运维的角度感知到服务的存在。规模上介与Subsystem与FM之间的逻辑架构模型元素。Service的功能更加内聚，对外依赖少，接口稳定。
Component		组件，可独立加载、部署和运行的进制代码，采用轻量级通讯机制、松耦合高内聚的软件架构构建单元，部署时不能跨节点类型部署（计算机百科全书：组件是软件系统中具有相对独立功能、接口由契约指定、 和语境有明显依赖关系、可独立部署、 可组装的软件实体）。
MS		是指可独立设计开发部署测试、粒度较小采用轻量级通讯机制、松耦合高内聚的软件单元。一般来说，用户感知不到微服务的存在。
Platform		表示逻辑对象引用的平台，包括名称（Name）、描述（Description）、架构负责人（Design Owner）、标准名称（artifactName）、版本号（artifactVersion）、平台类型（cpuType）、下载地上（repo）等。
Frame		框架，包含自研或开源框架。
DataBase		数据库。
Middleware		中间件。

元素名	图标	含义
OperatingSystem		操作系统。
Realization		实现，是一种类与接口的关系表示类是接口所有特征和行为的实现。
Usage		使用，是一种使用的关系，表明一个模块在运行的时候，需要使用另外一个模块。

1.4 开发视图

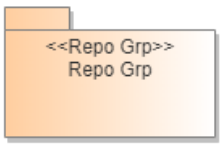
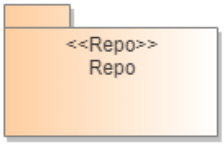
1.4.1 概述

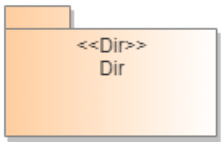
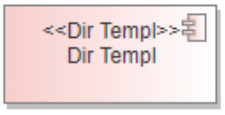
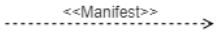
开发视图面向系统开发及软件管理，描述系统代码结构，构建结构的视图，主要解决系统技术实现和开发的问题，它依托逻辑视图，描述代码、构建结构。

模型类别	描述
代码模型(必选)	代码模型定义代码结构以及代码元素逻辑模型中逻辑元素的对应关系，建立逻辑元素到代码仓或者代码目录的映射关系，以实现软件源代码的显示管理。
构建模型(必选)	构建模型定义软件编译构建结构及工具链，构建模型建立代码到运行期文件的映射和追溯关系。

1.4.2 代码模型

元素介绍

元素名	图标	含义
Repo Grp		代码仓组是代码模型分组辅助元素，不对应具体的代码仓，仅表示一个集合。 一个设计对象对应多个代码仓的情况，建议使用Repo Grp标识出来，供构建模型整体引用。
Repo		表示一个代码仓。

元素名	图标	含义
Dir		表示一个代码目录。 Dir不单独出现，定是挂在某个代码仓或者上级目录之下。
File		表示代码仓中的文件，名称中包含文件名+文件类型后缀。
Dir Templ		目录模板。
Manifest		Repo和对应的逻辑设计对象使用Manifest连接。 表示由此代码仓的代码实现此设计对象的功能。连线方向由代码元素指向逻辑元素。
Composition		组合，是整体与部分的关系，但部分不能离开整体而单独存在。菱形箭头为整体所在一边。
Aggregation		聚合，是整体与部分的关系，且部分可以离开整体而单独存在。菱形箭头为整体所在一边。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。
Usage		使用，是一种使用的关系。表明一个模块在运行的时候，需要使用另外一个模块。
Build From		构建关系，表示当前构建结果从某一代码目录或者代码文件构建而来，仅适于构建元素与代码元素之间的关系，连线方向由构建元素指向代码元素。

前提条件

因为代码模型主要是描述创建出来的代码元素与逻辑元素的Manifest连线关系，所以在代码模型设计前必须要先完成逻辑模型的设计。

建模步骤

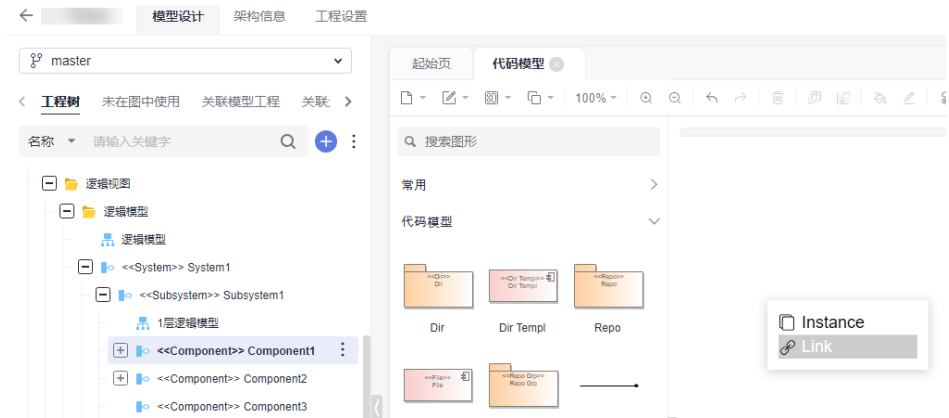
步骤1 创建代码模型图。

创建新的代码模型图或者在已有的代码模型图中进行画图设计，如果设计内容过多，可根据实际情况将内容进行拆分，创建多个代码模型图，在对应的代码模型图中去建立关系。

步骤2 引用逻辑元素到代码模型。

在代码模型中不能创建新的逻辑元素，必须要从逻辑模型中引用到代码模型中，引用逻辑元素的操作方式有两种

方式一：直接从工程树上将逻辑元素节点拖入到打开的代码模型图中，选择Link方式。

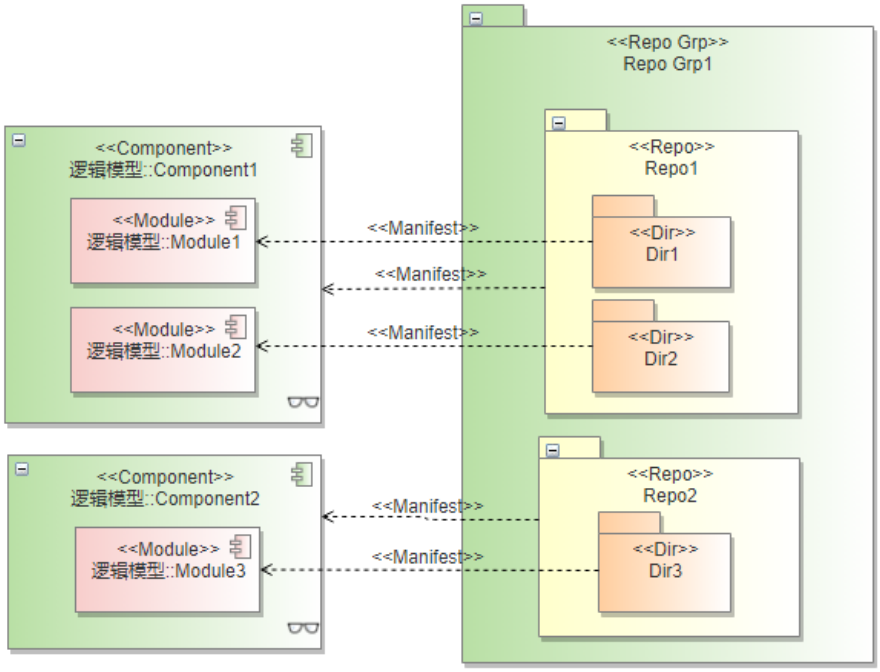


方式二：在逻辑模型图中按Ctrl键**多选**或者**框选多个**逻辑元素Ctrl+C复制，然后再到代码模型中Ctrl+V粘贴，粘贴方式选择引用方式，可以保留嵌套组合的结构样式。

上述两种方式都可以将逻辑元素引用到代码模型图中，在代码模型中只需要引用需要建立映射关系的逻辑元素即可。

步骤3 创建代码元素并与逻辑元素建立Manifest连线关系。

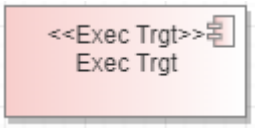
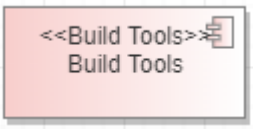
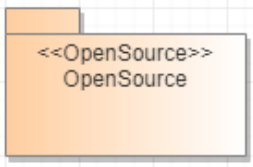
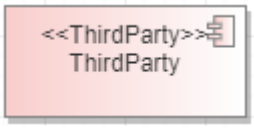
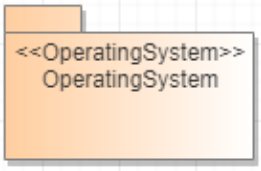
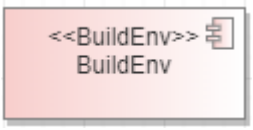
在步骤2中将逻辑元素引用到代码模型中后，再从工具箱中拖入代码仓元素，如果存在一个代码仓组下的多个代码仓元素，可以选代码仓组元素，将多个代码仓元素包含起来，如果具体模块对应的是代码仓中某一目录中的代码，则需要在对应的代码仓元素中创建Dir目录元素，再建立对应逻辑元素与代码元素的Manifest关系。

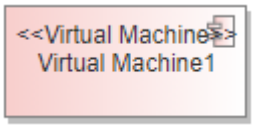
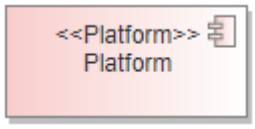


----结束

1.4.3 构建模型

元素介绍

元素名	图标	含义
Exec Trgt		表示逻辑对象构建的二进制结果（.so/.bin/rpm等）。
Compile Trgt		表示逻辑对象构建的二进制编译结果（.o/obj/.a等），专指提供二进制编译结果给其他对象使用的场景，其他场景不用体现Compile Trgt。
Build Tools		表示逻辑对象构建时所需的构建工具。 【建议】Build Tools嵌套可以作为分组来区分不同类型的工具集合，作为构建时引用。
OpenSource		表示逻辑对象构建时所需的开源软件代码。
ThirdParty		表示构建中所使用的第三方元素。
OperatingSystem		表示逻辑对象构建时所需的操作系统。
BuildEnv		表示构建时适用的构建环境信息。
Container		表示逻辑对象构建时所需的容器。

元素名	图标	含义
Virtual Machine		表示逻辑对象构建时所需的虚拟机。
Platform		表示逻辑对象引用的平台。
Composition		组合，是整体与部分的关系，但部分不能离开整体而单独存在。
Aggregation		聚合，是整体与部分的关系，且部分可以离开整体而单独存在。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。
Usage		使用，是一种使用的关系。表明一个模块在运行的时候，需要使用另外一个模块。
Deployed To		部署关系是一种依赖关系，在部署图中，指一个工件被部署到一个节点或可执行目标上。
Build From		构建关系，表示当前构建结果从某一代码目录或者代码文件构建而来，仅适于构建元素与代码元素之间的关系，连线方向由构建元素指向代码元素。

前提条件

因为构建模型主要是描述创建出来的构建元素与代码元素的Build From构建关系，所以在画构建模型设计前必须要先完成代码模型的设计。

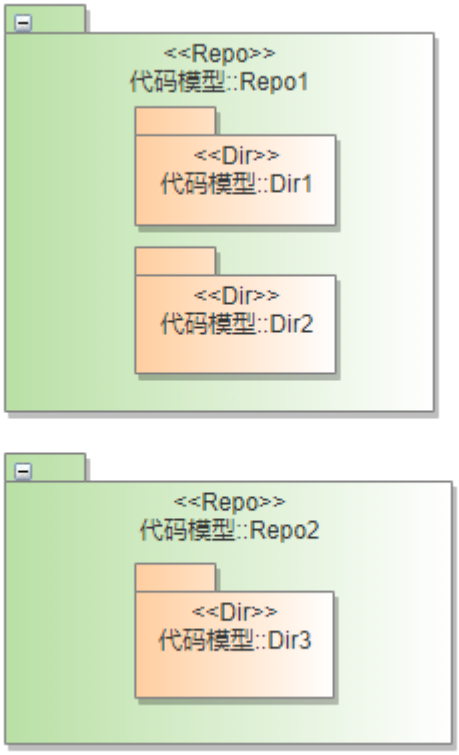
建模步骤

步骤1 创建构建模型。

创建新的构建模型图或者在已有的构建模型图中进行画图设计，如果设计内容过多，可根据实际情况将内容进行拆分，创建多个构建模型图，在对应的构建模型图中去建立关系。

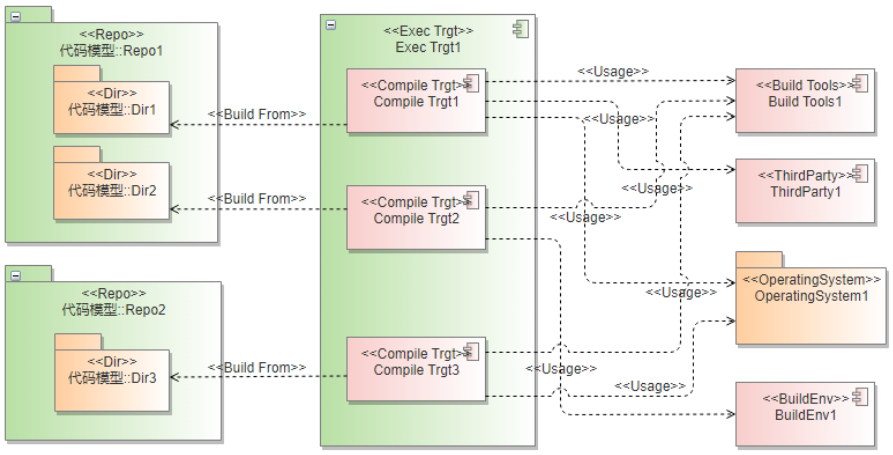
步骤2 引用代码元素到构建模型。

将代码元素引用到构建模型中跟代码模型中的步骤2一样，有两种方式，从工程树上将代码元素拖入到构建模型图中选link方式引用；另一种从代码模型图中多选复制元素，以引用方式粘贴到构建模型图中。



步骤3 建立代码元素与构建元素的Build From构建关系。

在步骤2中将代码元素引用到构建模型图后，再从工具箱中构建模型图形库中拖入构建元素，创建与代码元素需要建立关系的构建元素，并建立构建元素与代码元素的Build From关系，同时需要创建一些构建过程中构建元素使用到的构建工具和依赖的构建环境、平台等信息，并建其中的连线关系。



----结束

1.5 部署视图

1.5.1 概述

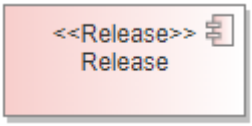
部署视图面向系统部署，描述系统的交付、安装、部署的视图，主要解决系统安装部署的问题，描述系统的交付、安装、部署关系。

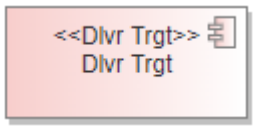
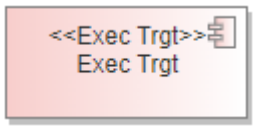
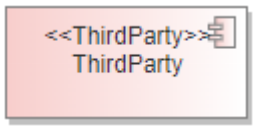
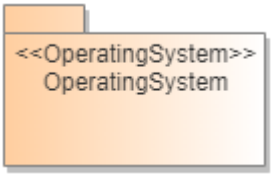
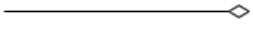
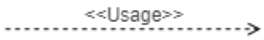
表 1-1 部署视图

模型类别	描述
交付模型(必选)	交付模型定义的是从构建结果和外部软件一起打包成最终交付给客户的Release Offering的模型设计过程。
部署模型(必选)	部署模型定义产品的部署关系，它依托于构建模型或交付模型，描述每个构建文件或者交付件以及相应的软件部署实体的部署依赖关系和部署约束。

1.5.2 交付模型

元素介绍

元素名	图标	含义
Release		指产品最终发布的release版本，按照公司发布版本命名规定release中自带版本号。

元素名	图标	含义
Dlvr Trgt		指通过Exec Trgt、Exec Trgt+DlvrTrgt、Dlvr Trgt+外部软件打包后的package.Dlvr Trgt 一般是tar/gz包。
Exec Trgt		来源于构建视图中的构建结果，一般场景下不在交付模型图中创建该该元素，都是从构建模型中引用到交付模型使用。
ThirdParty		需要作为软件一起打包交付给客户的第三方件。
OperationSystem		需要作为软件一起打包交付给客户的操作系统。
Package Tools		打包工具，在打包过程使用到的工具都可以用该元素表示，以名称作区分。
Composition		组合，是整体与部分的关系，但部分不能离开整体而单独存在。
Aggregation		聚合，是整体与部分的关系，且部分可以离开整体而单独存在。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。
Usage		使用，是一种使用的关系。表明一个模块在运行的时候，需要使用另外一个模块。

前提条件

因为交付模型主要是描述构建模型中的结构元素打包成交付文件的过程，所以必须先完成构建模型的设计才能进行交付模型。

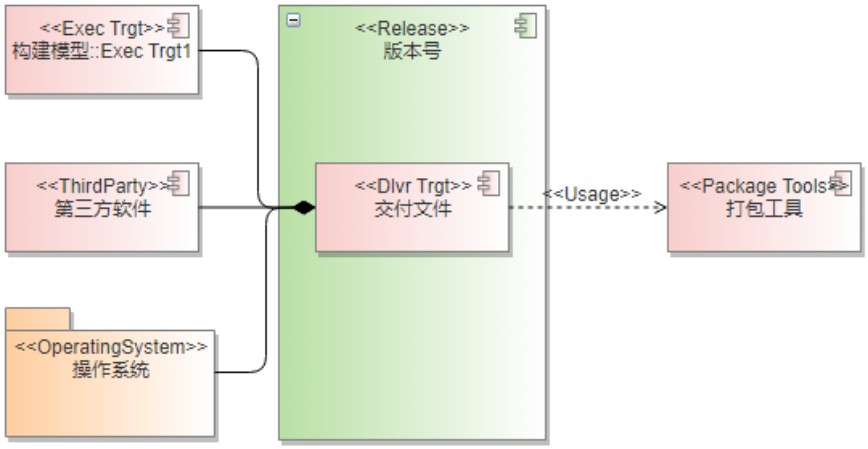
建模步骤

步骤1 创建交付模型。

创建新的交付模型图或者在已有的交付模型图中进行画图设计，如果设计内容过多，可根据实际情况将内容进行拆分，创建多个交付模型图，在对应的交付模型图中去建立关系。

步骤2 建立构建元素与交付元素的组合关系。

将构建模型中生成的构建元素引用到交付模型图中，并创建打包所需要的第三方软件或者操作系统，如果涉及打包工具，也可以在图中描述。

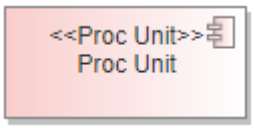
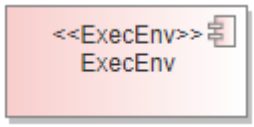
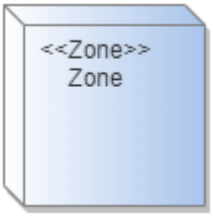
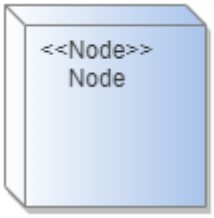
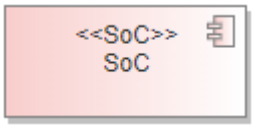
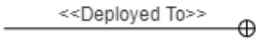


----结束

1.5.3 部署模型

元素介绍

元素名	图标	含义
FRU		现场可更换单元（Field-Replaceable Unit）。

元素名	图标	含义
Proc Unit		处理单元（Process Unit）。
Process		进程。
ExecEnv		执行环境（Execution Environment），可以是VM、docker等。
Zone		部署区域。
Node		部署节点。
SoC		片上系统（System-On-a-Chip）。
Deployed To		部署关系是一种依赖关系，在部署图中，指一个工件被部署到一个节点或可执行目标上。
Composition		组合，是整体与部分的关系，但部分不能离开整体而单独存在。
Aggregation		聚合，是整体与部分的关系，且部分可以离开整体而单独存在。
CommunicationPath		通信路径。定义两个部署目标能够交换信号和消息的通信路径。

前提条件

部署模型描述产品打包交付件的部署场景，所以画部署模型需要完成前面的构建模型或交付模型。

因为有些特殊产品没有交付打包过程，只有构建过程，在部署时使用的是构建过程生成文件来部署到部署模型中，描述部署的场景。

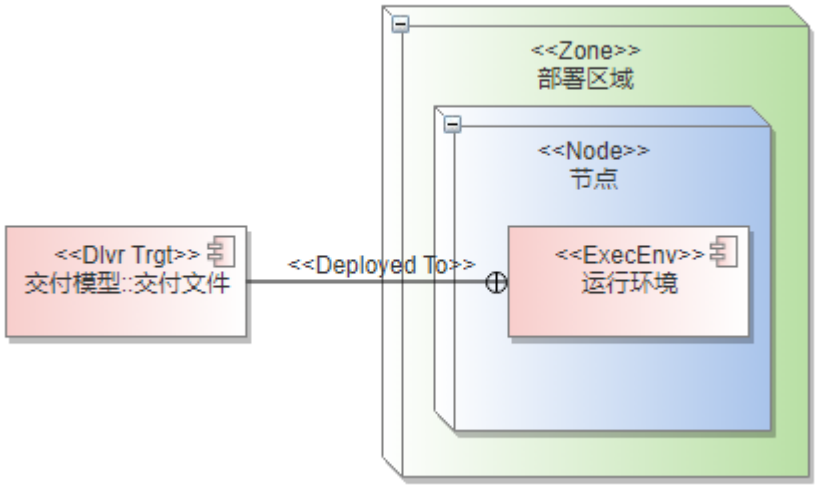
建模步骤

步骤1 创建部署模型。

创建新的部署模型图或者在已有的部署模型图中进行画图设计，如果部署模型场景较多，可根据实际情况将内容进行拆分，按实际部署场景创建多个部署模型图。

步骤2 建立交付元素与部署元素的部署关系。

从工具箱拖入部署元素创建到部署模型图中，描述部署场景，再将交付模型中定义的打包交付件引用到图中，并与部署元素建立部署关系。



----结束

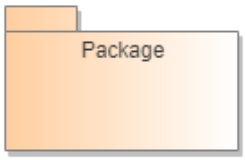
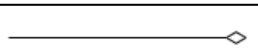
1.6 运行视图

1.6.1 概述

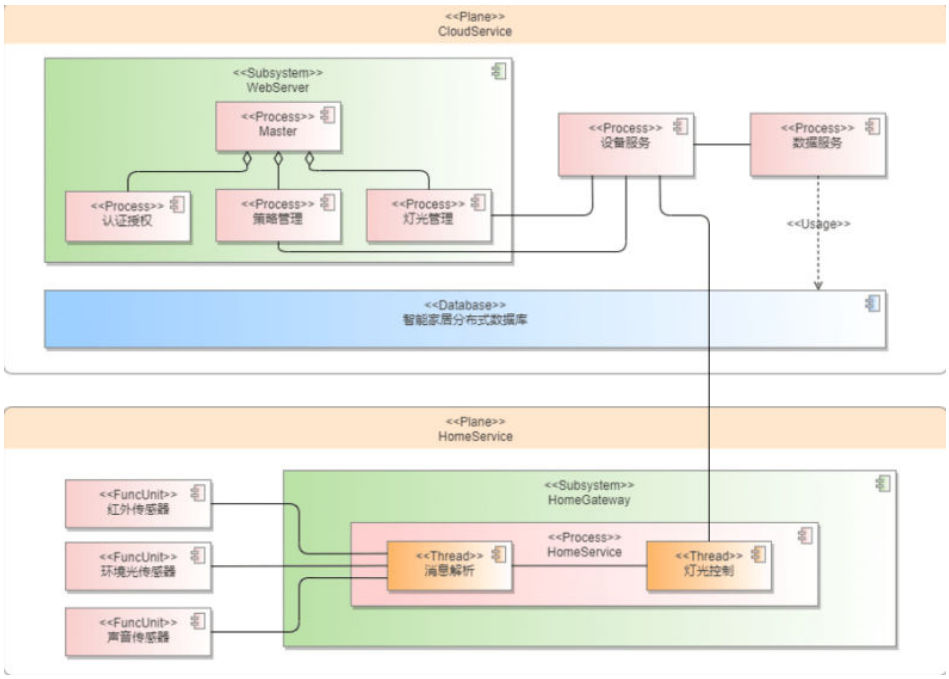
运行视图面向系统运行，描述系统启动过程、运行期交互的视图，主要解决系统运行期交互，描述各可执行交付件在运行期的交互关系。

1.6.2 运行模型

元素介绍

元素名	图标	含义
Process		进程，加载的组件、服务/微服务列表[1..*]。
Thread		线程，加载的组件、服务/微服务列表[1..*]。
Package		进程组，包含进程列表。
Mutex		锁/临界区，锁类型（自旋锁、排它锁、分布式锁、共享锁等）。
Composition		组合，是整体与部分的关系，但部分不能离开整体而单独存在。
Aggregation		聚合，是整体与部分的关系，且部分可以离开整体而单独存在。
Association		关联，是一种拥有的关系，它使一个类知道另一个类的属性和方法。

建模步骤



1.6.3 运行模型（顺序图）

元素介绍

运行模型-顺序图中的元素都来自于上下文模型中的用户角色、外部系统或者逻辑模型中定义的逻辑元素，不需要在顺序图中创建新元素，只需要使用到UML顺序图中的消息连线。

元素名	图标	含义
Messgae		同步消息连线,消息的发送者把控制传递给消息的接收者，然后停止活动，等待消息的接收者放弃或者返回控制。
Async Message		异步消息连线，消息发送者通过消息把信号传递给消息的接收者，然后继续自己的活动，不等待接受者返回消息或者控制。异步消息的接收者和发送者是并发工作的。
Reply Message		返回消息连线，返回消息表示从过程调用返回，一般与Message配套使用。
Self Message		自消息连线，表示方法的自身调用或者一个对象内的一个方法调用另外一个方法。

元素名	图标	含义
Create Message		创建对象消息连线，这个消息指向对象以后，对象的位置就不会出现在顶部，而是创建消息所在的位置。
Delete Message		终止对象消息连线，这个消息线指向对象以后，对象生命线下方出现终止符，表示对象不再接收消息调用。

前提条件

因为运行模型-顺序图中的元素都是来源于逻辑模型或上下文模型中的元素，所以需要先完成上下文模型和逻辑模型中的设计。

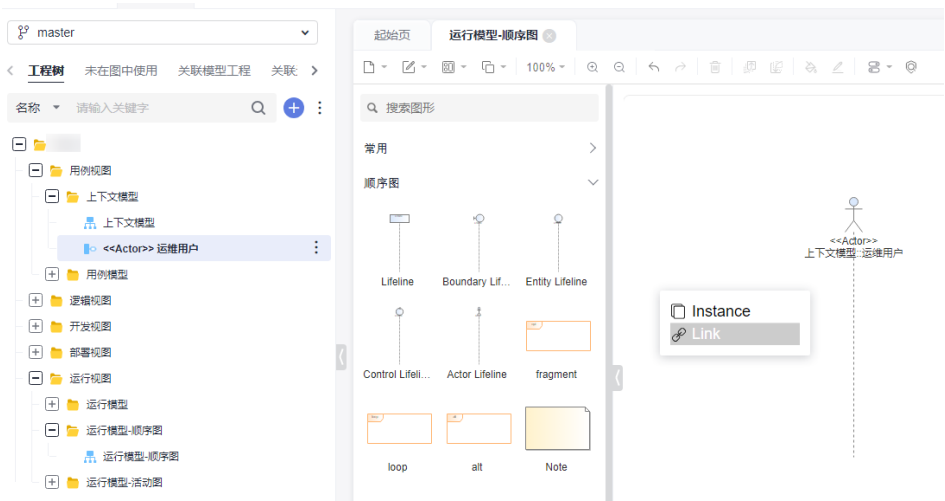
建模步骤

步骤1 创建运行模型-顺序图。

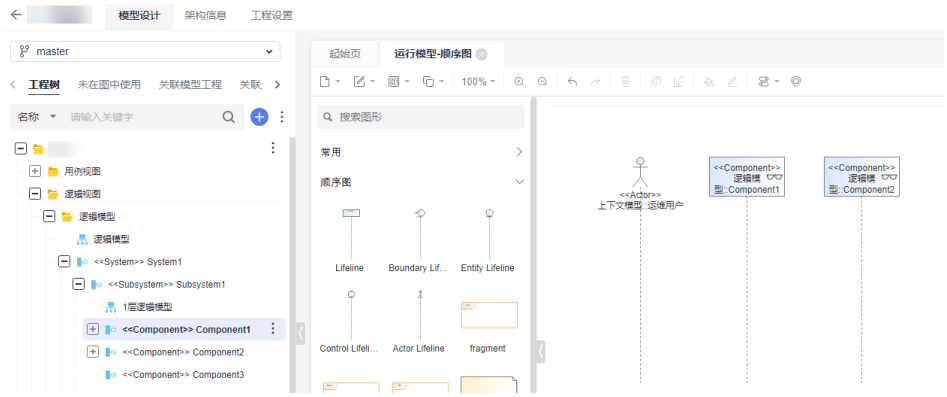
在目录或者元素节点右键菜单，选择“新增图”，可以在对应的目录或者元素节点下面创建“运行模型>顺序图”。

步骤2 引用角色和逻辑对象，描述消息交互过程。

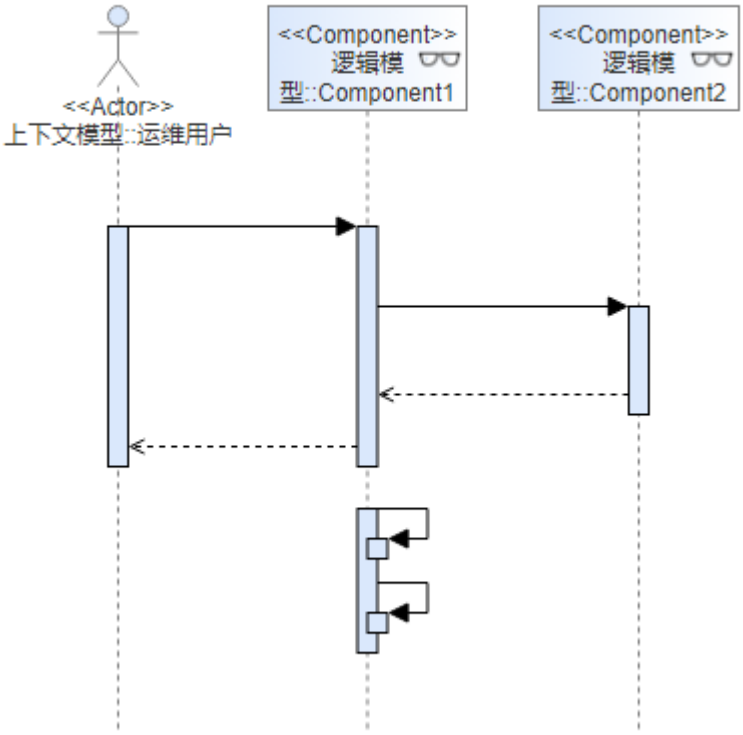
将工程树中上下文模型中定义的用户角色、外部系统引用到运行模型-顺序图中，拖入的元素选引用的方式，自动会变成Actor Lifeline样式，这是由于图类型为“运行模型-顺序图”，图类型不对，则不会变成Actor Lifeline样式。



再从工程树上将逻辑模型中定义的涉及交互场景的逻辑元素引用到“运行模型-顺序图”中。



当将需要引入的逻辑元素拖入到图中后，再去绘制交互消息的关系连线，顺序图消息连线画法可参考**绘制消息线**。



----结束

1.6.4 运行模型（活动图）

“运行模型-活动图”同UML的活动图一样，元素介绍参见**活动图**。

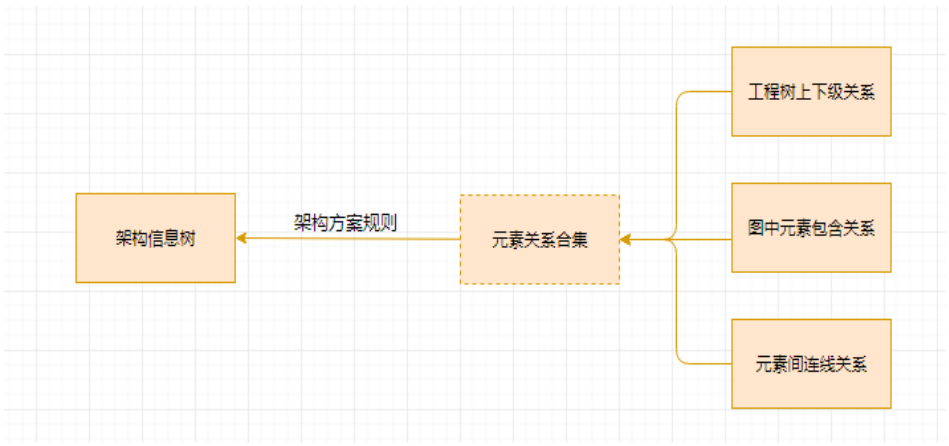
1.7 架构信息

1.7.1 架构信息树

架构信息树页签中可展示基于各种模型的架构方案规则，根据该类模型的元素关系集合生成一棵全量关系结构树。

架构方案规则的配置只有工程的管理员级别角色（带有配置资源操作权限的角色）才可配置，一般是产品的架构师来配置，制定产品的各模型中元素关系约束规则，例如逻辑模型的默认架构方案规则，方案中约束System下级节点只能是Subsystem、Domain、Service、MS这四类元素，如果在实际画图中出现了这四种以外的逻辑元素，则在最后的逻辑架构信息树中不会挂出。

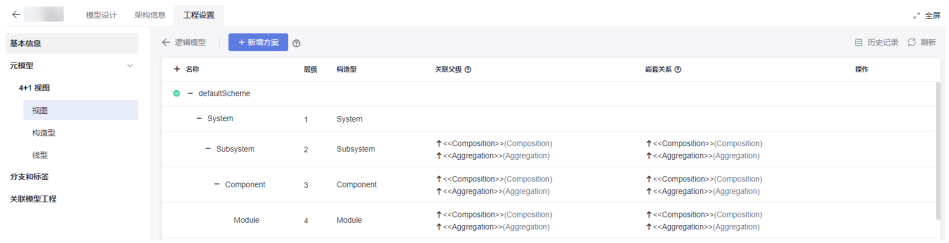
每种模型只能启用一种默认方案，但可以配置多种架构方案规则，在查询架构信息树时可以选择不同的方案，按不同的方案生成不同的信息树，**同一工程不同分支共用的是同一套架构方案及启用默认方案。**



有些必选模型是带有默认方案，默认方案不可修改，可供参考，也可以导出默认方案，修改方案名称后，再导入生成新的方案，基于该方案来修改为自己想要的结构方案。

以逻辑模型的默认方案其中一条System->Subsystem->Component->Module结构为例展示构树过程。

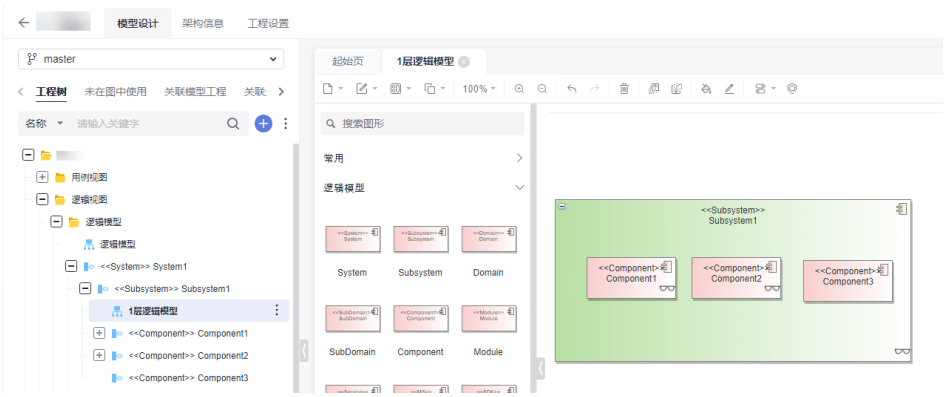
架构方案规则如下：



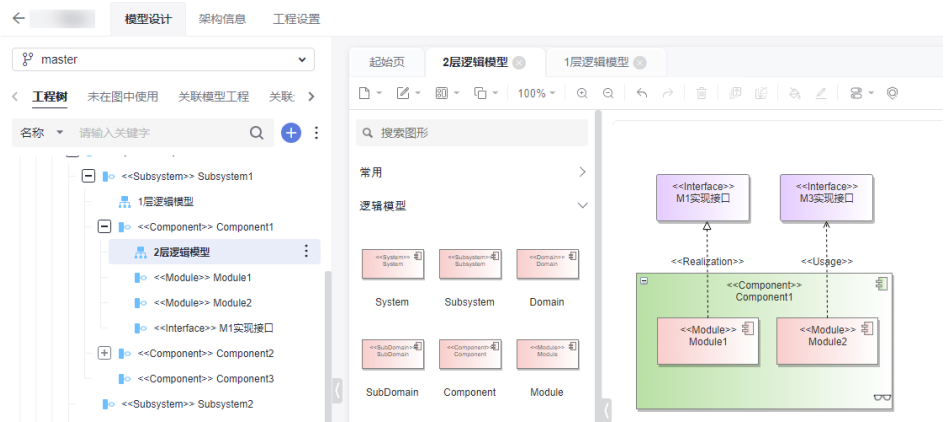
工程中的关系集合：

工程树与图中关系一致时，只取一种有效关系参与构树，下图是在0层模型中System与Subsystem关系。

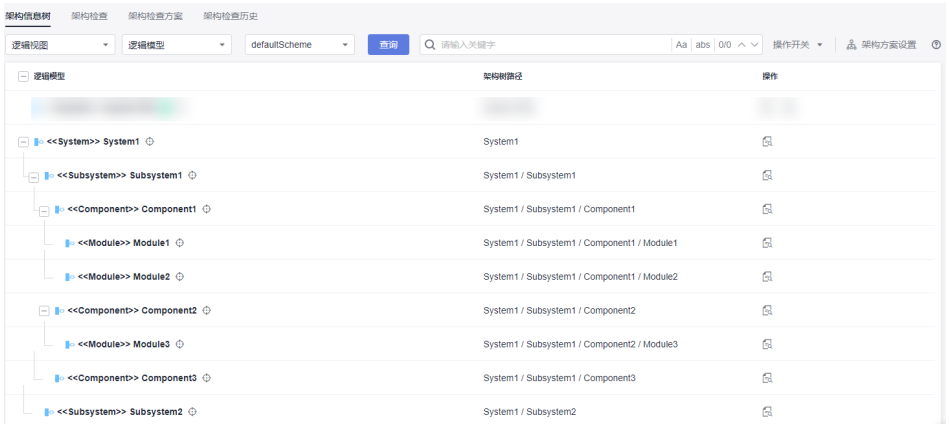
在一层逻辑模型中是Subsystem与Component的关系，图中包含关系，工程树中上下级关系，取其中一种有效关系参与构树。



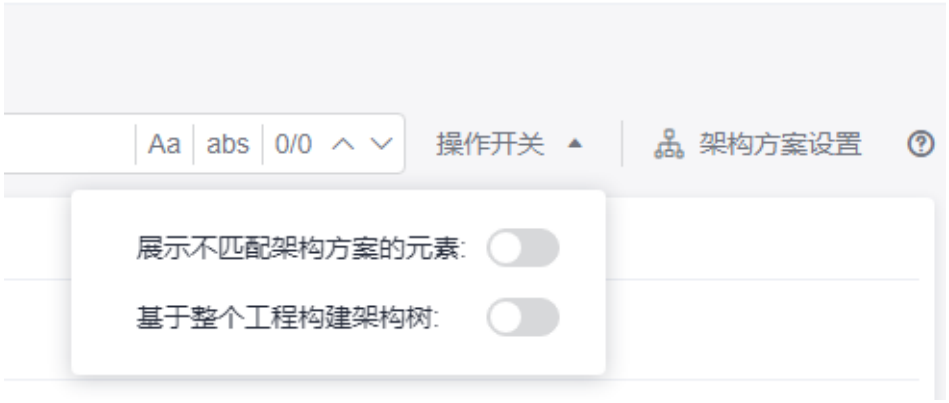
在2层逻辑模型中，是Component与Module的关系，图中包含关系，工程树中上下级关系，取其中一种有效关系参与构树，但是在2层模型中还在Module下定义了实现接口，而方案中该路径下的Module节点下没有配置Interface，故在正常构树时，Interface是不会出现在最后的架构信息树中。



最后单击“查询”按钮，实时生成的逻辑架构信息树如下：

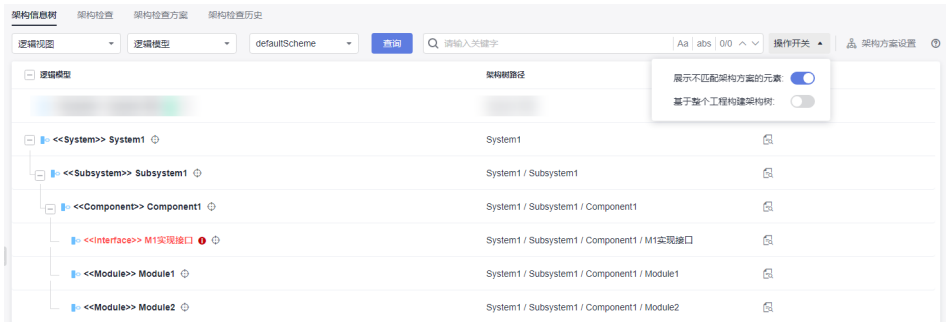


架构信息树中还有两个操作开关配置，一个是“展示不匹配架构方案的元素”，另一个是“基于整个工程构建架构树”



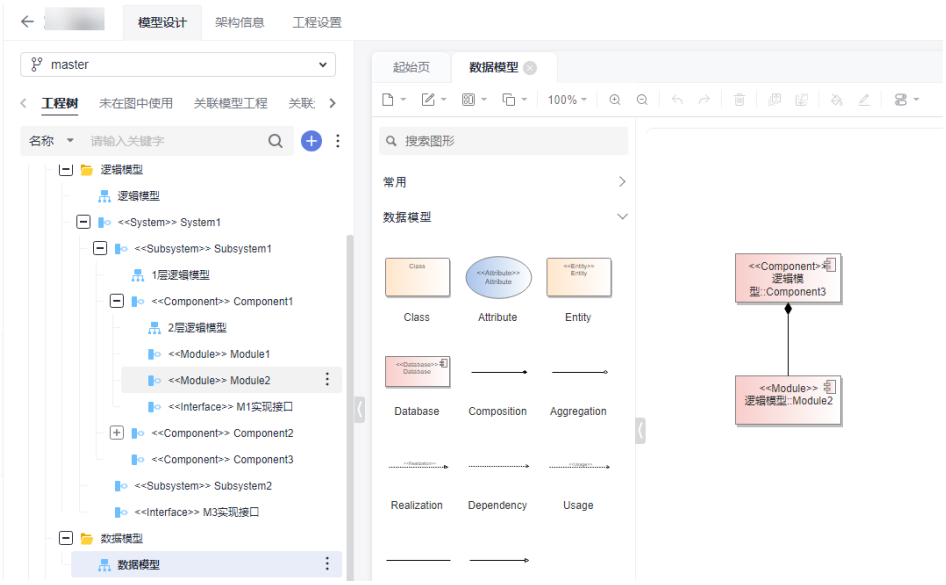
展示不匹配架构方案的元素：在架构信息树中展示出不符合架构方案规则的元素节点，并以红色文本加叹号的方式在架构信息树中标记出。

如下图所示，在方案中，component下面是未配置interface元素，但是在工程树上组件下面挂了接口元素，导致该组件与接口存在了不符合架构方案规则 的上下级关系，当开启了“展示不匹配架构方案的元素”时，该接口就会以红色状态显示在架构信息树中。

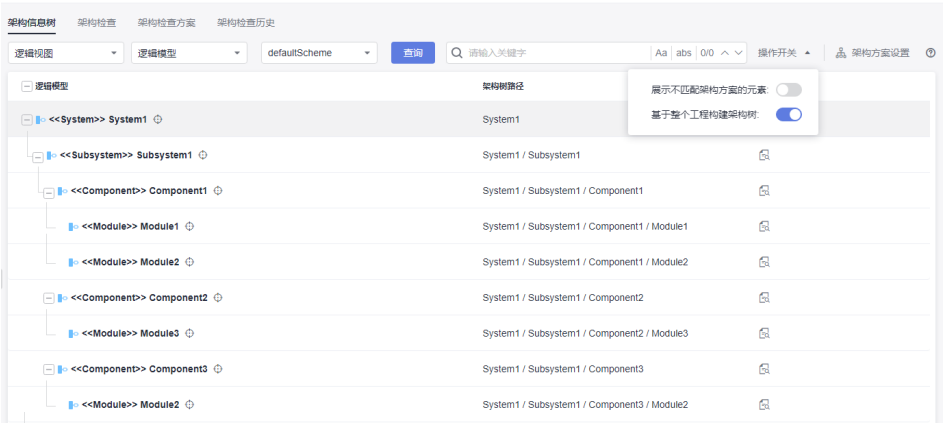


基于整个工程构建架构树：不局限于当前模型类型图中的关系参与构树，会将元素在所有模型图中的连线关系、包含关系算到关系集合中，按架构方案规则生成架构信息树。

如下图所示，在数据模型中对Component3与Module2建立符合架构规则的组合关系。



启用“基于整个工程构建架构树”开关后，上述非逻辑模型图中的组合关系参与逻辑模型的构树。



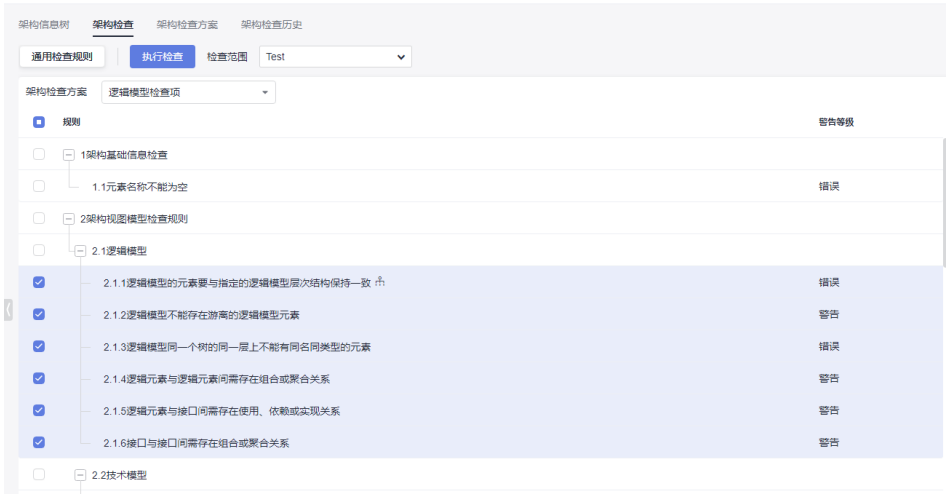
1.7.2 架构检查方案

架构检查方案功能是基于架构检查中的规则项，设置一个检查规则集合，可将该检查集合设置为架构检查中默认启用的检查规则集，该检查会生成检查任务到架构检查历史中。

单击“新增架构检查方案”，输入方案名称，规则集中的检查项来源基于通用检查规则，选择要配置到规则集中的方案。

新建名称为逻辑模型检查项，勾选对应检查规则。

“架构检查>通用检查规则”即可看到新增规则。



1.7.3 架构检查历史

在架构检查历史中可以查看过往检查的历史记录，默认打开是当前用户的检查记录。在“我的检查”下拉选项中，可以切换显示出所有人的检查结果，也支持按时间过滤查询检查的记录。

在检查历史记录下“查看”可以打开检查规则结果通过情况。

在是否通过详情下“查看”具体元素定位‘修复指导和关系查询。

1.8 架构检查

1.8.1 通用检查规则

1.8.1.1 架构基础信息检查

1.1 元素名称不能为空

详细描述

建模设计的元素名称不能为空，如果存在名称为空的元素，在检查结果中都会列出。

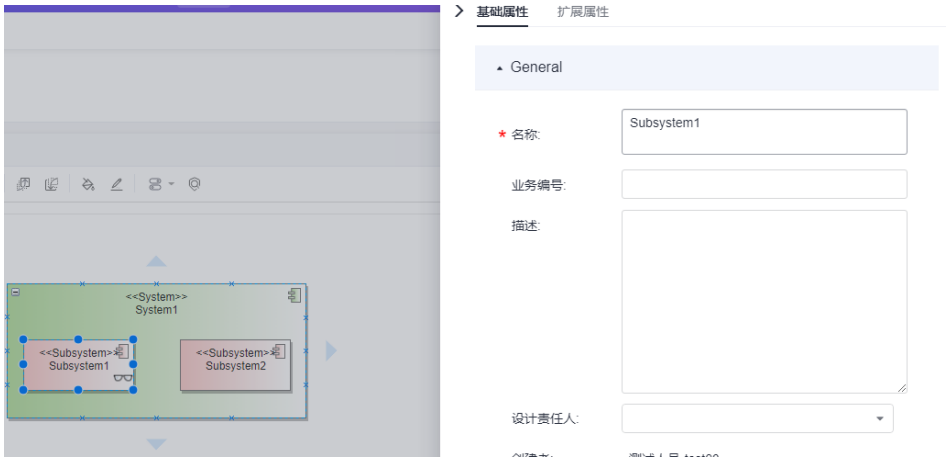
检查范围

在图上创建的元素在工程树中出现对应的节点，即为建模元素，都在被检查范围内。

如何检查

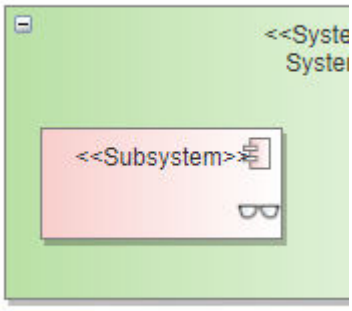
查询模型工程内所有建模元素，检查出名称为空元素。

正确示例



错误示例

Subsystem名称为空。



1.8.1.2 架构视图模型检查规则

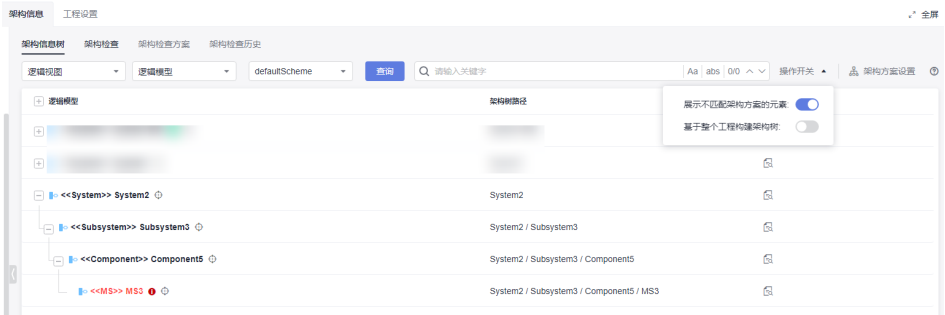
1.8.1.2.1 逻辑模型

2.1.1 逻辑模型的元素要与指定的逻辑模型层次结构保持一致

详细描述

在逻辑模型中创建逻辑元素，逻辑元素在架构树中与上下级元素的关系层级结构要与逻辑模型架构方案配置定义的层次结构一致，即该逻辑元素与上层父级元素、下层子级元素的父子关系（也称上下层级关系）、以及它们之间的连线关系和方向指向，都要与层级规则中定义的保持一致。

该规则项检查出的是架构信息树中按模型图构树后显示红色叹号的告警元素。



检查范围

当前模型工程所有逻辑模型图中的逻辑元素。

逻辑元素定义：“工程设置>元素构造型”下，绑定到4+1视图：逻辑模型的基础构造型与自定义构造型元素以及逻辑模型架构方案配置的构造型。

包括：在逻辑模型图上创建出来的逻辑元素，引用到逻辑模型中的逻辑元素（包含关联空间中的引用的逻辑元素）。

如何检查

查询基于模型图（只有逻辑模型图内的逻辑元素参与构树）构出的逻辑模型架构树，找出与架构方案不匹配（标红）的元素。

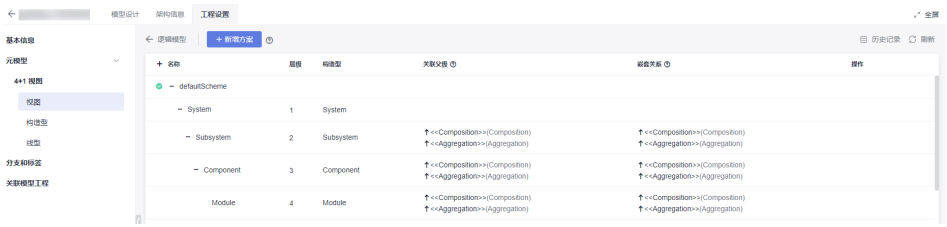
正确示例

架构层级规则示例：

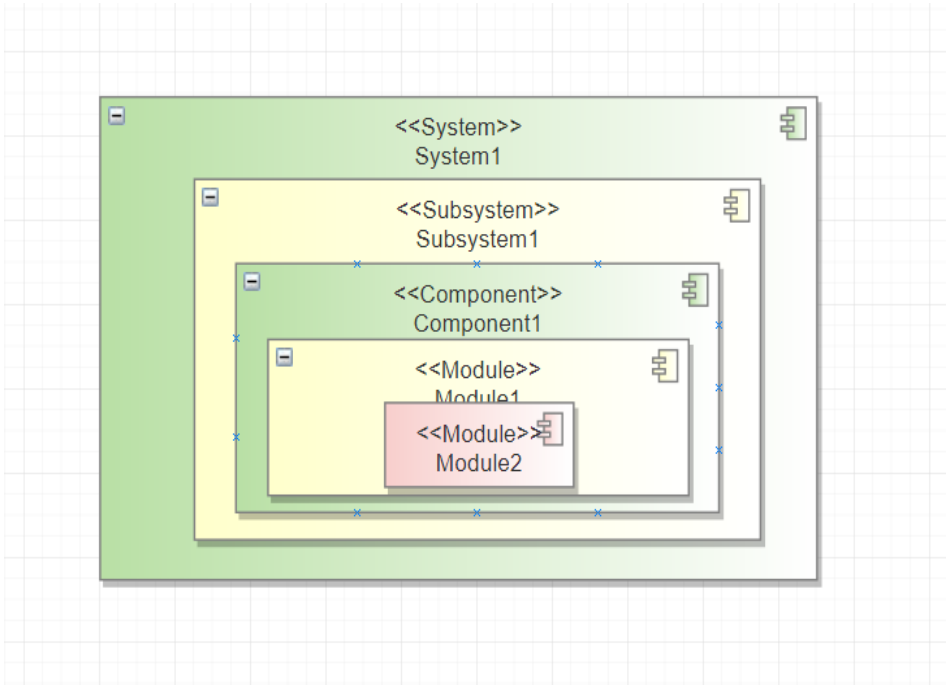
关联父级：配置的是当前层级元素与上一层级的元素之间的连线类型和父子关系指向。

嵌套：是否支持当前类型的元素与同类型元素建立关系。

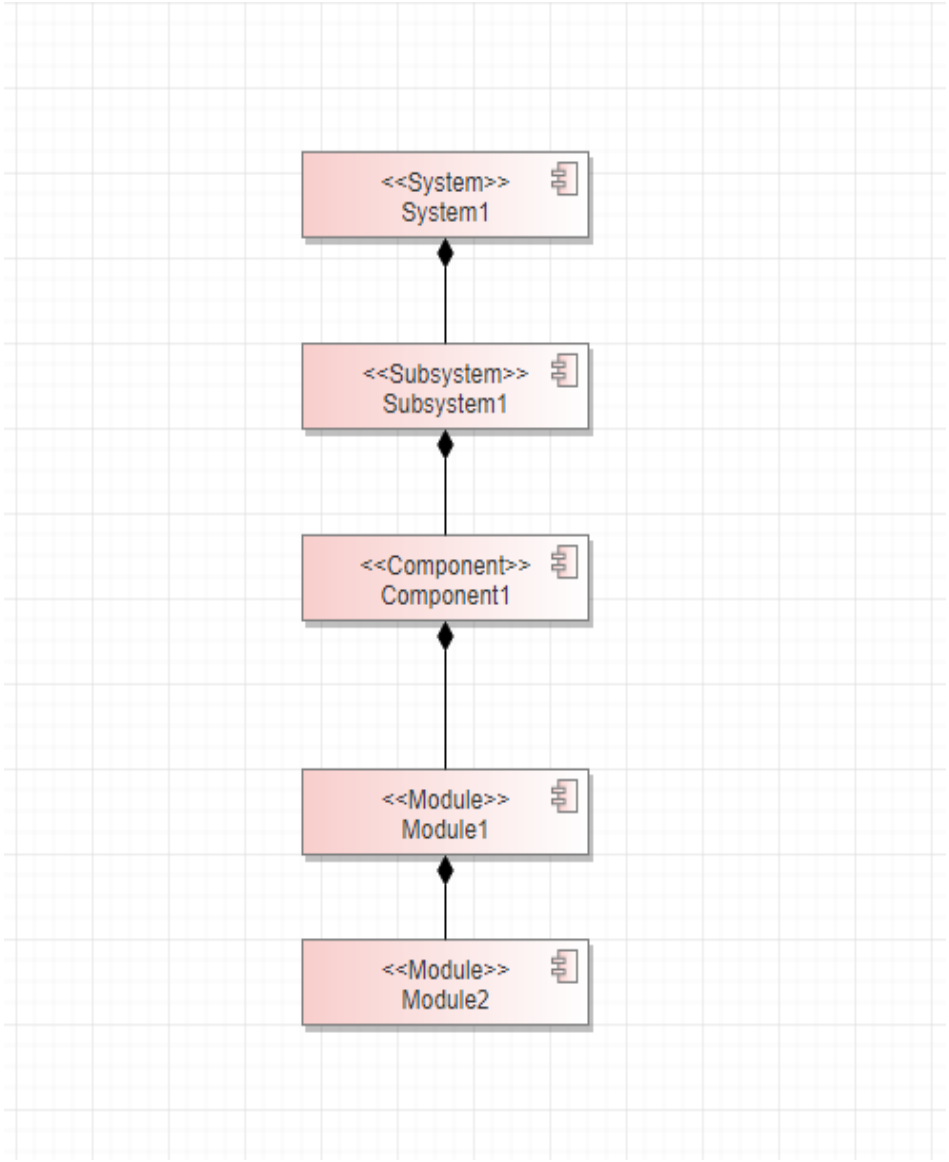
嵌套关系：当前类型的元素与同类型元素建立连线关系类型，指向关系默认为子指向父（即被指向的一方为子）。



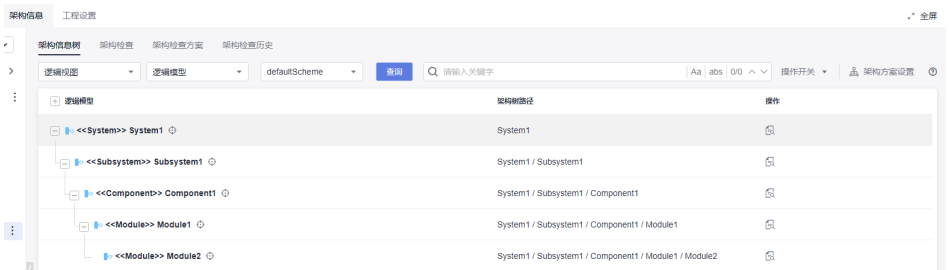
符合架构层级方案的画法示例1--包含的父子关系：



符合架构层级方案的画法示例2–连线的父子关系：



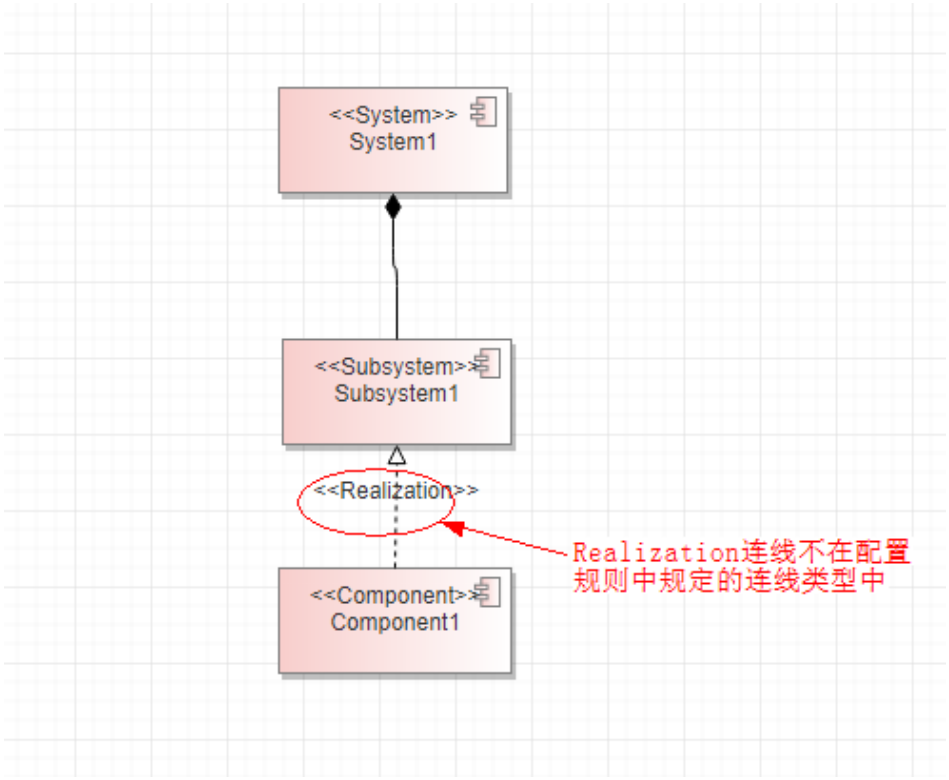
架构信息树展示结果：



当架构树上没有标红元素，就没有2.1.1的检查错误结果。

错误示例

连线类型不对。



架构信息树中报红。

架构检查结果。

2.1.2 逻辑模型不能存在游离的逻辑模型元素

详细描述

逻辑模型元素不能独立存在于逻辑架构树之外，必须要与架构树上的逻辑元素建立关联关系。

检查范围

当前模型工程所有逻辑模型图中的逻辑元素。

（逻辑元素定义：工程设置->构造型下，绑定到4+1视图：逻辑模型的基础构造型与自定义构造型元素以及逻辑模型架构方案配置的构造型）。

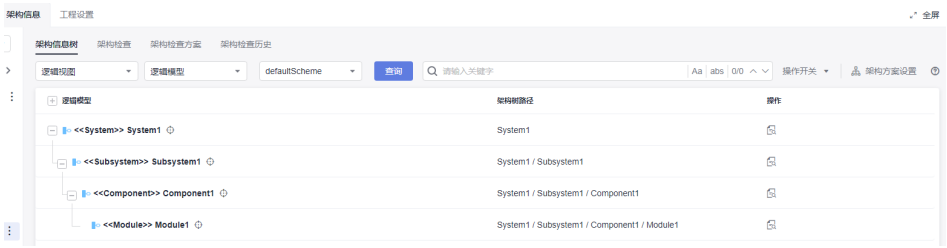
包括：在逻辑模型图上创建出来的逻辑元素，引用到逻辑模型中的逻辑元素（包含关联空间中的引用的逻辑元素）；

如何检查

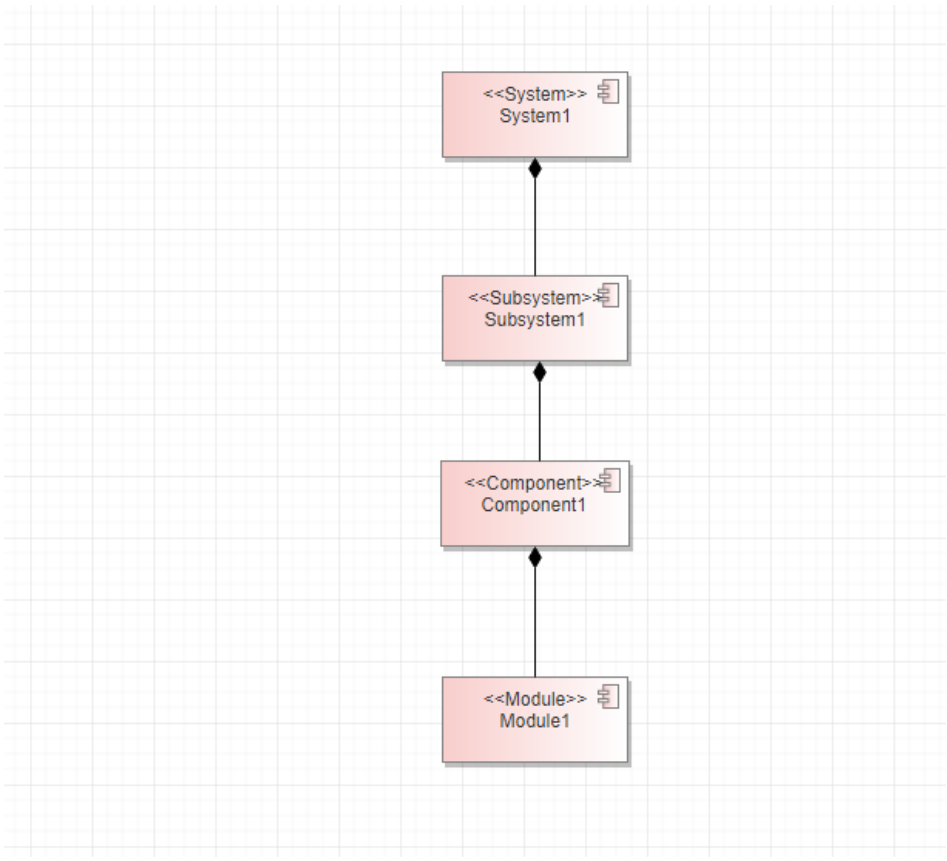
查询基于模型图（只有逻辑模型图内的逻辑元素参与构树）并展示不匹配元素构出的逻辑模型架构树，找出所有逻辑元素中不在架构树中的逻辑元素。

正确示例

按逻辑架构方案构建的架构信息树：

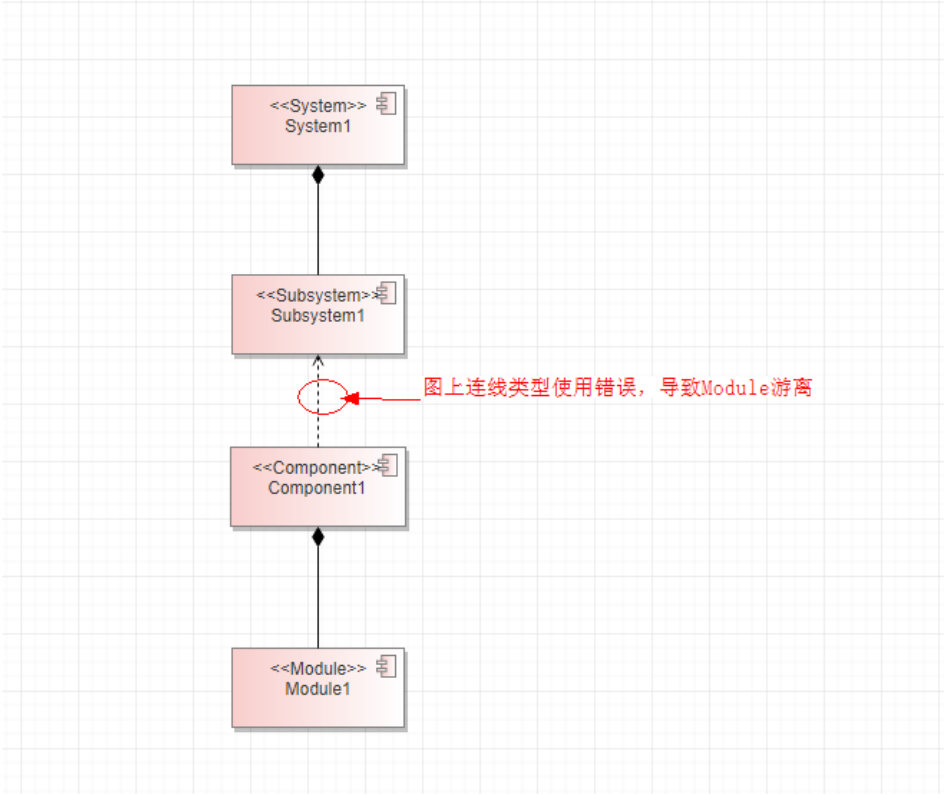


模型图示例：



错误示例

场景一：Component1与父级元素连线错误，导致子元素Module1变成游离的元素。



检查结果：

2.1.3 逻辑模型同一个树的同一层上不能有同名同类型的元素

详细描述

在逻辑架构信息树上，同一个父元素节点下面，不能存在类型相同，名称也相同的元素。

检查范围

当前模型工程所有逻辑模型图中的逻辑元素

(逻辑元素定义：工程设置->构造型下，绑定到4+1视图：逻辑模型的基础构造型与自定义构造型元素以及逻辑模型架构方案配置的构造型)。

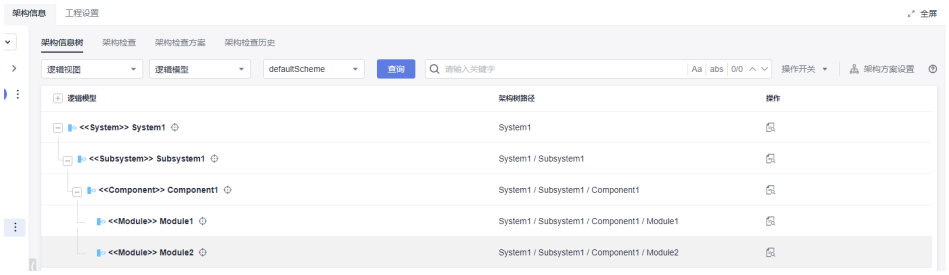
包括：

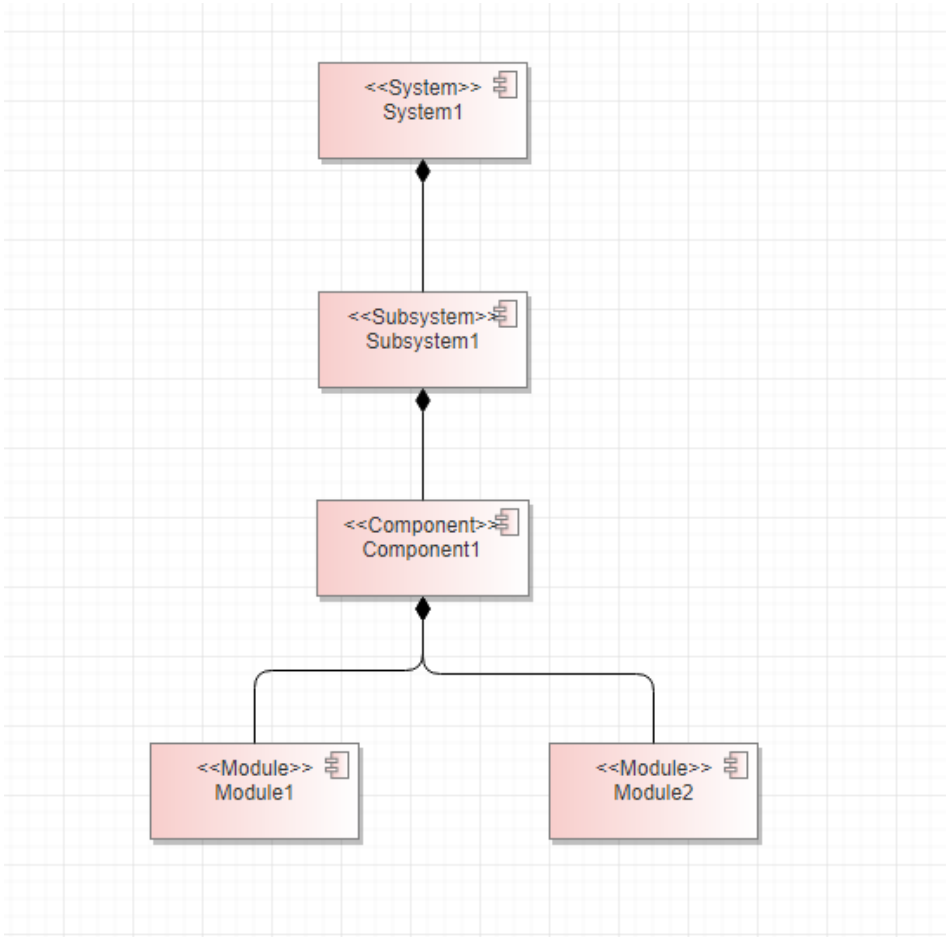
1. 在逻辑模型图上创建出来的逻辑元素；
2. 引用到逻辑模型中的逻辑元素（包含关联空间中的引用的逻辑元素）；

如何检查

查询基于模型图（只有逻辑模型图内的逻辑元素参与构树）构出的逻辑模型架构树，找出同一节点下同名同类型的逻辑元素。

正确示例

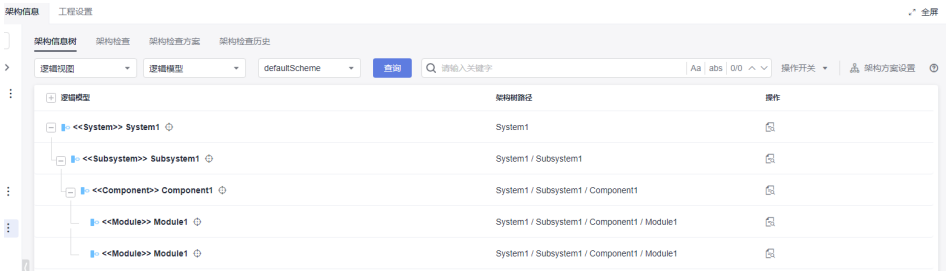




错误示例

场景一：同父元素下面存在同类型且同名称的元素。

按逻辑规则构建的架构信息树，树上不会显示异常：



2.1.4 逻辑元素与逻辑元素间需存在组合或聚合关系

详细描述

逻辑元素与逻辑元素之间如果存在连线关系，必须为组合或者聚合关系。

检查范围

当前模型工程所有逻辑模型图中的逻辑元素

（逻辑元素定义：工程设置->构造型下，绑定到4+1视图：逻辑模型的基础构造型与自定义构造型元素以及逻辑模型架构方案配置的构造型）。

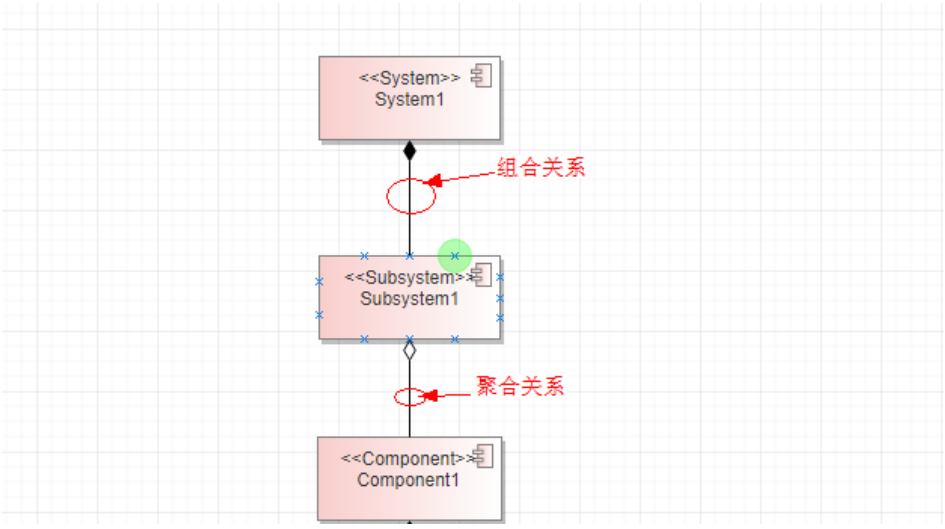
1. 在逻辑模型图上创建出来的逻辑元素之间的连线关系；
2. 引用到逻辑模型中的逻辑元素（包含关联空间中的引用的逻辑元素）之间的连线关系；
3. 不包含接口元素；

如何检查

找出逻辑模型图里的在检查范围内的逻辑元素间存在连线关系但连线关系中没有组合或聚合关系的元素（即使层级规则方案中配置了除组合聚合之外的指定的连线类型，也会检查出来）。

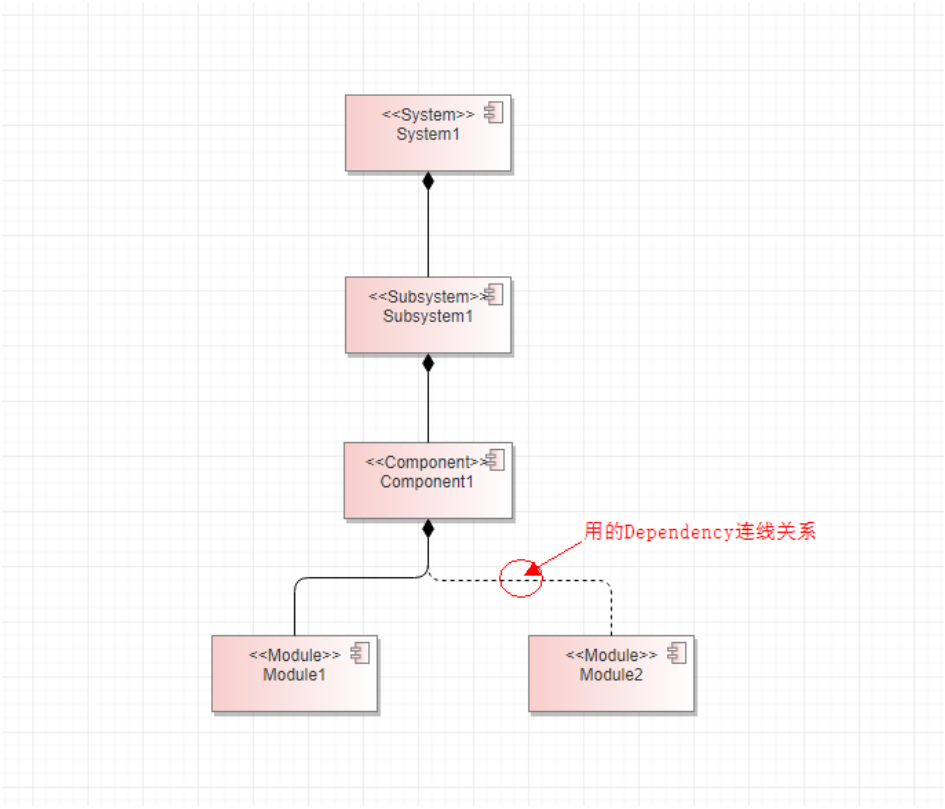
正确示例

组合或聚合关系。



错误示例

场景一：使用非组合或者聚合连线关系。



即使方案中配置了组件与模块之间有使用连线关系，但是该规则依然会检查出来。

2.1.5 逻辑元素与接口间需存在使用、依赖或实现关系

详细描述

逻辑元素与接口之间如果存在连线关系，必须为使用、实现、依赖关系。

检查范围

当前模型工程中的所有符合定义规则的逻辑元素（定义规则：工程设置>构造型下，绑定到4+1视图：逻辑模型的基础构造型与自定义构造型元素才认定为逻辑元素）。

1. 在逻辑模型图上创建出来的逻辑元素与接口连线之间的连线关系；
2. 引用到逻辑模型中的逻辑元素（包含关联空间中的引用的逻辑元素）与接口之间的连线关系；
3. 接口只指实体接口，即Interface元素，暴露接口Provided Interface和请求接口Required Interface 非实体接口，不在检查目标中；

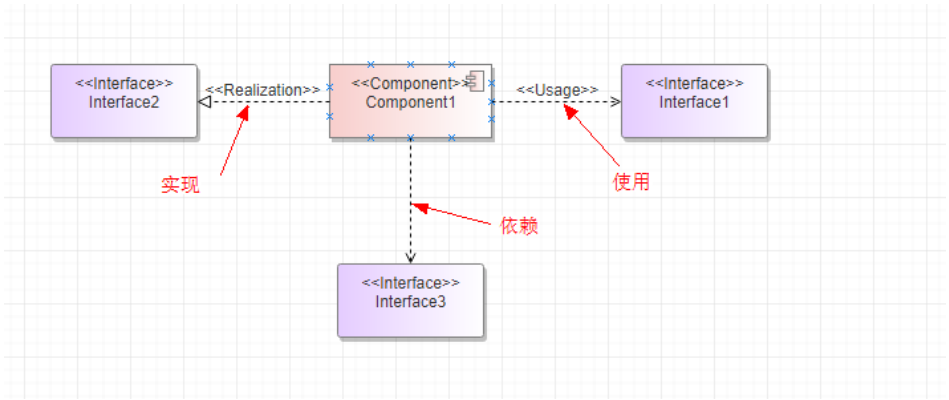
如何检查

找出逻辑模型图里的在检查范围内的逻辑元素与实体接口间存在连线关系但连线关系不是使用、实现、依赖关系的逻辑元素和接口

（即使层级规则方案中配置了除使用、实现、依赖之外的指定的连线类型，也会检查出来）。

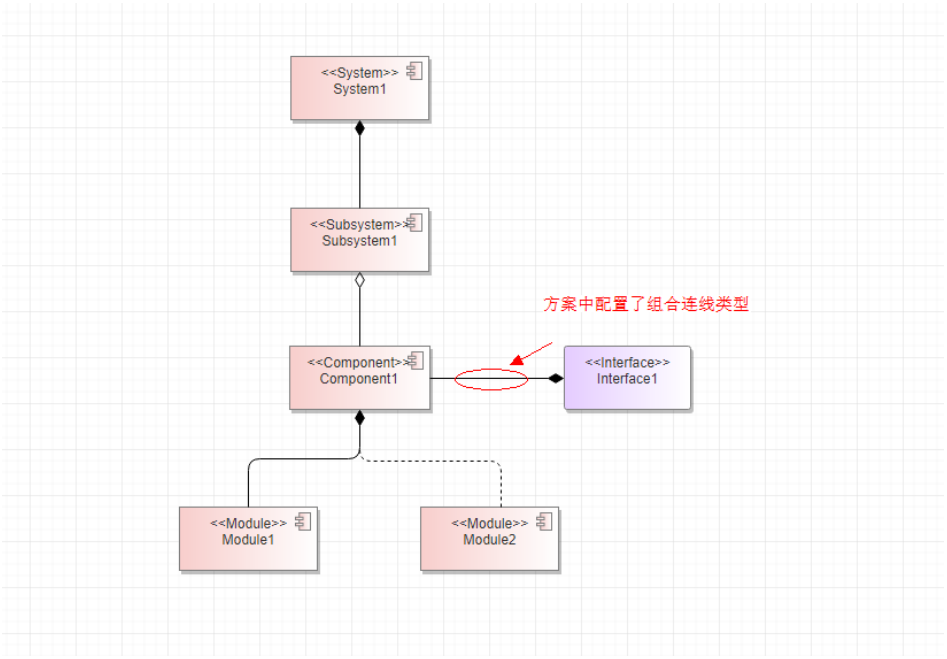
正确示例

使用、实现、依赖关系：



错误示例

场景一：使用非实现、使用、依赖之外的连线关系（即使关系已经配置到方案中）。



即使方案中配置了组件与模块之间有组合连线关系，但是该规则依然会检查出来。

2.1.6 接口与接口间需存在组合或聚合关系

详细描述

接口与接口之间如果存在连线关系，必须为组合或者聚合关系。

检查范围

当前模型工程中的所有符合定义规则的接口（定义规则：工程设置>构造型下，绑定到4+1视图：逻辑模型的接口Interface元素）。

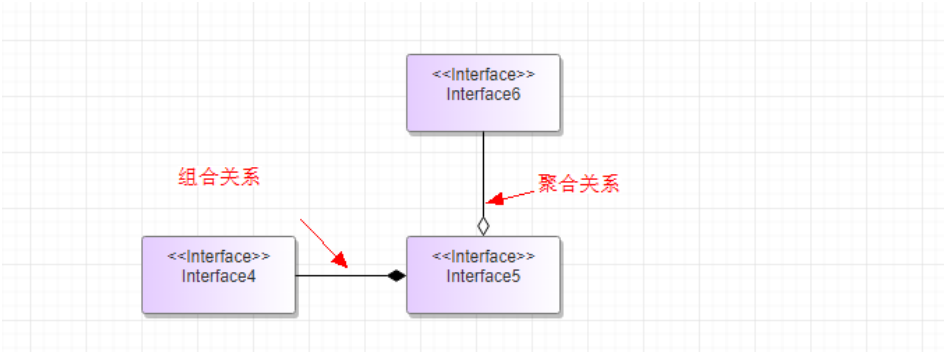
1. 在逻辑模型图上创建出来的接口元素之间的连线关系。
2. 引用到逻辑模型中的接口（包含关联空间中的引用的逻辑元素）之间的连线关系。

如何检查

找出逻辑模型图里的接口间存在连线关系但连线关系中没有组合或聚合关系的接口元素（即使层级规则方案中配置了除组合聚合之外的指定的连线类型，也会检查出来）。

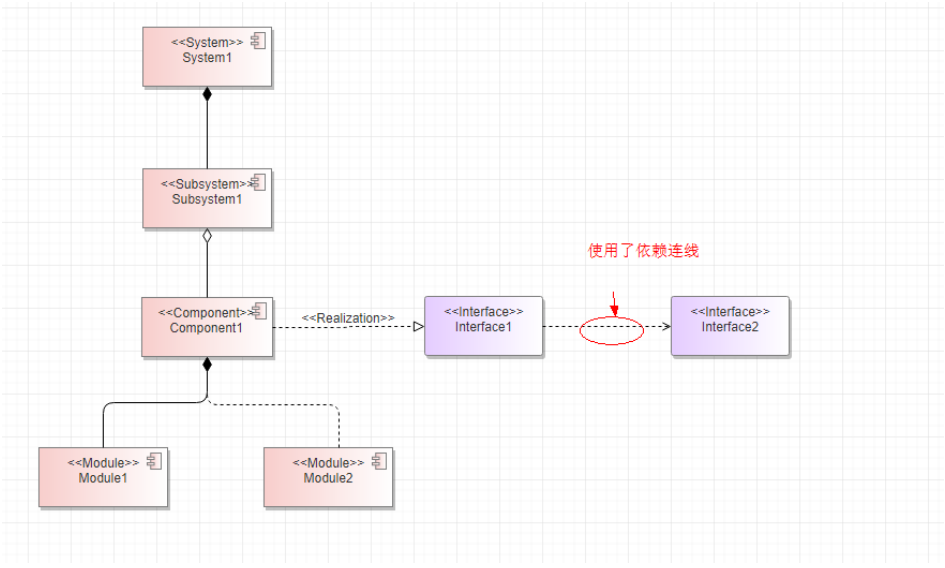
正确示例

组合或聚合关系：



错误示例

场景一：使用非组合或者聚合连线关系。



即使方案中配置了接口与接口之间有使用连线关系，但是该规则依然会检查出来。

1.8.1.2.2 技术模型

2.2.1 技术模型的元素要与指定的技术模型层次结构保持一致

详细描述

在技术模型中创建技术元素，技术元素在架构树中与上下级元素的关系层级结构要与技术模型架构方案配置定义的层次结构一致，即该技术元素与上层父级元素、下层子级元素的父子关系（也称上下层级关系）、以及它们之间的连线关系和方向指向，都要与层级规则中定义的保持一致。

检查范围

当前模型工程中的所有符合定义规则的技术元素（定义规则：工程设置>构造型下，绑定到4+1视图：技术模型的基础构造型与自定义构造型元素才认定为技术元素）。

1. 在技术模型图上创建出来的技术元素；
2. 引用到技术模型中的技术元素（包含关联空间中的引用的技术元素）。

如何检查

查询基于模型图（只有技术模型图内的技术元素参与构树）构出的技术模型架构树，找出与架构方案不匹配（标红）的元素。

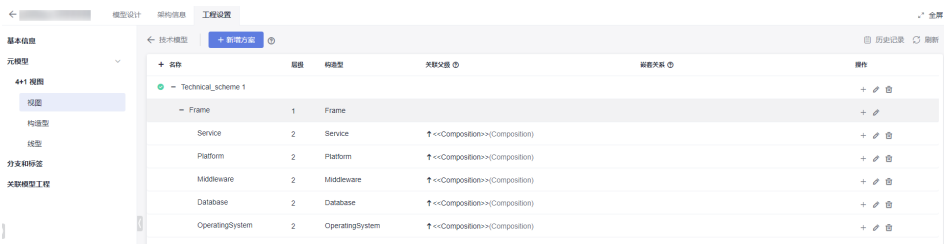
正确示例

架构层级规则示例：

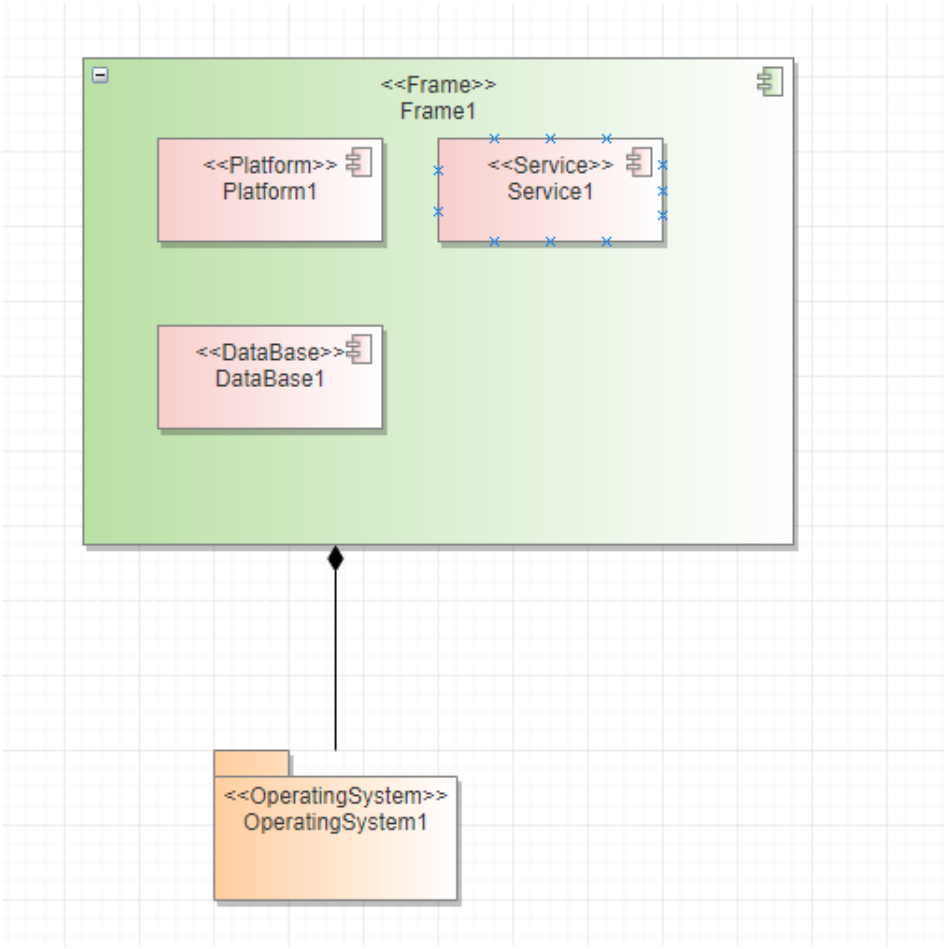
关联父级：配置的是当前层级元素与上一层级的元素之间的连线类型和父子关系指向。

嵌套：是否支持当前类型的元素与同类型元素建立关系。

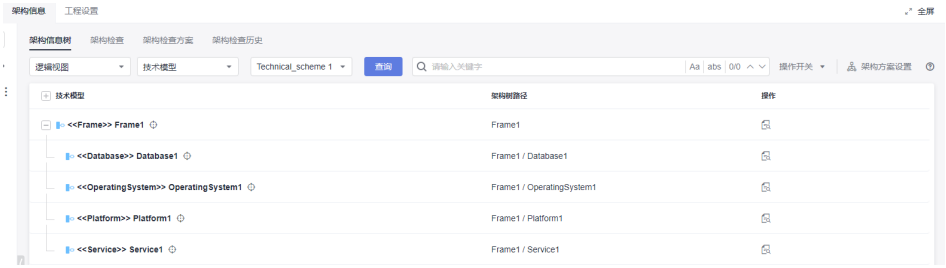
嵌套关系：当前类型的元素与同类型元素建立连线关系类型，指向关系默认为父指向子（即被指向的一方为子）。



图中画法示例1–包含的父子关系和连线关系：



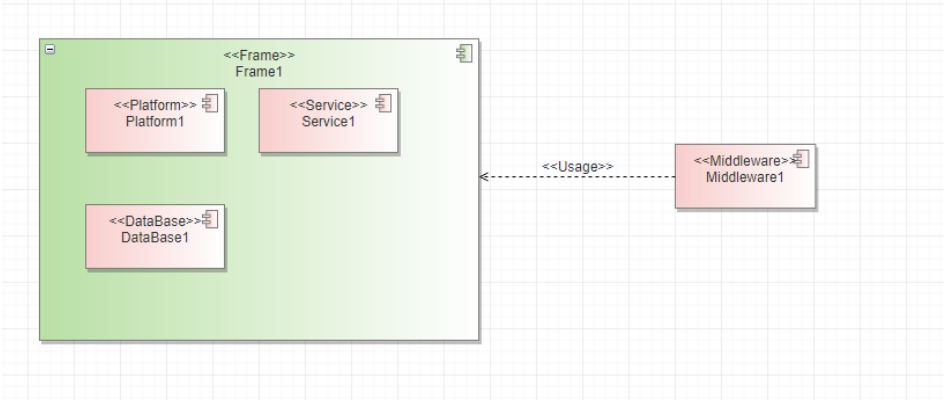
架构信息树展示结果：



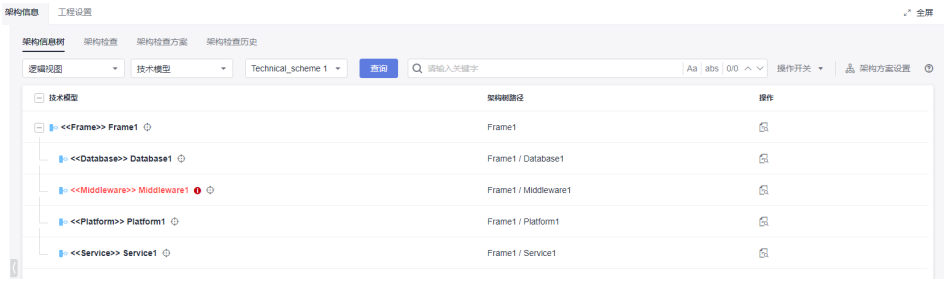
当架构树上没有标红元素，就没有2.2.1的检查错误结果。

错误示例

错误示例-连线类型不对：



架构信息树中报红：



架构检查结果：

2.2.2 技术模型不能存在游离的元素

详细描述

技术模型元素不能独立存在于技术架构树之外，必须要与架构树上的技术元素建立关联关系。

检查范围

当前模型工程中的所有符合定义规则的技术元素（定义规则：工程设置>构造型下，绑定到4+1视图：技术模型的基础构造型与自定义构造型元素才认定为技术元素）。

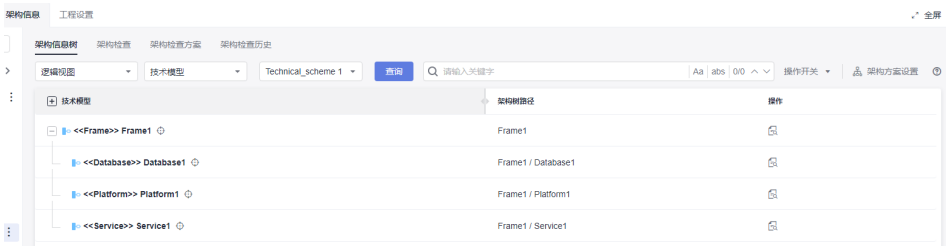
1. 在技术模型图上创建出来的技术元素；
2. 引用到技术模型中的技术元素（包含关联空间中的引用的技术元素）；

如何检查

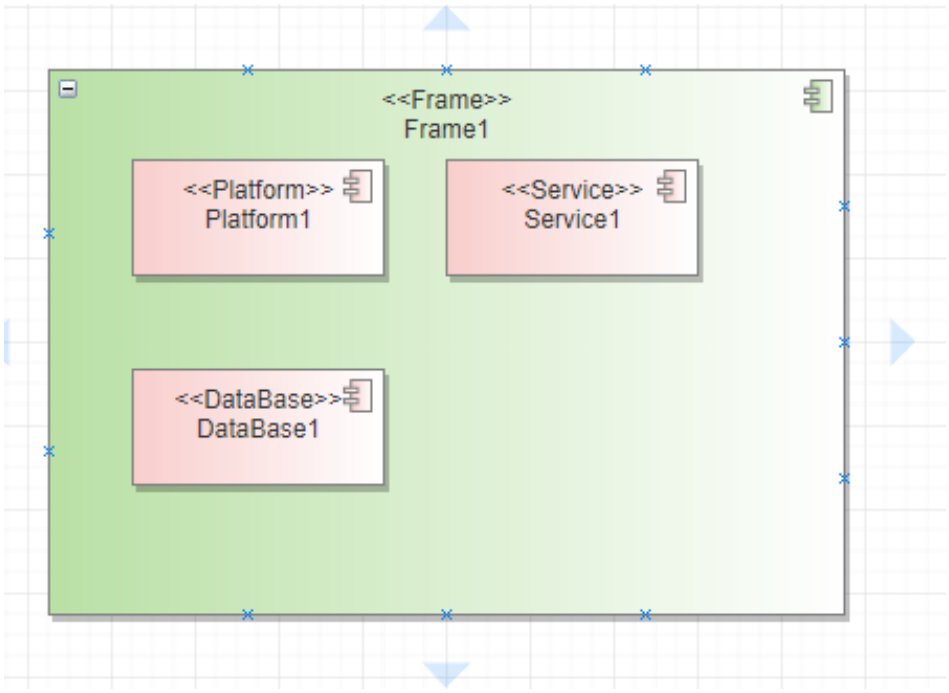
查询基于模型图（只有技术模型图内的技术元素参与构树）并展示不匹配元素构出的技术模型架构树，找出所有技术元素中不在架构树中的技术元素。

正确示例

按逻辑规则构建的架构信息树：

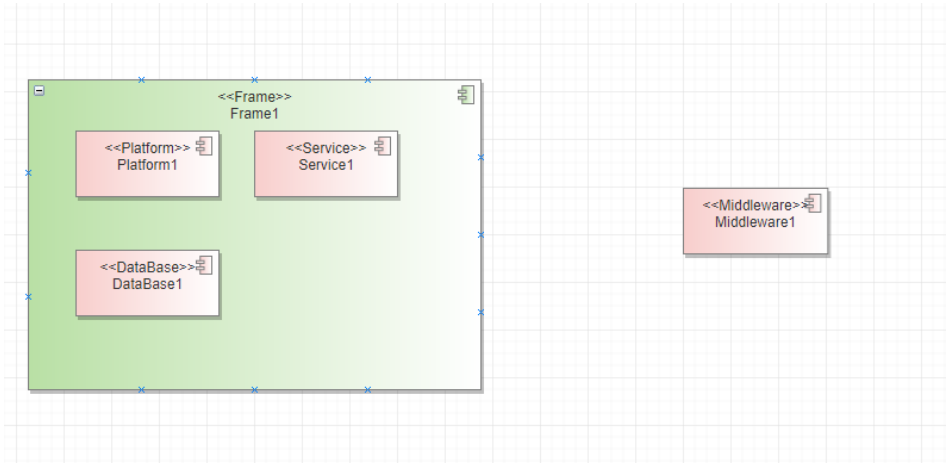


模型图示例：



错误示例

场景一：独立存在在技术模型图上的技术模型元素。



检查结果：

2.2.3 技术模型同一个树的同一层上不能有同名同类型的元素

详细描述

在同一棵技术架构信息树上，在同一个父元素节点下面，不能存在扩展类型相同，并且名称也相同的元素。

检查范围

当前模型工程中的所有符合定义规则的技术元素（定义规则：工程设置>构造型下，绑定到4+1视图：技术模型的基础构造型与自定义构造型元素才认定为技术元素）。

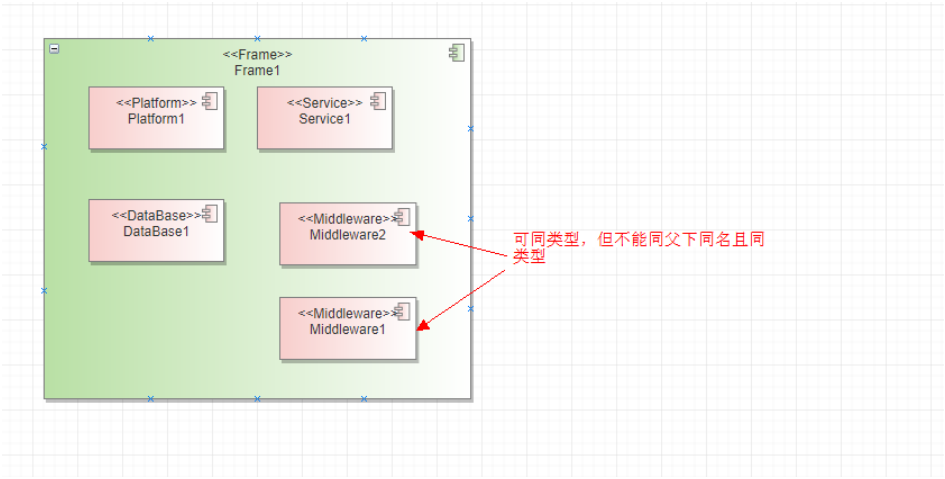
1. 在技术模型图上创建出来的技术元素；
2. 引用到技术模型中的技术元素（包含关联空间中的引用的技术元素）；

如何检查

查询基于模型图（只有技术模型图内的技术元素参与构树）构出的技术模型架构树，找出同一节点下同名同类型的技术元素。

正确示例

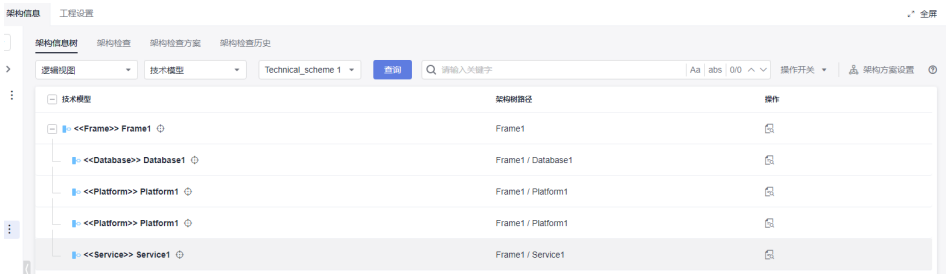
同一父节点下可以同类型或者同名，但是不能同类型且又同名的。



错误示例

场景一：同父元素下面存在同类型且同名称的元素。

按逻辑规则构建的架构信息树，树上不会显示异常：



2.2.4 技术元素与逻辑元素需存在技术元素实现逻辑元素或者逻辑元素使用或依赖技术元素关系

详细描述

技术模型元素与逻辑元素之间如果存在连线关系，必须为使用或者实现类型的连线关系，不能存在其它连线类型的关系。

检查范围

当前模型工程中的所有符合定义规则的技术元素（定义规则：工程设置>构造型下，绑定到4+1视图：技术模型的基础构造型与自定义构造型元素才认定为技术元素）。

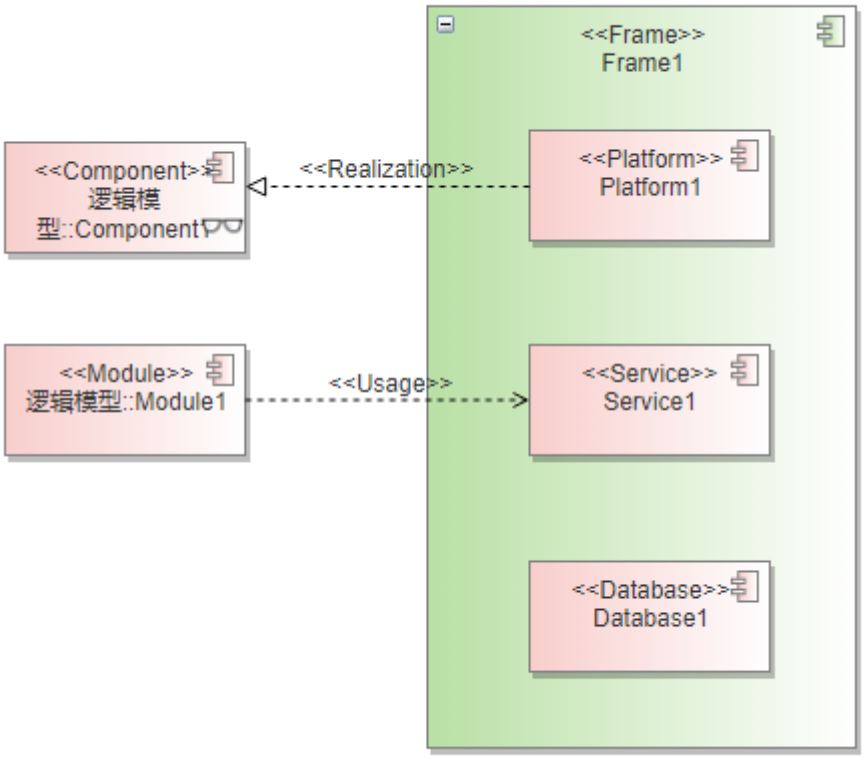
1. 在技术模型图上创建出来的技术元素；
2. 引用到技术模型中的技术元素（包含关联空间中的引用的技术元素）；
3. 从逻辑模型中引用（Link）到技术模型中的逻辑元素（逻辑元素的定义参考2.1.1 逻辑模型检查范围定义）。

如何检查

找出技术元素与逻辑元素中存在连线关系但不是 技术元素实现逻辑元素 或者 逻辑元素使用或依赖技术元素关系的元素。

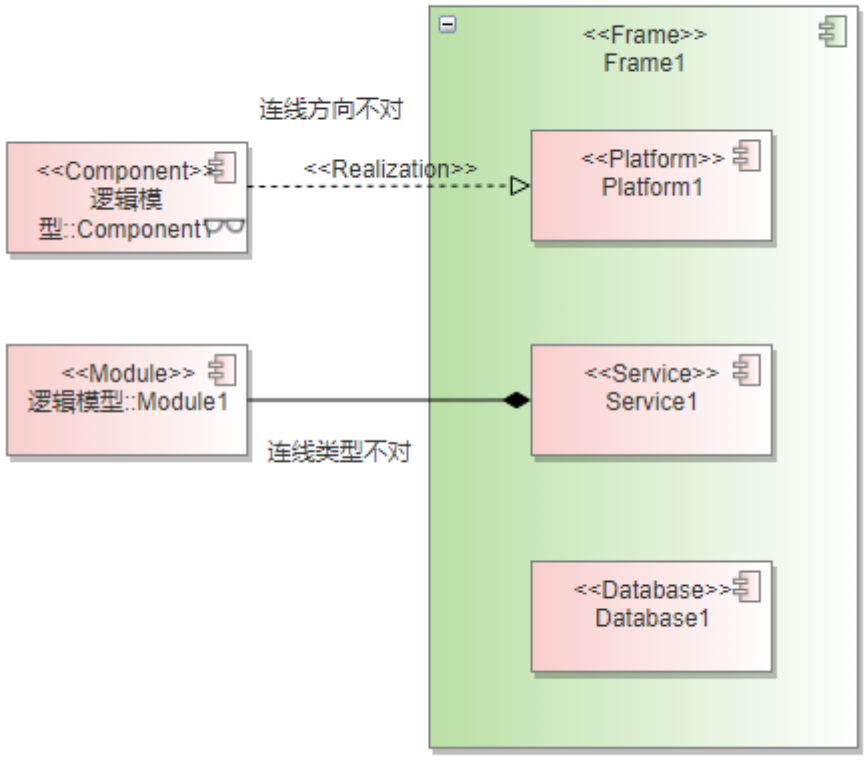
正确示例

模型图示例：正常为逻辑元素指向技术元素，用Usage连线；技术元素指向逻辑元素，用Realization连线。



错误示例

场景一：技术元素与逻辑元素连线类型不对或者连线类型正确但是指向不对。



检查结果：

连线方向不对和连线类型不对的两端的元素都会被检查出来。

1.8.1.2.3 代码模型

2.3.1 代码模型的元素要与指定的代码模型层次结构保持一致

详细描述

在代码模型中创建代码元素，代码元素在架构树中与上下级元素的关系层级结构要与代码模型架构方案配置定义的层次结构一致，即该代码元素与上层父级元素、下层子级元素的父子关系（也称上下层级关系）、以及它们之间的连线关系和方向指向，都要与层级规则中定义的保持一致。

检查范围

当前模型工程中的所有符合定义规则的代码模型元素（定义规则：工程设置>构造型下，绑定到4+1视图：代码模型的基础构造型与自定义构造型元素才认定为代码模型元素）。

1. 在代码模型图上创建出来的代码模型元素；
2. 引用到代码模型中的代码元素（包含关联空间中的引用的代码元素）；

如何检查

查询基于代码模型图构出的代码模型架构树，找出与架构方案不匹配（标红）的元素。

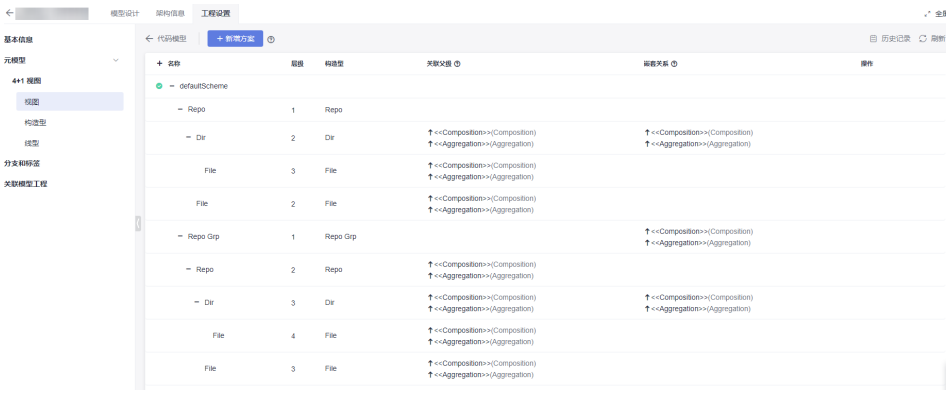
正确示例

架构层级规则示例：

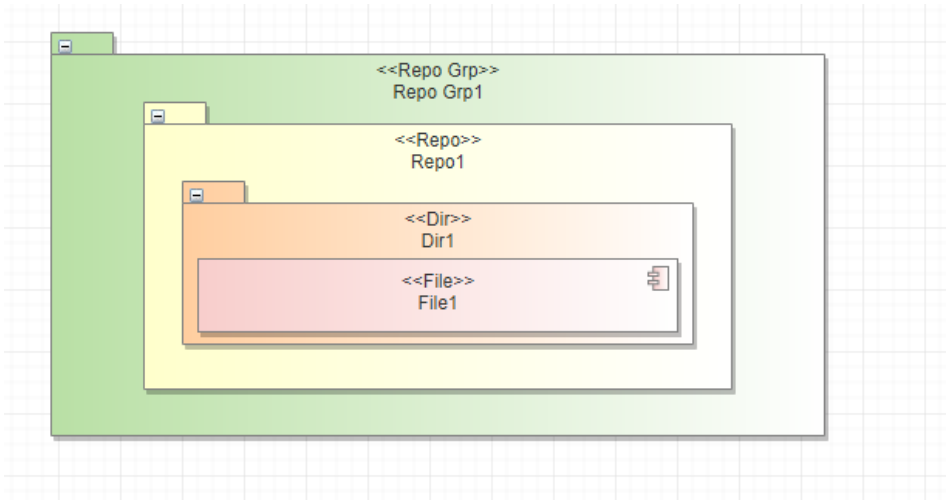
关联父级：配置的是当前层级元素与上一层级的元素之间的连线类型和父子关系指向。

嵌套：是否支持当前类型的元素与同类型元素建立关系。

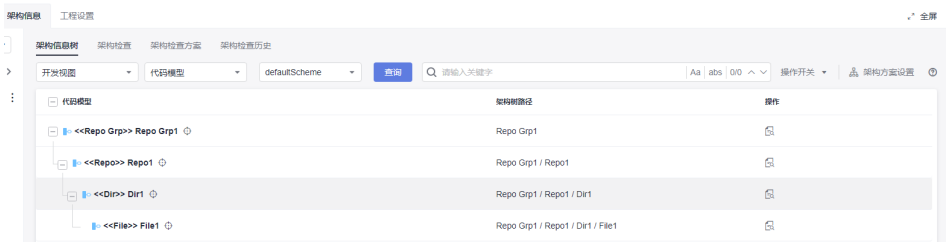
嵌套关系：当前类型的元素与同类型元素建立连线关系类型，指向关系默认为父指向子（即被指向的一方为子）。



图中画法示例1--包含的父子关系：



架构信息树展示结果：

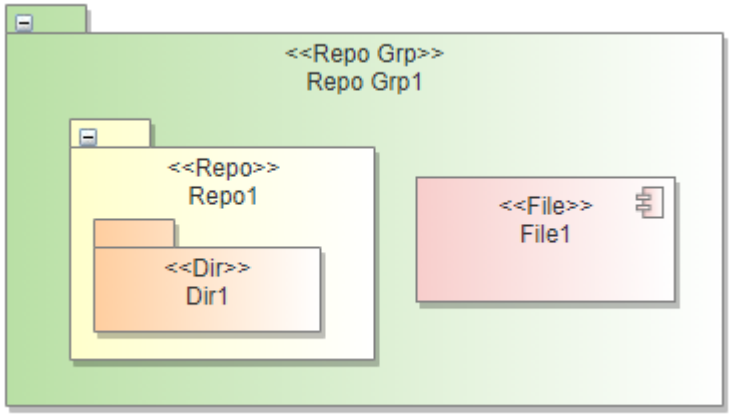


当架构树上没有标红元素，就没有2.3.1的检查错误结果。

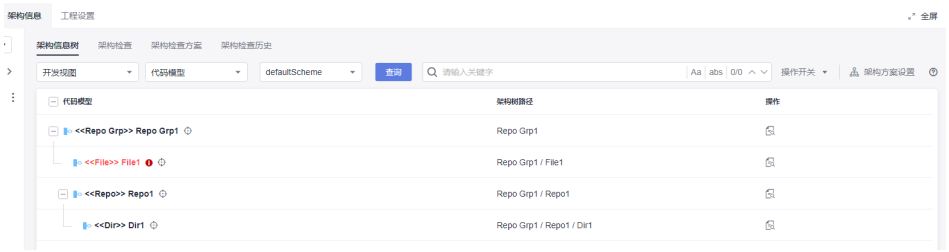
错误示例

场景一：方案中没有配置子节点，但是在画图设计中绘制了子节点。

File画在Repo Grp下是子节点，但方案中Repo Grp没有配置File为子节点。



架构信息树中报红：



架构检查结果：

2.3.2 代码模型不能存在游离的代码模型元素

详细描述

代码模型元素不能独立存在于代码架构树之外，必须要与架构树上的代码元素建立关联关系。

检查范围

当前模型工程中的所有符合定义规则的代码模型元素（定义规则：工程设置>构造型下，绑定到4+1视图：代码模型的基础构造型与自定义构造型元素才认定为代码模型元素）。

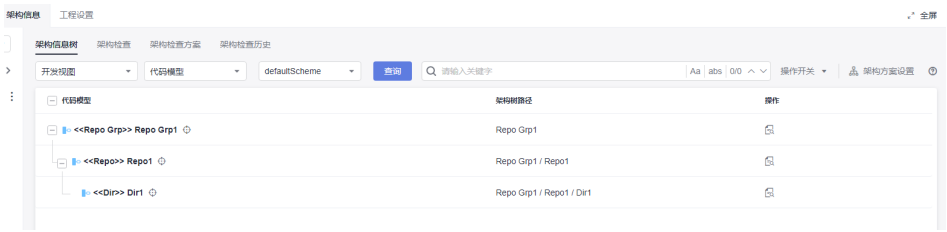
1. 在代码模型图上创建出来的代码模型元素；
2. 引用到代码模型中的代码元素（包含关联空间中的引用的代码元素）；

如何检查

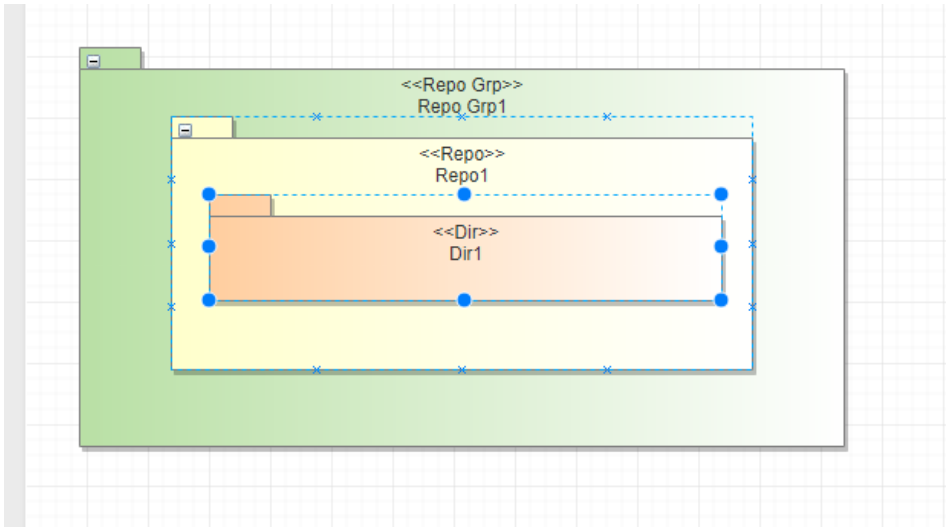
查询基于模型图（只有代码模型图内的代码元素参与构树）并展示不匹配元素构出的代码模型架构树，找出所有代码元素中不在架构树中的代码元素。

正确示例

按代码架构方案构建的架构信息树：

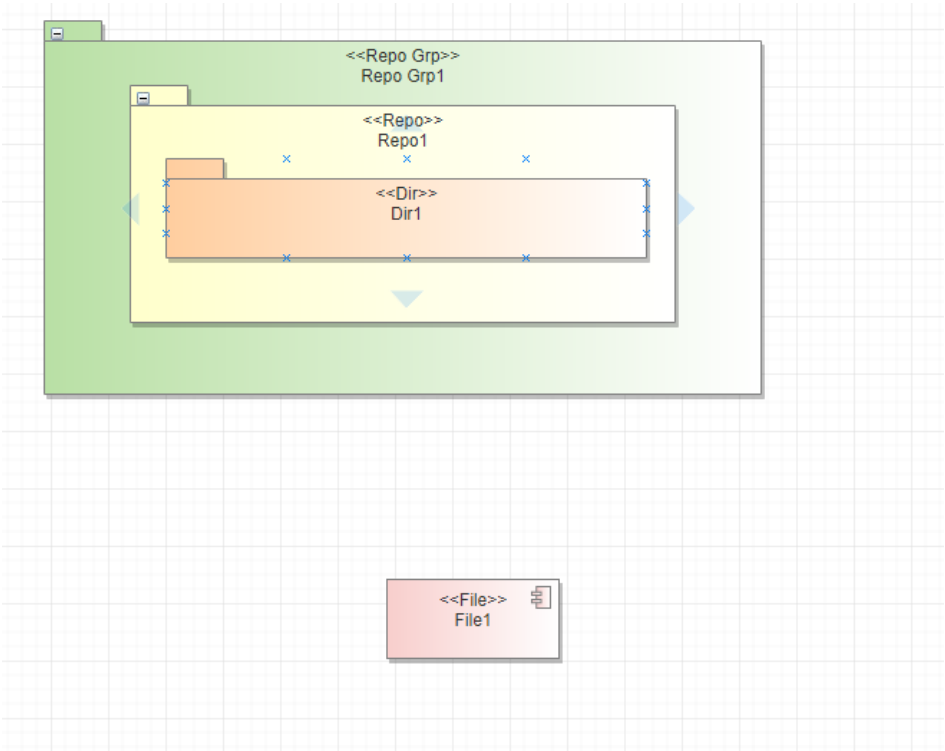


模型图示例：



错误示例

场景一：存在没有连线且没有与包含关系的独立元素。



检查结果：

2.3.3 代码模型同一个树的同一层上不能有同名同类型的元素

详细描述

在同一棵代码架构信息树上，在同一个父元素节点下面，不能存在类型相同，并且名称也相同的元素。

检查范围

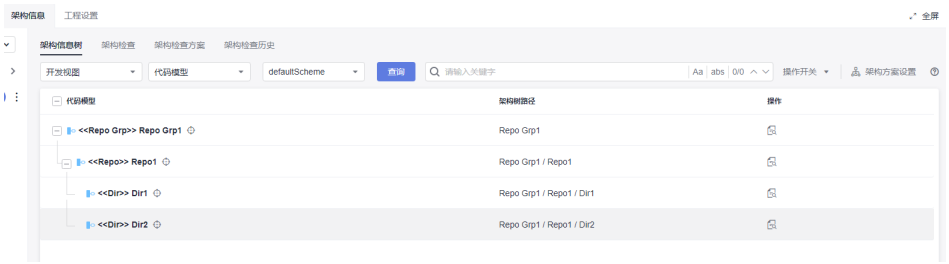
当前模型工程中的所有符合定义规则的代码模型元素（定义规则：工程设置>构造型下，绑定到4+1视图：代码模型的基础构造型与自定义构造型元素才认定为代码模型元素）。

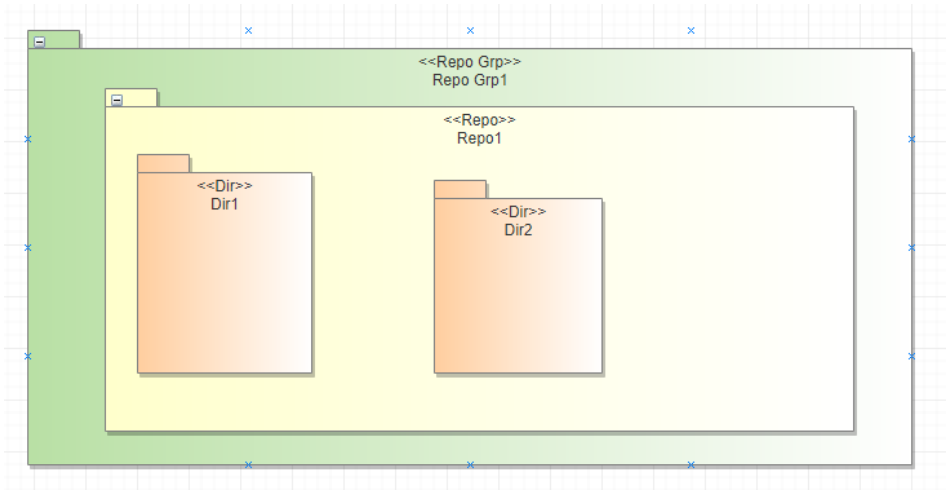
1. 在代码模型图上创建出来的代码模型元素；
2. 引用到代码模型中的代码元素（包含关联空间中的引用的代码元素）；

如何检查

查询基于代码模型图（只有代码模型图内的代码元素参与构树）构出的代码模型架构树，找出同一节点下同名同类型的代码元素。

正确示例

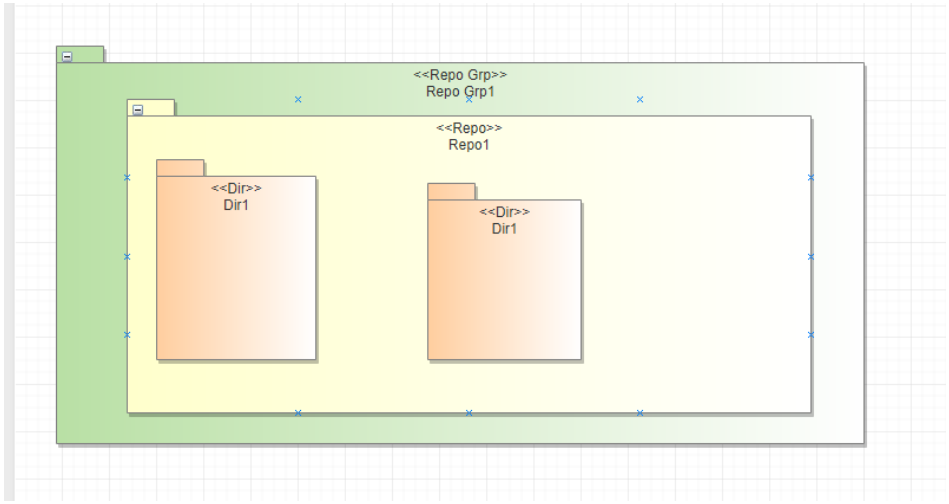


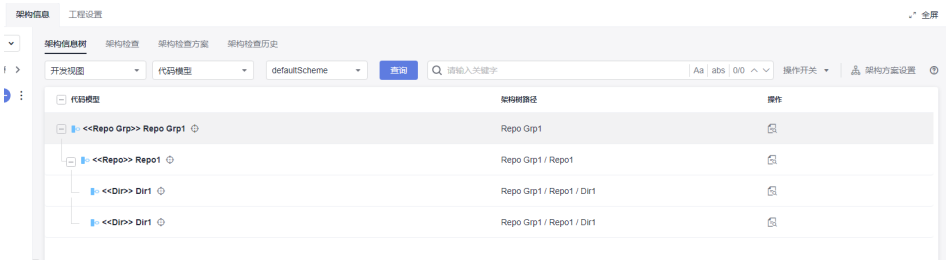


错误示例

场景一：同父元素下面存在同类型且同名称的元素。

按逻辑规则构建的架构信息树，树上不会显示异常。





检查结果：

2.3.4 代码元素与逻辑元素只能是 manifest 关系，且代码元素只能对应一个逻辑元素

详细描述

代码元素与逻辑元素之间的连线类型只能使用manifest连线，且指向方向由代码元素指向逻辑元素；一个代码元素只能连到一个逻辑元素上，而逻辑元素可以连线多个代码元素，即由多个代码元素指向构成。

检查范围

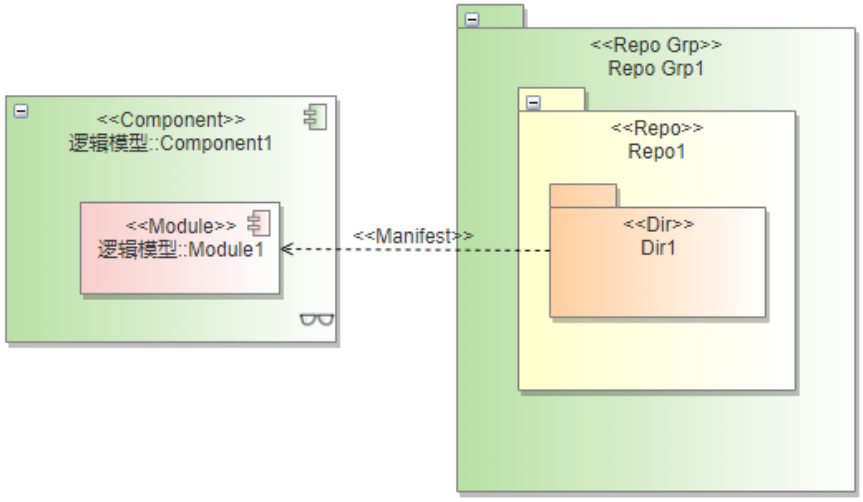
当前模型工程中的所有符合定义规则的代码模型元素（定义规则：工程设置>构造型下，绑定到4+1视图：代码模型的基础构造型与自定义构造型元素才认定为代码模型元素）。

1. 在代码模型图上创建出来的代码模型元素；
2. 引用到代码模型中的代码元素（包含关联空间中的引用的代码元素）；

如何检查

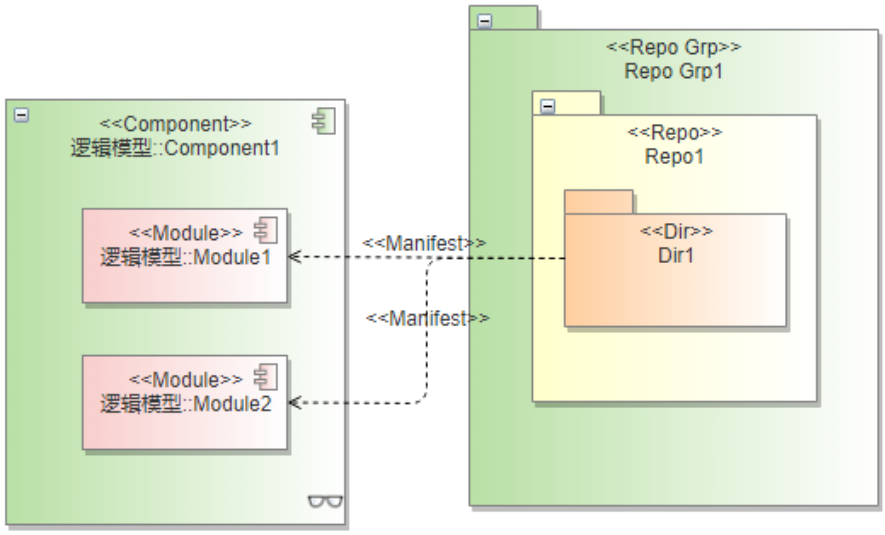
检查所有的代码模型图中的代码元素与逻辑元素之间的连线关系是否为manifest连线关系，并检查代码元素是否只与一个逻辑元素有manifest关系，如果有2个及以上的逻辑元素则不合规则，会列到检查结果中。

正确示例



错误示例

场景一：一个代码元素对应到两个及两个以上的逻辑元素（一对多）。



架构规则检查结果，列出不符合检查项的代码元素：

2.3.5 逻辑元素至少与一个代码元素存在 manifest 关系

详细描述

逻辑模型中的逻辑元素或从逻辑模型引用到代码模型中的逻辑元素至少要与一个代码元素中间有manifest连线关系。

当前规则支持配置检查类型后，已包含3.1.1的检查项，建议使用2.3.6检查项即可，3.1.1可不再重复检查。

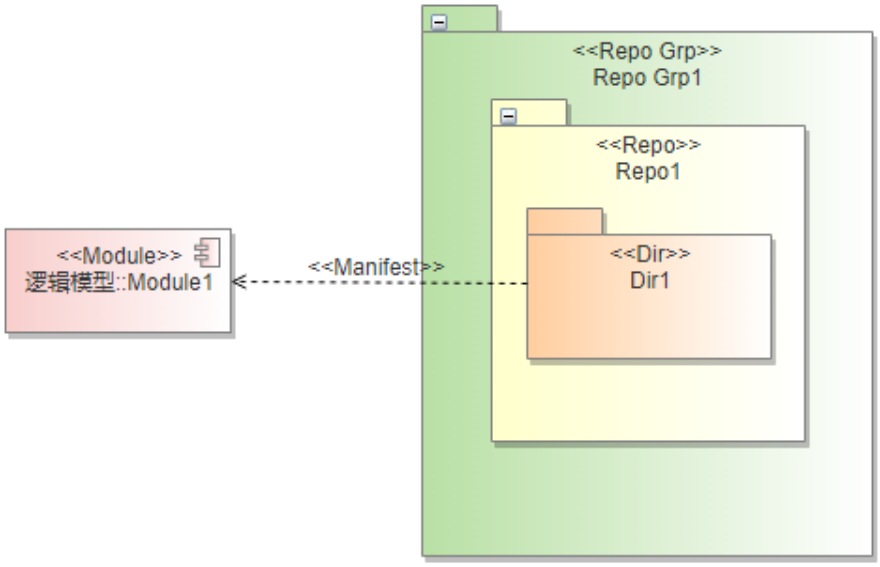
检查范围

1. 在逻辑模型图上创建出来的逻辑模型元素；
2. 引用到代码模型中的逻辑元素；
3. 排除Interface、Provided Interface、Required Interface元素。

如何检查

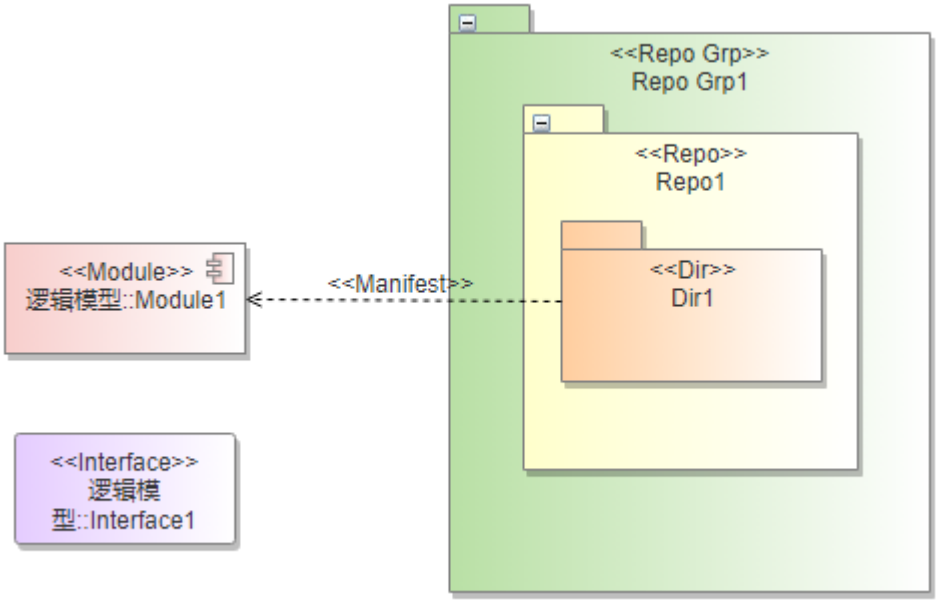
检查规则配置中勾选要检查的元素类型，服务、微服务、组件、模块是默认强制勾选的检查类型，检查这类元素在代码模型图中是否与代码元素存在manifest连线关系，由代码元素指向逻辑元素，不存在对应的代码元素则不符合规则 ，将该类逻辑元素列出到检查结果中。

正确示例



错误示例

引用过来的逻辑元素Interface没有对应任何代码元素。



1.8.1.2.4 构建模型

2.4.1 构建模型的元素要与指定的构建模型层次结构保持一致

详细描述

在构建模型中创建构建元素，构建元素在架构树中与上下级元素的关系层级结构要与构建模型架构方案配置定义的层次结构一致，即该构建元素与上层父级元素、下层子级元素的父子关系（也称上下层级关系）、以及它们之间的连线关系和方向指向，都要与层级规则中定义的保持一致。

检查范围

当前模型工程中的所有符合定义规则的构建元素（定义规则：工程设置>构造型下，绑定到4+1视图：构建模型的基础构造型与自定义构造型元素才认定为构建元素）。

1. 在构建模型图上创建出来的构建元素；
2. 引用到构建模型中的构建元素（包含关联空间中的引用的构建元素）；

如何检查

查询基于模型图构出的构建模型架构树，找出与架构方案不匹配（标红）的元素。

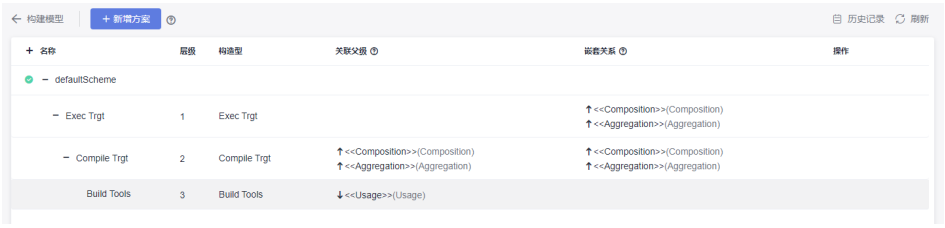
正确示例

架构层级规则示例：

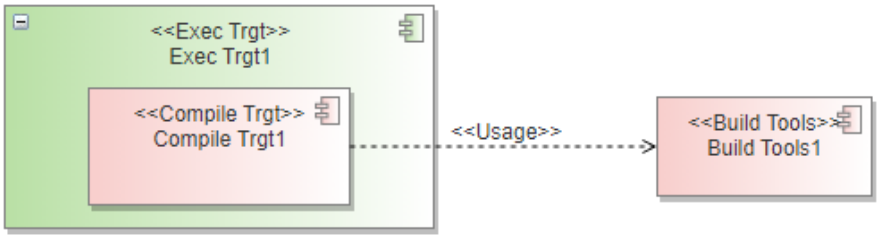
关联父级：配置的是当前层级元素与上一层级的元素之间的连线类型和父子关系指向。

嵌套：是否支持当前类型的元素与同类型元素建立关系。

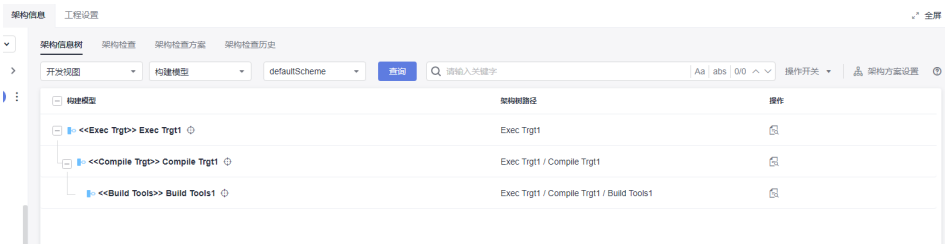
嵌套关系：当前类型的元素与同类型元素建立连线关系类型，指向关系默认为父指向子（即被指向的一方为子）。



图中画法示例1–包含的父子关系和连线关系：

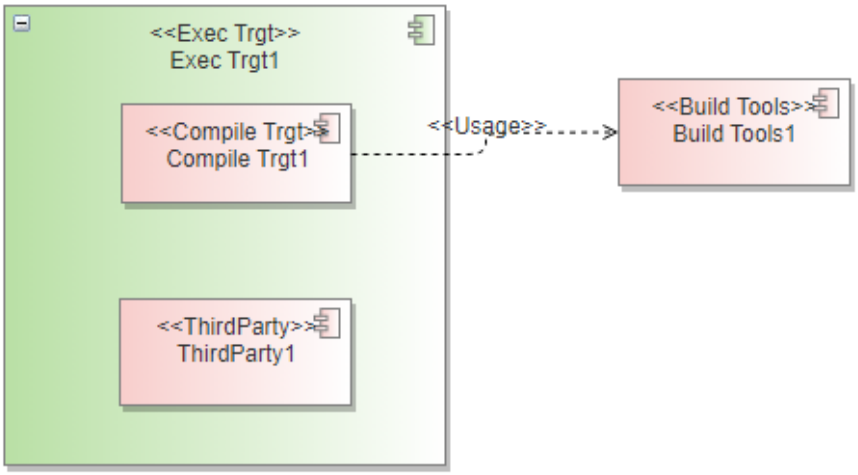


架构信息树展示结果：

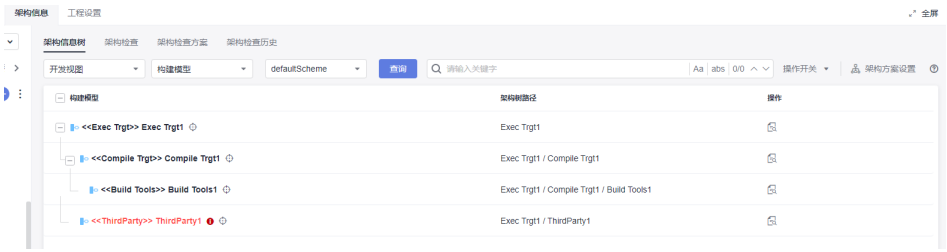


错误示例

错误示例场景1：方案中未配置元素关系，但是绘图中新增了关系。



架构信息树中报红：



架构检查结果：

2.4.2 构建模型不能存在游离的构建模型元素

详细描述

构建模型元素不能独立存在于构建模型架构树之外，必须要与架构树上的构建元素建立关联关系。

检查范围

当前模型工程中的所有符合定义规则的构建元素（定义规则：工程设置>构造型下，绑定到4+1视图：构建模型的基础构造型与自定义构造型元素才认定为构建元素）。

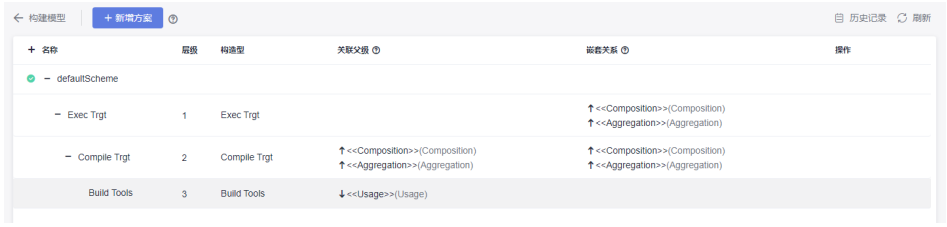
1. 在构建模型图上创建出来的构建元素；
2. 引用到构建模型中的构建元素（包含关联空间中的引用的构建元素）；

如何检查

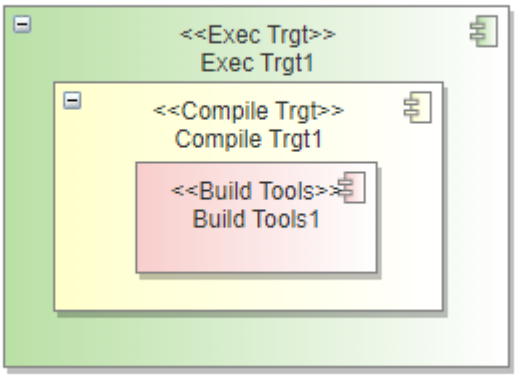
查询基于模型图（只有构建模型图内的构建元素参与构树）并展示不匹配元素构出的构建模型架构树，找出所有构建元素中不在架构树中的构建元素。

正确示例

按构建规则构建的架构信息树：

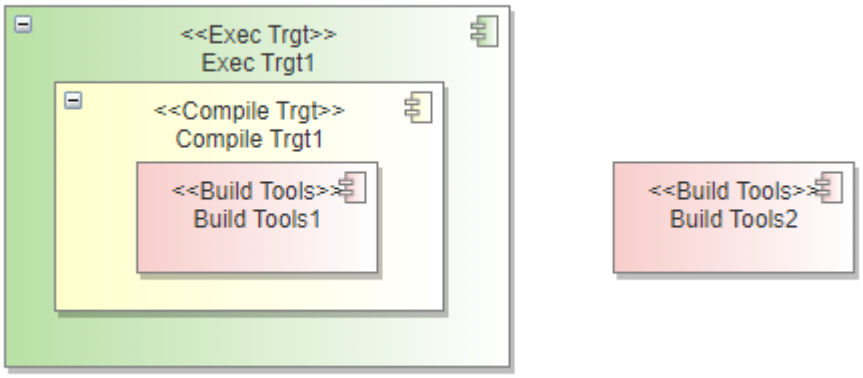


模型图示例：



错误示例

场景一：独立存在的构建元素



检查结果：

2.4.3 构建模型同一个树的同一层上不能有同名同类型的元素

详细描述

在同一棵构建架构信息树上，在同一个父元素节点下面，不能存在类型相同，并且名称也相同的构建元素；

检查范围

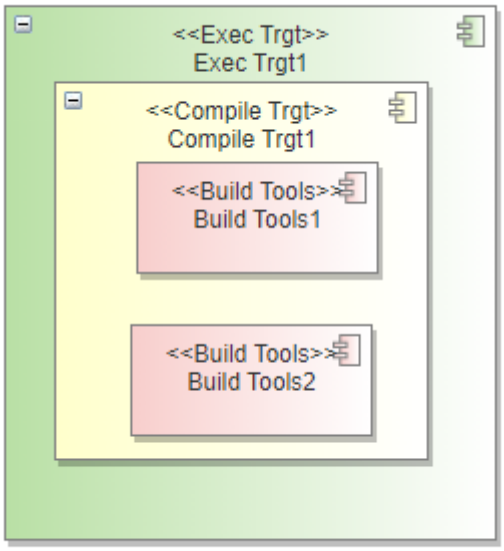
当前模型工程中的所有符合定义规则的构建元素（定义规则：工程设置>构造型下，绑定到4+1视图：构建模型的基础构造型与自定义构造型元素才认定为构建元素）。

1. 在构建模型图上创建出来的构建元素；
2. 引用到构建模型中的构建元素（包含关联空间中的引用的构建元素）；

如何检查

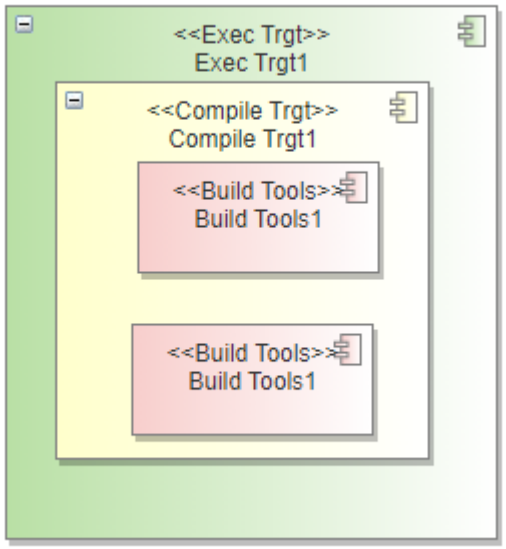
查询基于构建模型图（只有构建模型图内的构建元素参与构树）构出的构建模型架构树，找出同一节点下同名同类型的构建元素。

正确示例

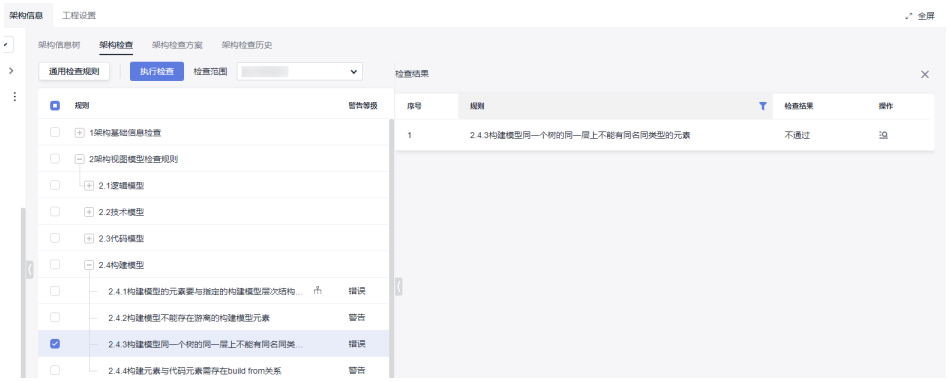


错误示例

错误示例场景1：同父节点下面存在类型相同，名称相同的构建元素。



检查结果：



2.4.4 构建元素与代码元素需存在 build from 关系

详细描述

在构建模型图上，构建元素与从代码模型中引用过来的代码元素如果存在连线关系，必须为build from的构建关系，且方向指向得从构建元素指向代码元素。

检查范围

当前模型工程中的所有符合定义规则的构建元素（定义规则：工程设置>构造型下，绑定到4+1视图：构建模型的基础构造型与自定义构造型元素才认定为构建元素）。

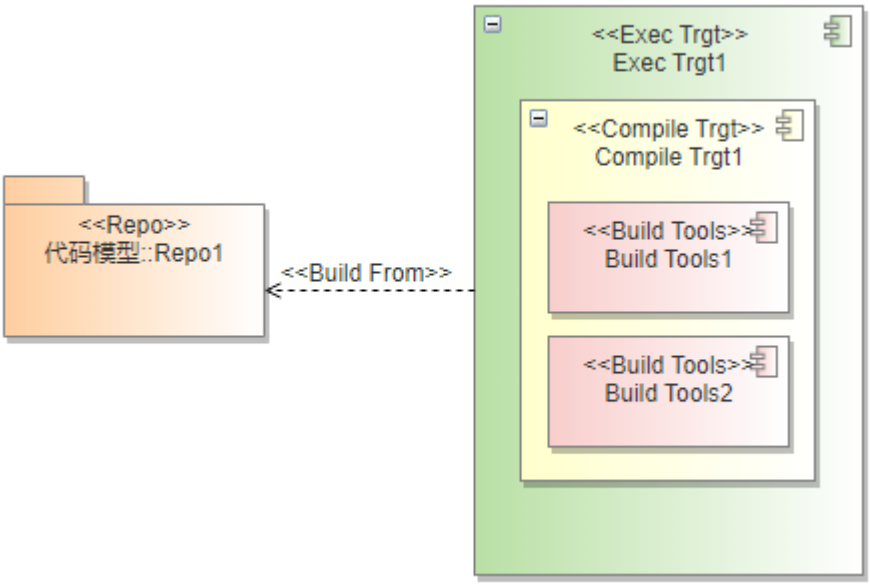
1. 在构建模型图上创建出来的构建元素；
2. 引用到构建模型中的构建元素（包含关联空间中的引用的构建元素）；
3. 引用到构建模型中的代码元素（代码元素的定义参考代码模型检查章节）；

如何检查

检查构建模型中构建元素与引用过来的代码元素之间的连线关系类型是否为build from，如果不存在，则列出这类不符合规则的构建元素。

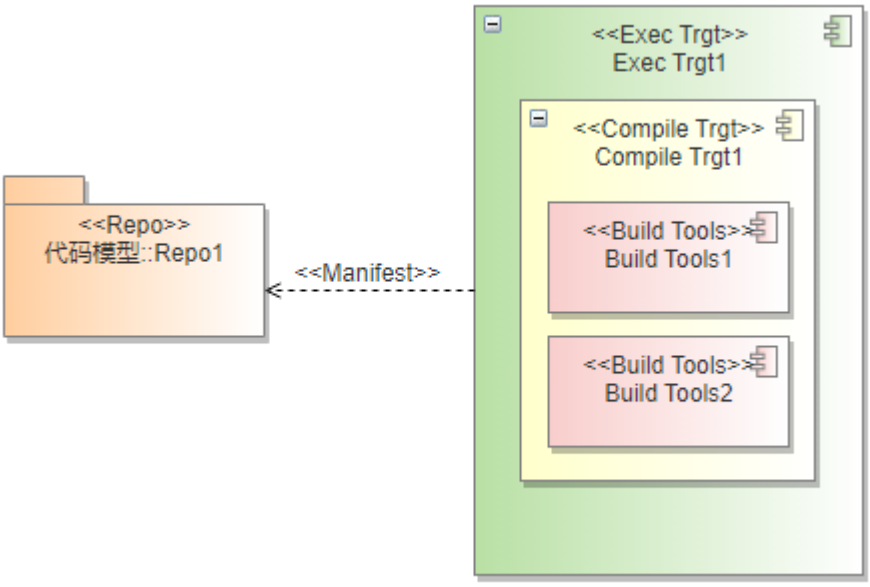
正确示例

构建元素与引用过来的代码元素存在连线关系，关系为Build From类型，且方向由构建元素指向代码元素。



错误示例

错误示例场景1：构建元素与代码元素之间的连线关系类型不对。



检测结果：

1.8.1.2.5 交付模型

2.5.1 交付模型的元素要与指定的交付模型层次结构保持一致

详细描述

在交付模型中创建交付元素，交付元素在架构树中与上下级元素的关系层级结构要与交付模型架构方案配置定义的层次结构一致，即该交付元素与上层父级元素、下层子级元素的父子关系（也称上下层级关系）、以及它们之间的连线关系和方向指向，都要与层级规则中定义的保持一致。

该规则项检查出的是架构信息树中按模型图构树后显示红色叹号的告警元素。

检查范围

当前模型工程中的所有符合定义规则的构建元素（定义规则：工程设置>构造型下，绑定到4+1视图：交付模型的基础构造型与自定义构造型元素才认定为交付元素）。

1. 在交付模型图上创建出来的交付元素；
2. 引用到交付模型中的交付元素（包含关联空间中的引用的交付元素）；

如何检查

查询基于模型图构出的交付模型架构树，找出与架构方案不匹配（标红）的元素

正确示例

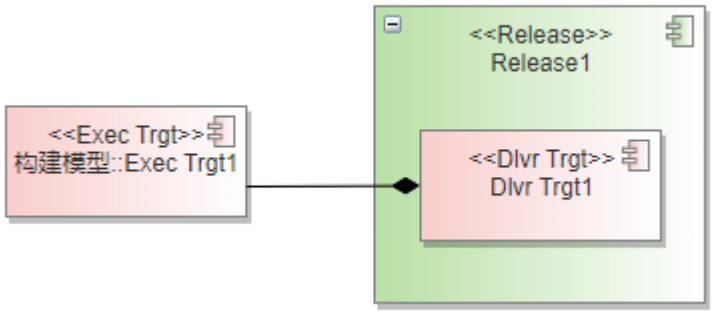
架构层级规则示例：

关联父级：配置的是当前层级元素与上一层级的元素之间的连线类型和父子关系指向。

嵌套：是否支持当前类型的元素与同类型元素建立关系。

嵌套关系：当前类型的元素与同类型元素建立连线关系类型，指向关系默认为父指向子（即被指向的一方为子）。

图中画法示例–构建元素连向交付元素，使用组合关系：

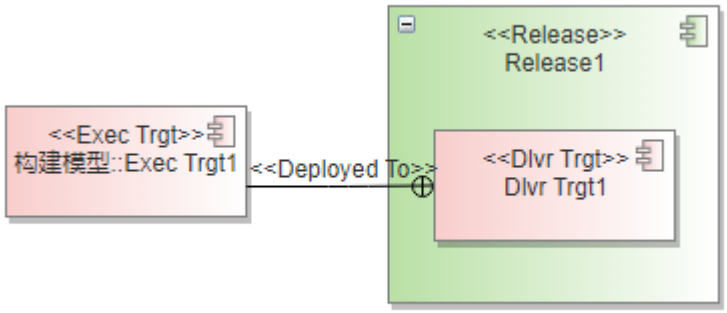


架构信息树展示结果：

当架构树上没有标红元素，就没有2.5.1的检查错误结果。

错误示例

错误示例场景1：连线类型与配置方案中规定的不一致。



架构信息树中报红：

架构检查结果：

2.5.2 交付模型不能存在游离的交付模型元素

详细描述

交付模型元素不能独立存在于交付架构树之外，必须要与架构树上的交付元素建立关联关系。

检查范围

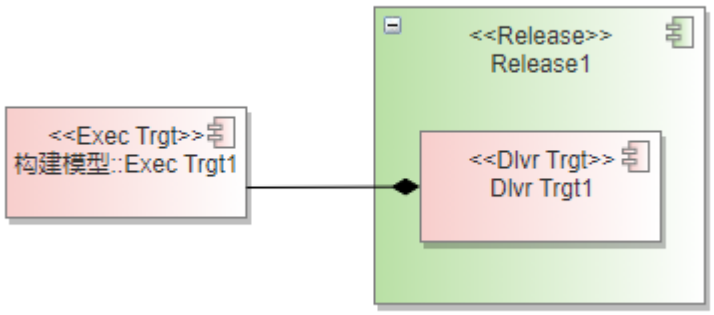
当前模型工程中的所有符合定义规则的构建元素（定义规则：工程设置>构造型下，绑定到4+1视图：交付模型的基础构造型与自定义构造型元素才认定为交付元素）。

1. 在交付模型图上创建出来的交付元素；
2. 引用到交付模型中的交付元素（包含关联空间中的引用的交付元素）；

如何检查

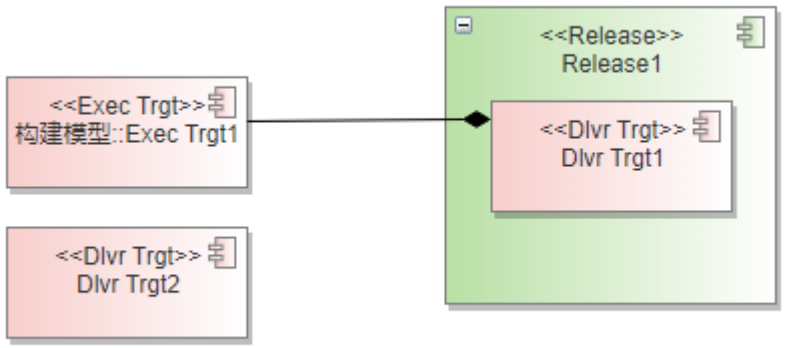
查询基于模型图（只有交付模型图内的交付元素参与构树）并展示不匹配元素构出的交付模型架构树，找出所有交付元素中不在架构树中的交付元素。

正确示例

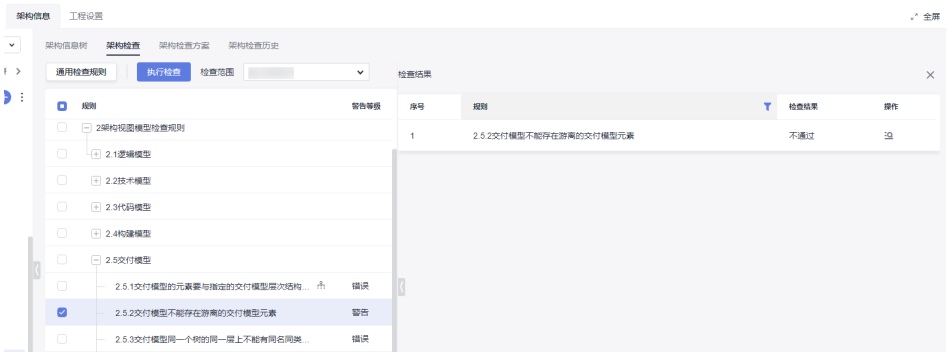


错误示例

错误示例场景1：独立的交付元素Dlvr Trgt存在图上。



架构检查结果：



2.5.3 交付模型同一个树的同一层上不能有同名同类型的元素

详细描述

在同一棵交付架构信息树上，在同一个父元素节点下面，不能存在类型相同，并且名称也相同的交付元素。

检查范围

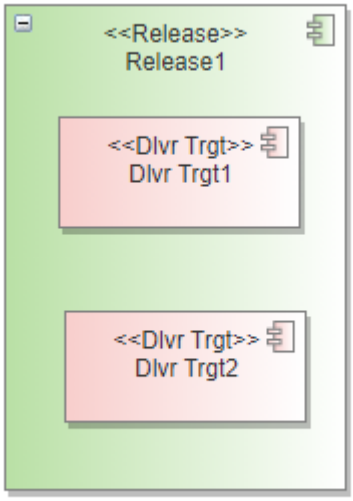
当前模型工程中的所有符合定义规则的交付元素（定义规则：工程设置>构造型下，绑定到4+1视图：交付模型的基础构造型与自定义构造型元素才认定为交付元素）。

1. 在交付模型图上创建出来的交付元素；
2. 引用到交付模型中的交付元素（包含关联空间中的引用的交付元素）；

如何检查

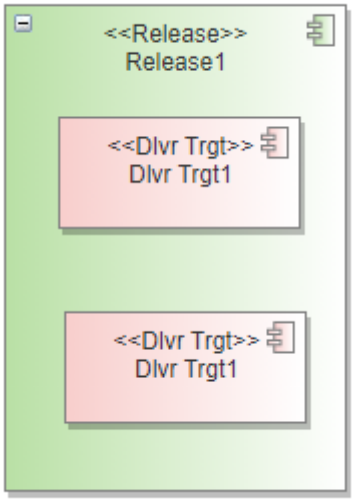
查询基于交付模型图（只有交付模型图内的交付元素参与构树）构出的交付模型架构树，找出同一节点下同名同类型的交付元素。

正确示例



错误示例

错误示例场景1：同父节点下面存在类型相同，名称相同的构建元素。



检测结果：

2.5.4 交付模型中构建元素与交付元素需存在组合或聚合关系

详细描述

在交付模型图上，交付元素与引用过来的构建元素如果存在连线关系，必须为组合或者聚合的关系，且方向指向得从构建元素指向交付元素。

检查范围

当前模型工程中的所有符合定义规则的构建元素（定义规则：工程设置>构造型下，绑定到4+1视图：交付模型的基础构造型与自定义构造型元素才认定为交付元素）。

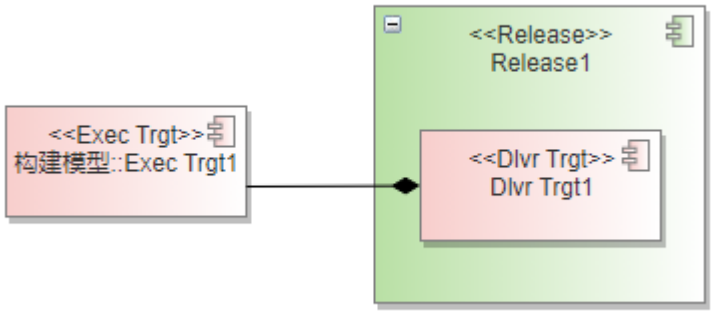
1. 在交付模型图上创建出来的交付元素；
2. 引用到交付模型中的交付元素（包含关联空间中的引用的交付元素）；
3. 引用到交付模型中的构建元素；（构建元素的定义参考代码模型检查章节）；

如何检查

检查交付模型中的引用来的构建元素与交付元素之间的连线关系类型是否为组合Compostion或者聚合Aggregation关系类型，如果为其它类型连线关系，则将该类型元素列出到检查结果中。

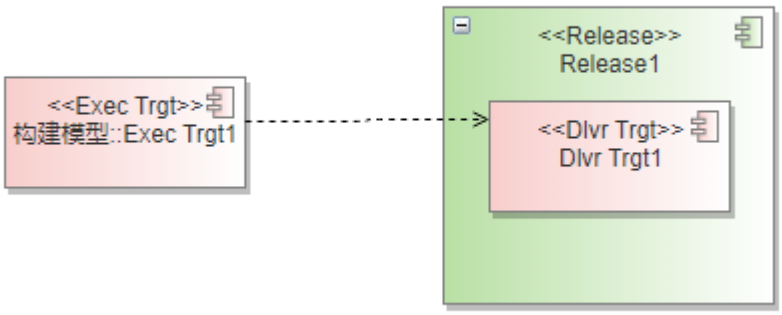
正确示例

交付元素与引用过来的构建元素存在连线关系，关系为组合Composition连线类型，且方向由构建元素指向交付元素。

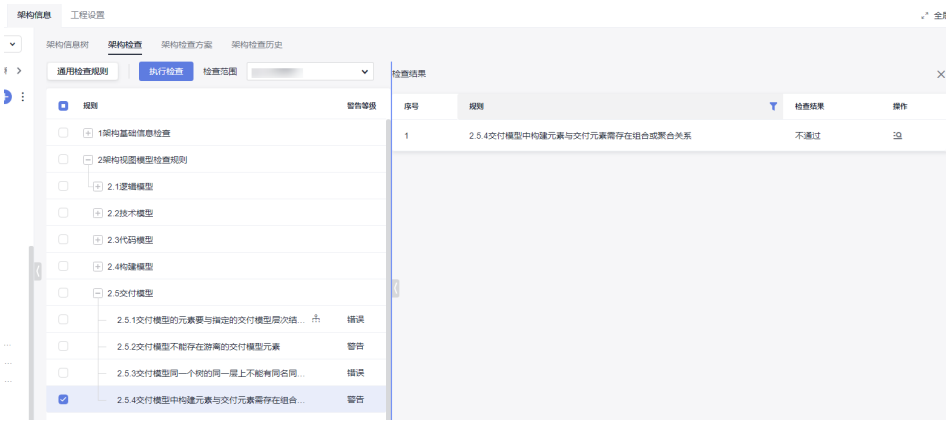


错误示例

错误示例场景1：构建元素与交付元素之间的连线关系类型不对。



检测结果：



1.8.1.2.6 部署模型

2.6.1 检查部署模型的元素是否与架构层次结构是否一致

详细描述

在部署模型中创建部署元素，部署元素在架构树中与上下级元素的关系层级结构要与部署模型架构方案配置定义的层次结构一致，即该部署元素与上层父级元素、下层子级元素的父子关系（也称上下层级关系）、以及它们之间的连线关系和方向指向，都要与层级规则中定义的保持一致。

该规则项检查出的是架构信息树中按模型图构树后显示红色叹号的告警元素。

检查范围

当前模型工程中的所有符合定义规则的部署元素（定义规则：工程设置>构造型下，绑定到4+1视图：部署模型的基础构造型与自定义构造型元素才认定为部署元素）。

1. 在部署模型图上创建出来的部署元素；
2. 引用到部署模型中的部署元素（包含关联空间中的引用的部署元素）；

如何检查

查询基于模型图构出的部署模型架构树，找出与架构方案不匹配（标红）的元素。

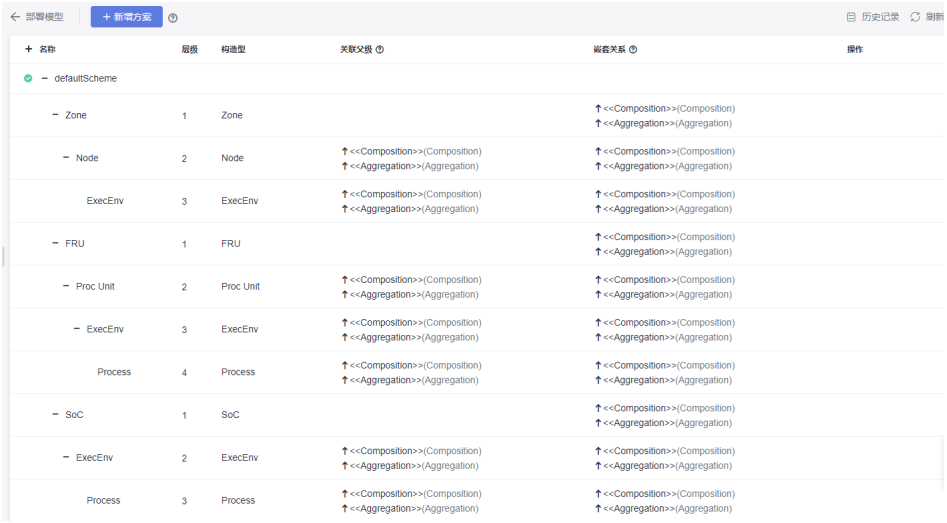
正确示例

架构层级规则示例：

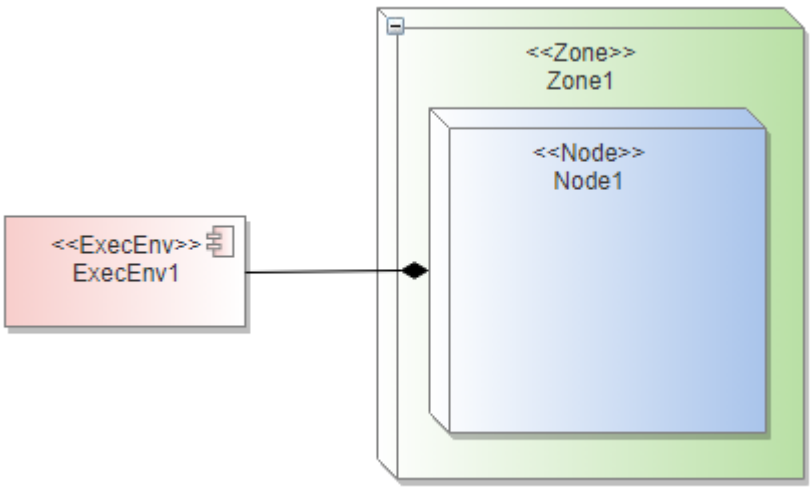
关联父级：配置的是当前层级元素与上一层级的元素之间的连线类型和父子关系指向。

嵌套：是否支持当前类型的元素与同类型元素建立关系

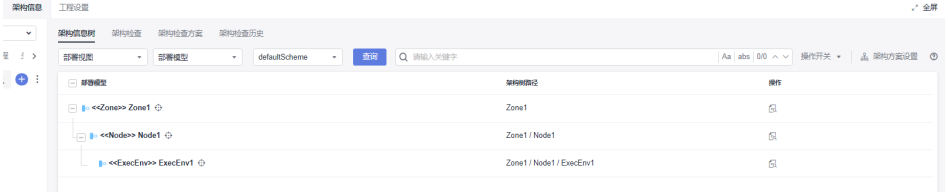
嵌套关系：当前类型的元素与同类型元素建立连线关系类型，指向关系默认为父指向子（即被指向的一方为子）。



包含的父子关系：



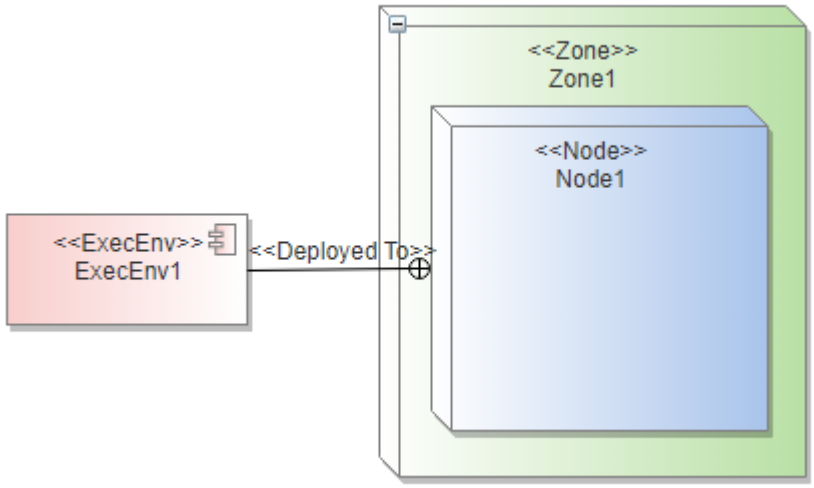
架构信息树展示结果：



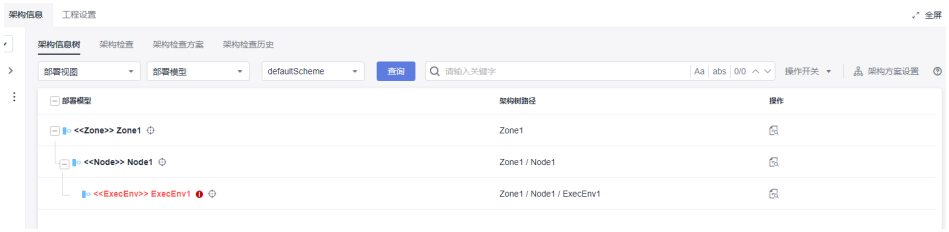
当架构树上没有标红元素，就没有2.6.1的检查错误结果。

错误示例

错误示例场景1：连线类型使用不对。



架构信息树中报红：



架构检查结果：

2.6.2 部署模型不能存在游离的部署模型元素

详细描述

部署模型元素不能独立存在于整个部署架构树之外，必须要与任何一个在架构树上的部署元素建立关系。

检查范围

当前模型工程中的所有符合定义规则的部署元素（定义规则：工程设置>构造型下，绑定到4+1视图：部署模型的基础构造型与自定义构造型元素才认定为部署元素）。

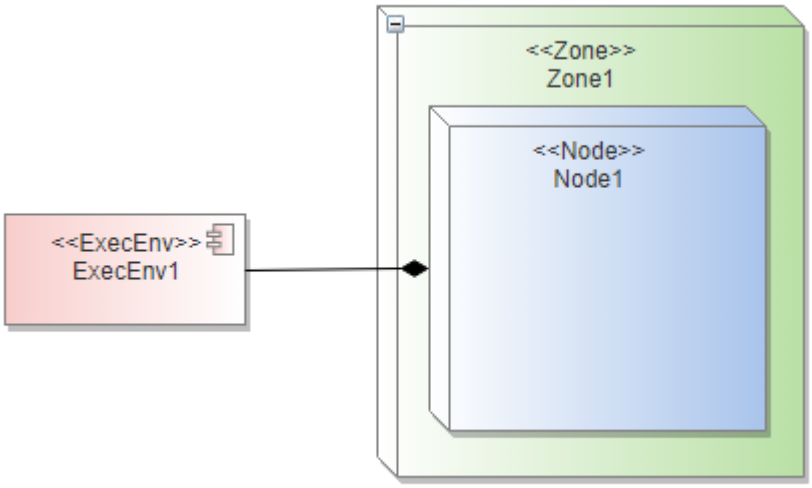
1. 在部署模型图上创建出来的部署元素；
2. 引用到部署模型中的部署元素（包含关联空间中的引用的部署元素）；

如何检查

查询部署模型图内元素类型为架构方案配置构造型的所有元素，查询基于模型图构出的部署模型架构树。

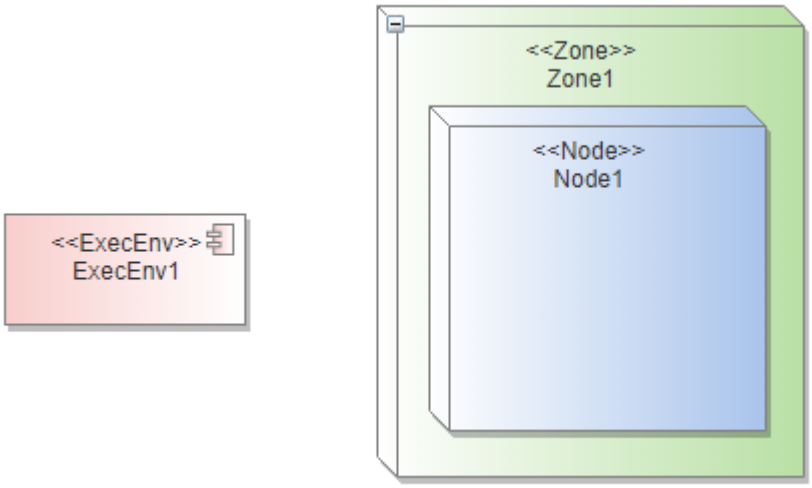
正确示例

每个部署元素都有连线关系和上下级关系（包含关系）。



错误示例

错误示例场景1：存在没有建立任务关系的部署元素在图上。



架构检查结果：

2.6.3 部署模型同一个树的同一层上不能有同名同类型的元素

详细描述

在同一棵部署架构信息树上，在同一个父元素节点下面，不能存在类型相同，并且名称也相同的元素。

检查范围

当前模型工程中的所有符合定义规则的部署元素（定义规则：工程设置>构造型下，绑定到4+1视图：部署模型的基础构造型与自定义构造型元素才认定为部署元素）。

1.在部署模型图上创建出来的部署元素；

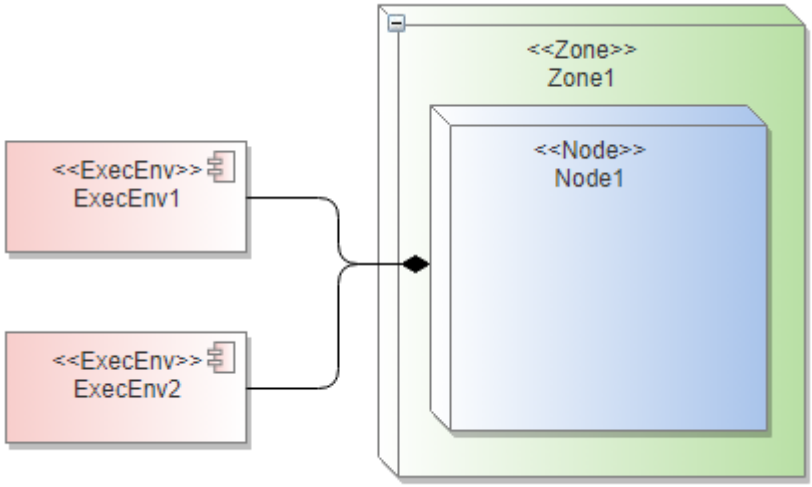
2.引用到部署模型中的部署元素（包含关联空间中的引用的部署元素）；

如何检查

查询基于模型图构出的部署模型架构树，找出架构树上同一层同名同类型的元素。

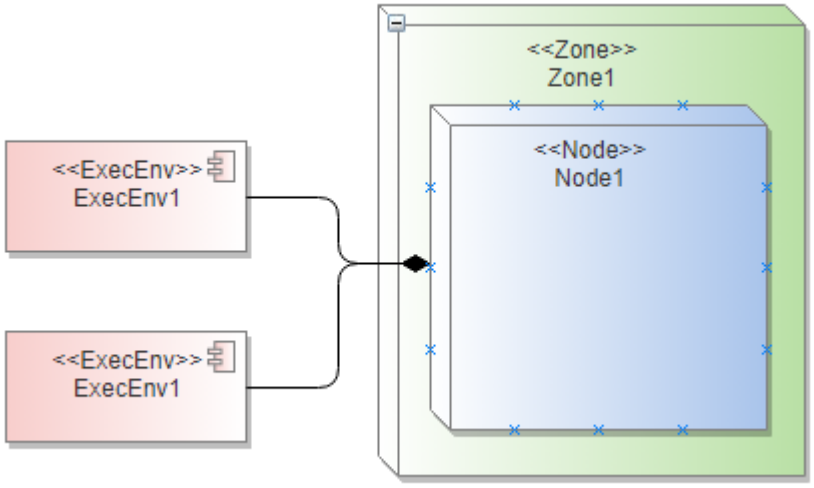
正确示例

同Node节点下面的不能有同类型同名元素。



错误示例

错误示例场景1：存在同名同类型的部署元素。



架构规则检查结果：

2.6.4 交付目标和可执行目标至少与一个部署元素存在部署关系

详细描述

交付目标和可执行目标（Dlvr Trgt、Exec Trgt）至少与一个部署元素有部署关系或者包含关系，部署关系指部署连线关系，即Deployed To连线关系；包含关系，即部署元素在图上包含交付目标和可执行目标（Dlvr Trgt、Exec Trgt）元素。

检查范围

当前模型工程中的所有符合定义规则的部署元素（定义规则：工程设置>构造型下，绑定到4+1视图：部署模型的基础构造型与自定义构造型元素才认定为部署元素）。

1.在部署模型图上创建出来的部署元素；

2.引用到部署模型中的部署元素（包含关联空间中的引用的部署元素）；

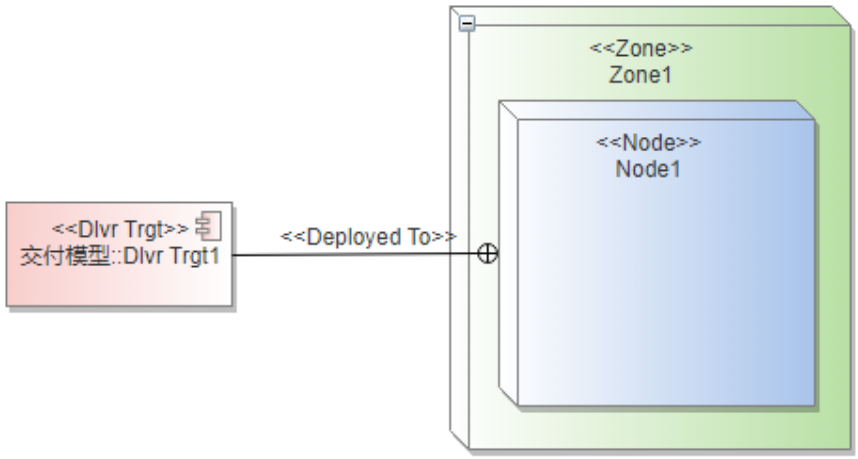
3.交付目标和执行目标，即Dlvr Trgt和Exec Trgt元素；

如何检查

查询部署模型图中生成的部署元素，查询部署模型图中的交付目标和可执行目标（Dlvr Trgt、Exec Trgt），找出交付目标和可执行目标中既无Deployed To部署元素连线关系也没有父子关系（部署元素为父，交付目标和可执行目标为子）的元素。

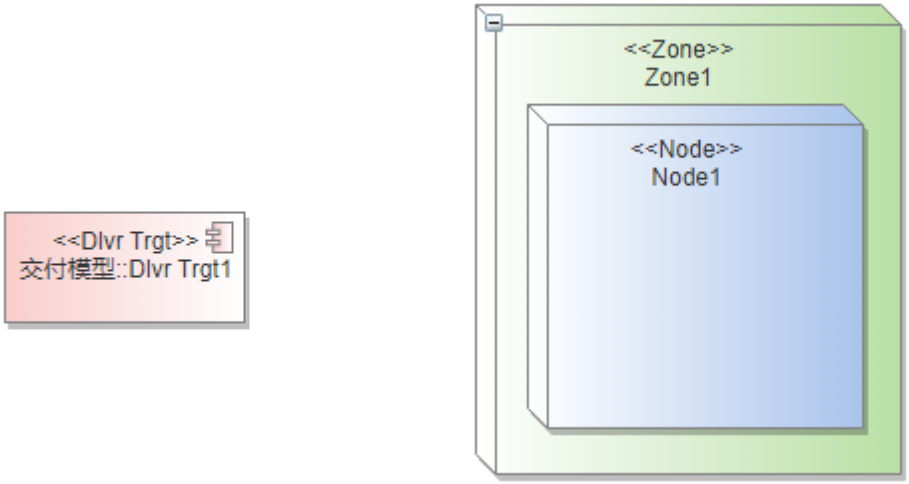
正确示例

交付目标与部署元素存在部署连线关系，且由交付元素指向部署元素。

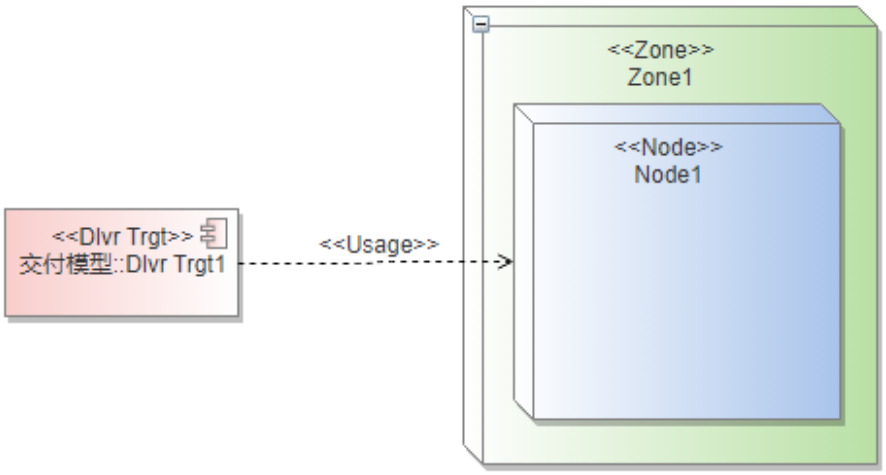


错误示例

错误示例场景1：交付元素与部署元素没有连线关系也没有包含关系。



错误示例场景2：交付元素与部署元素连线关系类型不正确。



1.8.1.2.7 上下文模型

2.7.1 上下文模型中只能有一个 System

详细描述

在上下文模型中只能存在一个类型为System的元素；其它的三方交互的对象用ExternalSystem或者Actor元素表示。

检查范围

当前模型工程中的所有符合定义规则的System元素，工程设置>构造型下，绑定到4+1视图：上下文模型的System元素。

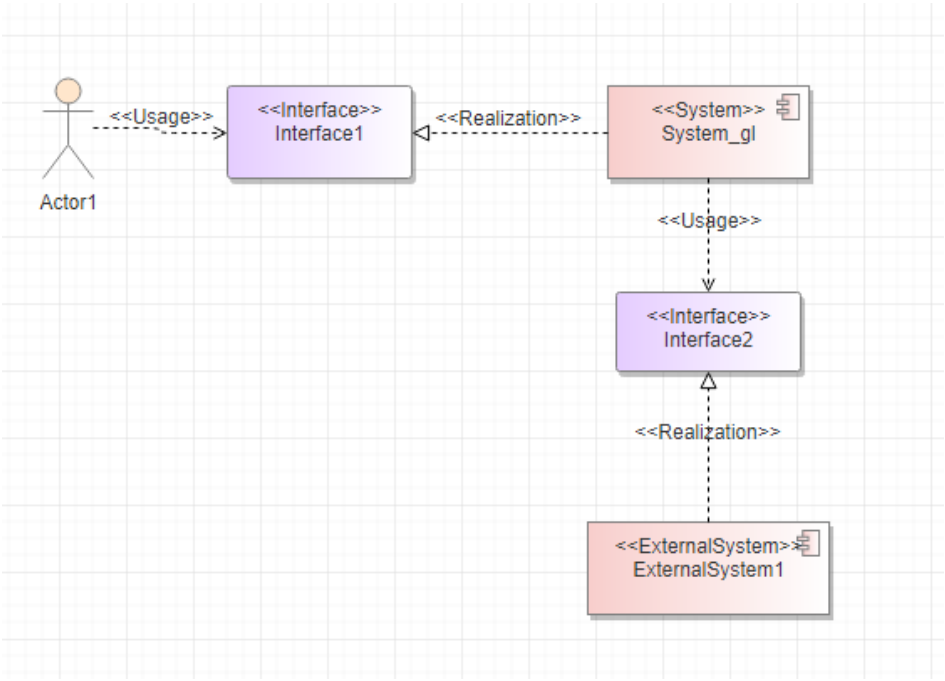
1.在上下文模型图上创建出来的System元素；

2.引用到上下文模型中的System元素（包含关联空间中的引用的system元素）；

如何检查

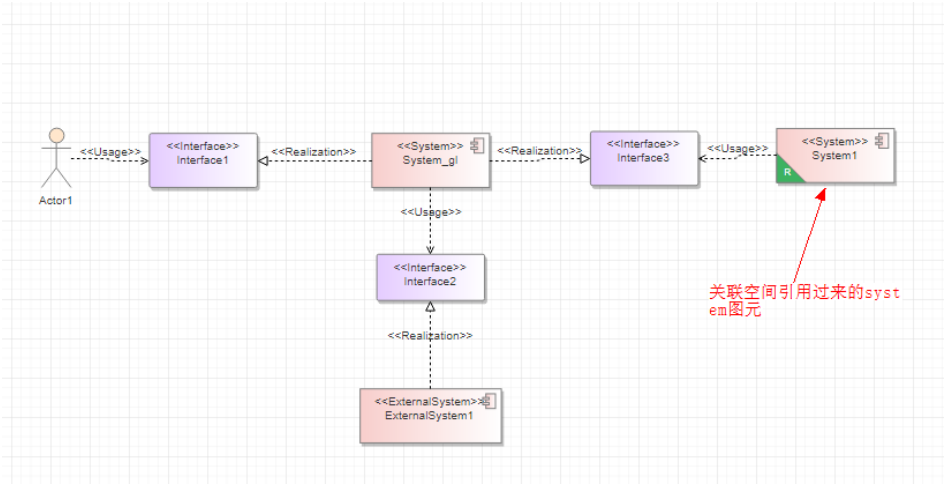
查询上下文模型图中的所有元素，从中找出类型为System的元素，如果存在多个System元素，则全部列出到检查结果中，不符合规则。

正确示例



错误示例

错误示例场景1：关联空间引用system元素到上下文。



架构规则检查结果：

2.7.2ExternalSystem 和 Actor 只能存在下面两种关系中的一种：ExternalSystem 和 Actor 使用或依赖 System 提供的接口；ExternalSystem 和 Actor 提供了接口给 System 使用或依赖

详细描述

ExternalSystem和Actor元素与System之间只能通过接口交互，不能直接使用连线关系表达交互，只能由ExternalSystem和Actor实现（Realization连线）接口，并由System使用（usage连线）该接口；或者由System实现（Realization连线）接口，由ExternalSystem和Actor使用（usage连线）该接口；其中使用关系可以由依赖Dependency连线关系代替。

检查范围

当前模型工程中的所有符合定义规则的System、ExternalSystem和Actor元素，工程设置>构造型下，绑定到4+1视图：上下文模型的System、ExternalSystem和Actor元素。

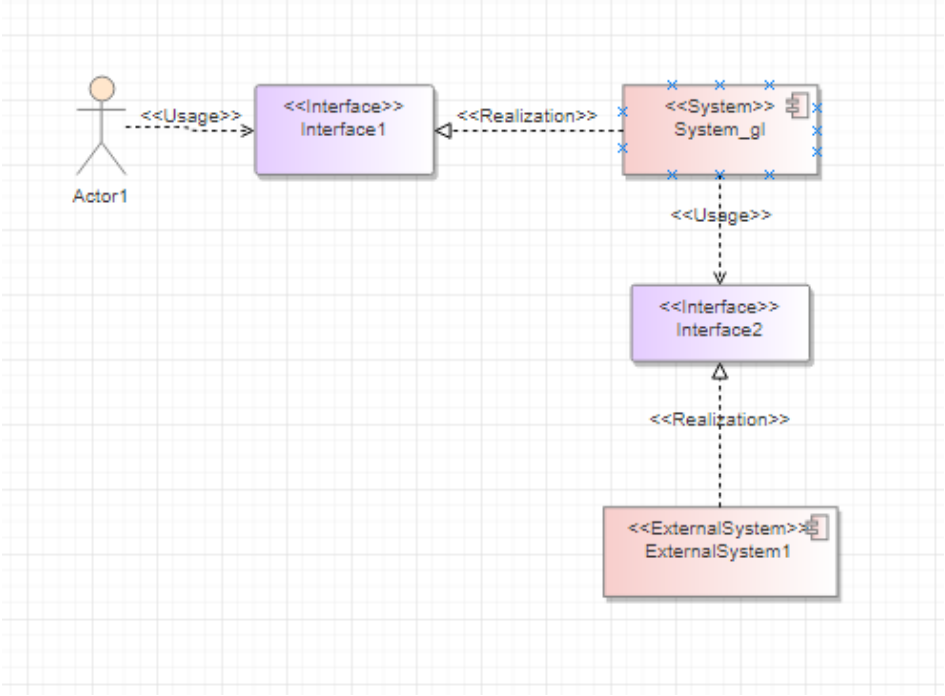
1.在上下文模型图上创建出来的System、ExternalSystem和Actor元素；

2.引用到上下文模型中的System、ExternalSystem和Actor元素（包含关联空间中的引用的System、ExternalSystem和Actor元素）；

如何检查

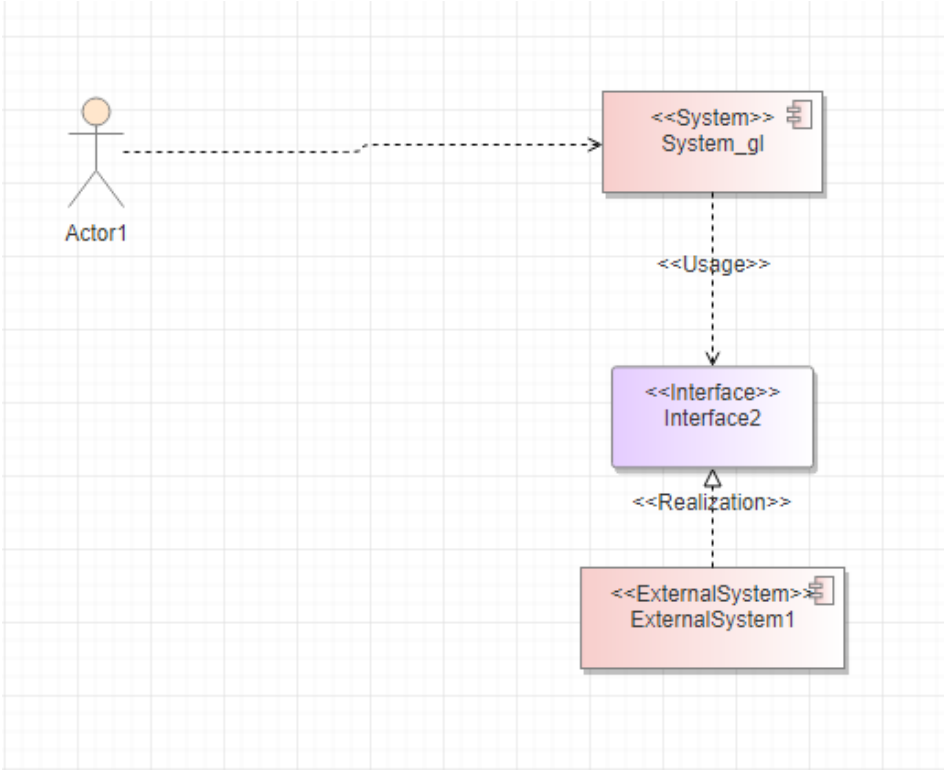
检查上下文模型中的ExternalSystem和Actor元素与System之间是否通过接口相关联，与接口之间存在使用，实现或者依赖关系连线，如果存在其它类型的连线也不符合规则，会列出不符合规则 的ExternalSystem和Actor元素在检查结果列表中。

正确示例



错误示例

错误示例场景1：没有通过接口交互，Actor与system直接用连线表示交互关系。



1.8.1.2.8 运行模型

2.8.1 运行模型、运行模型-顺序图、运行模型-活动图中不能产生新的逻辑元素

详细描述

在运行模型中不能创建新的逻辑元素，只能从逻辑模型中引用或者实例化到运行模型中来进行设计。

检查范围

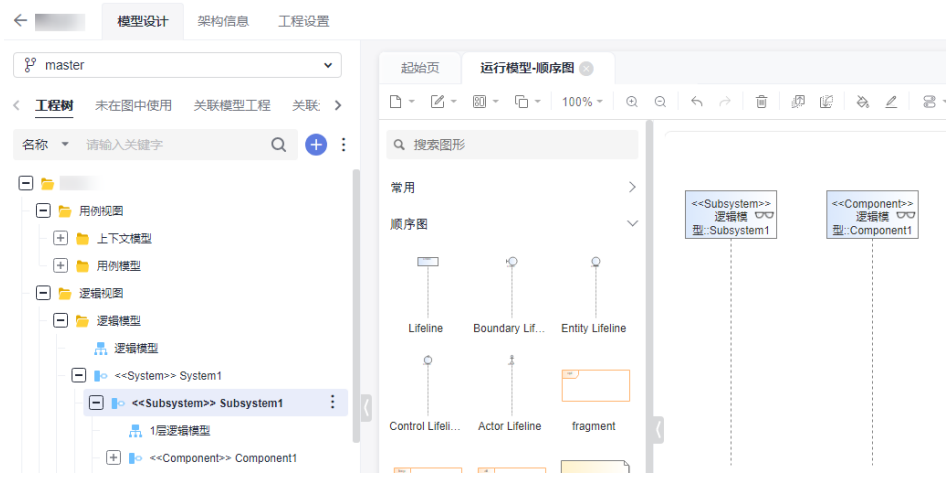
当前模型工程中的所有模型图类型为运行模型图上的逻辑元素，逻辑元素的定义参考逻辑模型检查章节。

如何检查

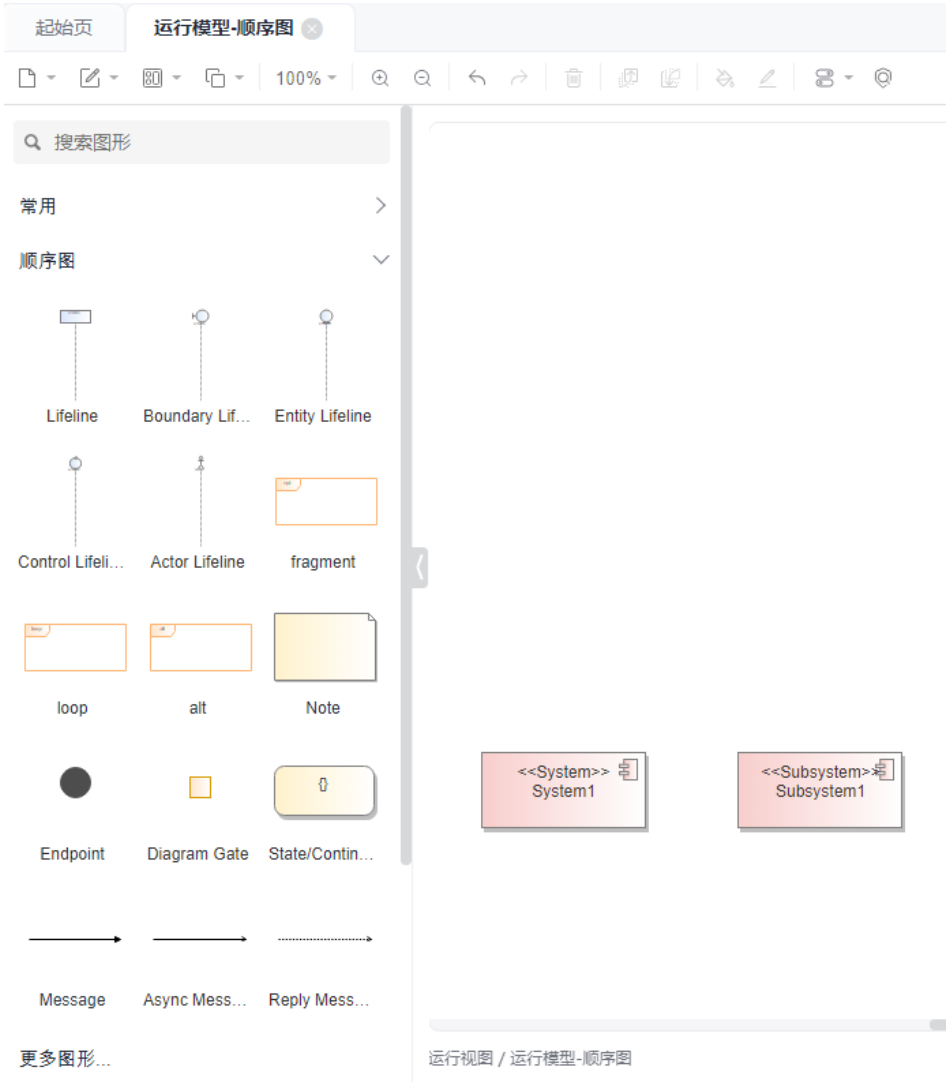
查询所有运行模型图中的元素，找出在运行模型图中创建生成出来的逻辑元素。

正确示例

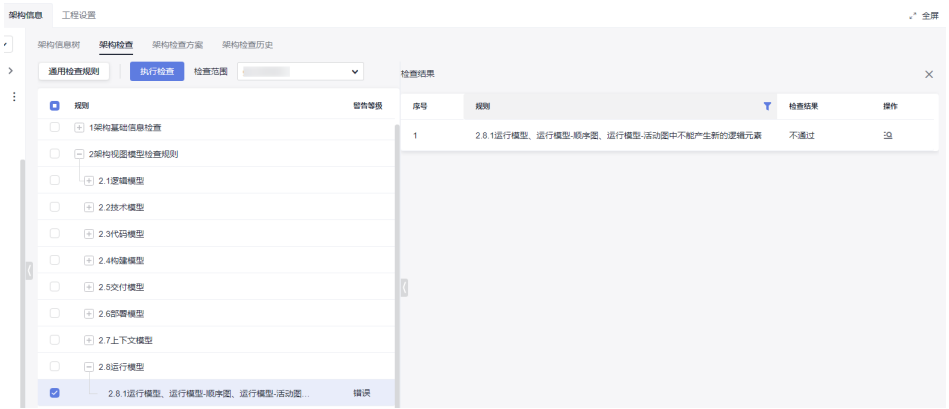
引用自逻辑模型的中定义的逻辑元素。



错误示例



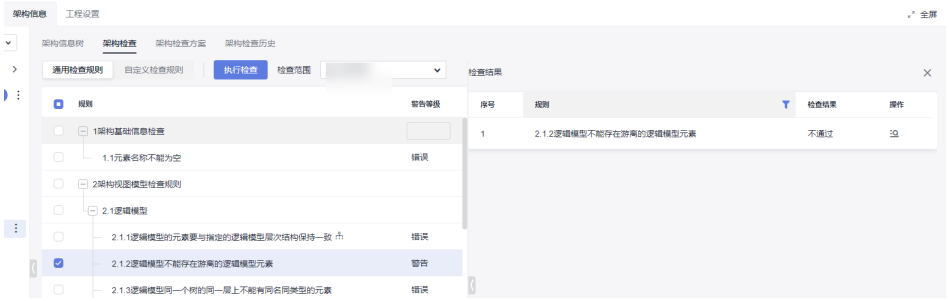
检测结果：



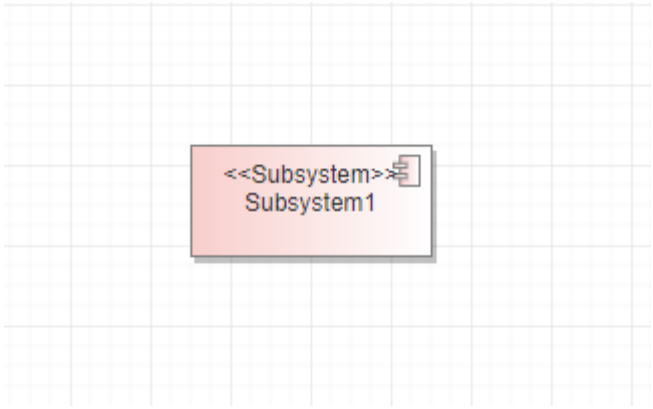
1.8.2 4+1 视图规范一致性检查错误修复指导

XX 模型不能存在游离的逻辑模型元素

以逻辑模型为例:



游离原因：元素没有在逻辑模型架构信息树中出现。

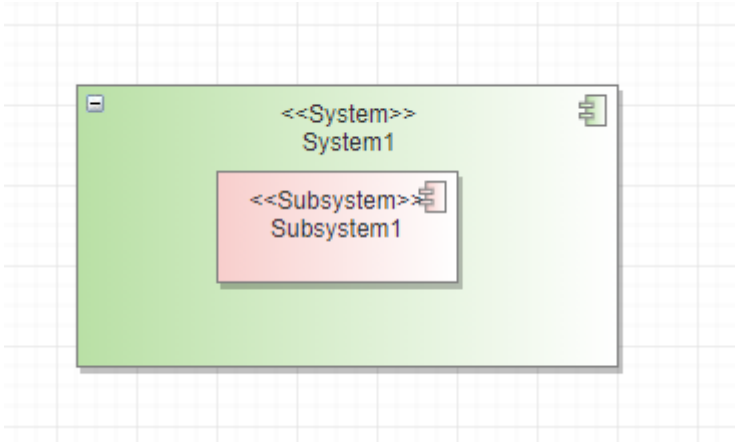


1. 查看逻辑模型架构方案设置。

2. 找到游离元素构造型相关的架构配置信息。

Subsystem需要与System有Composition/Aggregation关系或父子关系。

3. 在模型图中构建架构关系。

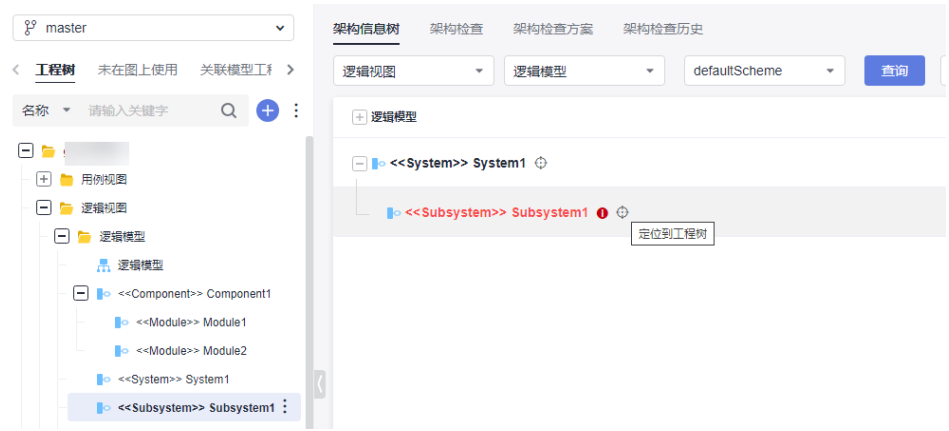


XX 模型的元素要与指定的 XX 模型层次结构保持一致

以逻辑模型为例:

1. 查询逻辑模型架构信息树，右侧操作开关把展示不匹配架构方案的元素打开。

2. 架构信息树构出后 根据错误元素名称查询定位到其所在架构树节点。



3. 查询错误元素与其他元素关系。

对比架构方案设置。

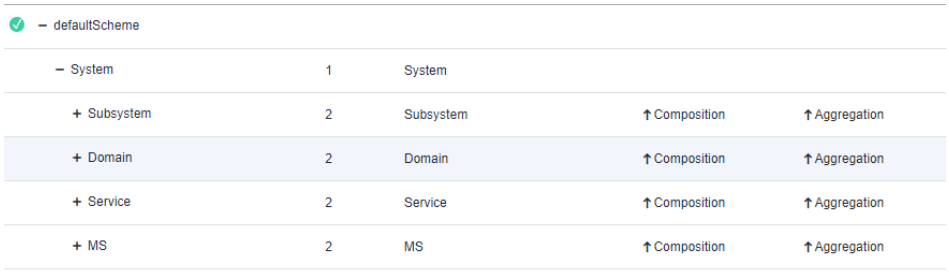
Subsystem1报错是因为与System1（架构信息树上的父节点）存在错误架构关系，对比发现实际模型图中使用的是Dependency连线 而架构配置方案要求Composition/Aggregation。

4. 在模型图中修改连线类型为Composition/Aggregation。

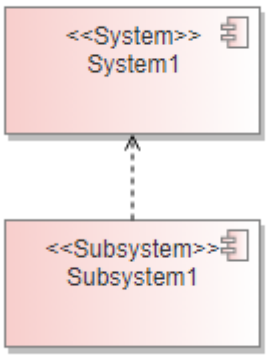
常见错误场景：

- 连线类型不符合架构配置方案。

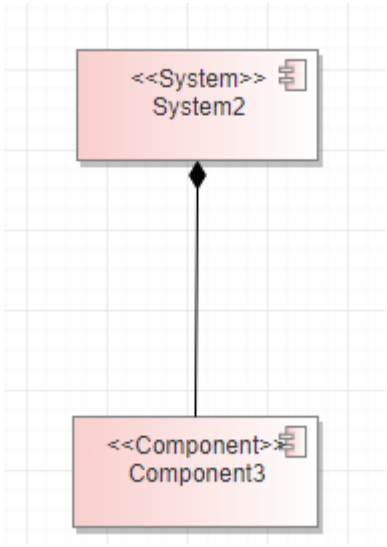
  规则

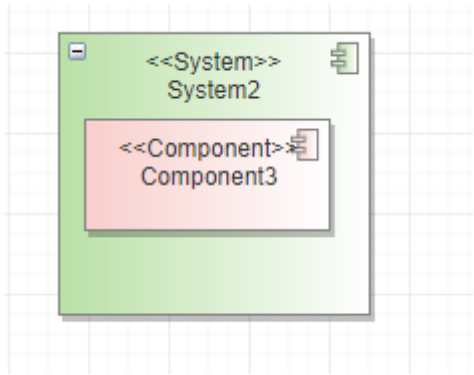


实际



- 子元素构造型不符合架构配置方案。





System下层子元素按架构配置方案只能是Subsystem、Domain、Service、MS 图中是Component。

2 UML 建模

2.1 概述

什么是 UML

UML是Unified Modeling Language缩写，译为统一建模语言，是始于1997年一个OMG标准，是一种面向对象的可视化建模语言。详情可查看UML教程 **https://sparxsystems.com/resources/tutorials/uml2/index.html**。

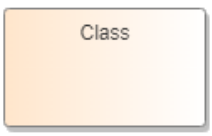
UML 有什么

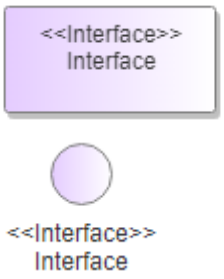
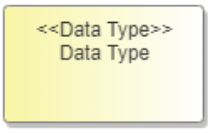
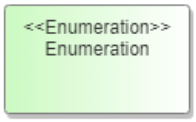
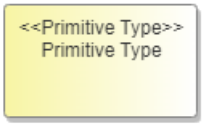
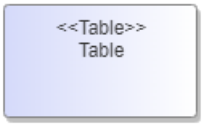
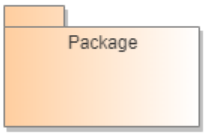
UML规范定义了两种主要的UML图，分别为结构图和行为图。

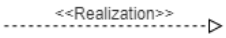
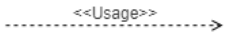
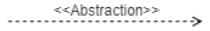
1. 结构图：显示了系统及其部件在不同抽象和实现级别上的静态结构以及它们如何相互关联。结构图中的元素表示系统的有意义的概念，并且可以包括抽象的，现实的和实现的概念。包括：类图、对象图、包图、组件图、复合结构图、部署图、配置文件图。
2. 行为图：显示了系统中对象的动态行为，可以将其描述为系统随时间的一系列更改。包括：用例图、活动图、状态图、顺序图、时序图、通信图、交互概览图。

2.2 类图

元素介绍

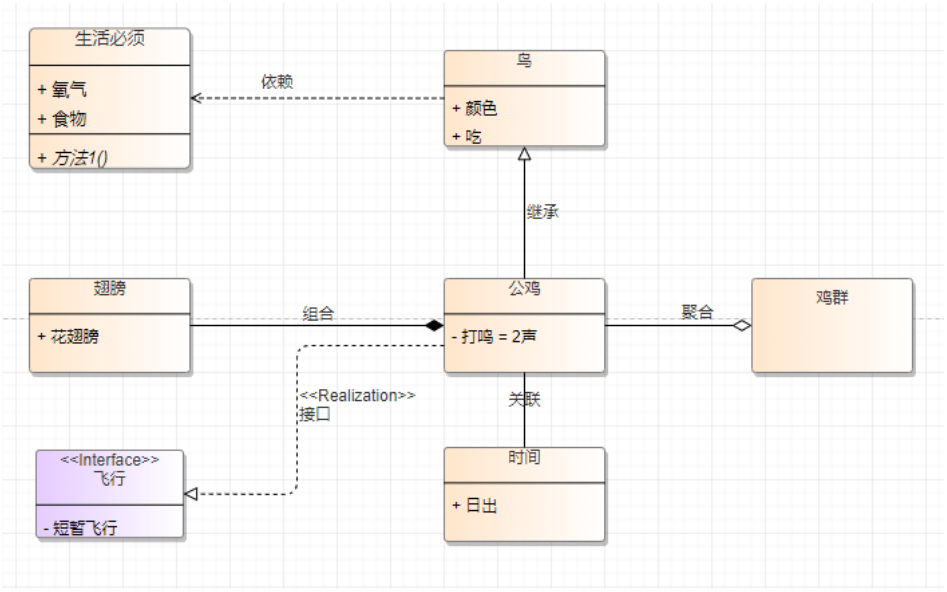
元素名	图标	含义
Class		是对象的集合，展示了对象的结构以及与系统的交互行为。

元素名	图标	含义
Interface		接口，可以是单个接口，也可以是抽象的一组接口的组合。 圆形接口与矩形接口意义相同，仅形状不同。
Data Type		数据类型包括原始预 定义的类型和用户自定义的类型。原始类型右：数字、字符串、乘方。用户定义的类型 是枚举类型。程序语言中用于实现的匿名数据类型可以用语言类型定义。
Enumeration		枚举是一种数据结构，它的实例构成了有名字的字面值。通常，同时声明枚举名和其字面值的名字。
Primitive Type		简单类型就是一个事先定义好了的基本数据类型，比如整数或者字符串。
Signal		对象之间异步通讯的声明。信号可以带有表示为属性的参数。
Table		代表一个数据库表的构造型组件。
Association Node		关联节点。
Package		包。
Generalization		泛化，表示类与类、接口与接口之间的继承关系，由子一方指向父对象一方。
Composition		组合，是整体与部分的关系，但部分不能离开整体而单独存在。

元素名	图标	含义
Aggregation		聚合，是整体与部分的关系，且部分可以离开整体而单独存在。
Realization		实现，是一种类与接口的关系，表示类是接口所有特征和行为的实现。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。
Usage		使用，是一种使用的关系。表明一个模块在运行的时候，需要使用另外一个模块。
Instantiate		实例化，声明用一个类的方法创建了另一个类的实例。
Constraint		是一个语义条件或者限制的表达式。UML 预定义了某些约束，其他可以由建模者自行定义。
Anchor		锚点。
Containment		内嵌，表示嵌在内部的类。
Abstraction		抽象是确认一件事物本质特征的行为，这种行为将这个事物与其他所有事物区分开来。 抽象依赖关系表示成从客户元素指向提供者元素的箭头。
Information flow		信息流表示任何图中两个元素之间的信息项（信息项元素或分类器）的流。
Association		关联，是一种拥有的关系，它使一个类知道另一个类的属性和方法。

类（Class Diagram）是对象的集合，展示了对象的结构以及与系统的交互行为。类主要有属性（Attribute）和方法（Operation）构成属性代表对象的状态，如果属性被保存到数据库，称为持久化，方法代表对象的操作行为，类具有继承关系，可以继承于父类，也可以与其他的Class进行交互。

类图展示了系统的逻辑结构，类和接口的关系。



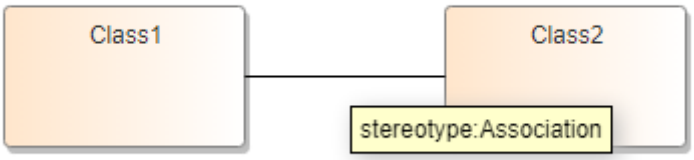
添加属性和方法

类元素添加属性和方法，选中元素右键“属性&方法”，属性&方法的编辑方式参考**如何添加元素属性和方法**。

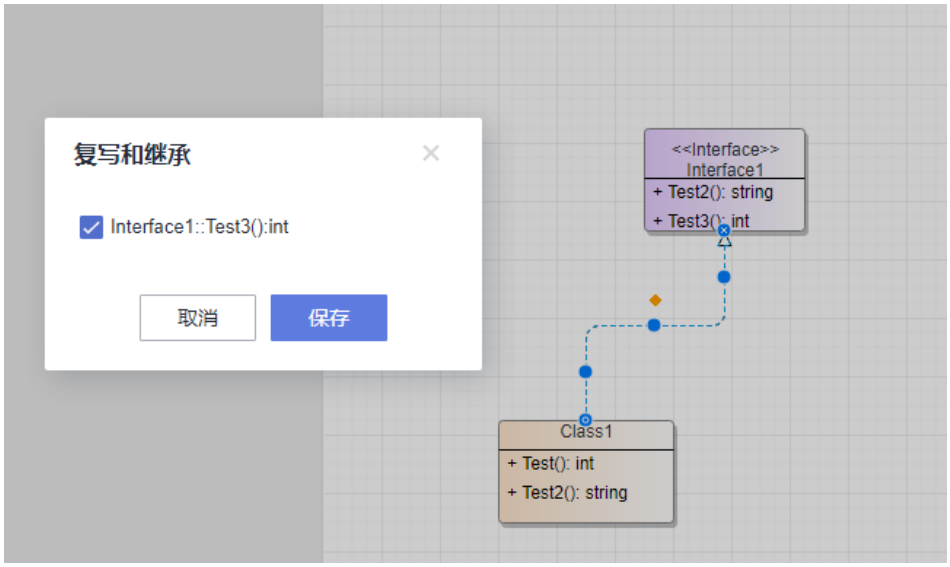
类连线的几种关系

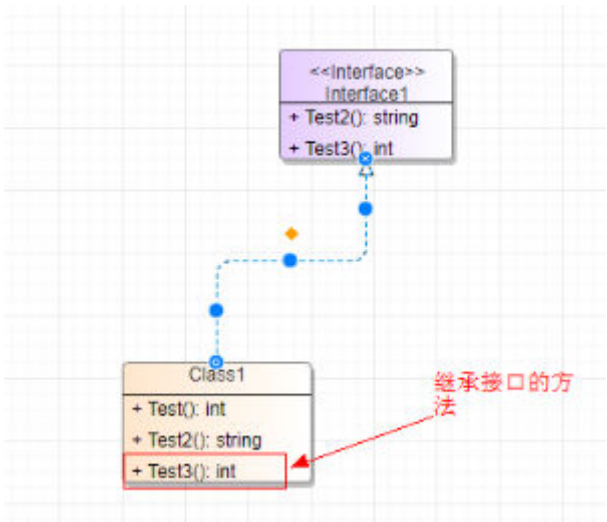
类连线的几种关系表示：关联关系、继承关系、实现关系、依赖关系、聚合关系、组合关系。

- 关联关系：使用Association连线表示类之间的双向关联关系。



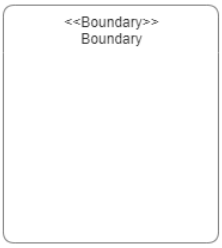
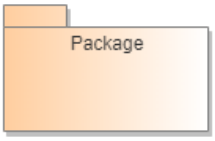
- 继承关系：Class类元素通过Realization关系连线快速继承Interface的operation。只支持Class指向Interface，当Interface的operation名称等发生改变，再次Realization连线，会生成新的Operation记录。

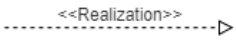
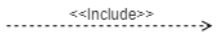
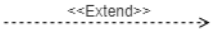
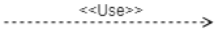




2.3 用例图

元素介绍

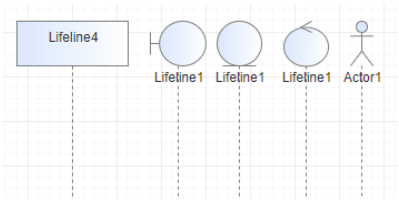
元素名	图标	含义
Use Case		用例，代表的是一个完整的功能。
Test Case		测试用例，是原型的用例元素。通过将元素属性和功能应用于由另一个元素或更确切地说是元素集表示的功能的测试，您可以使用它来扩展“测试”窗口的功能。
Actor		角色，是与系统交互的人或事物。
Boundary		边界。
Collaboration		是对对象和链总体安排的一个描述，这些对象和链在上下文中通过互操作完成一个行为，例如一个用例或者操作。
Collaboration Use		使用协作用于在复合结构图中将协作定义的模式应用于特定情况。
Package		包。

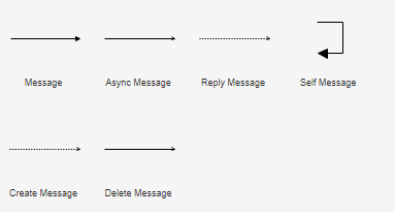
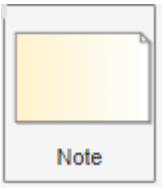
元素名	图标	含义
Generalization		组合，是整体与部分的关系，但部分不能离开整体而单独存在。
Realization		实现，是一种类与接口的关系，表示类是接口所有特征和行为的实现。
Association		关联，是一种拥有的关系，它使一个类知道另一个类的属性和方法。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。
Constraint		是一个语义条件或者限制的表达式。UML 预定义了某些约束，其他可以由建模者自行定义。
Anchor		锚点。
Containment		内嵌，表示嵌在内部的类。
Include		基用例与包含用例之间的关系。说明如何将包含用例中定义的行为插入基用例定义的行为中。基用例可以看到包含用例，并依赖于包含用例的执行结果。但是二者不能访问对方的属性。
Extend		是指扩展用例与基用例之间的关系。特别是如何将扩展用例定义的行为插入基用例定义的行为序列。
Use		使用，是一种使用的关系。表明一个模块在运行的时候，需要使用另外一个模块。

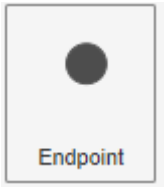
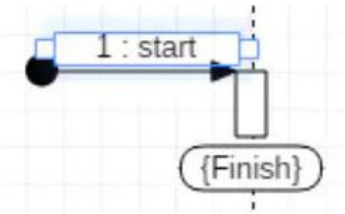
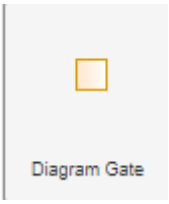
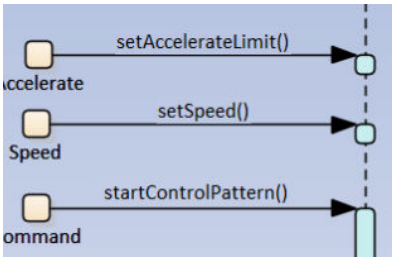
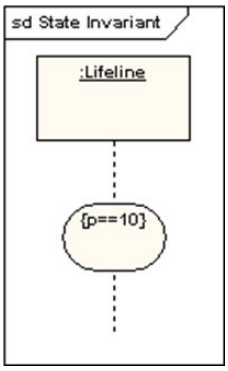
2.4 顺序图

2.4.1 元素介绍

顺序图是显示对象之间交互的图，这些对象是按时间顺序排列的。顺序图中显示的是参与交互的对象及其对象之间消息交互的顺序。

元素名	图标	含义
生命线（Lifeline） 边界生命线（Boundary Lifeline） 实体生命线（Entity Lifeline） 控制生命线（Control Lifeline） 使用者生命线（Actor Lifeline）		**Lifeline**：在时序图中表示为从对象图标向下延伸的一条虚线，表示对象存在的时间。 **Boundary Lifeline**：表示一个系统的边界,或者系统中的一个软件元素. 作为例子,与用户交互的接口界面,数据库网关,或者菜单,就是边界。 **Control Lifeline**：表示一个控制实体或管理者. 它组织和调度在边界(boundary)和实体(entities)间的交互,并作为两者之间的中介者。 **Entity Lifeline**：系统数据. 作为例子,在顾客服务应用中,顾客实体将管理所有与顾客相关的数据。 **Actor Lifeline**:使用者是系统的一个用户；用户可以意味着人类的用户，一台机器，或甚至另一个系统。
fragment（组合片段） loop（循环/迭代） alt（选择）		**fragment**:一组合片段反映了一片段或者多个片段的交互（称为交互操作数）由交互运算符控制，其相应的布尔条件被称为互动约束。它将显示为一个透明的窗口并以水平虚线分割。 **loop**：片段重复一定次数，可以在临界中指示片段重复的条件。 **alt**：用来指明在两个或更多的消息序列之间的互斥的选择，相当于经典的if…else…。

元素名	图标	含义
Messgae(同步消息线) Async Message(异步消息线) Reply Message(返回消息线) Self Message（自关联消息线） Create Message（创建对象消息线） Delete Message（销毁对象消息线）		**message**：消息的发送者把控制传递给消息的接收者，然后停止活动，等待消息的接收者放弃或者返回控制。用来表示同步的意义。 **Async Message**：消息发送者通过消息把信号传递给消息的接收者，然后继续自己的活动，不等待接受者返回消息或者控制。异步消息的接收者和发送者是并发工作的。 **Reply Message**：返回消息表示从过程调用返回。 **Self Message**：表示方法的自身调用或者一个对象内的一个方法调用另外一个方法。 **Create Message**：这个消息指向对象以后，对象的位置就不会出现在顶部，而是创建消息所在的位置。 **Delete Message**：这个消息指向对象以后，对象生命线底部出现一个终止符，表示该对象不再接收新的消息。
note		文本框。

元素名	图标	含义
Endpoint（结束点）	 示例： 	**Endpoint**：流程结束、异常退出的地方用“结束”表示。 **目前暂支持当做startpoint使用，图指向point后续优化中。**
Diagram Gate（门/边界）	 示例： 	**diagram gate**：表示图的门口，用法是可以链接到另外一张图。 **目前暂支持门指向图，图指向门后续优化中。** **该元素与激活块不同，不建议当作激活块使用。**
State Continuation（状态常量 / 延续）	 示例： 	**State Continuation**：状态常量是生命线的约束，运行时始终为“真”。 延续虽与状态常量有同样的标注，但是被用于复合片段，并可以延伸跨越多条生命线。

2.4.2 创建顺序图

绘制顺序图时，必须保证图的类型为顺序图，否则可能导致无法绘制对应消息线。

已创建的模型图修改图类型具体请参考**如何查看和修改模型图类型**。

步骤1 单击新增图。

步骤2 弹出新建图弹窗选择“UML>顺序图”，填写顺序图基本信息。

----结束

2.4.3 创建生命线

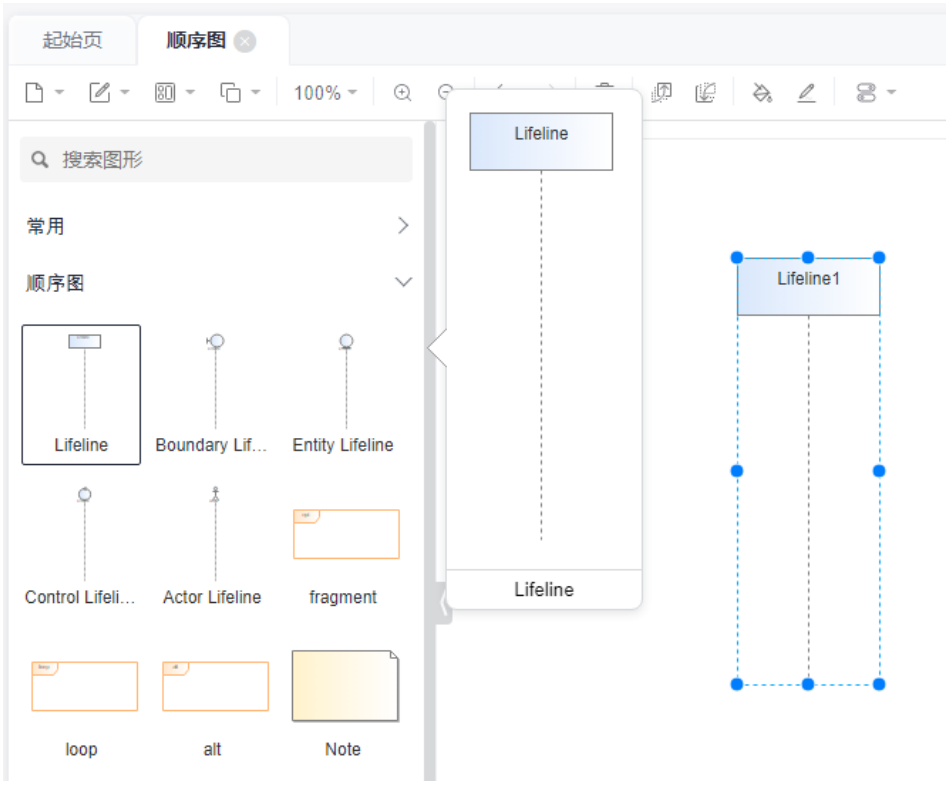
生命线介绍

通用生命线（Lifeline），边界生命线（Boundary Lifeline），实体生命线（Entity Lifeline），控制生命线（Control Lifeline），参与者生命线（Actor Lifeline）。生命线的类型不会影响消息线的连线逻辑。

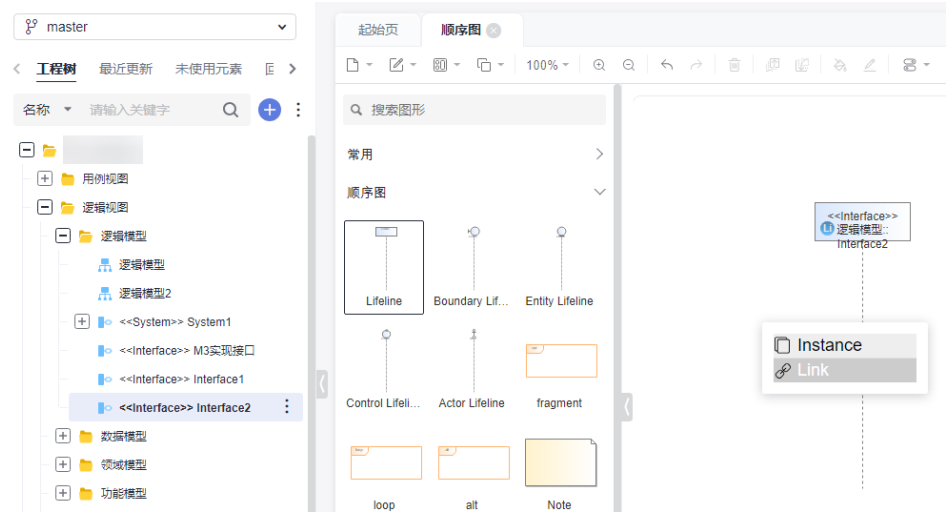
创建生命线

在创建生命线前，需要先考虑哪种类型的生命线能更好地表示你所要表达的对象。您可以通过以下三种方式创建生命线。

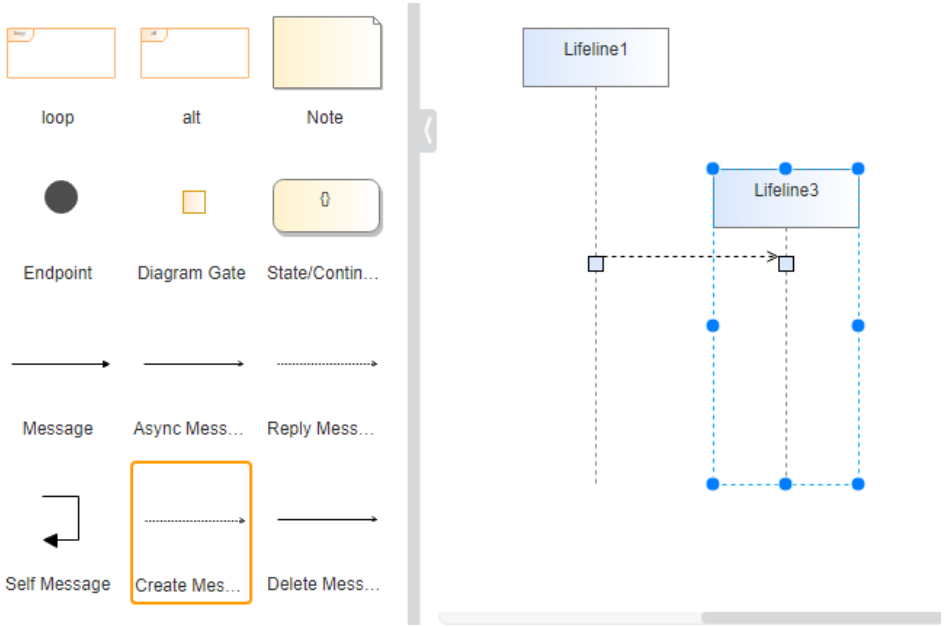
1. 工具画板中选择一个生命线拖拽至画布中。



2. 从工程树中其它模型视图中的元素拖拽至画布中生成对应生命线，可以使用Link方式引用。

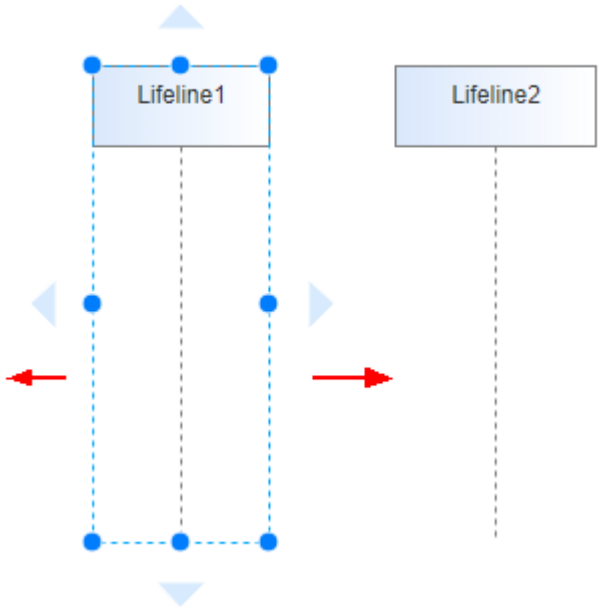


3. 生命线也支持由创建消息线（Create Message）触发创建，其位置通常低于别的生命线。

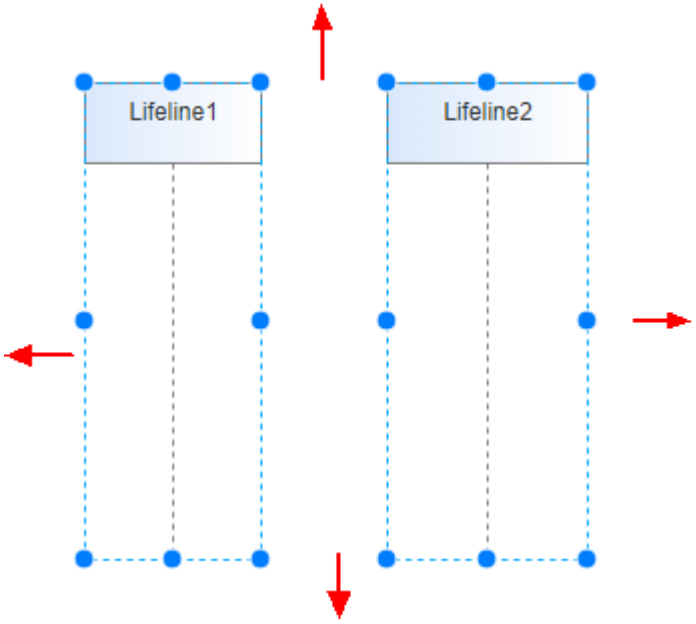


生命线移动、延长

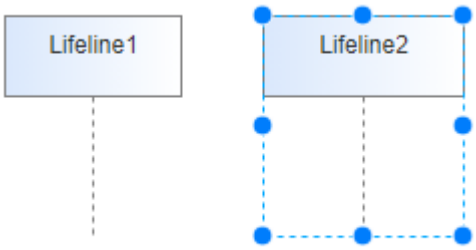
若画布中存在两条及以上生命线，单个生命线不支持上下垂直移动，只支持左右水平移动。



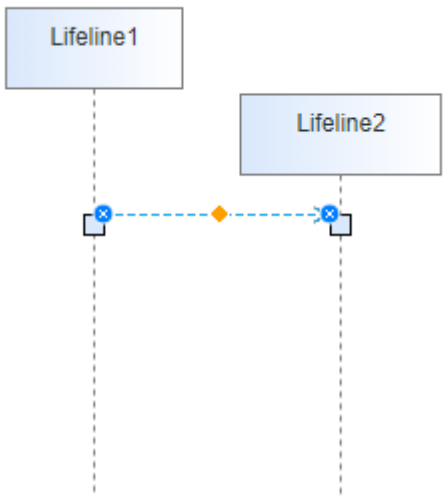
全选生命线后，可以在垂直方向上移动，调整元素在画布中的位置。



调整任一生命线的高度，其余生命线会同步调整，保证所有生命线底部对齐。

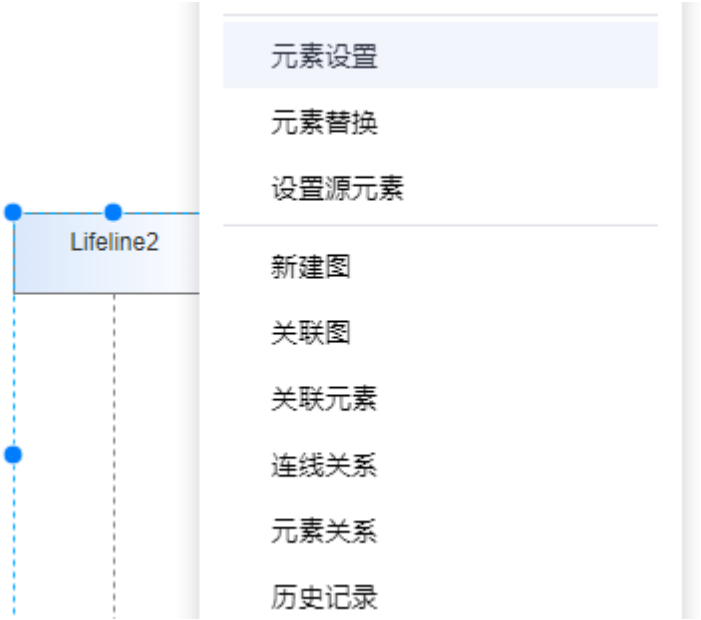


如果生命线是通过Create Message创建的，则可以操作该Create Message对目标生命线进行上下移动，但时间上不能早于之前最早生命线。



生命线变更

右键单击生命线，选择“元素设置/替换”可以修改生命线类型及替换成模型工程中其它元素。



元素设置：修改元素的类型及构造型。

元素替换：替换成其它元素。

2.4.4 绘制消息线

说明

根据建模规范限制，建模的连线不能独立存在于图中，两端必须是连接在元素上的，因此不允许消息线直接拖拽至画布上。且绘制顺序图时，必须保证图的类型为顺序图，否则可能导致无法绘制对应消息线。具体请参见**如何查看和修改模型图类型**。

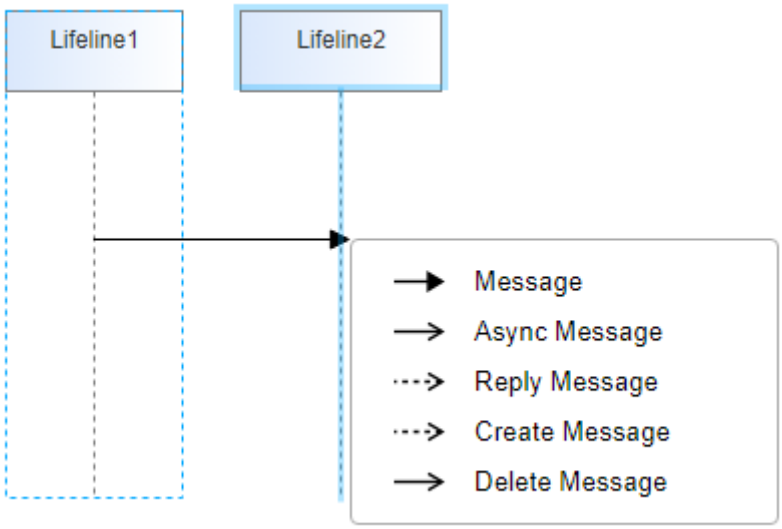
消息线介绍

顺序图用消息线描绘元素之间的工作流或者活动。软件模型中，消息线可以用来代表源端或目标端元素的操作或者属性。您可以根据需求绘制消息线，绘制消息线后，也可以提升/降低消息线层级。

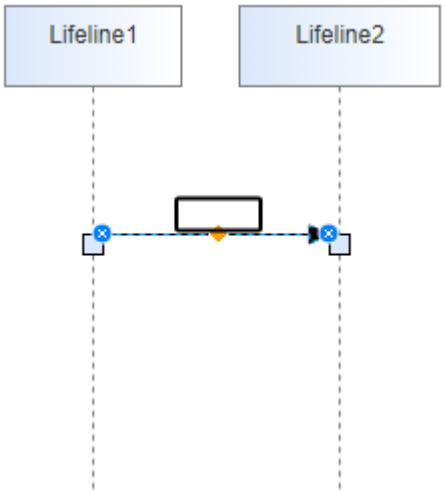
绘制消息线

您可以通过如下方式绘制生命线：

1. 选择生命线，从生命线的一端小三角单击拉线至另一个生命线上，选择消息类型。消息类型介绍请参见**消息线连线规则**。

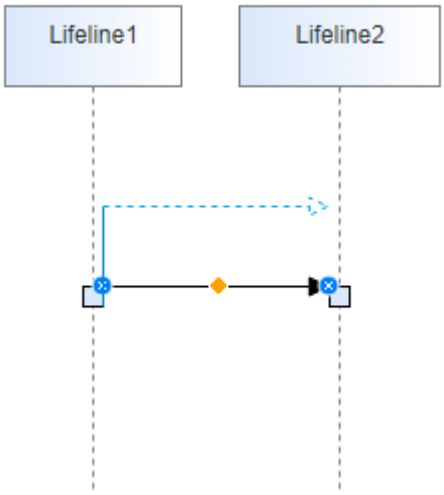


2. 选择连线按F2或者右键单击消息线选择“编辑”，可以编辑消息线名称。

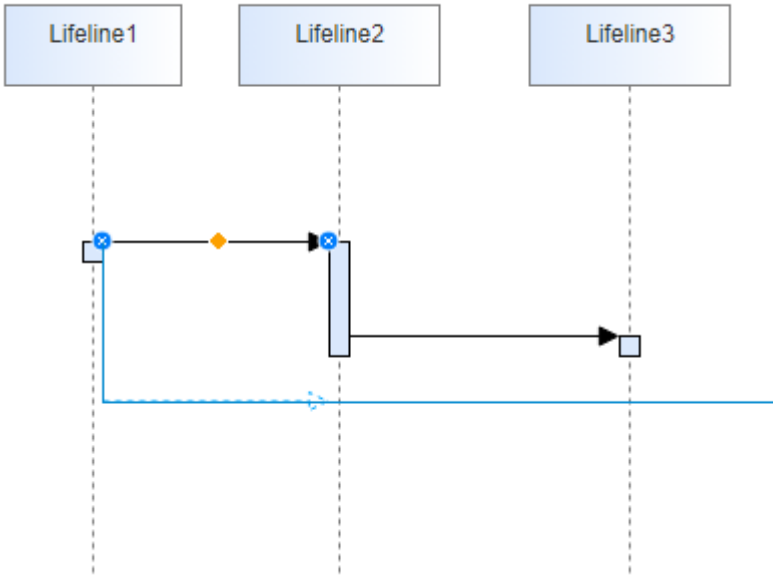


消息线移动

选中消息线，左键按住消息线可以垂直上下移动。

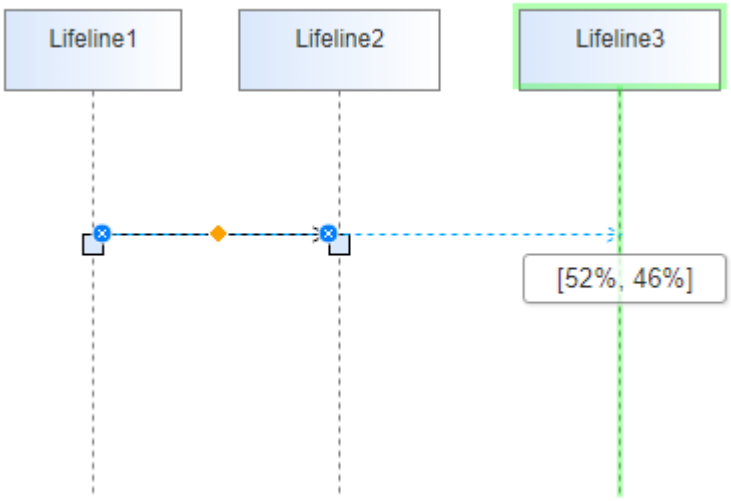


按住Ctrl后可以在垂直方向上进行移动，支持跨线移动，不支持水平移动。



消息线变更连接端

左键选中消息线目标端，向其余生命线拖去，可以将连接点变更到新生命线上。



2.4.5 消息线连线规则

顺序图连线逻辑:

元素概念:

1. 消息线（Message）：同步消息线（Message），异步消息线（Async Message），返回消息线（Reply Message），自连消息线（Self Message），创建消息线（Create Message），终止消息线（Delete Message）。
2. 激活块（Activation，术语同EA）：代表了一段生命周期，激活块内可包含子激活块。激活块的断开代表了其生命周期的终止。
3. 生命线（Lifeline）：通用生命线（Lifeline），边界生命线（Boundary Lifeline），实体生命线（Entity Lifeline），控制生命线（Control Lifeline），参与者生命线（Actor Lifeline）。生命线的类型不会影响消息线的连线逻辑。

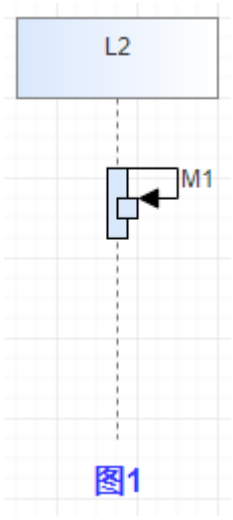
连线规则:

1. 同步消息线（Message）与异步消息线（Async Message）会在其指向的目标上创建一个激活块（如果指向生命线，则在生命线上创建；如果指向某个激活块，则在其内创建一个子激活块）。
2. 返回消息线（Reply Message）指向到激活块上，不会额外创建子激活块。如果返回消息指向的是生命线则会创建一个激活块。

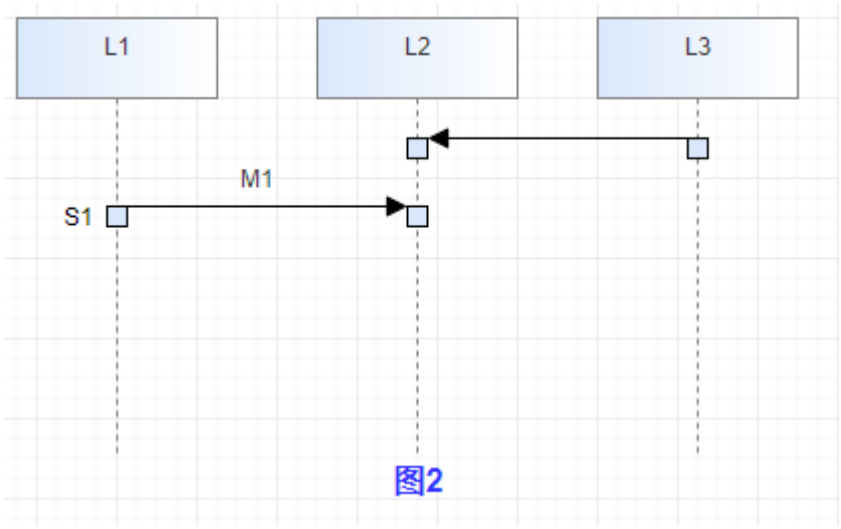
连线算法:

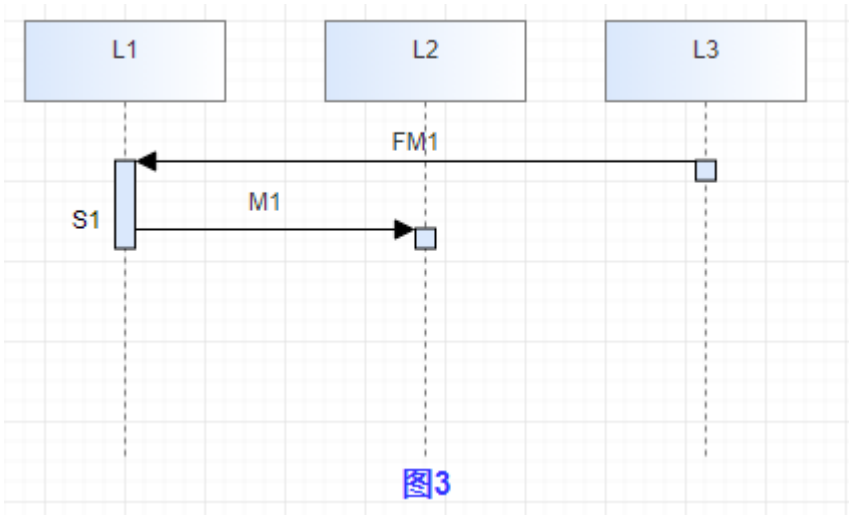
1. 目标端寻源：一条消息线M1尝试连接到生命线L2上。

从当前消息线M1出发，如果该消息线M1的源端S1在生命线L2上，则S1为M1的目标端（同时也是源端），在S1内创建子激活块并将M1指向它【图1】；否则继续如下流程；



查找源端S1的引入消息线（From Message），如果找不到，则终止寻源过程，目标端寻源失败，在L2上新创建一个激活块并将M1指向它【图2】；如果能找到，则命名为FM1，继续如下流程【图3】；





检测消息线FM1的源端FS1，查看其是否是生命线L2上的激活块，如果是则FS1为M1的目标端，在FS1内创建子激活块并将M1指向它【图4】；如果不是，则重复步骤b【图5】；

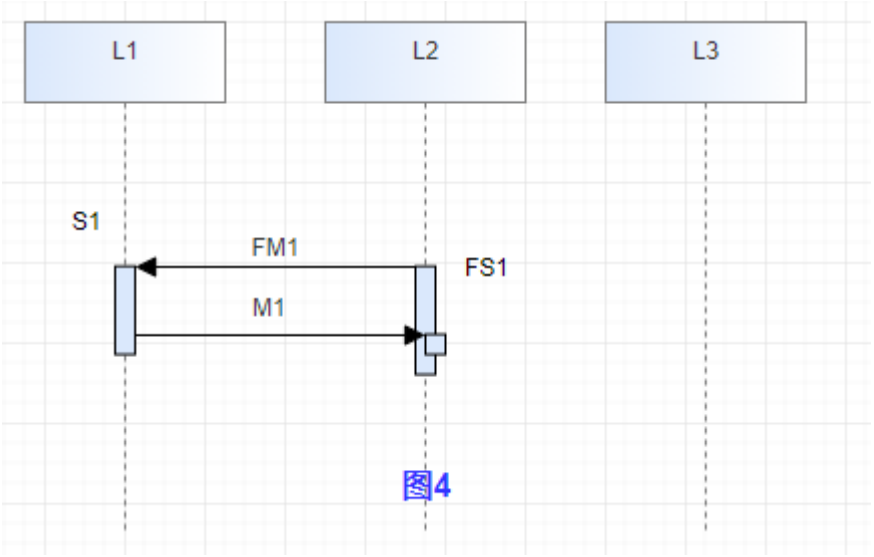


图4

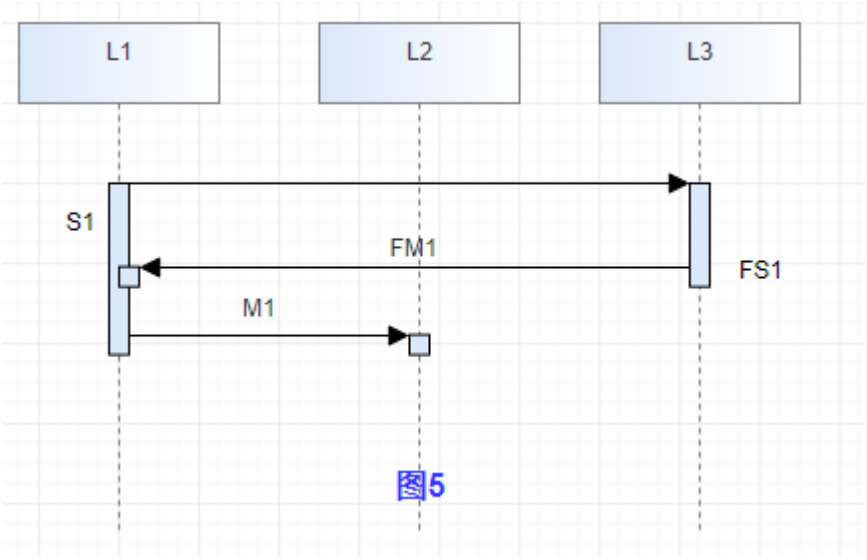
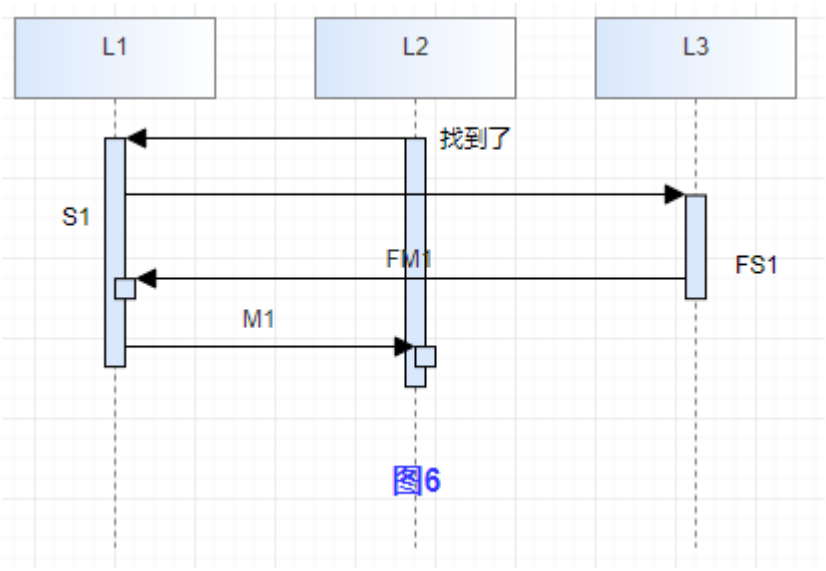


图5

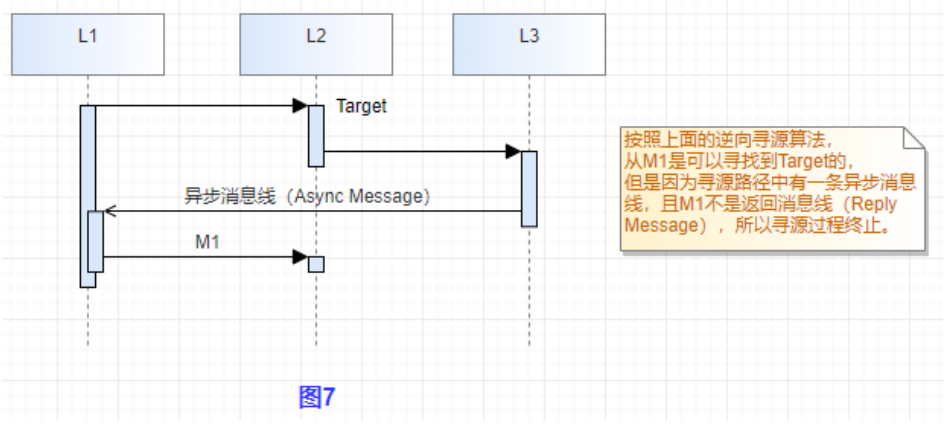
当查找到顺序图第一根消息线依旧未找到目标端，则此次寻源过程终止，目标端寻源失败，在L2上新创建一个激活块并将M1指向它【图5】；如果找到则在其内创建子激活块并将M1指向它【图6】。



说明

图6中M1的逆向寻源不会经过2、3两根消息线，因为M1的源端S1的引入消息线是第一条线，直接能找到目标。

逆向寻源过程中，如果寻源过程中遇到异步消息线（Async Message）或者返回消息线（Reply Message），则终止寻源过程；此时如果M1不是返回消息线（Reply Message），则在L2上新创建一个激活块并将M1指向它。



寻源过程中断后或者未找到对应激活块，并且M1是返回消息线：1. 那么如果M1有源端引入线（源端引入连线类型不是返回消息线），则从M1的目标端所在生命线找到其内位置最低的激活点作为寻源目标（图7.1）。2. 如果M1没有源端连线，或者源端连线是返回消息线（Reply Message），则在L2上新创建一个激活块并将M1指向它（图7.2，图7.3）。

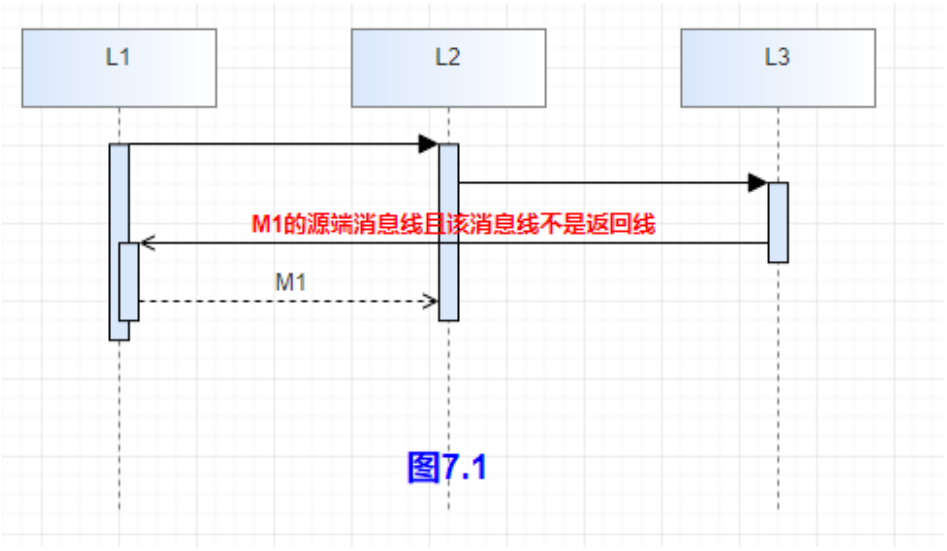


图7.1

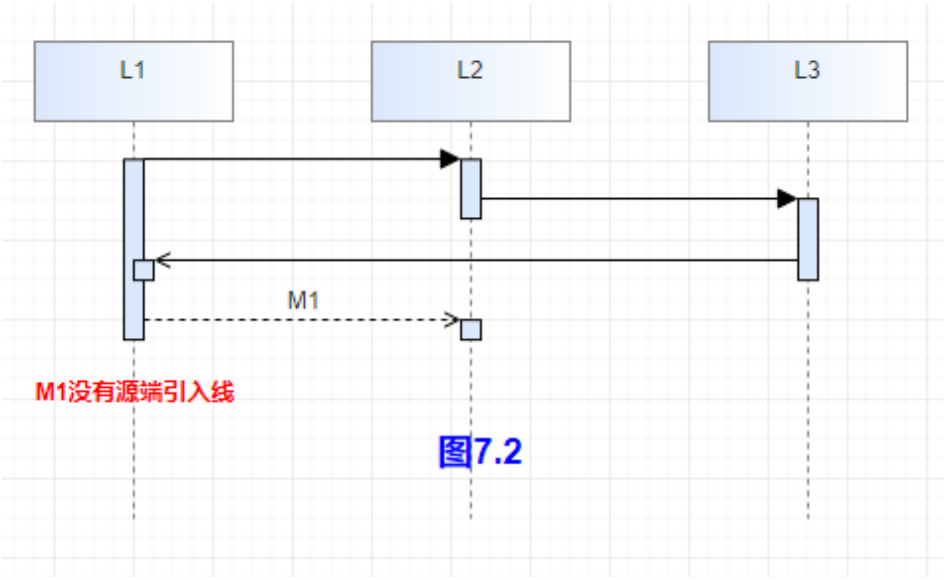


图7.2

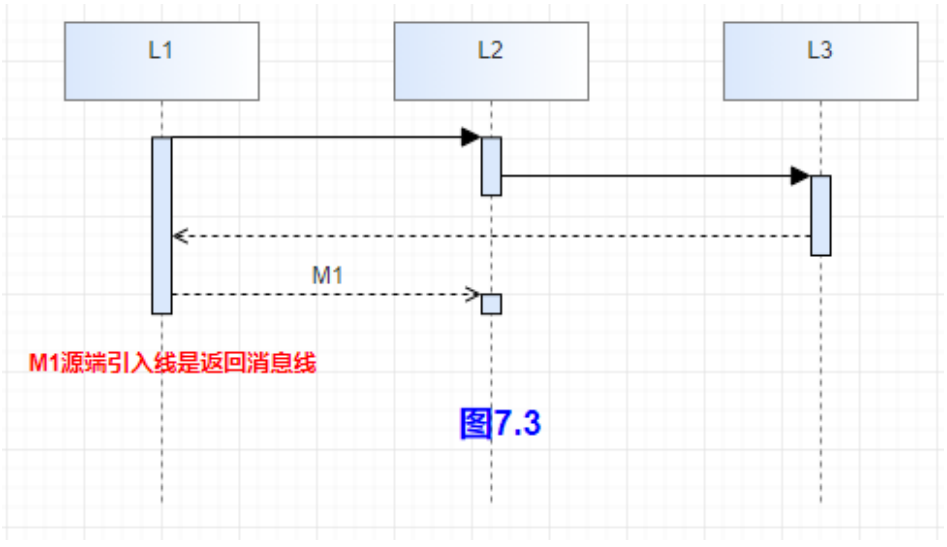
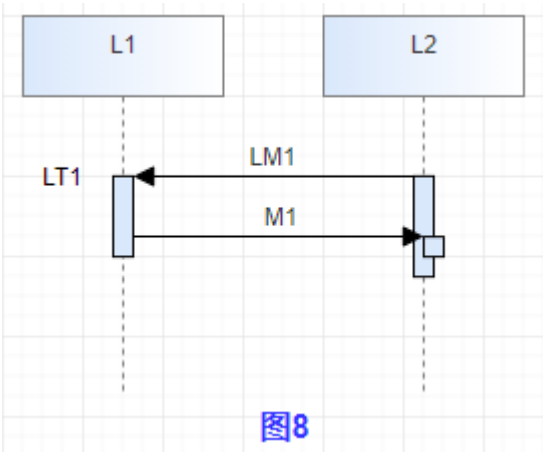


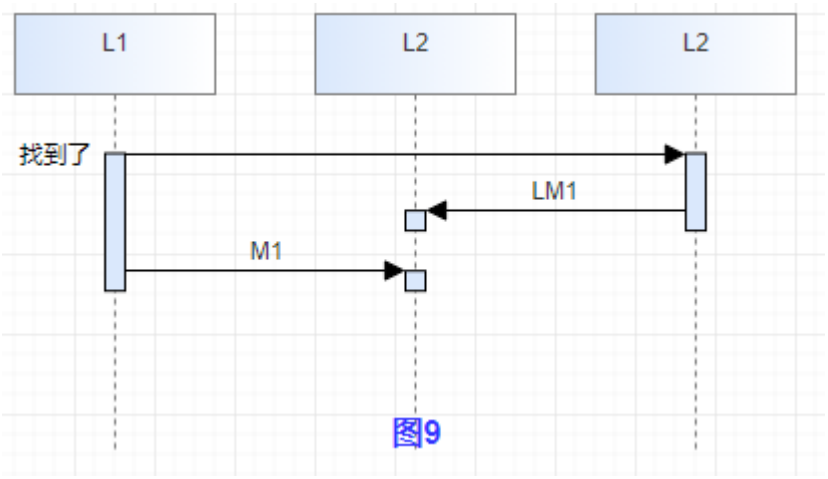
图7.3

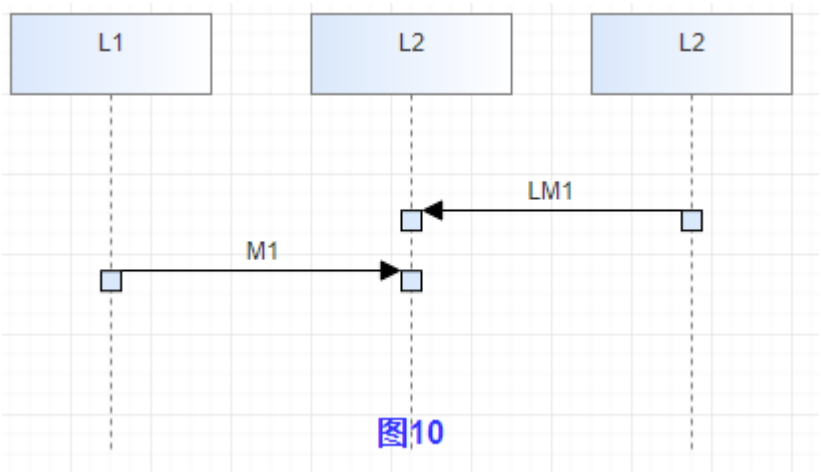
2. 源端寻源：一条消息线M1从生命线L1上出发。算法规则与目标端寻源类似，差异在于源端不以当前M1为出发点，而是以M1的上一条消息线LM1为出发点，并且不受异步与返回消息线的限制。

找到当前消息线M1的上一条消息LM1（注意，是位置在其上的第一条消息线），从LM1出发，如果LM1的目标端LT1在生命线L1上，则LT1为M1的源端【图8】；如果不在，则继续如下流程；

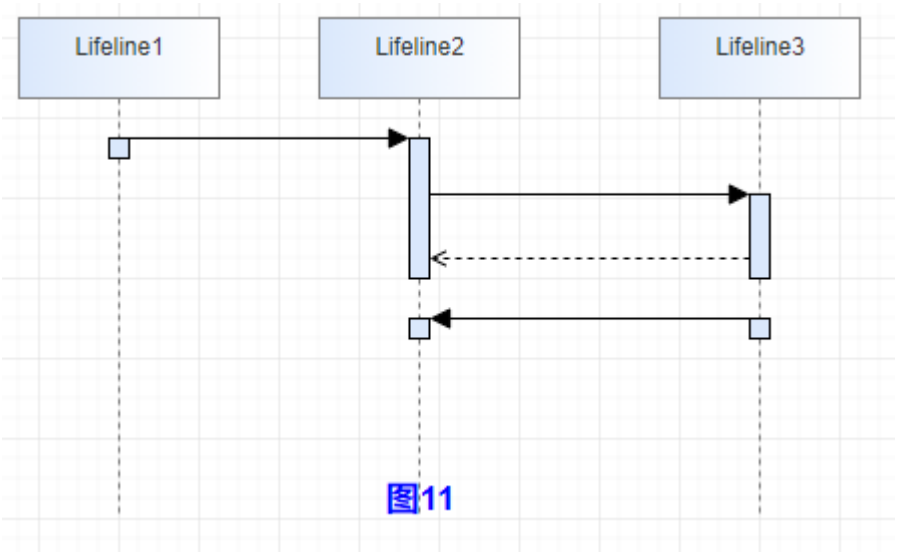


用LM1逆向寻源（同目标端逆向算法），如果能找到在生命线L1上的激活块，则该激活块为M1的源端【图9】；如果找不到生命线L1上的激活块，在生命线L1上新创建一个激活块并将M1从它引出【图10】。





逆向寻源过程中，如果某个消息线是返回消息线（Reply Message），则其源端不参与寻源判断，直接略过它继续向前寻源。



2.4.6 自定义激活块

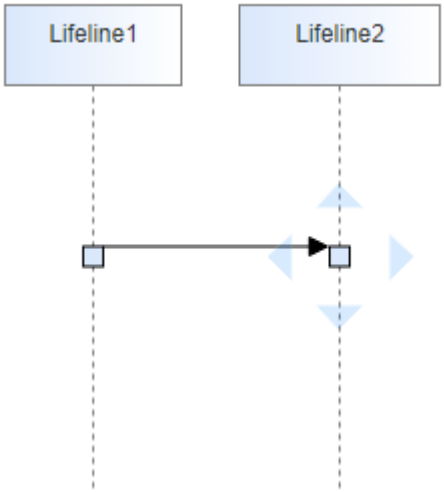
激活块代表了一段生命周期，激活块内可包含子激活块。激活块的断开代表了其生命周期的终止。

1. 同步消息线（Message）与异步消息线（Async Message）会在其指向的目标上创建一个激活块。如果指向生命线，则在生命线上创建，如果指向某个激活块，则在其内创建一个子激活块。
2. 返回消息线（Reply Message）指向到激活块上，不会额外创建子激活块。如果返回消息指向的是生命线则会创建一个激活块。

工具提供了如下五种方式控制激活块，可以根据需要合并或者断开激活块。

激活块生成

消息线连线到生命线成功后会自动生成激活块。

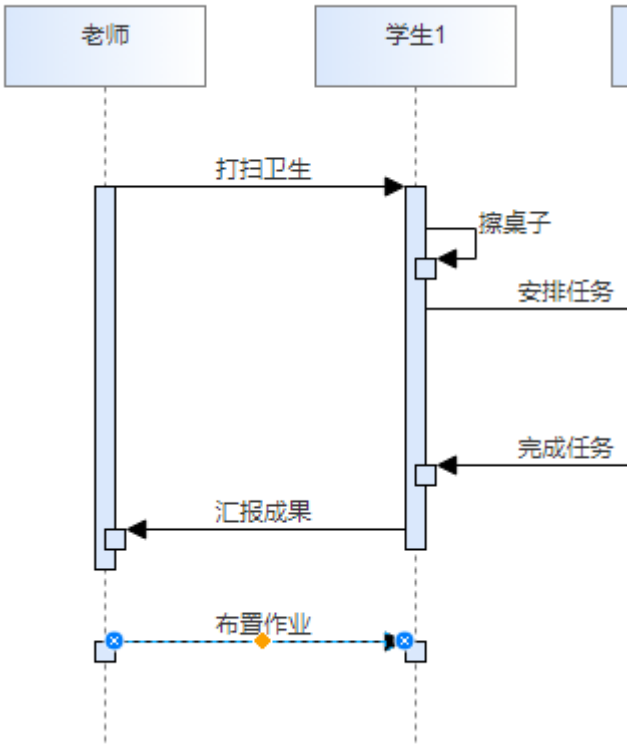


开始新的消息组

在当前模型图中开始新一轮进程，在单张模型图中描述多个进程场景。

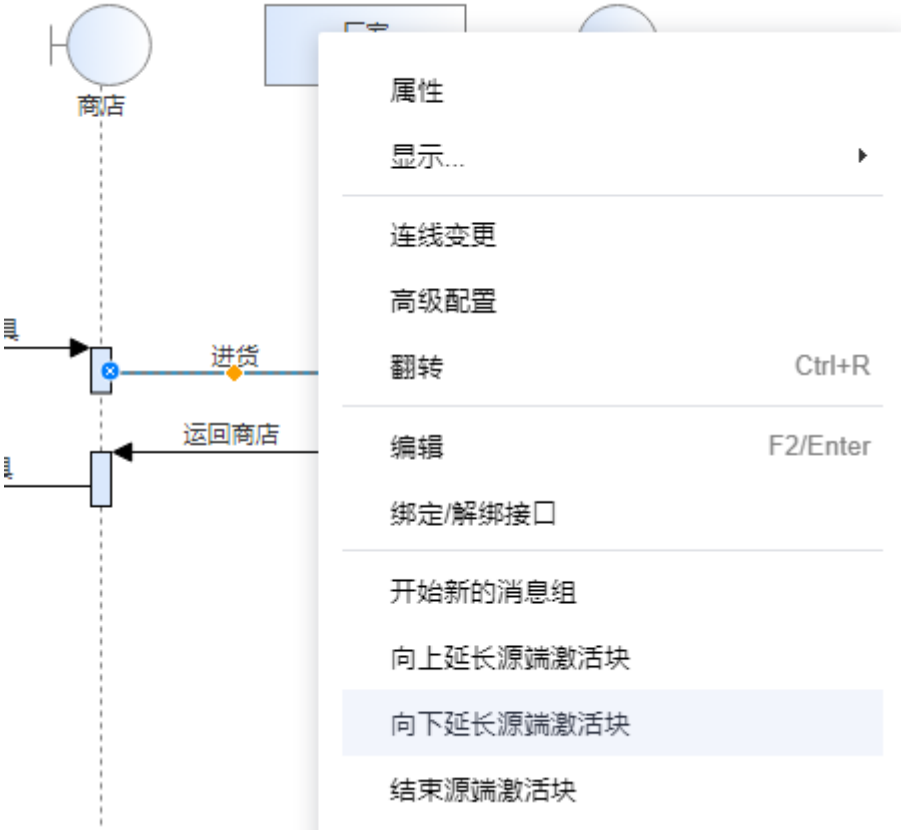
在准备开始新的消息线右键菜单“开始新的消息组”。

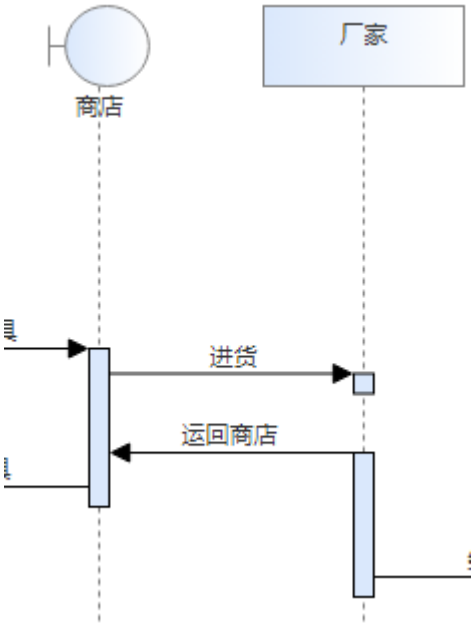
打扫卫生结束后开始布置作业新的消息组。



向下延长源端激活块

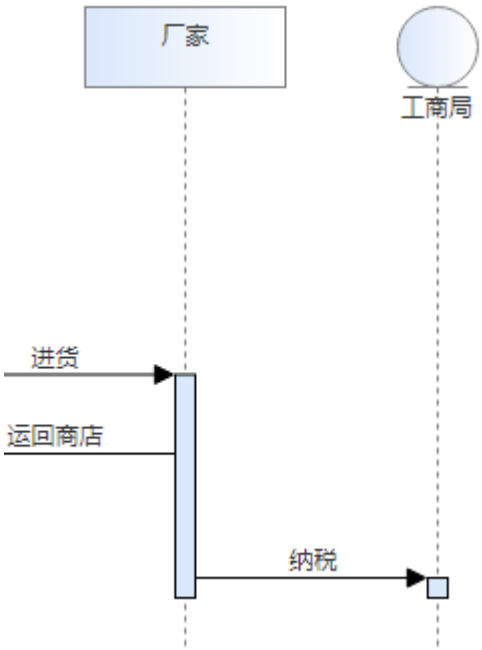
强制指定某个元素在正常进程周期外保持激活状态，表示该元素所属进程与其余进程共存；





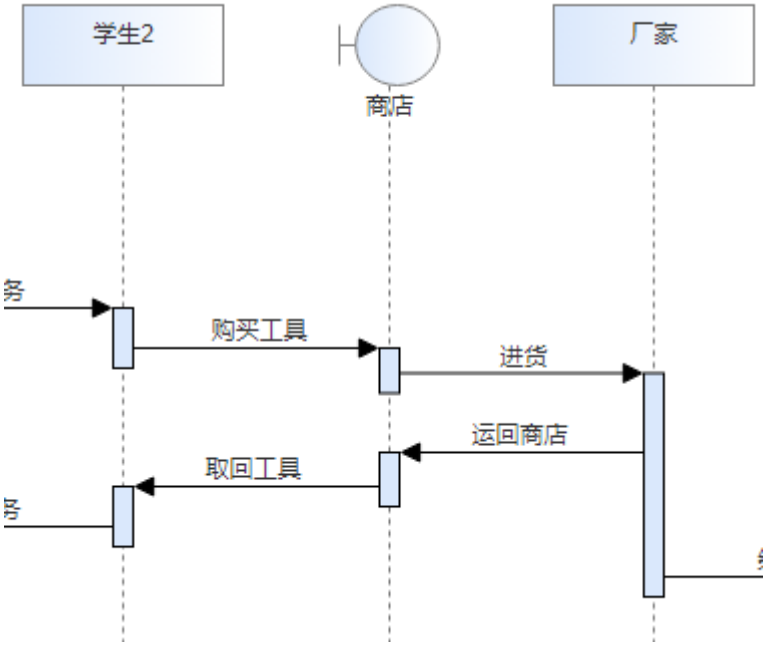
向上延展源端激活块

强制某个元素上的激活块向上延展。



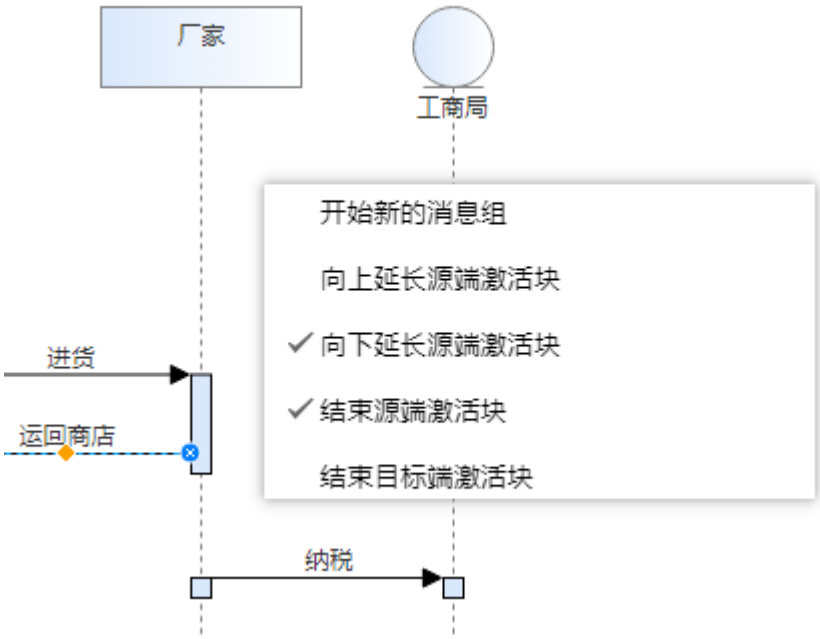
结束源端激活块

源端元素在该条消息线之后，结束其上激活块的进程周期，通常用来表示一条异步消息过后该源端元素变为空闲状态。



结束目标端激活块

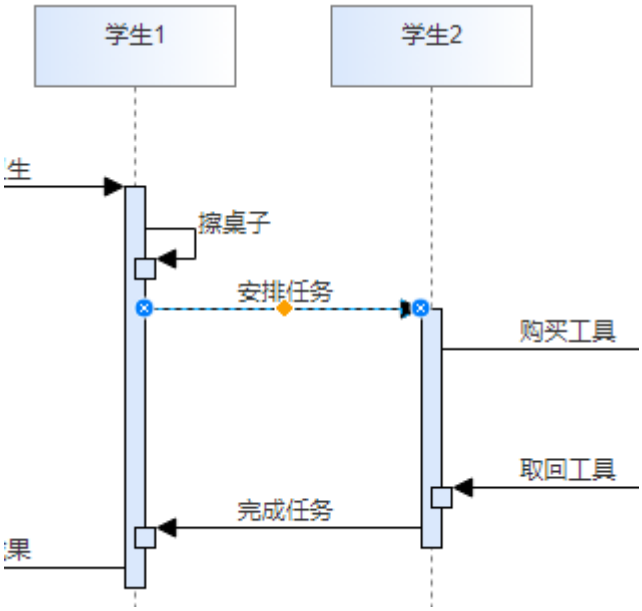
当前激活块是由其上某条消息线设置了“延长源端激活块”而延展下来的，可以通过该设置结束其延展能力。



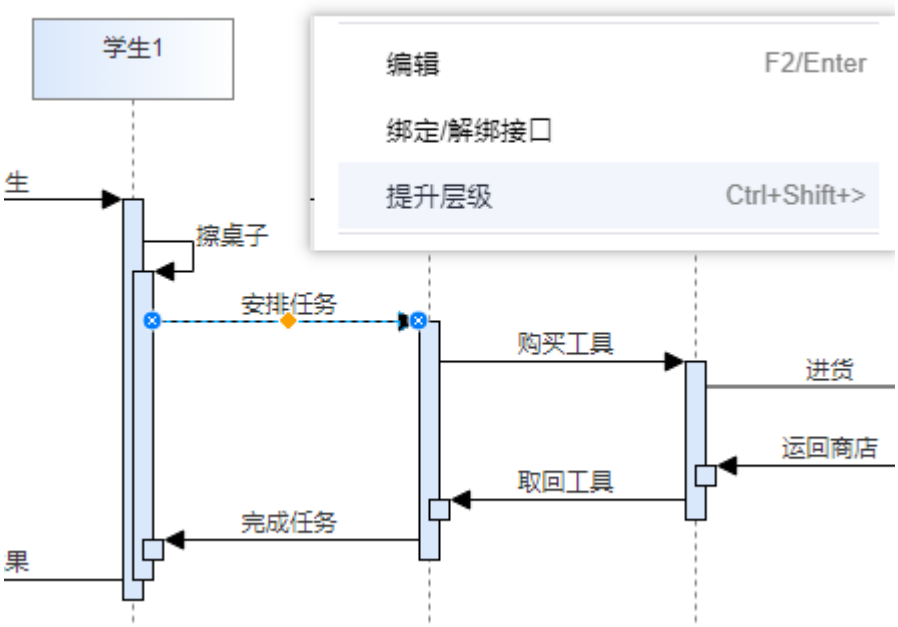
2.4.7 提升或降低消息线层级

激活块层级表示当前消息任务与前一个消息任务之间是并列还是包含关系，支持对消息线源端升降级。

右键单击消息线，选择提升层级或者降低层级。



提升层级后效果。



2.4.8 绘制组合片段

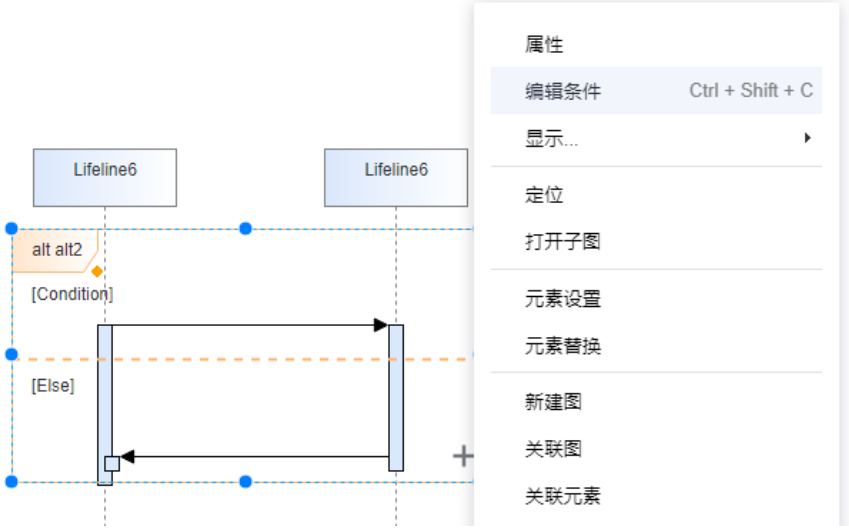
组合片段fragment中类型介绍说明

片段类型	片段名称	说明
opt	选择	包含一个可能发生或可能不发生的序列。 可以在临界中指定序列发生的条件。
alt	抉择	包含一个片段列表，这些片段包含备选消息序列。 在任何场合下只发生一个序列。 可以在每个片段中设置一个临界来指示该片段可以运行的条件。 **else** 的临界指示其他任何临界都不为 True 时应运行的片段。 如果所有临界都为 False 并且没有 **else**，则不执行任何片段。
loop	循环	片段重复一定次数，可以在临界中指示片段重复的条件。
break	中断	如果执行此片段，则放弃序列的其余部分。 可以使用临界来指示发生中断的条件。
par	并行	并行处理。 片段中的事件可以交错。
critical	关键	用在 Par 或 Seq 片段中。 指示此片段中的消息不得与其他消息交错。
seq	弱顺序	有两个或更多操作数片段。 涉及同一生命线的消息必须以片段的顺序发生。 如果消息涉及的生命线不同，来自不同片段的消息可能会并行交错。
strict	强顺序	有两个或更多操作数片段。 这些片段必须按给定顺序发生。

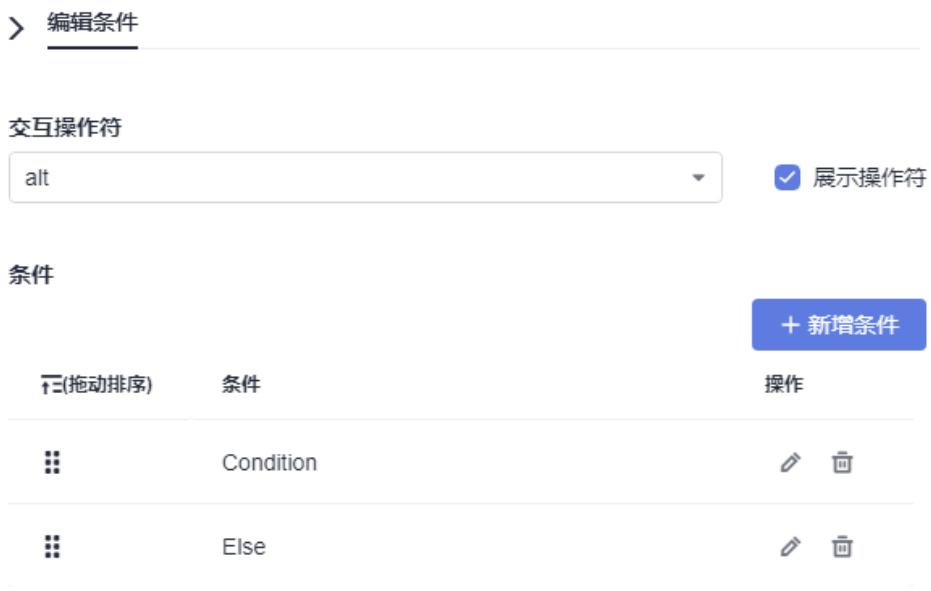
片段类型	片段名称	说明
consider	考虑	指定此片段描述的消息列表。 其他消息可发生在运行的系统中，但对此描述来说意义不大。
ignore	忽略	此片段未描述的消息列表。 这些消息可发生在运行的系统中，但对此描述来说意义不大。
assert	断言	操作数片段指定唯一有效的序列。 通常用在 consider 或 ignore 片段中。
neg	否定	此片段中显示的序列不得发生。 通常用在 consider 或 ignore 片段中。

由于loop与alt类型使用频率较高，故直接放到工具箱中作为可直接使用的元素，与通过fragment设置成loop与alt的意义相同，无任何区别。

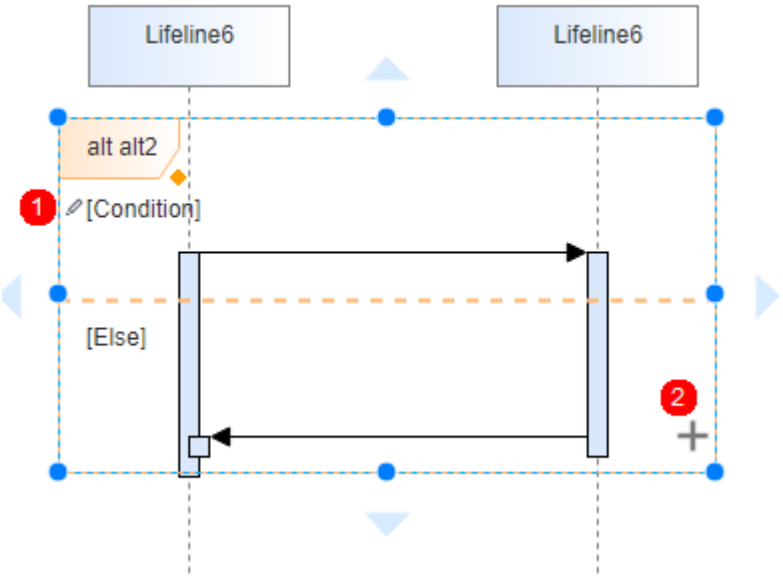
在工具画板中选择分段类型，拖拽至画布对应位置，右键单击分段可以进行“编辑条件”。



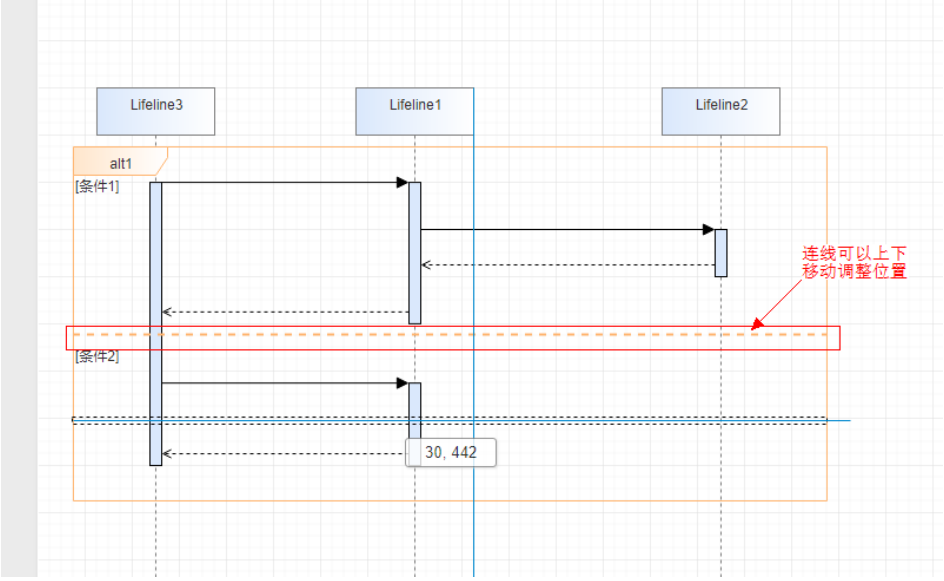
单击“新增条件”，输入分片区域或条件名称。支持拖动调整顺序。



也可以直接点击组合片段元素进行修改/新增条件框。



鼠标移到分层连线上方变成白色十字箭头后，fragment中的连线可以上下移动，调整位置。



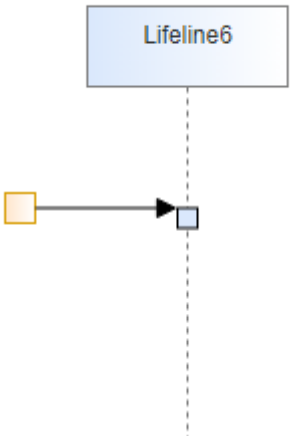
2.4.9 Diagram Gate 使用

图表门是一种简单的图形方式，用于指示可以将消息传输到交互片段和从交互片段传出的点。可能需要一个片段来接收或传递消息。在内部，有序消息反映了这一要求，并在片段帧的边界上指示了门。任何与此内部消息“同步”的外部消息必须适当地对应。门可以出现在交互图（顺序，时序，通信或交互概述），交互事件和组合片段（以指定表达式）上。

Diagram Gate并不是激活块（Activation），顺序图在绘制连线时会自动生成激活块，激活块的使用请参考**自定义激活块**。

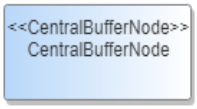
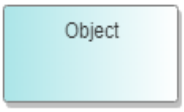
使用示例

拖动工具栏中的Diagram Gate图标到顺序图中，然后根据需要输入名称即可。

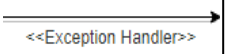


2.5 活动图

元素介绍

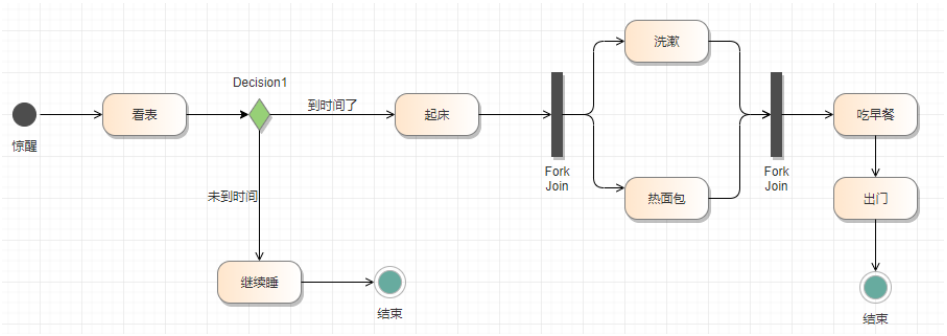
元素名	图标	含义
Action		动作是可执行的原子计算，它导致模型状态的改变和返回值。
Activity		活动是状态机内正在进行的非原子执行。
StructuredActivity		结构化活动是一个活动节点，可以将下级节点作为独立的活动组。
CentralBufferNode		中央缓冲区节点是一个对象节点，用于管理活动图中表示的来自多个源和目标的流。
Datastore		数据存储区定义了永久存储的数据。
ExceptionHandlerNode		异常处理程序元素定义发生异常时要执行的一组操作。
Object		封装了状态和行为的具有良好定义界面和身份的离散实体，即对象实例。
Decision		是状态机中的一个元素，在它当中一个独立的触发可能导致多个可能结果，每个结果有它自己的监护条件。
Merge		状态机中的一个位置，两个或多个可选的控制路径在此汇合或"无分支"。

元素名	图标	含义
Send		即发送者对象生成一个信号实例并把它传送到接收者对象以传送信息。
Receive		接收就是处理从发送者传送过来的消息实例。
Partition		分区元素用于逻辑组织活动的元素。
Partition		分区元素用于逻辑组织活动的元素。
Initial		用来指明其默认起始位置的伪状态。
Final		组成状态中的一个特殊状态，当它处于活动时，说明组成状态已经执行完成。
Flow Final		Flow Final元素描述了系统的退出，与Activity Final相反，后者代表Activity的完成。
Synch		一个特殊的状态，它可以实现在一个状态机里的两个并发区域之间的控制同步。

元素名	图标	含义
Fork Join		(Fork)复杂转换中，一个源状态可以转入多个目标状态，使活动状态的数目增加。 (Join)状态机活动图或顺序图中的一个位置，在此处有两个或以上并列线程或状态归结为一个线程或状态。
Fork Join		(Fork)复杂转换中，一个源状态可以转入多个目标状态，使活动状态的数目增加。 (Join)状态机活动图或顺序图中的一个位置，在此处有两个或以上并列线程或状态归结为一个线程或状态。
Region		并发区域。
Control Flow		（控制流）在交互中，控制的后继轨迹之间的关系。
Object Flow		（对象流）各种控制流表示了对象间的关系、对象和产生它（作输出）或使用它（作输入）的操作或转换间的关系。
Constraint		是一个语义条件或者限制的表达式。UML 预定义了某些约束，其他可以由建模者自行定义。
Exception Handler		异常处理。捕获异常根据异常类型查找到对应的异常处理方法，然后执行对应的方法。
Interrupt Flow		中断流是用于定义异常处理程序和可中断活动区域的连接器的两个UML概念的连接。中断流是活动边缘的一种。它通常用于活动图中，以对活动过渡进行建模。
Anchor		锚点。
Containment		内嵌，表示嵌在内部的类。

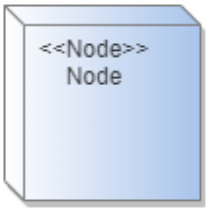
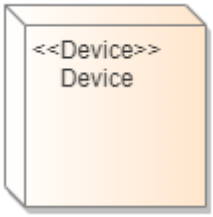
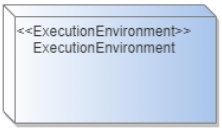
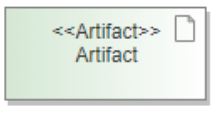
活动图对用户和系统遵循流程的行为进行建模，它们是流程图或工作流的一种，但是它们使用的形状略有不同。

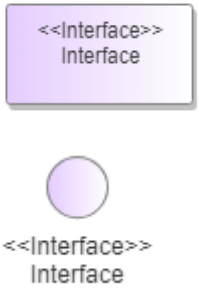
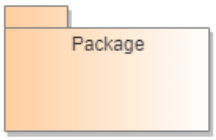
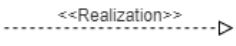
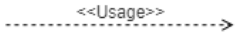
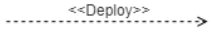
活动图示例如下所示：



2.6 部署图

元素介绍

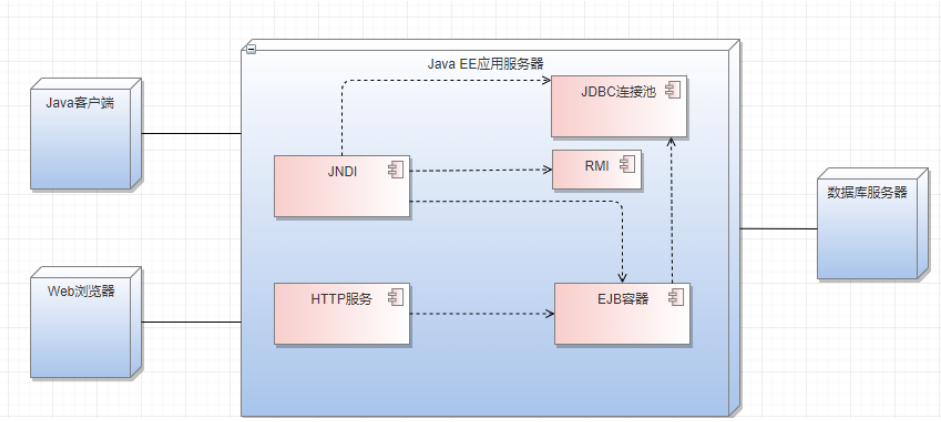
元素名	图标	含义
Node		部署节点。
Device		设备节点。
DeploymentSpecification		部署规格。
ExecutionEnvironment		执行环境。
Artifact		制品是被软件开发过程所利用或通过软件开发过程所生产的一段信息，如外部文档或工作产物。 制品可以是一个模型、描述或软件。

元素名	图标	含义
Component		组件，可独立加载、部署和运行的二进制代码，采用轻量级通讯机制、松耦合高内聚的软件架构构建单元，部署时不能跨节点类型部署（计算机百科全书：组件是软件系统中具有相对独立功能、接口由契约指定、和语境有明显依赖关系、可独立部署、可组装的软件实体）。
Interface		接口，可以是单个接口，也可以是抽象的一组接口的组合。 圆形接口与矩形接口意义相同，仅形状不同。
Package		包。
Composition		组合，是整体与部分的关系，但部分不能离开整体而单独存在。
Aggregation		聚合，是整体与部分的关系，且部分可以离开整体而单独存在。
Realization		实现，是一种类与接口的关系，表示类是接口所有特征和行为的实现。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。
Usage		使用，是一种使用的关系。表明一个模块在运行的时候，需要使用另外一个模块。
Generalization		通用化，是一种继承关系，一个类（通用元素）的所有信息（属性或操作）能被另一个类（具体元素）继承，继承某个类的类中不进可以有属于自己的信息，而且还拥有了被继承类中的信息。
Manifest		Repo和对应的逻辑设计对象使用"Manifest”连接 表示由此代码仓的代码实现此设计对象的功能。
Deployment		描述现实世界环境运行系统的配置的开发步骤。
Association		关联，是一种拥有的关系，它使一个类知道另一个类的属性和方法。

部署图用于大型和复杂系统的另一张专门图，其中软件部署在多个系统上。

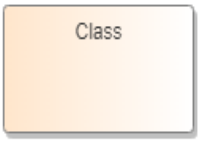
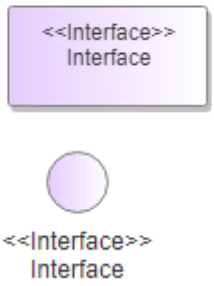
部署图一般用于：

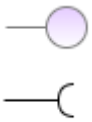
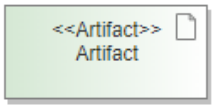
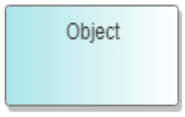
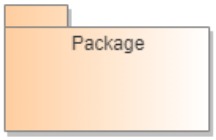
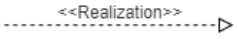
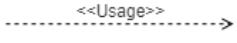
1. 嵌入式系统建模（硬件之间的交互）。
2. 客户端/服务器系统建模（用户界面与数据的分离）。
3. 分布式系统建模（多级服务器）。



2.7 组件图

元素介绍

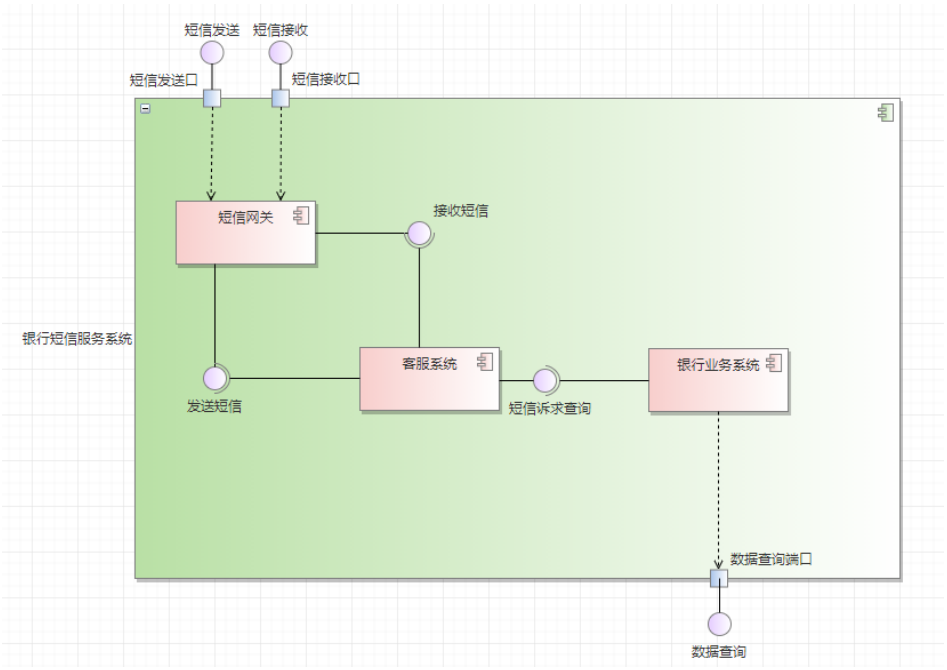
元素名	图标	含义
Class		是对象的集合，展示了对象的结构以及与系统的交互行为。
Interface		接口，可以是单个接口，也可以是抽象的一组接口的组合。 圆形接口与矩形接口意义相同，仅形状不同。
Component		组件，可独立加载、部署和运行的二进制代码，采用轻量级通讯机制、松耦合高内聚的软件架构构建单元，部署时不能跨节点类型部署(计算机百科全书：组件是软件系统中具有相对独立功能、接口由契约指定、和语境有明显依赖关系、可独立部署、可组装的软件实体)

元素名	图标	含义
Interface		Required Interface和Provided Interface之间可以建立Dependency，表明一个组件需要的接口是由另外一个组件提供的。
port		端口
PackagingComponent		包装组件。
Artifact		制品。
Document Artifact		文档制品。
Object		对象。
Package		包。
Aggregation		聚合，是整体与部分的关系，且部分可以离开整体而单独存在。
Composition		组合，是整体与部分的关系，但部分不能离开整体而单独存在。
Realization		实现，是一种类与接口的关系，表示类是接口所有特征和行为的实现。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。
Usage		使用，是一种使用的关系。表明一个模块在运行的时候，需要使用另外一个模块。
Constraint		是一个语义条件或者限制的表达式。UML 预定义了某些约束，其他可以由建模者自行定义。

元素名	图标	含义
Anchor		锚点。
Containment		内嵌，表示嵌在内部的类。
Generalization		泛化，表示类与类、接口与接口之间的继承关系，由子一方指向父对象一方。
Association		关联，是一种拥有的关系，它使一个类知道另一个类的属性和方法。

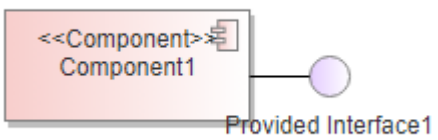
组件图显示了复杂软件系统中的各个组件如何相互关联以及如何使用接口进行通信。它们不用于更简单或更直接的系统。

组件图示例如下所示：

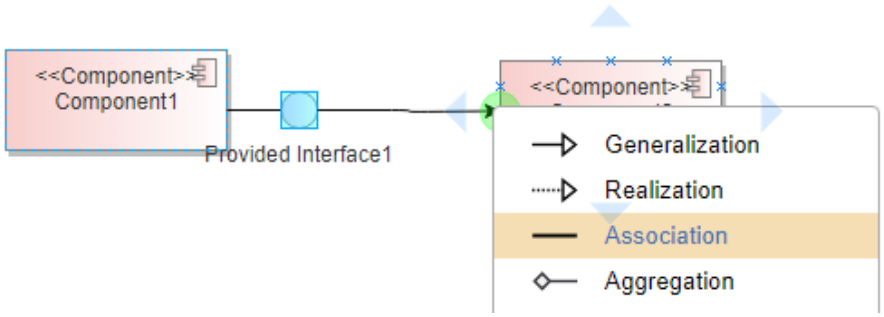


在画暴露接口与请求接口时，可以通过Association关联连线将两种接口合并。

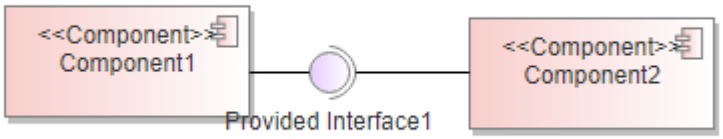
步骤1 在左侧工具画板中中选择“Provided Interface”，将其拖拽至需要连接的图形上。



步骤2 拖拽完成松开左键，在弹出的连线选择列表中选择“Association”关联连线。



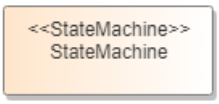
步骤3 松开鼠标后即可形成接口合并。



----结束

2.8 状态机图

元素介绍

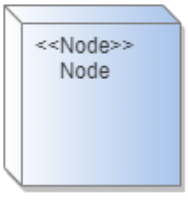
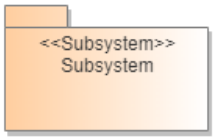
元素名	图标	含义
State		对象的生命中的满足一定条件，执行一定操作，或者等待某事件的条件或者情况。
StateMachine		状态机是展示状态与状态转换的图。通常一个状态机依附于一个类，并且描述一个类的实例对接受到的事件所发生的反应。
Fork Join		(Fork)复杂转换中，一个源状态可以转入多个目标状态，使活动状态的数目增加。 (Join)状态机活动图或顺序图中的一个位置，在此处有两个或以上并列线程或状态归结为一个线程或状态。

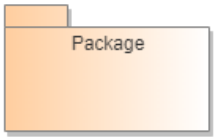
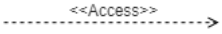
元素名	图标	含义
Fork Join		(Fork)复杂转换中，一个源状态可以转入多个目标状态，使活动状态的数目增加。 (Join)状态机活动图或顺序图中的一个位置，在此处有两个或以上并列线程或状态归结为一个线程或状态。
Initial		用来指明其默认起始位置的伪状态。
Junction		（ 结合状态 ）状态机中作为一个综合转换一部分的伪状态。它在转换执行中不打断运行至完成步骤。
Deep History		历史状态可以记忆浅历史和深历史。深历史状态记忆组成状态中更深的嵌套层次的状态。要记忆深状态，转换必须直接从深状态中转出。
Shallow History		浅历史状态保存并激活与历史状态在同一个嵌套层次上的状态。
EntryPoint		进入某一状态时执行的动作
ExitPoint		离开某一状态时执行的动作。
Final		组成状态中的一个特殊状态，当它处于活动时，说明组成状态已经执行完成。

元素名	图标	含义
Flow Final		Flow Final元素描述了系统的退出，与Activity Final相反，后者代表Activity的完成。
Synch		一个特殊的状态，它可以实现在一个状态机里的两个并发区域之间的控制同步。
Choice		选择，代表多个路径选择。
Terminate		终止。
Transition		转换用实线箭头表示，从一个状态（源状态）到另一个状态（目标状态），用一条转换线标注。
Object flow		各种控制流表示了对象间的关系、对象和产生它（作输出）或使用它（作输入）的操作或转换间的关系。

2.9 包图

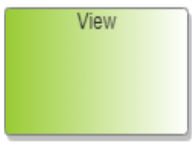
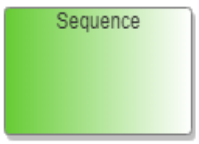
元素介绍

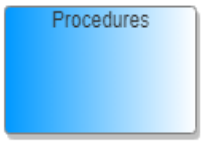
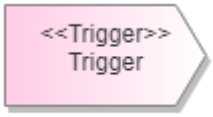
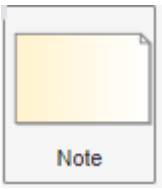
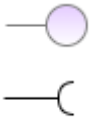
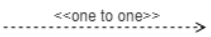
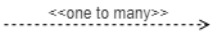
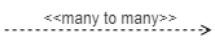
元素名	图标	含义
Node		部署节点。
Subsystem		作为且有规范、实现和身份的单元的包。

元素名	图标	含义
Package		包。
Access		访问依赖关系用一个从客户包指向提供者包的虚箭头表示。
Merge		合并连接器，定义了源包元素与目标包同名元素之间的泛化关系。源包元素的定义被扩展来包含目标包元素定义。当源包元素与目标包内没有同名元素时，目标包元素的定义不受影响。
Import		用虚线箭头从得到访问权限的包指向提供者所在的包。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。 依赖关系用两个模型元素之间的虚线箭头表示。箭尾处的模型元素（客户）依赖于箭头处的模型元素（服务者）。

2.10 实体关系图

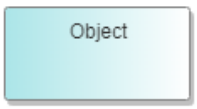
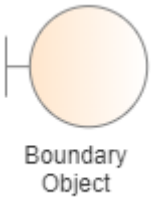
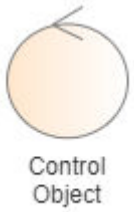
元素介绍

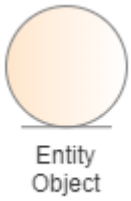
元素名	图标	含义
Entity		实体。
View		视图。模型的一个影射，它可从一个透视图或优势位置看到，而且它省略了与这个透视图无关的实体。
Sequence		顺序。

元素名	图标	含义
Procedures		过程。
ResultSet		结果集。
Trigger		触发动作。
note		文本框。
Provided/ Required Interface		Required Interface和Provided Interface之间可以建立Dependency，表明一个组件需要的接口是由另外一个组件提供的。
one to one		一对一。
one to many		一对多。
many to many		多对多。

2.11 对象图

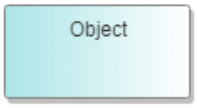
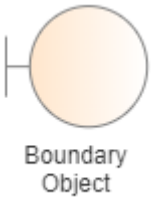
元素介绍

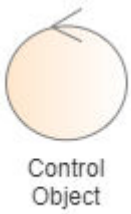
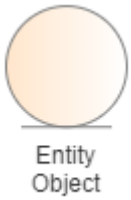
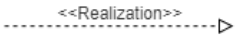
元素名	图标	含义
Actor		角色，是与系统交互的人或事物。
Object		封装了状态和行为的具有良好定义界面和身份的离散实体；即对象实例。
Collaboration		是对对象和链总体安排的一个描述，这些对象和链在上下文中通过互操作完成一个行为，例如一个用例或者操作。
Collaboration Use		使用协作用于在复合结构图中将协作定义的模式应用于特定情况。
Information Item		信息项元素表示数据的抽象，该数据可以在两个对象之间传递。
Boundary Object		边界对象。
Control Object		控制对象。

元素名	图标	含义
Entity Object		实体对象。
Dependency		依赖，是一种使用的关系，即一个类的实现需要另一个类的协助。
Information flow		信息流表示任何图中两个元素之间的信息项（信息项元素或分类器）的流。
Association		关联，是一种拥有的关系，它使一个类知道另一个类的属性和方法。

2.12 通信图

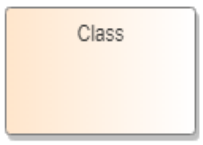
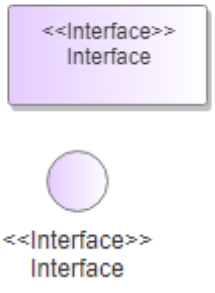
元素介绍

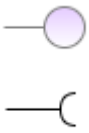
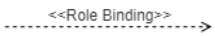
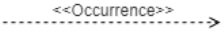
元素名	图标	含义
Actor		角色，是与系统交互的人或事物。
Object		封装了状态和行为的具有良好定义界面和身份的离散实体；即对象实例。
Boundary Object		边界对象。

元素名	图标	含义
Control Object		控制对象。
Entity Object		实体对象。
Nesting		嵌套，即一个类的嵌套到另一个类。
Realization		实现，是一种类与接口的关系，表示类是接口所有特征和行为的实现。
Association		关联，是一种拥有的关系，它使一个类知道另一个类的属性和方法。

2.13 组合结构图

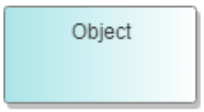
元素介绍

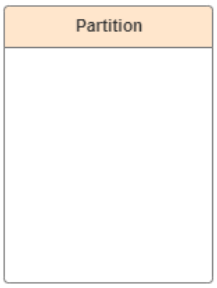
元素名	图标	含义
Class		是对象的集合，展示了对象的结构以及与系统的交互行为。
Interface		接口，可以是单个接口，也可以是抽象的一组接口的组合。 圆形接口与矩形接口意义相同，仅形状不同。

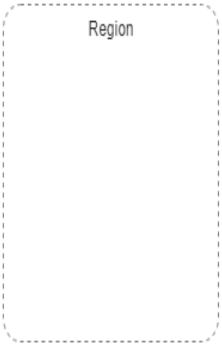
元素名	图标	含义
Property		特性就是表示传递有关模型元素信息的值的一般性术语。属性具有语义效果，在 UML中一部分属性已经事先定义好了，其他的特性是用户定义的。
Port		端口定义了分类器与其环境之间的交互。
Collaboration		协作定义了一组协作角色及其连接器。
Occurrence		使用协作用于在复合结构图中将协作定义的模式应用于特定情况。
Provided Interface Required Interface		Required Interface和Provided Interface之间可以建立Dependency，表明一个组件需要的接口是由另外一个组件提供的。
Connector		连接器通常在“组合结构”图中说明零件之间的通信链接以实现结构的目的。
Delegate		委托连接器在组件图上定义了组件外部端口和接口的内部组件。
Role Binding		角色绑定是协作使用的内部角色和实现特定情况所需的各个部分之间的映射，通常在复合结构图中。
Ocurrence		在组合结构图中，发生关系表示协作表示分类器。
Represents		表示连接器指示在分类器（通常在“组合结构”图中）中使用了协作。

2.14 交互概述图

元素介绍

元素名	图标	含义
Action		动作是可执行的原子计算，它导致模型状态的改变和返回值。
Activity		活动是状态机内正在进行的非原子执行。
StructuredActivity		结构化活动是一个活动节点，可以将下级节点作为独立的活动组。
CentralBufferNode		中央缓冲区节点是一个对象节点，用于管理活动图中表示的来自多个源和目标的流。
Datastore		数据存储区定义了永久存储的数据。
Object		封装了状态和行为的具有良好定义界面和身份的离散实体；即对象实例。
Decision		是状态机中的一个元素，在它当中一个独立的触发可能导致多个可能结果，每个结果有它自己的监护条件。
Merge		状态机中的一个位置，两个或多个可选的控制路径在此汇合或"无分支"。
Send		发送者对象生成一个信号实例并把它传送到接收者对象以传送信息。

元素名	图标	含义
Receive		接收就是处理从发送者传送过来的消息实例。
Partition		分区元素用于逻辑组织元素。
Partition		分区元素用于逻辑组织元素。
Initial		用来指明其默认起始位置的伪状态。
Final		组成状态中的一个特殊状态，当它处于活动时，说明组成状态已经执行完成。
Flow Final		Flow Final元素描述了系统的退出，与Activity Final相反，后者代表Activity的完成。
Synch		一个特殊的状态，它可以实现在一个状态机里的两个并发区域之间的控制同步。

元素名	图标	含义
Fork Join		(Fork)复杂转换中，一个源状态可以转入多个目标状态，使活动状态的数目增加。 (Join)状态机活动图或顺序图中的一个位置，在此处有两个或以上并列线程或状态归结为一个线程或状态。
Fork Join		(Fork)复杂转换中，一个源状态可以转入多个目标状态，使活动状态的数目增加。 (Join)状态机活动图或顺序图中的一个位置，在此处有两个或以上并列线程或状态归结为一个线程或状态。
Region		并发区域。
Exception Handler		异常处理。捕获异常根据异常类型查找到对应的异常处理方法，然后执行对应的方法。
Control Flow		（控制流）在交互中，控制的后继轨迹之间的关系。
Object Flow		（对象流）各种控制流表示了对象间的关系、对象和产生它（作输出）或使用它（作输入）的操作或转换间的关系。
Interrupt Flow		中断流是用于定义异常处理程序和可中断活动区域的连接器的两个UML概念的连接。

