# 微服务剖分

## 原则

### 1.1 微服务剖分相关概念

1. 微服务架构定义

微服务架构是一种从业务角度对应用进行功能切分，并以分布式运行的组件化方式。微服务架构切分后的组件应满足以下要求：各组件具备单一但完整的业务功能，可以独立运行、独立升级；各组件具备定义良好的轻量接口；组件运行力求自给自足，自己管理自己的数据。尽量减少组件间的依赖和影响，只在必要时才通过接口调用其他组件。通过以上的功能切分，形成多个可分散治理、功能完备、耦合度低、拥有独立生命周期、可灵活拼装的业务单元，各个独立的单元就是微服务。

简而言之，微服务就是一些协同工作的小而自治的服务。在很多软件开发的组织中会存在这样一个问题，随着新需求的增加，所要维护的代码量就会越来越庞大，以至于即使需要添加一个很小的功能也需要大面积修改代码库，或者很小的代码改动都很难保证不影响其他功能。虽然可以选择模块化，但是模块与模块之间的界限确是很难确定和维护的。

2. 微服务的粒度

微服务粒度太小会带来效率和可用性降低问题，而过大又会导致灵活性降低。微服务的“微”并不是限定微服务的大小尺寸，而是限定微服务所包含的业务功能。微服务架构定义要求切分的组件需要“具备单一但完整的业务功能”，所以“微”是指业务功能单一、但完整。

从业务角度看，“单一”且“完整”功能的微服务具备以下特点。

（1）“单一”：业务功能原子性，执行过程不可出错或中断，否则已执行部分的状态就应该回滚或重做。如果服务中某部分执行得到的结果，与其他部分执行出错与否没有关系，那么这样粒度的微服务的业务功能就不是“单一”的，这一部分的逻辑应该拆分出来成为一个新的微服务。以此类推继续拆分，直到违反“完整”原则为止。由此可见，“单一”原则用来限定微服务粒度的上限。

（2）“完整”：如果继续细分，得出的更小粒度服务将失去业务意义。如果说合适粒度的微服务完成一个业务“任务”，继续细分只能得到一系列完成这项“任务”所需的“步骤”。单从“步骤”上看无法判断这个“步骤”是在完成哪个“任务”。显然，划分到“步骤”这个级别，对于微服务粒度来说就太细了。可见，“完整”原则用来衡量微服务粒度的下限。

### 1.2 微服务剖分原则

微服务剖分的问题在于，服务有多小才算合适，或者说代码量有多小才算合适。如果简单地从代码的数量来讨论感觉并不十分合理，因为不同语言表达力并不相同，有些高级语言可以只写简单的几行代码完成很多底层语言需要写很多代码才能实现的功能。还有一个需要考虑的问题是不同业务特性对象其本身可能就比较复杂，对象本身描述的代码量就很大。除此之外，微服务的结构能否和公司的组织架构很好的协调也至关重要。

一个服务就可以看作为一个独立的实体。把多个服务部署在同一个实例中并不是什么明智是选择，如果服务承担的服务比较重的话还是建议单独部署，如果服务并不重，也要选择资源消耗类型相同的服务一起部署。然而隔离性可能会带来一些资源上的浪费，但是其却可以很大程度和是上简化分布式系统的构建。并且服务与服务之间都是使用网络通信完成服务之间的交互，这样就更强化了服务间的隔离性，进而降低服务之间的耦合性。

每个服务能够在不影响其他服务的前提下独自进行代码和服务的修改、开发和升级，而且单一服务的部署或者升级不需要消费者一起进行修改。服务与服务之间选择暴露统一的API进行通信，来降低集成难度，实现系统的统一标准开发。统一的API应该避免与服务消费者之间的耦合，所以要求使用的API技术实现必须跟具体的业务或者具体的技术没有关联。

微服务划分没有普适的标准，需要结合项目实际开发综合考虑以下多个因素：

（1）业务视角。服务可以分为业务服务和技术服务。根据对微服务架构的定义，微服务的切分应该从业务角度进行功能划分，每个微服务应该都提供的是业务服务。

（2）自给自足。常见到有项目把共用的技术处理逻辑从各微服务中提取出来，以一个独立的、提供“技术服务”的组件运行，并提供RESTful接口被其它微服务调用，例如“数据库访问服务”、“日志记录服务”等。出现这种切分方式的原因是人们沿袭了传统的组件化观念和习惯，目的是有利于代码复用，减少重复维护工作等。但微服务切分的目的不是为了复用，而是为了提高架构的灵活性。微服务架构要求每个微服务自己包含所需的技术处理逻辑，包括数据存取、加工、发送接收等，由此来保障微服务运行自给自足，尽量避免微服务对外界的依赖。这对保证微服务的独立运行、独立升级、灵活拼装能力等是非常重要的。

从对微服务架构的定义角度看，上面的“技术服务”组件，违反了从“业务视角”进行服务划分的原则；同时，这一做法也不符合“组件运行力求自给自足，自己管理自己的数据。尽量减少组件间的依赖和影响”的要求，因此不建议这样的切分方式用于微服务架构。为了提高代码的复用性，可以考虑把共用的技术处理逻辑封装成单独的公用库，在微服务代码中静态引用即可。这样在提高代码复用性的同时，仍然保证了微服务的独立性。

（3）功能单一但完整。按照“单一且完整”的定义，只要是实现单一业务功能需要的逻辑都应该放在这个微服务组件中，而不是放在另一个微服务中被调用，否则就违反了“完整”原则。反之，只要与实现某单一业务功能无关的逻辑，一定要把它拿出来，放在它应该属于的那个微服务中去，否则就将违反“完整”原则。

（4）非事务性和最终一致性。微服务架构中如果需要跨多个微服务维护事务，则在本质上是处理分布式系统的事务性和数据一致性问题。这往往带来高昂的成本，相比之下，只在一个微服务内维持事务性要容易得多。因此，在决定微服务拆分的边界时，应当考虑把有事务性要求、数据强一致性要求的逻辑放在一个服务中。微服务拆分后，微服务的边界应该符合非事务性、数据最终一致性即可的要求。

（5）平衡效率。微服务切分后形成的各个服务以分布式方式运行，因此相对于传统单体应用，微服务架构在可靠性和效率方面都有下降。一个完整的业务需要的微服务越多，它的可靠性和效率就越低。因此，微服务拆分应该同时关照业务的可靠性和效率，必要时可以考虑吧相关微服务进行合并，这实质上是在效率、可靠性和微服务的“功能单一”原则之间做权衡。

划分微服务的标准则有如下几点：

从设计角度：微服务应该是高内聚低耦合的，有明晰的界面，与外界尽量少的交互。

从开发角度，微服务应该是一个小的团队、在短时间内可以完成的，保持沟通的高效和管理的低成本。

从运维的角度：微服务的粒度不能太小，种类不能太多，否则会使得系统复杂性和管理成本急剧上升，有违微服务架构的初衷。

现行的微服务剖分的主要解决方案有以下两个：

1、根据业务能力拆分

业务能力是业务架构模型中的一个概念。业务模型经常对应于一个业务对象；业务能力经常组织成一个多层等级。比如说，一个企业应用也许有顶级的分类，如产品开发、产品交付、需求挖掘等。每个分类可以对应一个或几个用以实现该业务的微服务。这需要对业务有好的理解，需要对组织的目标、结构、业务流程做具体分析。这样做的好处主要有：

* 架构稳定，因为业务能力相对比较稳定
* 开发团队是自主的，围绕着交付业务价值而不是技术特性来组织
* 服务之间共同合作，松耦合

2、根据子域拆分

该定义对应于领域驱动设计（DDD）的子域的服务。一个领域由多个子域组成。每个子域对应了业务的不同组成部分。

子域有以下几个分类：

* 核心：业务的核心区分点，应用的最有价值的部分
* 支持：与业务是做什么的相关，但不是主要区分点。这个可以自己做或者外包
* 通用：不特定于业务，理想情况下使用现成的软件来实现

## 策略

### 2.1 业务服务特征建模

为了解决业务的服务精准建模问题，并在业务基础上充分考虑软件复用，提出一种以服务特征、服务特征间的关系为核心的面向服务的特征模型，从面向服务的视角来组织领域需求。采用一阶逻辑明确定义需求的语法和语义。在这个定义之下，产生出面向服务的特征模型的四个基本性质，这些性质是验证服务模型正确性的基本条件。本项目拟将服务引入领域模型的组织方式，可以提高领域模型的可复用程度，并且便于实现从需求模型到设计模型的平滑过渡。另一方面，在整个模型转换的过程中，会产生各个层次的模型，这些模型必须保持一致。

服务特征是指软件系统中用户可见的、显著的、具有特点的属性、品质、特点。服务特征是一个用户或客户可感知的系统特点。从需求规格说明的组织结构角度来看，服务特征提供了一种对需求的分割和组织方式，即以服务特征作为需求空间内的一阶实体，系统具有的服务特征及其相互关系构成了系统的需求空间；从需求的内涵来看，一个服务特征体现了系统具有的某种能力或特点，反映了需求获取的参与者对系统的某种要求或理解；从需求的类型上看，一个服务特征可能是一种功能性的需求，或是对系统质量属性的要求，或者是外部环境对系统的某种约束条件。根据前期调研，可以给出服务特征的概念模型如图2-1所示。

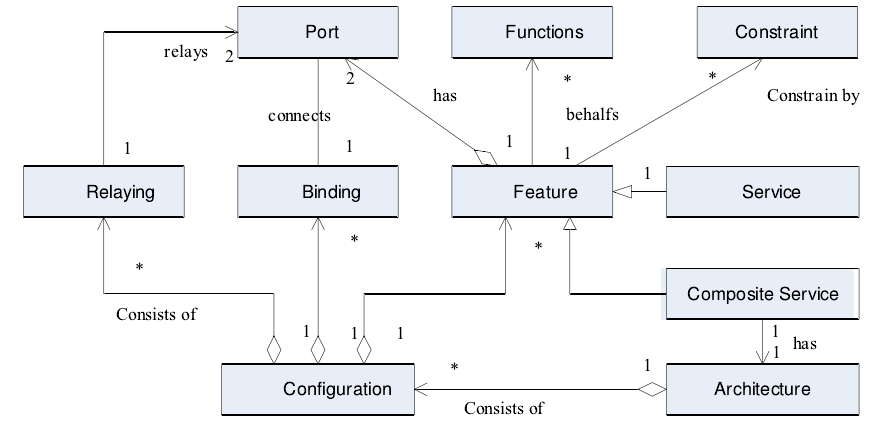


图2-1 服务特征概念模型

基于服务特征概念模型，从需求规格说明的组织角度给出一个服务特征的形式化描述。

定义1：服务特征。服务特征是指从面向服务的角度来描述系统具有的某种属性，它将需求表达为系统可以提供给最终用户的服务集合。每个服务特征都由9元组来表示

Fid是服务特征的唯一标识符，服务特征定义需要在领域术语词典中被引用。

Pa是服务特征的父服务特征，表示领域模型的层级结构。

Ps是指一个服务特征可以提供的一组服务。

T指服务特征的类型，包括强制类型、必选类型和可选类型三种。服务特征的类型描述的是同一层次的服务特征之间的选择关系。服务特征的类型必须明确指明。

Bt(Bindingtime)是指绑定时间。Bt需要表明服务特征是在编译时刻（静态时）、激活时刻（初始化阶段）或者运行时刻绑定。为了使得模型简化，绑定时间通常被指明为编译时刻。

Rs(Requiringservicefeatures)是指该特征所需的一系列服务特征，称为需求服务特征。它包括服务特征实现时所有需要的服务特征。

Es(Excludingservicefeatures)是指该特征所排斥的服务特征，称为互斥服务特征。它包括该服务特征被选择时，所有不能同时选择的特征。

C(Constraints)是指服务特征中必须遵循的一系列约束。

Cd(ChildFeatures)是指服务特征的所有子服务特征。

在服务特征中，对于服务特征功能的描述是通过服务特征可以提供的一组服务即Ps来给出的。Ps是服务特征中的服务集合属性，下面给出服务的形式化定义。

定义2：服务（Service）。一个服务由2元组Service=(Interfaces,Operation)构成。

Interfaces是服务提供给外部的接口集合，服务通过服务接口与其它服务进行通信，提供服务给外部系统，接受所需的服务以及信息以完成服务所提供的功能。一个服务往往会包含多个服务访问入口，而每一个访问入口都会有一个Interface元素来描述。

Operation实现接口声明的服务、对服务中所支持的操作的抽象描述，Operation描述了一个访问入口。

定义3：操作（Operation）。操作用3元组Operation=(OperationName,Parameters,DataDecl,Predication)表示。

DataDecl是数据声明，是说明服务所使用的数据的声明。是数据结构的抽象类型化定义。

Parameters是参数声明，说明一个操作的可变性。

Predication是对Operation所描述的操作中的约束部分。

基于上述定义，可以给出服务描述的元模型，如图2-2所示。

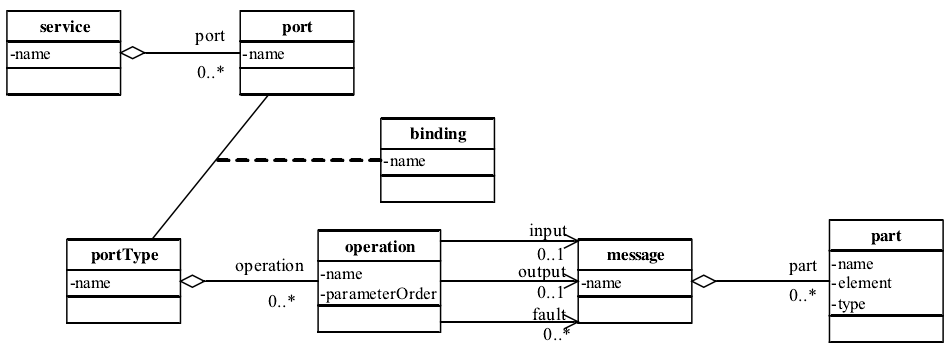


图2-2 服务元模型

基于服务特征和服务模型，可以给出面向服务的特征模型，该特征模型包含服务特征和服务特征之间的关系两个部分。图2-3给出了面向服务的特征模型的概念图。

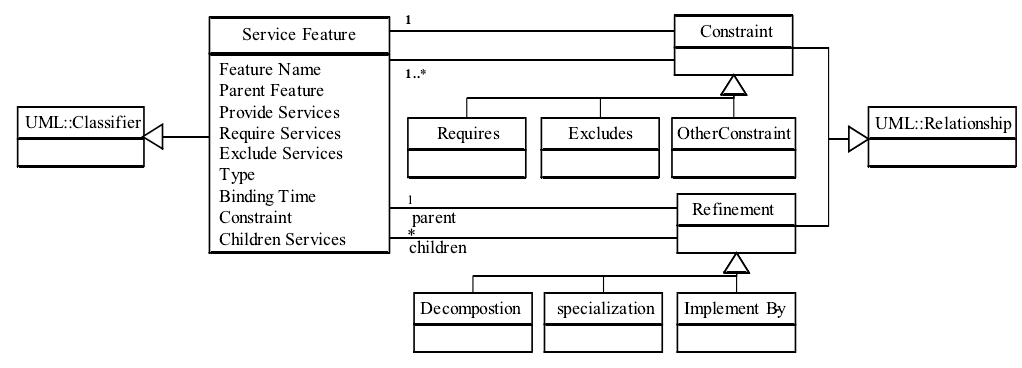


图2-3 服务特征模型概念图

定义4：面向服务的特征模型（Service-OrientedFeatureModel，SOFM）。一个面向服务的特征模型抽象表示了领域中的系统，包括服务特征集合和服务特征之间的关系集合，可以采用2元组表达SOFM=(F,R)，其中F是面向服务的特征模型所包含的服务特征集合。服务特征集合中的每一个服务特征必须预先定义。R是面向服务的特征模型中的服务特征之间存在的关系集合。

服务特征间的关系被分为两大类：约束关系(constraint)以及求精关系(refinement)。

约束关系是绑定状态的服务特征之间的静态关系。服务特征之间的约束关系被分为三类：需求约束、排斥约束以及其他约束。软件需求模型中必须满足这些约束，也就是说，从服务特征模型所描述的领域模型定制目标系统的需求模型时，必须验证这些约束关系，这些约束关系也指导服务特征模型的定制。只有那些不会违背服务特征间约束的决策才能产生有效的目标系统需求。只有通过明确的对这些约束建模，服务特征模型才能有一个质量良好的需求模型。

需求关系(require)：需求关系定义了两个服务特征之间的一种约束。它的含义是，在服务特征模型实例中，如果一个服务特征被选择，那么需要同时选择的其它服务特征为它提供必须的服务。如果F1和F2是服务模型中的两个服务特征，F1需求F2表示：F1需要F2提供的服务，F2提供服务给F1。表示为F1ReF2。需求关系是有方向的。

排斥关系(exclude)：排斥关系定义了两个服务特征之间的一种约束。它的含义是，在服务特征模型的实例中，如果一个服务特征被选择，它排斥另一个服务特征的选择，也就是说，另一个服务特征不能同时被选择。如果两个特征同时被选择了，那么就会产生矛盾。它可以表示为Ex。排斥关系表示为F1ExF2，这里F1和F2是面向服务特征模型中的两个服务特征。它意味着，在服务特征模型实例中，如果服务特征F1被选中，那么它要求不能同时选择服务特征F2。需要注意的是，排斥关系是相互的，也就是说，如果F1ExF2，那么一定有F2ExF1。

其它约束关系(otherconstrain)：其它约束是指服务特征之间的制约关系，它包括领域知识的原理，服务特征的组合原则，以及用户所做出的权衡决定（trade-off）。这类约束用于调整服务特征的选择，以及非功能需求问题的实施。

求精关系可以分为三类：分解求精(decomposition)、特例求精(specialization)以及可操作求精(operationalization)。

可以看出，面向服务的特征模型形成了一个结构化的需求空间，它给出了需求空间的一种层次分解，包括服务特征的定义，属性以及服务特征之间的关系。事实上，服务特征是一个特殊的面向服务的特征模型，在这个面向服务的特征模型中，关系是一个空集。服务特征集合为服务特征本身所组成的集合。一个应用程序就是一个对领域服务模型经过定制之后的目标系统服务特征模型的实例，通常来说，应用程序模型一定是一个闭合的服务特征模型。

针对具体应用，首先进行需求分析，建立该应用所属的服务领域特征模型，特征模型会包含多个服务特征，每个服务特征表示在该应用中强制或可选的功能。针对每个服务特征，需要明确该特征包含的服务集合，定义服务的接口和操作方式，最后需要明确不同的服务特征之间的关系。通过这种方式，可以以面向服务的方式明确刻画某个应用中应该包含的服务特征以及特征之间的关系，服务特征中包含的服务集合与约束，每个服务的接口规约和操作细节等，形成业务服务的领域模型、数据字典和接口协议等规约。

### 2.2 基于层次任务网络的任务剖分方法

复杂的任务模型往往结构庞大且功能繁多，其所对应的建模任务信息量大且内容多，因此应将其逐步细化分解。任务分解是为后续的任务规划和分派管理控制提供依据的关键一步，其按照一定的原则将抽象的建模总任务细化为多个简单而具体的子任务的替换集合，得到可执行的“建模子任务金字塔”，子任务间存在松散的关联耦合度，且具有各自的独立性，以便多个建模成员协同执行建模任务。结合分布式协同任务建模流程，其任务分解主要步骤如下：

1.依据待求解的任务问题抽取建模需求，确定建模总任务和目标；

2.将总任务细化、分解并扩展为子任务网络；

3.分析子任务的可行性，判断建模成员对子任务现有或潜在的规划和建模能力；

4.对于甄选失败的子任务将粒度继续分解，直至所有子任务分解粒度为可用一种可行的方法予以求解为止；

5.对于甄选成功粒度合适的子任务则将其分派给潜在的建模成员

任务建模任务分解依据：任务分解的关键在于分解依据与粒度的把握，分解是否合理将影响任务建模的顺利进行，若分解粒度太粗，则子任务复杂度大且难执行；而若分解粒度太细，则又会增加任务管理与控制的难度。分解后子任务的粒度是否合适，取决于建模成员按照何种分解依据对建模任务进行细化分解，即取决于成员的规划和建模能力。建模任务分解具有其特殊性，需要充分考虑任务建模特点，通常分布式协同任务建模中任务分解可按以下方式进行。

(1)功能结构：按照拟构建模型的功能结构进行任务分解，将总任务分解成多个子领域、子模型、子操作甚至是子运算。例如，将任务建模总任务分解为输入、处理、输出等功能子模块。

(2)组织多方性：分布式协同任务建模涉及的组织数量日益递增，需要众多部门和机构的积极参与，根据各部门的数据资源和建模能力，将建模任务分与不同的组织机构。

(3)时空尺度：时空尺度是考察任务事物特征与变化的时间和空间范围，包括客体、主体和时空维。不同时空尺度的任务模型的机理不同，例如将“流域”尺度建模任务划分成多个“子流域”尺度的子任务。

(4)计算复杂性：任务模型是为了求解复杂问题所构建的计算资源，涉及空间数据、存储资源和计算能力等资源组织方式，可分为空间复杂性和时间复杂性等方面。

一般而言，给定的建模任务大多是模糊的、非量化的、非形式化的，但这种描述方式存在许多问题：跨学科领域的建模成员间沟通困难、同一建模任务的抽象描述不一致、空间信息及地理模型重复采集和构建等，且不利于建模任务在计算机上的表达。因此，需要在建模之初就对所需分解的任务进行完整而正确的描述，即利用形式化手段对地理建模任务进行表达。

1.层次任务网络规划：HTN规划是在设计加工、项目管理、生产调度、装配规划等实际应用领域中使用最为广泛的一种规划技术，原因在于其面向工程实际且能够较好地利用领域知识进行推理分解，适用于大规模层次任务环境中的工作规划和流程的计算机辅助制定。国际上现已对HTN规划的理论模型进行了研究，提出了完备的模型，并给出可被证明正确的HTN规划程序：UMCP、SHOP2、JSHOP2等。HTP规划形式化表示成一阶谓词逻辑语言6元组。其中：

V为变量符号集合；

C是常量符号集合；

P为谓词符号集合；

F为基本任务（动作）符号集合；

T为复合任务符号集合；

N是任务标记（labeling）符号集合；

P、F、T两两相交为空集。

基本任务的语法形式为：，其中，，为项。

目标任务的语法形式为：，其中，是文字。

复合任务的语法形式为：，其中，，为项。

目标任务和复合任务统称为非基本任务。

任务网络的语法形式为：,其中，是任务，其可能是基本任务也可能是非基本任务；是的标记，以便与其他出现在网络中的任务相区别；表示任务之间的关系，包含逻辑算子和逻辑析取。

操作（算子）是一个基本的操作行为模式，语法形式为：，其中，是基本任务符号；是出现的变量；而是文字，表示基本动作即最底层任务的作用结果。

分解方法的语法形式为：，其中，为非基本任务；为任务网络，表示完成任务网络是实现任务的一种分解方法，分解方法将上层的抽象任务分解为下层的具体任务。

规划领域表示为：，其中为操作（算子）集合；为分解方法集合，的构建需要借助领域知识和经验，是智能策略的主要表现形式。

规划问题定义为四元组：，是描述规划系统要解决的具体问题的实例。其中，为系统初始状态；d为要规划的目标任务；D为规划领域，分解为操作集合和方法集合。

2.形式化表达：建模任务具有层次性，需要在科研机构、实验室以及建模成员之间自上而下完成任务主体的匹配和甄选。HTN规划正是基于分层抽象和领域知识的分布式智能规划技术和求解方法，与协同求解任务问题时进行建模任务规划的思考方式非常相似。基于HTN的语义和模型结构进行模型任务的形式化定义如下。

定义１：地理建模任务的HTN形式化框架

语言Ｌ的项集为Term，子项分为４类：Hol（建模成员）、Act（行为）、Htn（分层任务网络）、Obj（其他对象）。相应的：

，式中表示成员集合；表示行为集合，且有，主要分为基本行为集合和非基本行为集合两部分；表示基于行为的分层任务网络；而表示其他对象集合。

，表示建模成员之间直接包含的偏序关系。

Can：，给出能执行某个基本行为的建模成员集合，其中，是幂集符号。通过该定义可以间接体现出建模成员的执行能力，表征了成员的建模能力。

∏：，定义每个建模成员所拥有的分层任务网络集合，表征了成员的规划能力。

Meth：，定义了每个分层任务网络的目的，即该分层任务网络所用于求解的非基本任务行为。对应于ＨＴＮ规划方法的语法形式，即表明了任务网络d是实现该非基本任务a的方法。

定义２：任务建模（Military Mission）

表示：建模成员i接受建模子任务a，φ是与a有关的约束。为简化建模任务递归分解算法的说明，在形式化同时给出相应的符号约定：①i、j，k表示建模成员符号；②φ、φ′、φ″表示约束公式符号；③a、b表示基本行为符号；④α、β表示非基本行为符号；⑤ω表示分层任务网络符号，其中φ是空网络符号。

OTD算法一种高效ＨＴＮ规划算法，按照任务网络d中子任务的先后顺序，采用前向搜索依次进行任务分解，每次分解都基于对当前系统状态的判断而生成一个可行计划，并记录动作序列的代价信息，最终通过一次性回溯机制判定并返回一个优化的计划，从而减少了求解过程中的不确定性，使求解效率得到极大提高。基于OTD的任务建模分解算法如下图2-4所示:

图2-4 基于OTD的建模任务分解算法