(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 103164239 A
(43)申请公布日 2013.06.19

(21)申请号 201210537056.1
(22)申请日 2012.12.11
(71)申请人 广东电网公司电力科学研究院
地址 510080 广东省广州市越秀区东风东路水均岗 8 号
申请人 天津天大求实电力新技术股份有限公司
(72)发明人 胡亚平 高雅 刘振国 陈炯聪 余南华 黄曙 陈皓 谢国财 刘菲 夏亚君 黄缙华 徐兴辉 刘玮 李双佑 田艳华
(74)专利代理机构 广州华进联合专利商标代理有限公司 44224
代理人 王茹 曾旻辉
(51)Int.Cl.
G06F 9/445(2006.01)

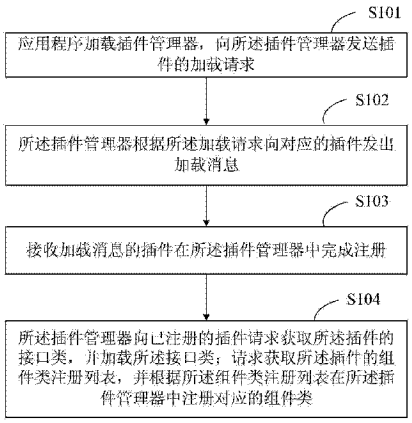
权利要求书1页 说明书6页 附图4页

(54)发明名称

基于微网可视化平台的插件实现方法

(57)摘要

本发明提供一种基于微网可视化平台的插件实现方法通过插件管理类加载插件，插件加载方式包括多个插件简单加载、一个插件简单加载、遍历目录下插件加载、静态库插件加载和通过配置文件对插件进行加载。插件中包括消息、命令、事件和接口功能，并通过上述功能实现菜单、工具条等界面交互和组件间功能调用。通过智能指针维护组件生命周期，保证数据的健壮性、安全性和唯一性。提供了静态库插件加载功能，这样发布软件时只需要一个主程序文件，在内部则是由多个静态库模块组成，这样既能发挥插件的可重用优点，又能减少程序文件数量，能够使插件提供跨平台支持，提高程序的复用性。适用于解决微网可视化平台与各个不同子系统插件的耦合。



CN 103164239 A

1. 一种基于微网可视化平台的插件实现方法，其特征在于，包括以下步骤：

应用程序加载插件管理器，向所述插件管理器发送插件的加载请求；

所述插件管理器根据所述加载请求向对应的插件发出加载消息；

接收所述加载消息的插件在所述插件管理器中完成注册；

所述插件管理器向已注册的插件请求获取所述插件的接口类，并加载所述接口类；请求获取所述插件的组件类注册列表，并根据所述组件类注册列表在所述插件管理器中注册对应的组件类。

2. 如权利要求 1 所述的基于微网可视化平台的插件实现方法，其特征在于，所述插件的接口类由纯虚函数组成，并且具有一个由 GUID 字符串定义的常量名作为组件类 ID。

3. 如权利要求 2 所述的基于微网可视化平台的插件实现方法，其特征在于，所述插件的组件类从所述接口类继承，实现组件类中定义的接口函数，并且所述组件类的构造函数为保护型。

4. 如权利要求 1 所述的基于微网可视化平台的插件实现方法，其特征在于，所述应用程序加载插件管理器的步骤包括：

所述应用程序检验插件管理类是否加载，如果没有加载，则加载所述插件管理类并获取所述插件管理类的指针，如果已加载则直接获取所述指针。

5. 如权利要求 1 所述的基于微网可视化平台的插件实现方法，其特征在于，所述插件管理器加载插件的方式包括多个插件加载、单个插件加载；遍历目录下所有插件加载、静态库插件加载和通过配置文件对插件进行加载。

6. 如权利要求 1 所述的基于微网可视化平台的插件实现方法，其特征在于，在所述插件管理器中注册对应的组件类之后进一步包括一个初始化的步骤，具体包括：

初始化消息、命令、接口和事件，并通过系统定义消息在平台中扩展菜单和工具栏。

7. 如权利要求 6 所述的基于微网可视化平台的插件实现方法，其特征在于，在收到应用程序关闭消息时，所述插件管理器则按照插件加载顺序逆序向各个插件发送释放插件消息，各个插件收到所述释放插件消息后释放内存，向所述插件管理器返回释放成功消息，所述插件管理器收到所述释放成功消息后向平台返回该消息。

8. 如权利要求 1 所述的基于微网可视化平台的插件实现方法，其特征在于，进一步包括以下步骤：

加载事件管理器；

各个所述插件接收所述插件管理器发送的初始化消息，向所述事件管理器注册相应的事件和响应函数；

当所述事件被触发时，直接受事件驱动的插件向所述事件管理器发送数据和通知消息；

所述事件管理器根据所述通知消息，向已注册的各个插件发送数据，各个所述插件接收所述数据后执行相应的响应函数进行处理。

基于微网可视化平台的插件实现方法

技术领域

[0001] 本发明涉及电力系统的微网可视化平台应用的技术领域，特别是涉及一种基于微网可视化平台的插件实现方法。

背景技术

[0002] 插件技术的出发点主要是为了使软件能够灵活地进行扩展功能，而不需要对主程序(框架程序)进行重新编译和发布。软件的功能由框架程序和插件来决定。在框架程序和插件之间具有一个特定的接口，它们两者须通过这个接口来传递数据或控制信息，以实现软件功能。

[0003] 目前的插件多为单一环境运行，不提供操作系统跨平台支持。同时，插件多为定制开发，与框架耦合度高，不利于程序、功能的复用，同时各插件之间通讯方式繁琐，开发工作复杂，容易出现错误。

发明内容

[0004] 针对上述背景技术中存在的问题，本发明的目的在于提供一种基于微网可视化平台的插件实现方法，能够使插件提供跨平台支持，提高程序的复用性。

[0005] 一种基于微网可视化平台的插件实现方法，包括以下步骤：

[0006] 应用程序加载插件管理器，向所述插件管理器发送插件的加载请求；

[0007] 所述插件管理器根据所述加载请求向对应的插件发出加载消息；

[0008] 接收所述加载消息的插件在所述插件管理器中完成注册；

[0009] 所述插件管理器向已注册的插件请求获取所述插件的接口类，并加载所述接口类；请求获取所述插件的组件类注册列表，并根据所述组件类注册列表在所述插件管理器中注册对应的组件类。

[0010] 本发明的基于微网可视化平台的插件实现方法中，通过所述插件管理器对各个插件进行管理，在应用程序加载插件时，可以通过所述插件管理器进行一个插件加载、多个插件加载、遍历插件目录加载等多种插件加载方式。并且各个插件在所述插件管理器中完成注册后，所述插件管理器先记载所述插件的接口类，再加载所述插件的组件类，则各个插件之间可以通过所述接口类完成插件之间的交互功能，插件间可通过特定方式通讯、调用、交互。提供了静态库插件加载功能，这样发布软件时只需要一个主程序文件，在内部则是由多个静态库模块组成，这样既能发挥插件的可重用优点，又能减少程序文件数量，能够使插件提供跨平台支持，提高程序的复用性。

附图说明

[0011] 图 1 是本发明基于微网可视化平台的插件实现方法的流程示意图；

[0012] 图 2 是本发明的基于微网可视化平台的插件实现方法中事件驱动的流程图；

[0013] 图 3 是本发明的一个实施例中插件加载过程的流程示意图；

[0014] 图 4 是本发明的一个实施例中事件驱动的执行流程。

具体实施方式

[0015] 请参阅图 1，图 1 是本发明基于微网可视化平台的插件实现方法的流程示意图。

[0016] 所述基于微网可视化平台的插件实现方法，包括以下步骤：

[0017] S101，应用程序加载插件管理器，向所述插件管理器发送插件的加载请求；

[0018] S102，所述插件管理器根据所述加载请求向对应的插件发出加载消息；

[0019] S103，接收所述加载消息的插件在所述插件管理器中完成注册；

[0020] S104，所述插件管理器向已注册的插件请求获取所述插件的接口类，并加载所述接口类；请求获取所述插件的组件类注册列表，并根据所述组件类注册列表在所述插件管理器中注册对应的组件类。

[0021] 本发明程序框架与插件的耦合并同时实现了各个插件之间、平台和插件之间的相互调用，从而提高了系统的健壮性、稳定性、扩展性和程序的复用性，减少了代码开发量，方便了系统的复用、扩展。尤其适合于解决微网可视化平台与各个不同子系统插件的耦合的问题。

[0022] 其中，对于步骤 S101，通过所述插件管理器对相关组件类 ID 和组件类创建的接口函数指针进行管理。优选使用 Boost 函数库中的智能指针类 boost::share_ptr 对函数的生命周期进行维护管理。插件生命周期包括程序启动时插件的加载(构造)，程序运行中插件的调用和程序退出时插件的释放(析构)。在步骤 S101 中，程序首先校验插件管理类(PluginManager)是否加载，如果没有加载则加载所述插件管理类，并获取该类指针，如果已加载则直接获取到该类指针。

[0023] 所述插件管理类加载插件的方式有多种，包括多个插件简单加载、一个插件简单加载、遍历目录下插件加载、静态库插件加载和通过配置文件对插件进行加载。例如：

[0024]

```
// 多个插件文件名，以 NULL 结尾
const char* plugins[] = {
    "PluginManager", "plMyInterface", NULL
};
AutoLoadPlugins autoload(plugins, "plugins");
//加载指定目录下所有插件
loadPlugins(strPath);
```

[0025] 对于步骤 S102，所述插件管理器根据所述应用程序发送的加载请求向对应的插件发出加载消息；

[0026] 各个接收所述加载消息的插件在接收到所述加载消息之后，向所述插件管理器发送确认信息。

[0027] 对于步骤 S103，所述插件管理器对接收到所述加载消息的插件进行注册。

[0028] 对于步骤 S104，插件创建是整个插件框架的核心，各个插件和平台可通过接口实现插件间、平台和插件间功能调用。

[0029] 优选地，所述插件的接口类由纯虚函数组成，并且具有一个由 GUID 字符串定义的常量名作为组件类 ID。

[0030] 亦即，定义插件接口类时，所述插件接口类没有基类，没有显式的接口函数，没有成员变量，全部由纯虚函数组成。同时直接使用 GUID 字符串定义一个常量名，需要用到组件类 ID 时使用该常量名，不依赖于编译环境；例如：

[0031]

```
const string idInterface = "D29108AA-403E-4e71-831C-384E2C32DA27";
class IMyInterface
{
    virtual void DoSomething() = 0;
};
```

[0032] 所述插件的组件类优选从所述接口类继承，实现组件类中定义的接口函数，并且所述组件类的构造函数为保护型。

[0033] 亦即，定义所述插件的组件类时，所述插件的组件类从接口类继承，实现组件类中定义的接口函数。为了针对组件编程将类的构造函数设为保护类型。一个组件类可以实现多个接口，组件类可以被继承。

[0034]

```
#include "MyInterface.h"
class CMyObject : public IMyInterface
{
protected:
    CMyObject ();
    virtual ~CMyObject ();
    virtual void DoSomething();
};
```

[0035] 所述插件管理器加载所述接口类和对应的组件类，完成对插件加载。

[0036] 优选地，在完成插件的加载之后，进一步包括一个初始化的步骤，具体包括：

[0037] 初始化消息、命令、接口和事件，并通过系统定义消息在平台中扩展菜单(Menu)和工具栏(ToolBar)。

[0038] 当系统收到应用程序关闭消息时，所述插件管理器则按照插件加载顺序逆序向各个插件发送释放插件消息，各个插件收到所述释放插件消息后释放内存，向所述插件管理器返回释放成功消息。当插件管理器收到所有返回成功消息后向平台返回消息，系统可关

闭。

[0039] 本发明中“消息”实现了平台对一个插件、平台对多个插件、一个插件对平台之间的调用，插件间的相互调用采用“接口”进行。本发明中对“接口”进行扩展提出“事件(Event)”，实现多个插件的函数回调以实现一个插件对多个插件的调用，下面介绍事件驱动模型：

[0040] 第一：事件驱动的原理。

[0041] 定义事件类型主要是定义唯一的字符串以便区分不同的事件，同时为了在编译阶段识别格式错误，自动定义了内部类型。注册响应函数是将多个相同格式的响应函数(回调函数)记录下来，同时关联到一个事件类型字符串。触发事件就是根据事件类型字符串找到对应的响应函数，按注册先后顺序依次调用这些响应函数(同步调用方式)，将事件参数传到响应函数的形参。本发明通过 Boost 库中 Signal2、Function 和 Bind 类库组合实现。

[0042] 第二：定义事件类型。

[0043] 在一个头文件中可定义多个事件类型，例如：

[0044]

```
#include <Triggerevent.h>

// void login(string strUser)

DEFINE_EVENT_1(EventLogin, string, "login.plg");

// void login (string strUser, string strPwd)

DEFINE_EVENT_2(EventLoginEx, string, string, "login.plg");

// bool login (string strUser)

DEFINE_EVENT_1Break(EventLoginDemo, string, "login.plg");
```

[0045] 在该文件中首先包含 Triggerevent.h，然后使用 DEFINE_EVENT_1 等宏定义每一个事件类型，使用格式如下：

[0046] DEFINE_EVENT_0、_1、_2 对应的响应函数的返回值类型为 void，每个响应函数都将依次调用。

[0047] DEFINE_EVENT_0(事件名称，事件名称标识后缀)；

[0048] DEFINE_EVENT_1(事件名称，形参类型，事件名称标识后缀)；

[0049] DEFINE_EVENT_2(事件名称，形参类型 1，形参类型 2，事件名称标识后缀)；

[0050] 事件名称将用于定义一个内部结构体和事件触发的辅助类(类名以 Trigger 开头)，在触发和响应事件时将使用该事件名称；形参类型为响应函数的形参类型，最多两个形参，可以使用普通类型(例如 char*、int)，或引用类型(例如 string&，const MyObj&)；事件名称标识后缀用于避免不同模块中出现相同的事件名称，内部将事件名称和标识后缀组合形成一个不重复的字符串常量，例如 "EventAdd.login.plg"、"EventAdd.login.plg"。

[0051] 第三：注册响应函数。

[0052] 使用静态函数来响应简单事件类型，函数的返回值类型和形参列表必须与事件类型的定义一致。可使用普通全局函数或类的静态函数，例如：

[0053] void InsertNodeFunc(int& result){...}

[0054] void MyClass::InsertNode(int& result){...}

[0055] 在响应函数所在模块的 InitializePlugin() 函数或其他函数中注册这些响应函数到对应的事件类型上。使用 REGISTER_OBSERVER 宏注册响应函数，例如：

[0056]

```
#include "myevents.h"
void registerHandlers()
{
    REGISTER_OBSERVER(EventAdd, InsertNodeFunc);
    REGISTER_OBSERVER(EventAdd, &MyClass::InsertNode);
}
```

[0057] 在响应函数所在模块的入口函数 CPP 文件(module.cpp 或 main.cpp，包含了 pluginimpl.h 或实现类 createObject 函数)中，需要包含 observerimpl.h 文件，在该模块中只能在一个 CPP 文件中包含 observerimpl.h 文件：#include<observerimpl.h>。

[0058] 因此，本发明的基于微网可视化平台的插件实现方法在完成插件加载之后，可进一步包括以下步骤，如图 2 所示：

[0059] S201，加载事件管理器；

[0060] S202，各个所述插件接收所述插件管理器发送的初始化消息，向所述事件管理器注册相应的事件和响应函数；

[0061] S203，当所述事件被触发时，直接受事件驱动的插件向所述事件管理器发送数据和通知消息；

[0062] S204，所述事件管理器根据所述通知消息，向已注册的各个插件发送数据，各个所述插件接收所述数据后执行相应的响应函数进行处理。

[0063] 本发明的基于微网可视化平台的插件实现方法适用于解决电力系统微网可视化平台与各个不同子系统插件的耦合。通过插件管理类加载插件，插件加载方式包括多个插件简单加载、一个插件简单加载、遍历目录下插件加载、静态库插件加载和通过配置文件对插件进行加载。插件中包括消息(Message)、命令(Command)、事件(Event)和接口(Interface)功能，并通过上述功能实现菜单(Menu)、工具条(ToolBar)等界面(UI)交互和组件间功能调用。通过智能指针(SharePtr)维护组件生命周期，保证数据的健壮性、安全性和唯一性。

[0064] 因此，具有以下优点：

[0065] 1) 接口定义简单灵活，采用普通的 C++ 接口，即由纯虚函数组成的结构体，不需要特殊的基类；同时可以使用 C++ 的各种变量类型，不受 COM 接口那样的约束。

[0066] 2) 接口与基于微网可视化平台的插件实现分离，向外部提供接口头文件，而在内部可以由一个组件来实现一个或多个接口，不需要对外导出该类或暴露实现细节。可以让多个模块并行开发，模块相互之间不存在编译依赖(不需要其他插件的 LIB 等文件)。

[0067] 3) 采用 Boost 智能指针来维护接口的生命周期，可从一个接口动态转换为另一个

接口，可以区分是来自于外部插件的引用还是来自于插件内部的引用。

[0068] 4) 组件透明部署，一个组件只需要使用其他组件的接口，不需要关心该接口是在哪个插件中实现的。可以根据需要将各个实现类进行合并或拆分，使其分布到不同插件中，而接口使用者不受影响。另外，插件部署于哪个目录也不影响插件接口的使用。

[0069] 5) 模块可替换、可扩展。可根据需要替换某个插件，只要该基于微网可视化平台的插件实现了相同的接口，即使内部功能不相同。这样就实现了插件可替换、按需组合的功能。通过在新的插件中支持更多的接口，可扩展更多的功能。可以在新插件中局部替换原有插件的某些接口或部分函数，实现重用和扩展。

[0070] 下面以具体的流程说明本发明的基于微网可视化平台的插件实现方法：

[0071] 请参阅图 3 和图 4，图 3 是本发明的一个实施例中插件加载过程的流程示意图；

[0072] 首先应用程序 Application 请求并加载插件管理器 LoadPlugInManager，加载完成后，发出加载插件的消息 LoadPlugIns。所述插件管理器 PlugInManager 根据所述加载请求向对应的 A 插件和 B 插件发出加载消息 LoadPlugIn；A 插件响应所述加载消息返回确认消息 LoadPlugInEx，所述插件管理器对 A 插件完成注册 RegisterPlugIn，请求获取所述插件的接口类 GetModuleInterface，并加载所述接口类 AddModule；请求获取所述插件的组件类注册列表 RegisterClassEntryTable、GetClassEntryTable，并根据所述组件类注册列表在所述插件管理器中注册对应的组件类 RegisterClass。完成加载后，发送初始化消息 InitPlugIn，进行初始化。

[0073] 图 4 是本发明的一个实施例中事件驱动的执行流程。

[0074] 所述插件管理器发送初始化消息 InitPlugIn 至各个插件，其中 C 插件为直接受事件驱动的插件，A 插件和 B 插件为受 C 插件回调的插件。A 插件和 B 插件收到所述初始化消息 InitPlugIn 之后，向事件管理器 ChangeManager 注册相应的事件和响应函数 RegisterObserver；

[0075] 当触发事件时，直接受事件驱动的 C 插件向所述事件管理器发送数据和通知消息 Data：：Notify()、ChangeNotify；所述事件管理器根据所述通知消息，向已注册的 A 插件和 B 插件发送数据 Updata()，所述 A 插件和 B 插件接收所述数据后执行相应的响应函数进行处理，响应所述事件 OnEventX()。

[0076] 本领域普通技术人员可以理解实现上述实施方式中的全部或部分流程，是可以通过计算机程序来指令相关的硬件来完成，所述的程序可存储于一计算机可读取存储介质中，该程序在执行时，可包括如上述各实施方式的流程。其中，所述的存储介质可为磁碟、光盘、只读存储记忆体（Read-Only Memory，ROM）或随机存储记忆体（Random Access Memory，RAM）等。

[0077] 以上所述实施例仅表达了本发明的几种实施方式，其描述较为具体和详细，但并不能因此而理解为对本发明专利范围的限制。应当指出的是，对于本领域的普通技术人员来说，在不脱离本发明构思的前提下，还可以做出若干变形和改进，这些都属于本发明的保护范围。因此，本发明专利的保护范围应以所附权利要求为准。

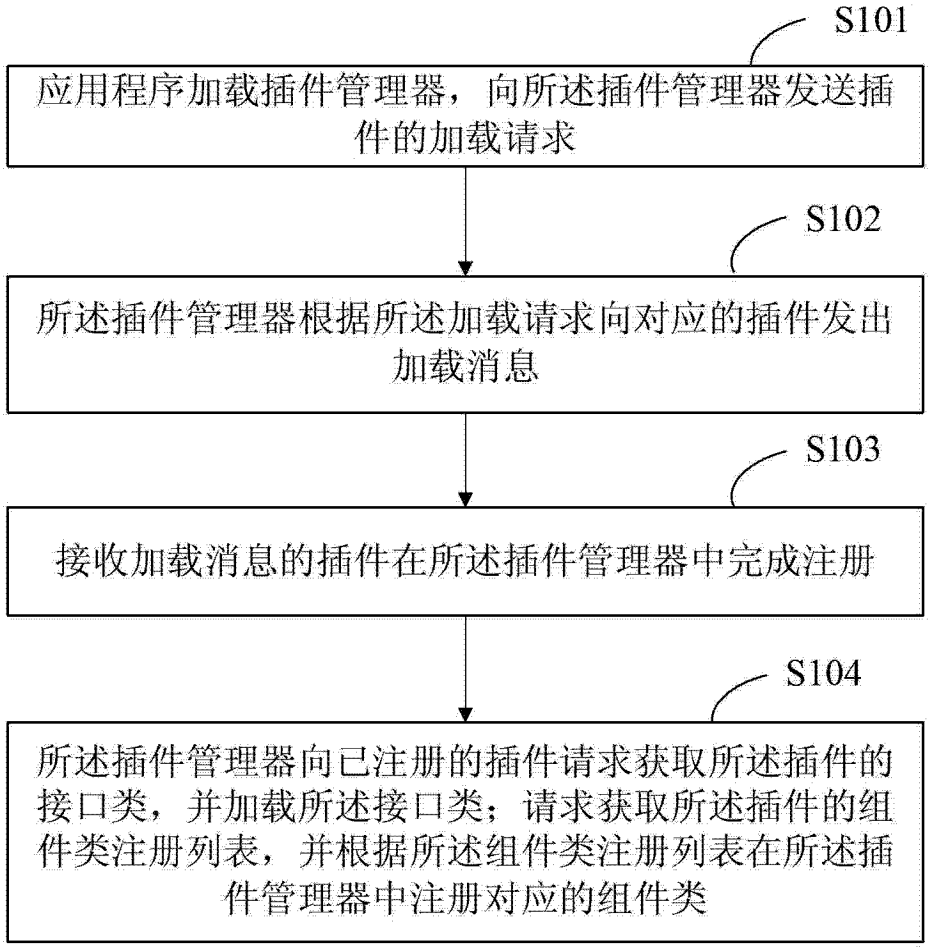

S101
应用程序加载插件管理器，向所述插件管理器发送插件的加载请求
S102
所述插件管理器根据所述加载请求向对应的插件发出加载消息
S103
接收加载消息的插件在所述插件管理器中完成注册
S104
所述插件管理器向已注册的插件请求获取所述插件的接口类，并加载所述接口类；请求获取所述插件的组件类注册列表，并根据所述组件类注册列表在所述插件管理器中注册对应的组件类

图 1

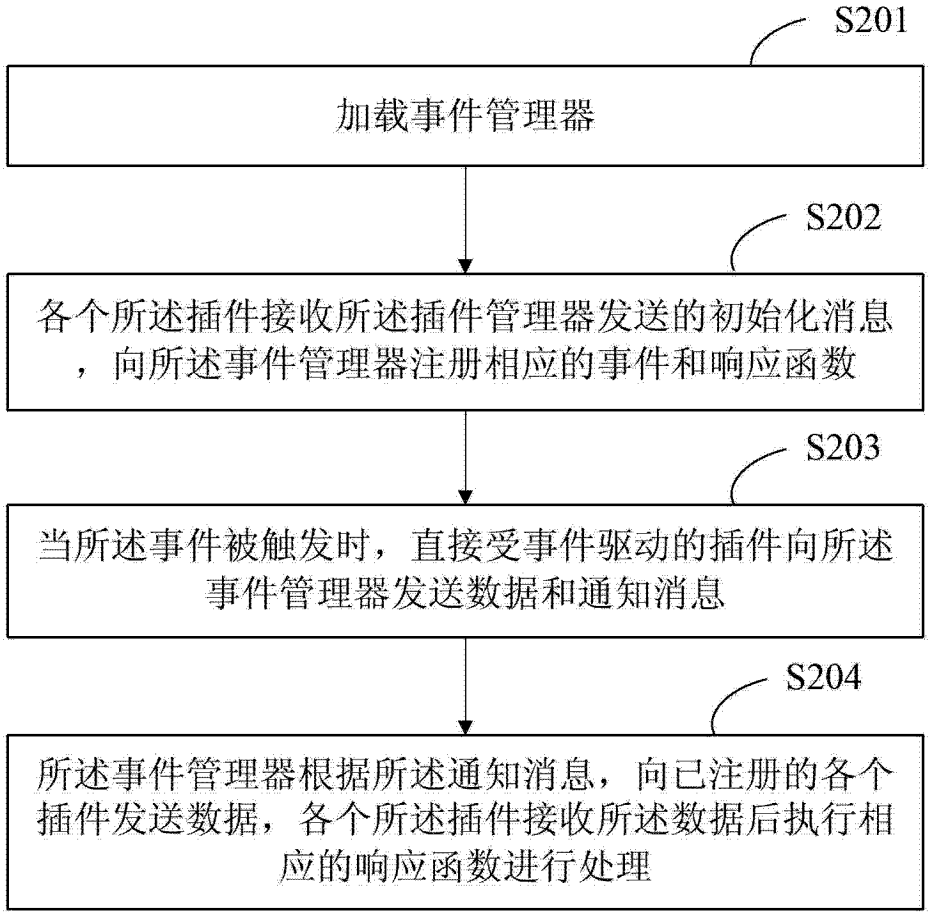

S201
加载事件管理器
S202
各个所述插件接收所述插件管理器发送的初始化消息，向所述事件管理器注册相应的事件和响应函数
S203
当所述事件被触发时，直接受事件驱动的插件向所述事件管理器发送数据和通知消息
S204
所述事件管理器根据所述通知消息，向已注册的各个插件发送数据，各个所述插件接收所述数据后执行相应的响应函数进行处理

图 2

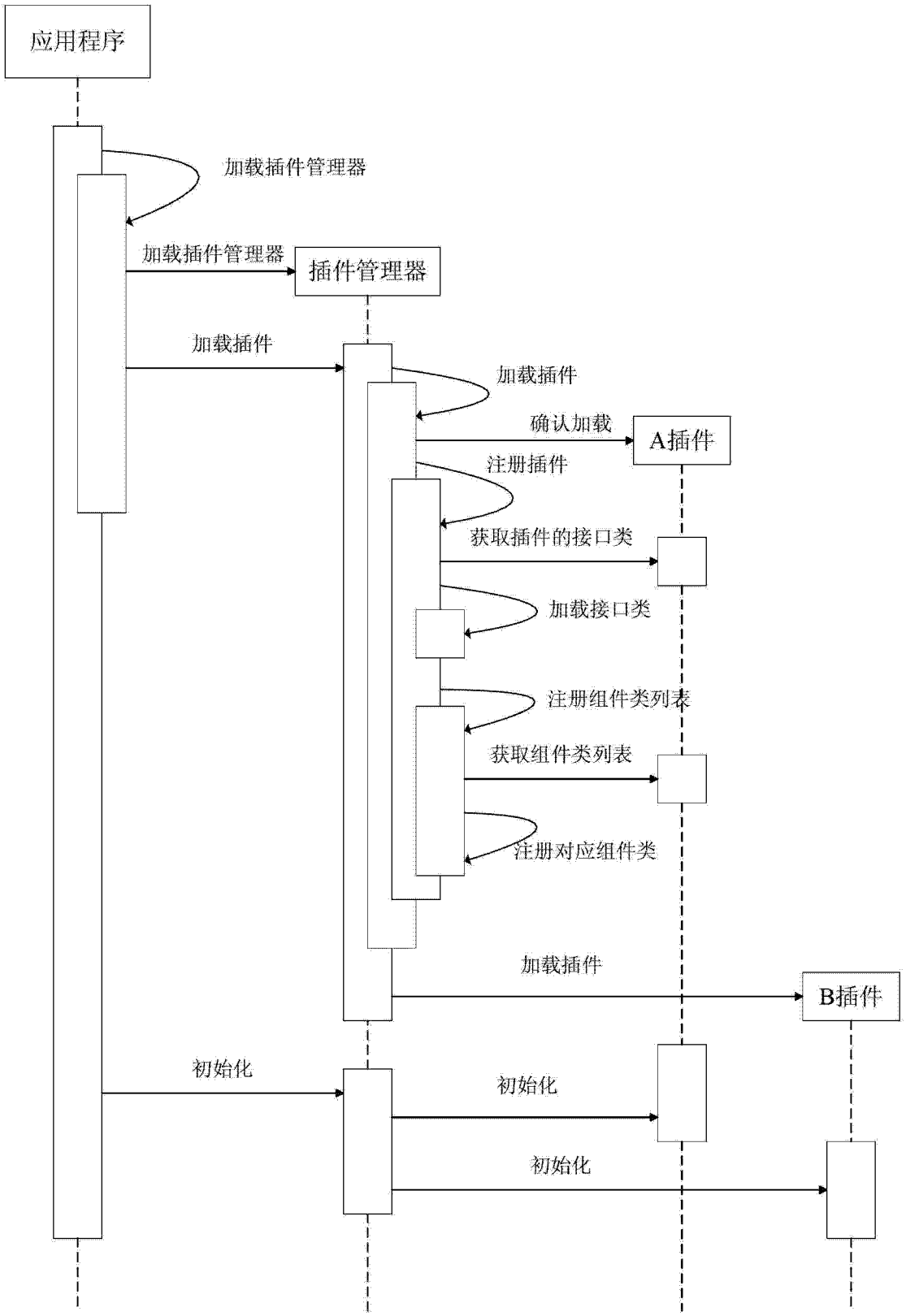

应用程序
加载插件管理器
加载插件管理器
插件管理器
加载插件
加载插件
确认加载
A插件
注册插件
获取插件的接口类
加载接口类
注册组件类列表
获取组件类列表
注册对应组件类
加载插件
B插件
初始化
初始化
初始化

图 3

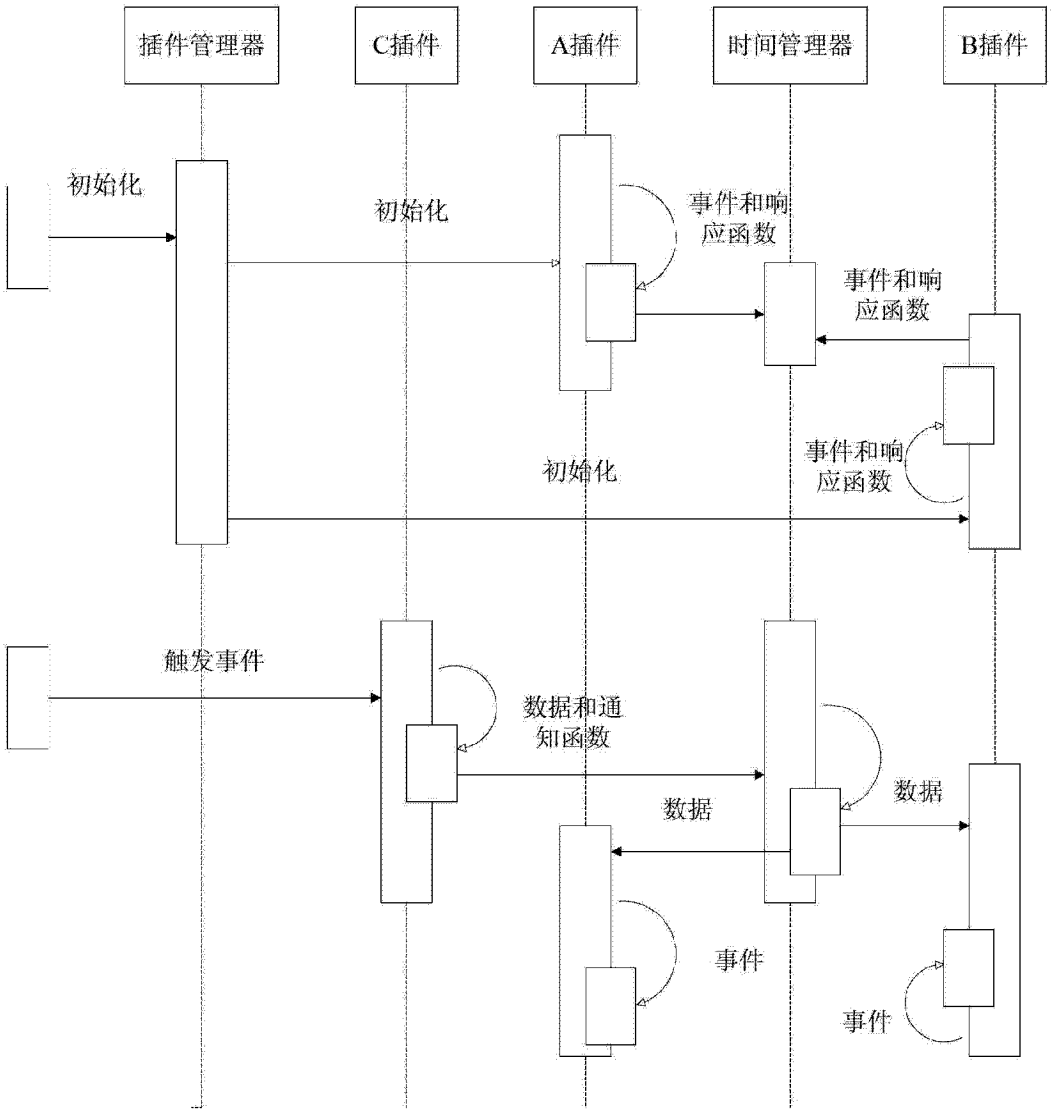

插件管理器
C插件
A插件
时间管理器
B插件
初始化
初始化
事件和响应函数
事件和响应函数
初始化
事件和响应函数
触发事件
数据和通知函数
数据
数据
事件
事件

图 4