



(12) 发明专利

(10) 授权公告号 CN 101937353 B

(45) 授权公告日 2014. 11. 05

(21) 申请号 201010290022. 8

(22) 申请日 2010. 09. 20

(73) 专利权人 中兴通讯股份有限公司

地址 518057 广东省深圳市南山区高新技术  
产业园科技南路中兴通讯大厦法务部

(72) 发明人 张新平 祝继洪

(74) 专利代理机构 北京元本知识产权代理事务  
所 11308

代理人 秦力军

(51) Int. Cl.

G06F 9/445 (2006. 01)

(56) 对比文件

CN 1968154 A, 2007. 05. 23,

CN 101488996 A, 2009. 07. 22, 全文 .

US 2003/0018694 A1, 2003. 01. 23, 全文 .

审查员 李慧

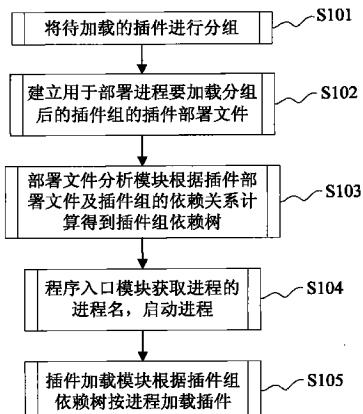
权利要求书2页 说明书6页 附图4页

(54) 发明名称

一种插件部署和加载的方法及装置

(57) 摘要

本发明公开了一种插件部署和加载的方法及装置,该方法包括以下步骤:将待加载的插件进行分组;建立用于部署进程要加载分组后的插件组的插件部署文件;部署文件分析模块根据插件部署文件及插件组的依赖关系计算得到插件组依赖树;程序入口模块获取进程的进程名,启动进程;插件加载模块根据插件组依赖树按进程加载插件。本发明通过分组形式加载插件,支持分布式管理,进程工作目录和程序目录分离,实现了不同功能进程共享程序文件。



1. 一种插件部署和加载方法,其特征在于,包括以下步骤:

A、将待加载的插件按类型进行插件分组,得到适于按进程加载的插件组;

B、建立用于部署进程要加载的所述插件组的插件部署文件;

C、部署文件分析模块根据所述插件部署文件及所述插件组的依赖关系计算得到插件组依赖树;

D、程序入口模块通过分析命令行函数获取所述进程的进程名,启动进程;以及

E、在进程启动后,插件加载模块根据基于所述插件部署文件及所述插件组的依赖关系计算得到插件组依赖树,按所述进程加载所述插件组。

2. 根据权利要求 1 所述的一种插件部署和加载方法,其特征在于,在所述步骤 A 之前,还包括以下步骤:

建立用于添加插件实现基础功能的基础扩展点接口和所述基础扩展点接口的插件基础扩展点接口实例函数;

建立派生于所述基础扩展点的用于管理不同类型插件的相应派生接口和所述派生接口的插件派生接口实例函数;以及

定义所述插件的接口的类型。

3. 根据权利要求 1 所述的一种插件部署和加载方法,其特征在于,在所述步骤 D 之前,建立用于定义进程启动和运行所需名值对的属性数据文件。

4. 根据权利要求 3 所述的一种插件部署和加载方法,其特征在于,在所述步骤 D 和步骤 E 之间,还包括以下步骤:

所述程序入口模块建立用于管理所述插件基础扩展点接口实例函数和所述插件派生接口实例函数的插件工厂接口;

所述程序入口模块建立用于描述运行环境的上下文接口;以及

所述程序入口模块解析所述插件部署文件、所述属性数据文件和所述插件组依赖树。

5. 一种插件部署和加载装置,其特征在于,包括:

插件分组模块,用于将待加载的插件按类型进行插件分组,得到适于按进程加载的插件组;

插件部署模块,用于建立部署进程要加载的所述插件组的插件部署文件;

部署文件分析模块,用于根据所述插件部署文件及所述插件组的依赖关系计算得到插件组依赖树;

程序入口模块,用于通过分析命令行函数获取所述进程的进程名,启动进程;以及

插件加载模块,用于在进程启动后,根据基于所述插件部署文件及所述插件组的依赖关系计算得到插件组依赖树,按所述进程加载所述插件组。

6. 根据权利要求 5 所述的一种插件部署和加载装置,其特征在于,还包括:

插件定义模块,用于定义插件基础扩展点接口、派生接口、所述插件基础扩展点接口的实例函数、所述派生接口的实例函数和所述插件接口的类型。

7. 根据权利要求 6 所述的一种插件部署和加载装置,其特征在于,所述插件定义单元包括:

插件基础扩展点定义单元,用于建立添加插件实现基础功能的基础扩展点接口和所述基础扩展点接口的插件基础扩展点接口实例函数;

插件派生接口定义单元,用于建立派生于所述基础扩展点的用于管理不同类型插件的相应派生接口和所述派生接口的插件派生接口实例函数;以及

插件描述文件定义单元,用于定义所述插件的接口的类型。

8. 根据权利要求 5 所述的一种插件部署和加载装置,其特征在于,还包括:

属性数据模块,用于定义进程启动和运行所需名值对的属性数据文件。

9. 根据权利要求 8 所述的一种插件部署和加载装置,其特征在于,所述程序入口模块还用于建立管理所述插件基础扩展点接口实例函数和所述插件派生接口实例函数的插件工厂接口;建立描述运行环境的上下文接口;以及解析所述插件部署文件、所述属性数据文件和所述插件组依赖树。

## 一种插件部署和加载的方法及装置

### 技术领域

[0001] 本发明涉及一种软件插件的加载技术,特别涉及一种插件部署和加载的方法及装置。

### 背景技术

[0002] 在大型软件开发中,为了方便对软件进行功能扩展,实现模块内的高内聚,模块间的低耦合,为实现发布软件功能的可定制性,往往采用插件式开发,将一个个功能封装在插件中实现,这些插件有多种类型,即遵循的插件接口不一样,而为了处理上的方便,这些不同的接口可以派生于同一个基础接口。

[0003] 在综合型的网管中要管理各种专业网产品,而每种专业网产品拥有很多种设备,专业网的特性也各有不同,由于不同设备之间存在差异,一个网管要管理数以万计的设备就需要多个进程、分布式管理。由于不同进程所需部署的功能也不同,因此提供一种软件架构来开发这种大型网管,管理不同的专业网,成为我们需要解决的问题。

### 发明内容

[0004] 本发明的目的在于提供一种插件部署和加载方法,采用大型软件插件分组管理、进程共享程序目录和分布式进程内插件部署及插件加载的方法,解决了大型网管管理不同的专业网的问题。

[0005] 本发明的另一目的在于提供一种插件部署和加载装置,采用大型软件插件分组管理、进程共享程序目录和分布式进程内插件部署及插件加载的方法,解决了大型网管管理不同的专业网的问题。

[0006] 根据本发明的一个方面,提供了一种插件部署和加载方法,包括以下步骤:

[0007] A、将待加载的插件进行分组;

[0008] B、建立用于部署进程要加载分组后的插件组的插件部署文件;

[0009] C、部署文件分析模块根据插件部署文件及插件组的依赖关系计算得到插件组依赖树;

[0010] D、程序入口模块获取进程的进程名,启动进程;

[0011] E、插件加载模块根据插件组依赖树按进程加载插件。

[0012] 根据本发明的另一方面,提供了一种插件部署和加载装置,包括:

[0013] 插件分组模块,用于将待加载的插件按类型划分成适于按进程加载的插件组;

[0014] 插件部署模块,用于建立部署进程要加载插件组的插件部署文件;

[0015] 部署文件分析模块,用于根据插件部署文件及插件组的依赖关系计算得到插件组依赖树;

[0016] 程序入口模块,用于获取所述进程的进程名,启动进程;

[0017] 插件加载模块,用于根据插件组依赖树按进程加载插件。

[0018] 与现有技术相比较,本发明的有益效果在于:本发明通过分组形式加载插件,支持

分布式管理,进程工作目录和程序目录分离,实现了不同功能进程共享程序文件。

## 附图说明

- [0019] 图 1 是本发明提供的插件部署和加载方法的流程示意图;
- [0020] 图 2 是本发明提供的插件部署和加载装置的结构示意图;
- [0021] 图 3 是本发明实施例提供的网管插件分组示意图;
- [0022] 图 4 是本发明实施例提供的进程启动过程和分析命令行获取进程名称的示意图;
- [0023] 图 5 是本发明实施例提供的软件发布目录结构图。

## 具体实施方式

[0024] 以下结合附图对本发明的优选实施例进行详细说明,应当理解,以下所说明的优选实施例仅用于说明和解释本发明,并不用于限定本发明。

[0025] 图 1 显示了本发明提供的插件部署和加载方法的流程示意,如图 1 所示:

[0026] 步骤 S101,将待加载的插件进行分组。

[0027] 在此之前,需对插件接口进行定义:建立程序的基础扩展点接口和该基础扩展点接口的插件基础扩展点接口实例函数;建立派生于该基础扩展点的派生接口和该派生接口的插件派生接口实例函数;定义插件的描述文件格式:定义插件 ID、插件接口类型和插件库名称。

[0028] 如按产品功能可将插件产品分为产品插件单元 PPU,PPU 是顶层组,一个 PPU 下有多个插件管理单元 PMU,PMU 下有多个插件功能组 PMUFunction,插件功能组下有多个插件或数据文件,这些插件功能组可以有交叉插件和数据文件。

[0029] 步骤 S102,建立插件部署文件,在该文件中部署进程要加载的分组后的插件组,一个进程可能有多个部署文件,进程按这些文件的并集加载插件。

[0030] 步骤 S103,部署文件分析模块根据插件部署文件及插件组的依赖关系计算得到插件组依赖树。

[0031] 步骤 S104,程序入口模块通过分析命令行函数获取进程的进程名,启动进程。

[0032] 此外,程序入口模块需先创建插件工厂接口,再创建上下文接口,然后解析插件部署文件、属性数据文件和插件组依赖树。

[0033] 在此之前,需建立属性数据文件,该文件是定义进程在启动或运行过程中需要的一些名值对,这些属性文件是分层的,根据需要可以定义多层。

[0034] 步骤 S105,插件加载模块根据插件组依赖树按进程加载插件。加载插件有两种方式:进程启动过程中加载某些插件或运行过程中按需加载插件。

[0035] 图 2 是本发明提供的插件部署和加载装置的结构示意,如图 2 所示,该装置包括插件定义模块、插件分组模块、插件部署模块、部署文件分析模块、程序入口模块、属性数据模块和插件加载模块。

[0036] 其中,插件定义模块包括插件基础扩展点定义单元、插件派生接口定义单元和插件描述文件定义单元。

[0037] 插件基础扩展点定义单元建立程序基础扩展点接口和该基础扩展点接口的实例函数。

[0038] 插件派生接口定义单元建立派生于基础扩展点接口的派生接口和该派生接口的实例函数。

[0039] 插件描述文件定义单元定义插件的描述文件格式,包括:插件 ID、插件接口的类型和插件库名称。

[0040] 插件分组模块将待加载的插件划分成适于按进程加载的插件组。

[0041] 插件部署模块建立部署进程要加载插件组的插件部署文件。

[0042] 部署文件分析模块根据插件部署文件及插件组的依赖关系计算得到插件组依赖树。

[0043] 程序入口模块获取所述进程的进程名,启动进程,并建立管理插件基础扩展点接口实例函数和插件派生接口实例函数的插件工厂接口和建立描述运行环境的上下文接口,以及解析插件部署文件、属性数据文件和插件组依赖树。

[0044] 属性数据模块定义进程启动和运行所需名值对的属性数据文件。

[0045] 插件加载模块根据插件组依赖树按进程加载插件。

[0046] 图 3 显示了本发明实施例提供的网管插件分组示意,如图 3 所示,顶层有 bn.ppu、uca.ppu 两个 PPU,其中 bn.ppu 下有 bn-core-c.pmu、bn-necommon-c.pmu 和 bn-res.pmu 多个 PMU,每个 PMU 下多个插件功能组 PMUFunction,如其中 bn-res.pmu 下有两个功能组 osf 和 emf,各包含有一些目录,且存在共同的目录 common,与 common 同级的目录下是一些数据文件、插件及描述文件,其中同级的 d11 中为普通的动态库。

[0047] 图 4 显示了本发明实施例提供的进程启动过程和分析命令行获取进程名称的示意,如图 4 所示,程序入口模块首先创建插件工厂接口,再创建上下文接口,在上下文的初始化插件函数中创建本地系统接口 ILocalSystem,本地系统接口负责分析部署及加载启动插件。插件工厂接口 IExtensionService、上下文接口 IContext 和 ILocalSystem 对于每个进程必须存在一个实例。

[0048] 图 5 显示了本发明实施例提供的软件发布目录结构,如图 5 所示,在 procs 下是程序和数据文件,在 works 下是一些进程对应的目录,进程对应的目录下有一些子目录,如 log 目录用来存放运行过程中产生的日志,而 deploy 下用来存放插件部署文件和属性数据文件。

[0049] 下面结合图 1 ~ 图 3 对统一网管的需求进行说明。

#### [0050] (一) 需求来源

[0051] 在统一网管中,需要将各种产品线的设备在一个网管中管理,各产品线有多种类型的设备,各设备的差异较大。在一个网管中管理多个产品线的设备,既节省资源又便于统一管理,在设备数量庞大的情况下,需要分布式的管理。

#### [0052] (二) 开发平台

[0053] 自适配通信环境 ACE(ADAPTIVE Communication Environment),编译器 vc7 或 gcc,支持 win32、solaris 和 linux 等多个操作系统。

#### [0054] (三) 插件式开发的实现

[0055] 1、定义基础扩展点。

[0056] class IExtension {

[0057] public:

```
[0058] IExtension(const char*version = " V 1.0" );
[0059] virtual ~IExtension() {}
[0060] virtual int Init(int argc, char*argv[]) ;
[0061] virtual int Finish() ;
[0062] virtual void SetVersion(const char*version) ;
[0063] virtual const char*GetVersion() ;
[0064] virtual void ProcessDebugCommand(std::string&cmd, ::StringList&paras,
std::ostream&os) ;
[0065] private:
[0066] _STD_string m_strVersion ;
[0067] } ;
```

[0068] Init 根据进程启动参数初始化插件。Finish 用于在扩展点销毁前做清理工作，如关闭句柄和释放内存等。GetVersion 和 SetVersion 用于设置插件版本。ProcessDebugCommand 是一个支持命令行调适的方法，类似基础功能可根据需要添加函数。

[0069] 插件基础扩展点实例创建函数 IExtension\*Create\_Extension(const char\*path, CProperties\*properties) 是插件动态库对唯一的导出函数，创建该插件基础扩展点的实例，path 是插件所在的目录路径，properties 是该插件属性的名值对，函数返回扩展点指针。

[0070] 插件的名称用 ‘.’ 分割字符串，如 ican.context.localservice.log 表示一个插件 id，ican.context.localservice 表示是一个 ILocalService 对应的插件，通过对插件 id 加扩展的方式表示其子插件，如 ican.context.localservice.log.xxplg 表示 ican.context.localservice 的子插件，这种 id 有点长，可以采用别名方式来简化插件 id，如 log 可以作为 ican.context.localservice.log 的别名。

[0071] 插件描述文件用 xml 文件定义，文件中定义插件 ID、版本、对应 dll 的名称等，也可以约定某种类型得子插件由其父插件来加载。

[0072] 2. 定义本地服务提供者接口 ILocalService、服务接口 IServant 和 IServant 的本地或远程代理 IService 接口，这些接口派生自 IExtension，这些插件实现了网管的具体功能，因此这些插件在数量上有很多，如果需要还可以定义一些其它插件接口。下面给出了这三个接口的主要方法：

```
[0073] 2.1 ILocalService 对本进程模块提供服务：
[0074] virtual void PreInit(IContext*ctx)
[0075] virtual IDefaultContext*GetCtx(void)
[0076] 其中，PreInit 用于设置上下文，GetCtx 用于获得上下文。
[0077] 2.2 IServant 对本进程或远程进程提供服务，服务提供主要通过 Get、Set 来实现：
[0078] virtual void PreInit(IContext*ctx)
[0079] virtual CMsg*Get(CMMsg&msgbuf, CMMsgHead&headinfo)
[0080] virtual void Set(CMMsg&msgbuf, CMMsgHead&headinfo)
[0081] PreInit 用于设置上下文，Get 是有应答报文处理函数，CMsg 表示报文，headinfo 表示命令码和用户等报文头信息，Set 是无应答报文处理函数。
```

[0082] 2.3 客户通过 IService 实现对 IServant 的 Get 和 Set 的调用，通过 IContext::HasServant 可以判断 servant 是否在本地。

[0083] virtual void PreInit(IContext\*ctx, const char\*servantName)

[0084] virtual void SetServant(IIcanServant\*servant)

[0085] virtual CMsg\*Get(CMsg&msgbuf, CMsgHead&headinfo)

[0086] virtual void Set(CMsg&msgbuf, CMsgHead&headinfo)

[0087] PreInit 用于设置上下文以及对应 IServant 的名称 servantName, SetServant 用于设置对应 IServant 的名称, Get 是有应答报文处理函数, Set 是无应答报文处理函数。

[0088] 3、定义插件分组，按产品功能分为多个 PPU，PPU 是顶层组，一个 PPU 下有多个 PMU，一个 PMU 下有多个 PMUFunction。PPU 对应一个目录，而 PMU 对应 PPU 下的一个子目录，一个插件功能组可以含有 PMU 的多个目录，每个目录下有多个插件或数据文件，这些插件功能组可能有交叉目录。PMUFunction 中还定义了对其它功能组的依赖关系。

[0089] 4、定义进程部署，在进程工作目录的文件中部署定义，该文件可以有多个，部署定义文件用于定义本进程部署哪些 PMUFunction。

[0090] 5、定义 IContext 和 IExtensionService 接口，这些接口可以派生自 IExtension。

[0091] 5.1 定义 IExtensionService 的方法：

[0092] virtual IExtension\*CreateExtension(const char\*extName, bool isSingleton) ;

[0093] virtual void DestroyExtension(IExtension\*pcExtension) = 0 ;

[0094] virtual StringList GetExtensions(const char\*pcSubject) = 0 ;

[0095] CreateExtension 按照是否以单例方式创建名为 extName 的基础扩展点实例。DestroyExtension 销毁扩展点。GetExtensions 获取主体 pcSubject 的所有子插件，即用‘.’分割的插件 ID 的部分前缀。

[0096] 5.2 IContext 在某个进程中创建和运行插件的上下文的方法：

[0097] virtual void PreInit(IExtensionService\*)

[0098] virtual ILocalService\*GetLocalService(const char\*extName, const char\*strpmufunc) ;

[0099] virtual IServant\*GetServant(const char\*extName, const char\*strpmufunc) ;

[0100] virtual IService\*GetService(const char\*extName, const char\*strpmufunc) ;

[0101] virtual IExtensionService\*GetExtService() ;

[0102] virtual bool HasServant(const char\*extName, const char\*strpmufunc) ;

[0103] PreInit 用于设置插件工厂。GetLocalService 获得插件功能组 strpmufunc 下的名为 extName 的 ILocalService, strpmufunc 为 NULL, 其中, 查找范围为本进程的所有 PMU。GetServant 获得插件功能组 strpmufunc 下的名为 extName 的 IServant。GetService 获得插件功能组 strpmufunc 下的名为 extName 的 IService。HasServant 判断本进程插件功能组 strpmufunc 是否存在名为 extName 的 IServant。GetExtService 获得插件工厂。

[0104] 6、ILocalSystem 派生于 ILocalService, 是本进程部署分析和插件组分析模块，

读取本进程属性数据文件中的名值对，分析本进程部署哪些插件组及插件组依赖树，在本实施方式中只要分析部署哪些功能组 PmuFunction 以及它们的依赖树，依赖树中叶子节点对父节点有依赖，然后从树根节点分层向叶子节点加载插件，同一个 PMUFunction 下的插件可以按目录或插件名定义先后加载顺序。约定某种原则在程序启动时加载某些类型插件，有些类型插件按需加载，如本实施方式中启动时加载具有 IServant 扩展点的插件。

[0105] 7、实现上面各接口功能或插件，发布软件。插件及 d11 按 PPU、PMU 分目录放在 Procs 目录下，PMUFunction 中含有那些目录和插件，每个功能组对应一个 xml 描述文件，在 Procs 同级别定义 works 目录，works 目录下是一些进程目录，存放与进程相关的一些数据文件，如进程部署文件和属性数据文件等。

[0106] 8、程序入口模块通过程序的命令行指定进程名称，通过进程名称知道进程的工作目录，这样进程入口模块可以加载不同的进程。创建 IContext 和 IExtensionService，传递进程工作目录给 ILocalSystem，ILocalSystem 负责按部署加载启动时需要的插件，运行时直接调用 IContext 按需加载插件。

[0107] 此外，可以根据本发明的技术方案的说明和具体实施方式做出各种可能的改变或替换，如 IContext 和 IExtensionService 合并，描述文件不采用 xml 格式，基础扩展点接口添加或减少一些方法，或是才用面向过程语言的结构及函数指针实现接口。

[0108] 综上所述，本发明具有以下技术效果：1、分组管理插件，便于大型软件对插件进行管理 2、可以通过文件定义插件 id 和类型。3、具有统一的插件基础接口，便于创建扩展点。4、支持分布式部署。5、进程工作目录和程序目录分离，便于多个进程共享程序文件。6、支持进程相关属性名值对，便于进程变量的定义。7、由于采用面向对象接口机制，有很高的抽象处理能力。此外，方法本身是通用的，不依赖于特定的操作系统和特定的面向对象语言，甚至在结构化语言中也能实现这种插件机制。

[0109] 尽管上文对本发明进行了详细说明，但是本发明不限于此，本领域技术人员可以根据本发明的原理进行各种修改。因此，凡按照本发明原理所作的修改，都应当理解为落入本发明的保护范围。

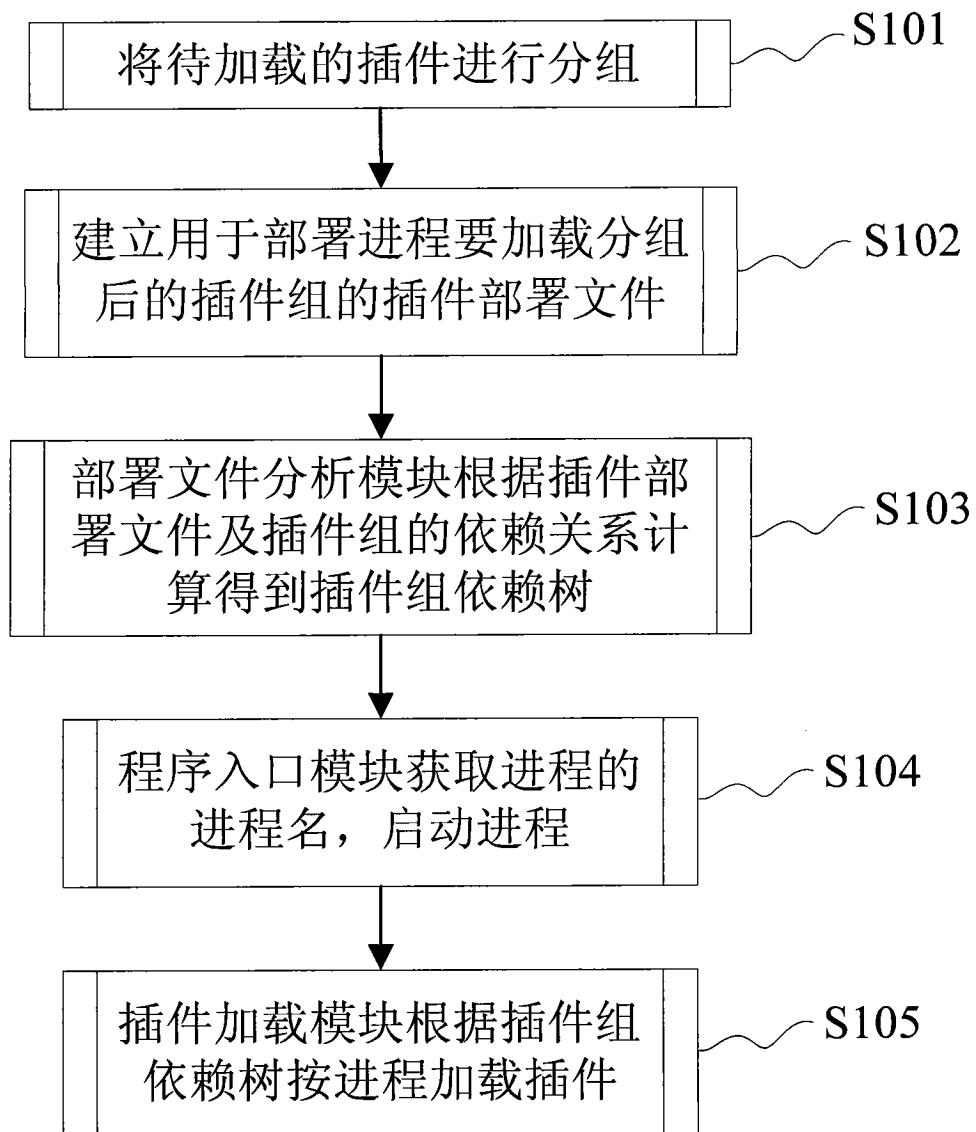


图 1

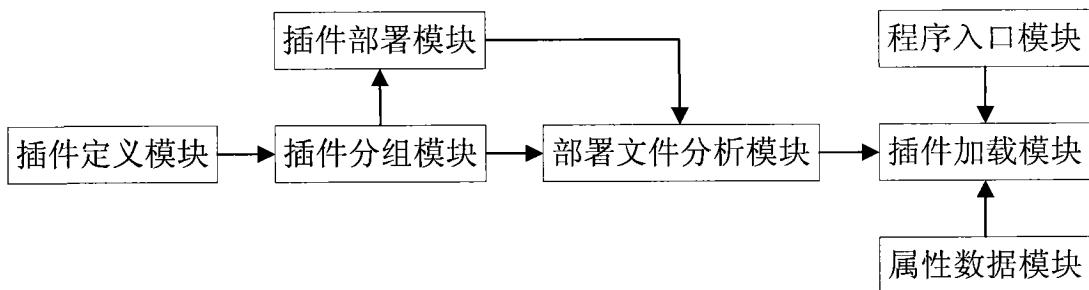


图 2

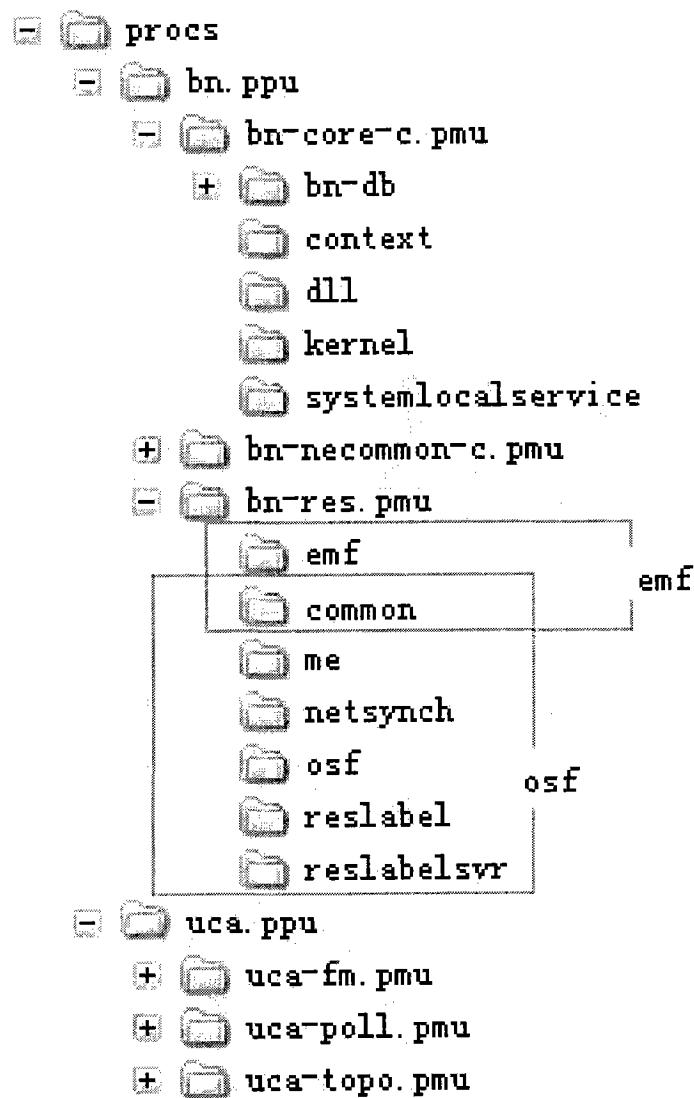


图 3

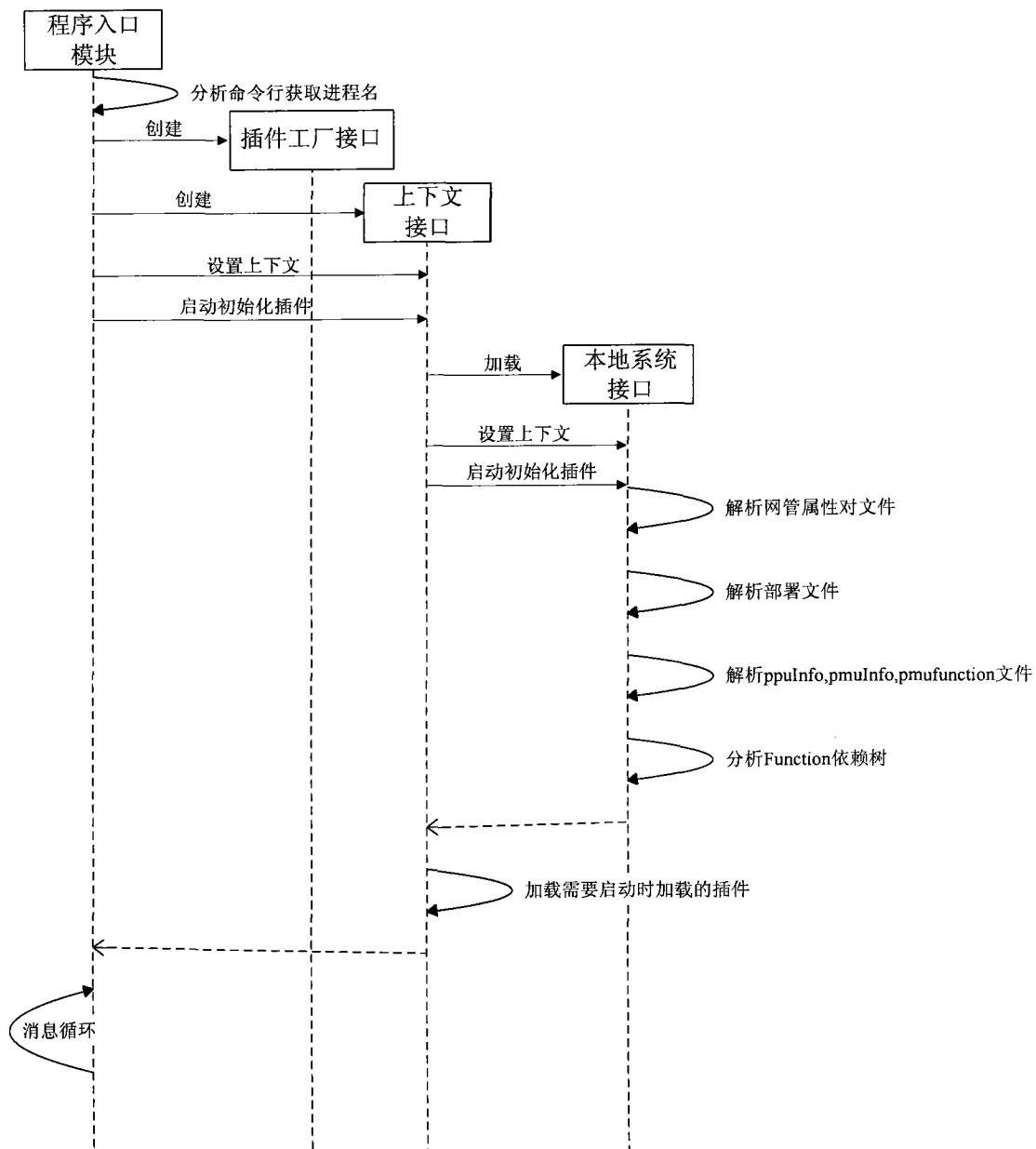


图 4

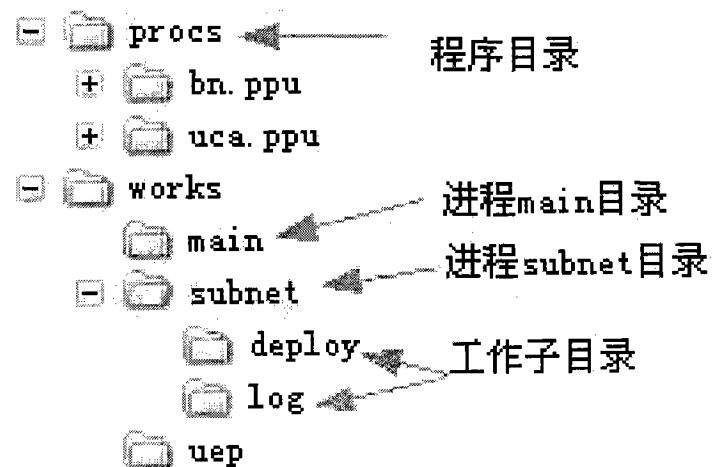


图 5