(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 106293863 B
(45)授权公告日 2019.10.25

(21)申请号 201610599948.2

(22)申请日 2016.07.26

(65)同一申请的已公布的文献号
申请公布号 CN 106293863 A

(43)申请公布日 2017.01.04

(73)专利权人 北京北森云计算股份有限公司
地址 100085 北京市海淀区上地东路35号颐泉汇7层

(72)发明人 闫观涛 刘生权 张庆化 熊品卿 徐东

(74)专利代理机构 北京路浩知识产权代理有限公司 11002
代理人 李相雨

(51)Int.Cl.
G06F 8/41(2018.01)
G06F 9/451(2018.01)

(56)对比文件
CN 104657189 A,2015.05.27,
US 5621169 A,1997.04.15,
CN 102799493 A,2012.11.28,
CN 103150155 A,2013.06.12,
US 2015347163 A1,2015.12.03,
何腾飞.“面向方面程序并行化技术研究”.《中国优秀硕士学位论文全文数据库 信息科技辑》.2011,(第S2期),第I138-156页.
郭栋等.一种基于微服务架构的新型云件PaaS平台.《信息网络安全》.2015,(第11期),第15-20页.

审查员 屈姗姗

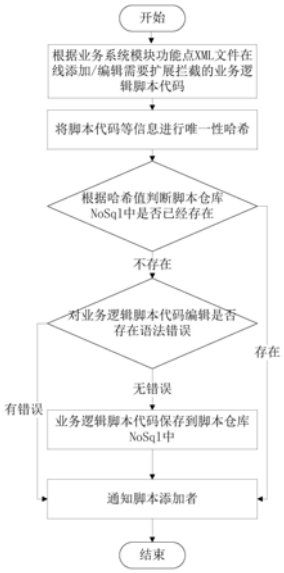
权利要求书2页 说明书14页 附图10页

(54)发明名称

多语言云编译实现系统功能动态扩展替换的方法及系统

(57)摘要

本发明公开了一种通过多语言云编译实现系统功能动态扩展替换的方法及系统,本发明根据拦截标签,在脚本存储库中存储对应于需要扩展的系统功能的业务逻辑拦截代码;本地编译时根据拦截标签在对应的位置添加处编译拦截装置代码,再根据拦截装置代码通过云编译再将对应的业务逻辑拦截代码进行编译并运行。本发明根据业务需求可以在线实时添加对应的业务逻辑拦截代码,从而可以在系统运行中通过云编译对业务逻辑拦截代码进行编译后直接引入并运行,对之前的代码毫无影响,实现了动态脚本引擎体系,继而实现了系统功能动态拦截扩展,提高了系统功能扩展的灵活性、开放性、可维护性以及可伸缩性。



CN 106293863 B

1.一种多语言云编译实现系统功能动态扩展替换的方法,其特征在于,所述方法包括以下步骤:

S1、根据业务需求,在需要扩展的方法上标识对应的拦截标签;

S2、对于每一个拦截标签,添加对应于需要扩展的系统功能的业务逻辑拦截代码;所述添加对应于需要扩展的系统功能的业务逻辑拦截代码的方式包括在线实时动态的添加方式;

S3、根据所述业务逻辑拦截代码生成标识码,并根据所述标识码判断脚本存储库中是否已经存在该业务逻辑拦截代码,其中所述脚本存储库中存储有所述业务逻辑拦截代码以及对应的标识码;

S4、若所述脚本存储库没有存储当前生成的业务逻辑拦截代码,则将当前生成的业务逻辑拦截代码保存到所述脚本存储库中;

S5、根据业务需求,本地编译时根据所述拦截标签在对应的位置处添加编译拦截装置代码,其中所述拦截装置代码用于织入对应的业务逻辑拦截代码,根据所述拦截装置代码通过云编译再将所述脚本存储库中存储的对应的业务逻辑拦截代码进行编译并运行。

2.根据权利要求1所述的方法,其特征在于,所述业务逻辑拦截代码包括以下信息:代码使用的语言种类、扩展拦截代码段、所述扩展拦截代码的用途、运行超时时间、执行错误处理方式、拦截方式、拦截级别和拦截作用于的客户。

3.根据权利要求2所述的方法,其特征在于,所述执行错误处理方式包括中断和继续运行。

4.根据权利要求2所述的方法,其特征在于,所述拦截方式包括同步拦截和异步拦截。

5.根据权利要求1所述的方法,其特征在于,所述步骤S3中生成所述标识码包括以下步骤:

对所述业务逻辑拦截代码进行虚拟类包装,并进行唯一性哈希,生成MD5字符串。

6.根据权利要求1所述的方法,其特征在于,所述步骤S4具体包括以下步骤:

若所述脚本存储库没有存储当前生成的业务逻辑拦截代码,则检查所述业务逻辑拦截代码是否存在语法错误,若不存在语法错误则将当前生成的业务逻辑拦截代码保存到所述脚本存储库中,若存在语法错误则提示操作者。

7.根据权利要求1所述的方法,其特征在于,所述步骤S1中,在同一个方法上添加多个所述拦截标签。

8.一种多语言云编译实现系统功能动态扩展替换的系统,其特征在于,所述系统包括拦截标签添加单元、拦截代码添加单元、拦截代码处理单元以及编译运行单元;

所述拦截标签添加单元用于根据业务需求,在需要扩展的方法上标识对应的拦截标签;

所述拦截代码添加单元用于对于每一个拦截标签,添加对应于需要扩展的系统功能的业务逻辑拦截代码;所述添加对应于需要扩展的系统功能的业务逻辑拦截代码的方式包括在线实时动态的添加方式;

所述拦截代码处理单元用于根据所述业务逻辑拦截代码生成标识码,并根据所述标识码判断脚本存储库中是否已经存在该业务逻辑拦截代码,其中所述脚本存储库中存储有所述业务逻辑拦截代码以及对应的标识码;若所述脚本存储库没有存储当前生成的业务逻辑

拦截代码，则将当前生成的业务逻辑拦截代码保存到所述脚本存储库中；

所述编译运行单元用于根据业务需求，本地编译时根据所述拦截标签在对应的位置处添加编译拦截装置代码，其中所述拦截装置代码用于织入对应的业务逻辑拦截代码，根据所述拦截装置代码通过云编译再将所述脚本存储库中存储的对应的业务逻辑拦截代码进行编译并运行。

9.根据权利要求8所述的系统，其特征在于，所述业务逻辑拦截代码包括以下信息：代码使用的语言种类、扩展拦截代码段、所述扩展拦截代码的用途、运行超时时间、执行错误处理方式、拦截方式、拦截级别和拦截作用于的客户。

10.根据权利要求9所述的系统，其特征在于，所述执行错误处理方式包括中断和继续运行。

多语言云编译实现系统功能动态扩展替换的方法及系统

技术领域

[0001] 本发明涉及软件动态扩展领域，更具体涉及一种通过多语言云编译实现系统功能动态扩展替换的方法及系统。

背景技术

[0002] 随着科学技术的快速发展，社会信息化程度的不断提高，计算机的应用已渗透到社会的各个领域，企业之间的竞争和多样化的客户需求，导致企业应用软件系统越来越庞大和复杂。客户对软件系统的个性化需求变化频繁，同时要求开发周期短，现实中通用的标准化的产品很难满足客户复杂的业务需求。系统开发完毕，已经满足用户的需求，但当用户不断提出新的要求时，原有的软件架构可能已经无法胜任新增任务的需求了，需要重新设计开发这个应用系统，严重浪费人力以及其他不必要的资源。即使重新开发了满足一个客户的定制化需求的应用系统，也不能同时保证用户的需求不会再次变更或有新的客户需求提出。因此如果软件系统的不断升级给客户做特殊的个性化定制，将给软件的质量、效率、管理带来严重问题。同时在原来的设计的基础上添加越多的代码，原有的代码会被不断腐蚀，系统的体系结构也将越来越难以维护，可见初期的架构设计只是满足了一部分的标准客户需求，并不能满足所有需求，企业的软件系统如果允许这种不可预料的变化开始，那整个软件系统的代码将不被控制。

[0003] 下面通过一个场景，举例说明客户要求的多变性：

[0004] 例：在某个在线招聘系统中，原来的逻辑是，当简历进入系统时，都是新添加一条记录，这样就会有大量重复的简历在系统中存在(邮箱、手机不同)，不利于面试官查看简历；现在客户A提出新的需求，当应聘者简历的信息自动进入系统时，系统需要依据应聘者的身份证号，来检测判断此应聘者是否之前在系统中是否存在，如果存在则将两个简历信息进行合并再保存，如果不存在则进行添加；而另外一个客户B的需求是，当系统进入简历时，根据手机号或邮箱判断，如果系统中已经存在此简历，则不再添加；客户C的需求是，当此简历在一段时间内频繁投递进入系统大于指定次数且符合职位所设置的基本条件，则自动给面试官发送一封邮件，表明此应聘者有强烈的入职愿望，面试官应给予关注；客户D的需求是，在简历进入系统时需要根据简历中的要素进行应聘者评分，比如985、211的学校加10分，英语6级加8分，英语4级加6分等，便于面试官进行综合排序筛选操作；客户E的需求是，需要根据客户的特定的自定义规则和特定条件自动筛选出符合条件的简历进而自动进入面试阶段或淘汰状态，如果进入面试阶段还需要给面试官发送短信和邮件安排面试，进行提醒等等。从以上案例可以看出，客户对于系统的业务扩展需求纷繁复杂，各不相同，如何以客户为中心，快速定制扩展开发出适应于客户的个性化软件系统，对于客户与企业至关重要。

[0005] 现有技术中，在系统中添加或扩展新的功能时，都是在对应的功能模块中使用软件开发工具直接添加代码逻辑，然后进行手工的重新编译、打包、部署，这种方法对于软件开发人员从某种程度上说工作量较大，工作效率低，而且由于是直接在现有系统模块中添

加代码，出错率较高，同时可能破坏了原系统的架构结构。虽然当前已经通过一定的技术Factory/IOC/AOP机制，在一定程度上解决了此问题，可以在系统在演变的过程中直接注入新的代码，而不改变原有系统的架构结构，但是这些方式还存在于种种局限性和不足，通过以下检索到的专利技术予以说明。

[0006] 1、专利号为CN 102708452 A的专利《一种支持动态编译的水质分析计算服务系统》，公开了以下技术特征：水质分析算法代码用户可以自行输入，然后通过调用动态编译服务器对文本代码进行动态编译，生成新的动态链接库或更新已存在的动态链接库，并实时更新水质分析算法库，以此来动态替换固定算法，达到扩展系统算法的目的，提高灵活性和开放性。此发明存在以下几个方面的不足：首先此发明所使用的是动态编译是微软CodeDOM ICodeCompiler编译方式，是利用.NET Framework中包含一个名为“代码文档对象模型”(CodeDOM)的机制，该机制使编写源代码的程序的开发人员可以在运行时，根据表示所呈现代码的单一模型，用编程语言文本生成源代码编译得到编译结果，它基本上是完全一个黑盒子，只能提供编译、编译后结果是成功还是失败；其次在每一次对代码进行编译时，都会在当前进程之外再启动另外一个新的csc.exe或vbc.exe编译进程，ICodeCompiler编译方式仅仅是对csc.exe和vbc.exe的包装，这种旧的方式，经过测试，如果在多个脚本代码同时编译时，CPU造成一定压力，使CPU占用率急剧上升，另外当前进程之外再启动新的进程去编译脚本代码，对本身对于一个系统的稳定性和性能，以及编译结果的输入输出有一定的影响。

[0007] 2、专利号为CN 101534324A的专利《基于DotNet平台的可插拔的Web服务动态调用方法》使用动态编译生成客户端代理类，根据配置文件决定是否需要调用远程Web服务，执行调用远程Web服务的业务逻辑方法，最后把方法执行的结果返回给软件系统，这在一定程度上降低了调用者与Web服务间的耦合度，实现具有可插拔性和动态调用。但是该发明中动态编译只应用在了客户端，用此来决定是否调用远程Web服务器的业务逻辑以及生成调用远程服务调用的动态代理，远程Web服务业务逻辑是固定的，如果系统需要扩展，还需要软件开发人员线下开发、手动编译、打包、部署。其次，在调用远程Web服务时，先读的配置文件，这个配置文件中只有远程服务是否打开的信息和单台远程Web服务器的消息，是否打开配置决定系统是否能够调用远程Web服务，远程Web服务器信息决定了系统要调用哪台服务器的Web服务，这里的配置只有一台服务器，而不是有多台服务器，这将不能实现将业务逻辑同时部署到多台服务器上，实现业务逻辑的负载均衡横向扩展及分布式部署，从而在一定程度上限制系统将不能适应多用户、高并发请求；再次该发明中所讲的WSProxy类是根据远程WSDL(Web Services Description Language)描述信息、生成客户端代理SProxy，并在内存中动态编译此代理类，这里使用的动态编译同样使用的也是CodeDOM ICodeCompiler编译方式、缺点与不足如上描述；再次该发明中使用的WebService传递协议是SOAP(Simple Object Access Protocol简单对象访问协议)，SOAP其内部原理是通过XML进行传递数据，由于XML较为冗长解析费时，相比于二进制文件来说要大很多，会消耗更多的网络资源和较多的CPU和内存资源，因此采用XML也成为一个弊端。

[0008] 3、专利号为CN 103823658 A的专利《一种基于桥架构配合反射动态技术》提供一种基于桥架构配合反射动态技术，方便于程序动态扩展灵活配置，可实现组件可插拔乃至代码级的灵活扩展变更。在此发明中第6步：引用动态反射，声明建立动态映射，从配置文件

读取值并对此赋值，从配置文件里读取要放射的类，抽象变化的基类。在这里有2个方面的不足：首先此发明使用了反射技术，反射技术实际上是一种解释的操作，在底层运行态内部需要做很多工作：绑定过程、元数据字符串比对、参数校验、类型校验，安全校验，生成大量临时对象等，会给CPU和GC垃圾回收器造成巨大负担，且在运行效率和性能上很低。其次，动态反射的业务逻辑是相对固定的，是属于系统外部的文件，而这些文件是需要软件开发人员手动开发、手动部署，从某种意义上不是真正的可插拔热部署，也没有实现代码级的灵活扩展。

[0009] 4、专利号为CN 104484182 A的专利《一种弹性可扩展的多数据源mvc模型架构》提供一种弹性可扩展的多数据源mvc模型架构，解决了在系统中同一个视图View界面上，可以同时灵活选择调用多种不同的数据源业务逻辑，用户只需要通过扩展增加相关配置和数据库表的映射代码，即可实现现有应用程序动态扩展多数据源和其他相关业务不易的问题。其内部原理使用是使用Spring的ioc模式对EB(EntityBean)层、Service层(业务逻辑层)、DAO层(持久化对象增删查改)及PO(对象关系型数据库的数据映射的持久化实体对象HibernateORM)进行无缝整合管控，通过配置文件动态组件实现。这其中有以下几点可以改进，首先此发明中使用了Spring的ioc模式，ioc可以通过代码和配置的方式添加，此发明是通过大量配置文件的方式进行数据源的扩展非常的繁琐，且这是需要人工手动进行配置，造成额外的维护成本，耦合关系全都在配置文件，并且只能只常规的关系型数库获取数据，其次这种由于使用IOC模式(反射)会产生性能问题，再次在Service业务逻辑层，如果需要对取出的数据源进行二次过滤加工处理，则需要人工线下开发、部署、配置。

[0010] 综合上面的陈述可知，现有技术中存在以下缺点：(1)扩展复杂：代码有直接耦合在现有系统中或经过反射配置到系统中，这使得整个扩展过程非常复杂繁琐，并且破还原有代码的结构；(2)代码重用率低：新扩展的代码逻辑只是为了当前所开发并耦合到系统中，这降低了代码的重用率；(3)性能损耗：通过反射或配置机制在系统中做扩展，会导致系统的性能下降一个级别。(4)开发周期长效率低：扩展或添加新的需求时，需要开发人员线下重新开发、编译、测试、上传、配置、打包、部署系统，这使得开发周期很长效率很低。(5)难以维护：系统开发人员和技术的应用有可能都已经迭代过无数次，系统中代码之间的互相调用、互相耦合也错综复杂，不是很了解系统的人不敢轻举妄动。随着系统的需求和功能越来越多，团队越来越大，相应的沟通成本、开发成本、管理成本都会成倍增加，当系统出现缺陷时，也会造成分析问题、定位问题、修复问题的时间更长；随着代码量的增加也有可能导致“修复越多，缺陷越多”的恶性循环；系统中一个小的改动就需要重新部署整个应用系统。(6)开发语言局限性：应用系统如果使用某种语言开发，那么再次需求扩展时则还必须已经此语言进行扩展，无法尝试新的编程语言或框架，这目前在现实中也不可能使用一个技术平台或方案来解决所有问题。(7)硬件成本高难以水平扩展：所有的程序代码在服务器的同一个进程中，会导致水平扩展难和成本高的局面，因为系统中不同的模块功能实际上对服务器的要求是不同的，有的是CPU密集计算机型，需要有性能较强的CPU处理器，有的是需要内存IO密集型，需要使用大量内存进行数据缓存，如果将系统所要求的服务器基础设施按同一种标准去采购，则硬件成本会非常高。(8)可靠性低：所有模块功能都运行在一个进程中，任何一个模块功能中的一个错误bug，比如内存泄露/CPU持续走高，将会有可能弄垮或拖慢整个进程，从而影响到整个应用的可靠性。(9)无法分布式部署支撑高并发：由于无法

将外部依赖的Dll或Jar与现有系统程序进行物理机进行有效的分离，随着系统的不断升级和扩展，这样庞大且复杂的系统，要伸缩必须为整体式的伸缩，无法有效分离式伸缩，系统中用户所需要经常操作的核心业务点可能只有30％，而现在却要将非核心的70％做同样的部署伸缩，不能达到资源的最大化集约使用。(10)影响持续集成和交付：由于业务系统的庞大复杂，加上无数次的扩展升级，所以构建一次的时间也会非常漫长，失败的次数也会大大增加，如果构建过程错误，排查问题也会非常困难。另外对系统做的任何一点细微的修改都需要将整个系统重新构建、重新部署。多个开发人员共同开发一个系统，还需要等待其他开发人员完成后才能部署，这大大降低了团队的灵活性和功能交付频率；(11)另外在项目打开和运行十分缓慢、系统难以开发、测试、持续集成困难、部署环境依赖复杂、技术栈绑定，编程语言、框架、数据库的升级和更换非常困难等等。

[0011] 如何应对这些问题，对软件系统的灵活性、开放性、可维护性、可伸缩性，以及软件系统对于市场和客户的反应能力都提出了更高的要求。如何快速地增加、扩展、升级当前标准的系统功能为客户量身定制，并且有效的保障提升软件开发质量和开发效率，已经成为软件行业的重要挑战。

发明内容

[0012] (一)要解决的技术问题

[0013] 本发明要解决的技术问题是如何提高系统功能扩展的灵活性、开放性、可维护性以及可伸缩性，在不对原有代码造成影响的基础上远程动态扩展系统功能。

[0014] (二)技术方案

[0015] 为了解决上述技术问题，本发明提供了一种多语言云编译实现系统功能动态扩展替换的方法，所述方法包括以下步骤：

[0016] S1、根据业务需求，在需要扩展的方法上标识对应的拦截标签；

[0017] S2、对于每一个拦截标签，添加对应于需要扩展的系统功能的业务逻辑拦截代码；

[0018] S3、根据所述业务逻辑拦截代码生成标识码，并根据所述标识码判断脚本存储库中是否已经存在该业务逻辑拦截代码，其中所述脚本存储库中存储有所述业务逻辑拦截代码以及对应的标识码；

[0019] S4、若所述脚本存储库没有存储当前生成的业务逻辑拦截代码，则将当前生成的业务逻辑拦截代码保存到所述脚本存储库中；

[0020] S5、根据业务需求，本地编译时根据所述拦截标签在对应的位置处添加编译拦截装置代码，其中所述拦截装置代码用于织入对应的业务逻辑拦截代码，根据所述拦截装置代码通过云编译再将所述脚本存储库中存储的对应的业务逻辑拦截代码进行编译并运行。

[0021] 优选地，所述业务逻辑拦截代码包括以下信息：代码使用的语言种类、扩展拦截代码段、所述扩展拦截代码的用途、运行超时时间、执行错误处理方式、拦截方式、拦截级别，拦截作用于的客户。

[0022] 优选地，所述执行错误处理方式包括中断和继续运行。

[0023] 优选地，所述拦截方式包括同步拦截和异步拦截。

[0024] 优选地，所述步骤S3中生成所述标识码包括以下步骤：

[0025] 对所述业务逻辑拦截代码进行虚拟类包装，并进行唯一性哈希，生成MD5字符串。

[0026] 优选地，所述步骤S4具体包括以下步骤：

[0027] 若所述脚本存储库没有存储当前生成的业务逻辑拦截代码，则检查所述业务逻辑拦截代码是否存在语法错误，若不存在语法错误则将当前生成的业务逻辑拦截代码保存到所述脚本存储库中，若存在语法错误则提示操作者。

[0028] 优选地，所述步骤S1中，在同一个方法上添加多个所述拦截标签。

[0029] 一种多语言云编译实现系统功能动态扩展替换的系统，所述系统包括拦截标签添加单元、拦截代码添加单元、拦截代码处理单元以及编译运行单元；

[0030] 所述拦截标签添加单元用于根据业务需求，在需要扩展的方法上标识对应的拦截标签；

[0031] 所述拦截代码添加单元用于对于每一个拦截标签，添加对应于需要扩展的系统功能的业务逻辑拦截代码；

[0032] 所述拦截代码处理单元用于根据所述业务逻辑拦截代码生成标识码，并根据所述标识码判断脚本存储库中是否已经存在该业务逻辑拦截代码，其中所述脚本存储库中存储有所述业务逻辑拦截代码以及对应的标识码；若所述脚本存储库没有存储当前生成的业务逻辑拦截代码，则将当前生成的业务逻辑拦截代码保存到所述脚本存储库中；

[0033] 所述编译运行单元用于根据业务需求，本地编译时根据所述拦截标签在对应的位置处添加编译拦截装置代码，其中所述拦截装置代码用于织入对应的业务逻辑拦截代码，根据所述拦截装置代码通过云编译再将所述脚本存储库中存储的对应的业务逻辑拦截代码进行编译并运行。

[0034] 优选地，所述业务逻辑拦截代码包括以下信息：代码使用的语言种类、扩展拦截代码段、所述扩展拦截代码的用途、运行超时时间、执行错误处理方式、拦截方式、拦截级别，拦截作用于的客户。

[0035] 优选地，所述执行错误处理方式包括中断和继续运行。

[0036] (三)有益效果

[0037] 本发明提供了一种通过多语言云编译实现系统功能动态扩展替换的方法及系统，本发明根据业务需求，在需要扩展的方法上标识对应的拦截标签；对于每一个拦截标签，添加对应于需要扩展的系统功能的业务逻辑拦截代码；之后根据业务逻辑拦截代码生成标识码，并根据标识码判断脚本存储库中是否已经存在该业务逻辑拦截代码，其中脚本存储库中存储有业务逻辑拦截代码以及对应的标识码；若脚本存储库没有存储当前生成的业务逻辑拦截代码，则将当前生成的业务逻辑拦截代码保存到所述脚本存储库中；最后根据业务需求，本地编译时根据拦截标签在对应的位置处添加编译拦截装置代码，其中所述拦截装置代码用于织入对应的业务逻辑拦截代码，根据拦截装置代码通过云编译再将所述脚本存储库中存储的对应的业务逻辑拦截代码进行编译并运行。本发明根据业务需求可以实时动态远程通过拦截标签以及拦截装置代码并运行对应的业务逻辑拦截代码，从而可以在系统运行中通过云编译业务逻辑拦截代码进行编译后直接引入并运行，对之前的代码毫无影响，实现了动态脚本引擎体系，继而实现了系统功能动态拦截扩展，提高了系统功能扩展的灵活性、开放性、可维护性以及可伸缩性。

[0038] 本发明通过设置拦截标签、拦截装置代码以及业务逻辑拦截代码，再利用动态脚本引擎体系是把软件开发中的模块功能细分，使开发者可以使用不同语言在线开发，动态

部署、动态织入，动态运行的方案，以减少重复代码编写，然后在系统运行需要时编译、加载、运行时并织入原有系统的框架，这套框架体系可以提高开发效率的编程思想。本发明中在线动态可扩展脚本的模块甚至可以交给第三方公司开发，提高行业内的合作，达到优势互补的目的。在开发完成后，脚本模块功能也可被其他程序系统所调用，能提高软件的可重用性。因此，本发明提出的通过拦截标签、拦截装置代码以及业务逻辑拦截代码实现的动态可扩展结构对于软件体系结构发展也具有广泛的实用意义。

[0039] 传统的软件开发方法越来越显示出其固有的局限性，而且要求在运行时改变其行为，支持动态扩展和升级。与传统的软件体系结构的一个程序集成所有功能不同，本发明能够将大部分扩展的功能放在外部远程服务器中，由于这种方法的方便和灵活性，能够通过插入动态脚本和改变外部动态脚本来实现软件功能的扩充、改进和提高，这也是本发明通过拦截标签实现动态脚本引擎体系结构开发的优势所在

附图说明

[0040] 为了更清楚地说明本发明实施例或现有技术中的技术方案，下面将对实施例或现有技术描述中所需要使用的附图作简单地介绍，显而易见地，下面描述中的附图仅仅是本发明的一些实施例，对于本领域普通技术人员来讲，在不付出创造性劳动的前提下，还可以根据这些附图获得其他的附图。

[0041] 图1是本发明中添加拦截标签的流程图；

[0042] 图2是本发明中添加业务逻辑拦截代码的流程图；

[0043] 图3是本发明中编译运行的流程图；

[0044] 图4是本发明中脚本存储库的结构示意图；

[0045] 图5是本发明中通过多语言云编译实现系统功能动态拦截扩展替换的装置示意图；

[0046] 图6是本发明中多语言动态编译/执行整体框图；

[0047] 图7是本发明中业务系统功能本地扩展模式结构图；

[0048] 图8是本发明中业务系统功能远程扩展模式结构图；

[0049] 图9是本发明中拦截标签添加位置示意图；

[0050] 图10A是传统系统的结构示意图；

[0051] 图10B是本发明中可扩展系统结构示意图；

[0052] 图11是本发明中多语言动态脚本代码列表示意图；

[0053] 图12是本发明中添加的业务逻辑拦截代码Rest的界面示意图；

[0054] 图13是本发明中添加业务逻辑拦截代码的界面示意图；

[0055] 图14是本发明中添加业务逻辑拦截代码的数据源代码界面的界面示意图。

具体实施方式

[0056] 下面结合附图和实施例对本发明作进一步详细描述。以下实施例用于说明本发明，但不能用来限制本发明的范围。

[0057] 一种通过多语言云编译实现系统功能动态扩展替换的方法，所述方法包括以下步骤：

[0058] S1、根据业务需求，在需要扩展的方法上标识对应的拦截标签；其中，在同一个方法上可以添加多个所述拦截标签；

[0059] S2、对于每一个拦截标签，添加对应于需要扩展的系统功能的业务逻辑拦截代码；其中业务逻辑拦截代码代码包括以下信息：代码使用的语言种类、扩展拦截代码段、所述扩展拦截代码的用途、运行超时时间、执行错误处理方式、拦截方式、拦截级别，拦截作用于的客户；所述执行错误处理方式包括中断和继续运行；所述拦截方式包括同步拦截和异步拦截；

[0060] S3、根据所述业务逻辑拦截代码生成标识码，并根据所述标识码判断脚本存储库中是否已经存在该业务逻辑拦截代码，其中所述脚本存储库中存储有所述业务逻辑拦截代码代码以及对应的标识码；

[0061] 其中，生成所述标识码包括以下步骤：对所述业务逻辑拦截代码进行虚拟类包装，并进行唯一性哈希，生成MD5字符串；

[0062] S4、若所述脚本存储库没有存储当前生成的业务逻辑拦截代码，则将当前生成的业务逻辑拦截代码保存到所述脚本存储库中；

[0063] S5、根据业务需求，本地编译时根据所述拦截标签在对应的位置处添加编译拦截装置代码，其中所述拦截装置代码用于织入对应的业务逻辑拦截代码，根据所述拦截装置代码通过云编译再将所述脚本存储库中存储的对应的业务逻辑拦截代码进行编译并运行。

[0064] 本发明根据业务需求可以在线实时动态添加对应的业务逻辑拦截代码，从而可以在系统运行中通过云编译对业务逻辑拦截代码进行编译后直接引入并运行，对之前的代码毫无影响，实现了动态脚本引擎体系，继而实现了系统功能动态拦截扩展，提高了系统功能扩展的灵活性、开放性、可维护性以及可伸缩性。

[0065] 进一步地，步骤S4具体包括以下步骤：

[0066] 若所述脚本存储库没有存储当前生成的业务逻辑拦截代码，则检查所述业务逻辑拦截代码是否存在语法错误，若不存在语法错误则将当前生成的业务逻辑拦截代码保存到所述脚本存储库中，若存在语法错误则提示操作者。

[0067] 对应于上述方法本发明还公开了一种通过多语言云编译实现系统功能动态扩展替换的系统，所述系统包括拦截标签添加单元、拦截代码添加单元、拦截代码处理单元以及编译运行单元；

[0068] 所述拦截标签添加单元用于根据业务需求，在需要扩展的方法上标识对应的拦截标签；

[0069] 所述拦截代码添加单元用于对于每一个拦截标签，添加对应于需要扩展的系统功能的业务逻辑拦截代码；

[0070] 所述拦截代码处理单元用于根据所述业务逻辑拦截代码生成标识码，并根据所述标识码判断脚本存储库中是否已经存在该业务逻辑拦截代码，其中所述脚本存储库中存储有所述业务逻辑拦截代码以及对应的标识码；若所述脚本存储库没有存储当前生成的业务逻辑拦截代码，则将当前生成的业务逻辑拦截代码保存到所述脚本存储库中；

[0071] 所述编译运行单元用于根据业务需求，本地编译时根据所述拦截标签在对应的位置处添加编译拦截装置代码，其中所述拦截装置代码用于织入对应的业务逻辑拦截代码，根据所述拦截装置代码通过云编译再将所述脚本存储库中存储的对应的业务逻辑拦截代

码进行编译并运行。

[0072] 本发明通过设置拦截标签、拦截装置代码以及业务逻辑拦截代码，在利用动态脚本引擎体系是把软件开发中的模块功能细分，使开发者可以使用不同语言在线开发，动态部署、动态织入，动态运行的方案，以减少重复代码编写，然后在系统运行需要时编译、加载、运行时并织入原有系统的框架，这套框架体系可以提高开发效率的编程思想。本发明中在线动态可扩展脚本的模块甚至可以交给第三方公司开发，提高行业内的合作，达到优势互补的目的。在开发完成后，脚本模块功能也可被其他程序系统所调用，能提高软件的可重用性。因此，本发明提出的通过拦截标签实现的动态可扩展结构对于软件体系结构发展也具有广泛的实用意义。

[0073] 上述方法或系统通过动态添加业务逻辑拦截代码的方式可以实现在线的而功能扩展、功能替换以及功能禁用等，方便灵活。

[0074] 下面通过一个具体的实施例对上述方法以及其他步骤进行详细的说明。

[0075] 如图1所示，本实施例的添加拦截标签的方法包括以下步骤：

[0076] (一)根据业务系统的需要或客户个性化定制需求的要求，确定在系统模块功能的所需要扩展的方法上通过IDE打上Attribute intercept拦截标签，其中这些拦截标签上的信息主要有：是否生成拦截装置代码，拦截枚举：方法前拦截、方法后拦截、方法前后拦截。拦截标签的位置，已经决定方法代码的所在模块、功能、方法名字空间、类信息，对需要扩展拦截的通用的核心业务预留出相应的接口。

[0077] 如图9所示，拦截位置可以根据实际情况选择具体为方法前拦截、方法后拦截或方法前后拦截。

[0078] 上面是通过打拦截标签的方式添加拦截点，也可以通过工具的方式可视化的将以上拦截标签信息添加到拦截配置文件中。添加的信息除了有以上信息，还有系统项目名称、中英文标识、模块名中英文标识、功能名中英文标识、方法名中英文标识。

[0079] (二)将以上拦截装置信息保存至XML信息中。

[0080] (三)通过IDE插件，在编译项目前读取拦截标签信息或XML文件信息。

[0081] (四)通过IDE插件，判断项目的每个类方法上或XML信息中是否有拦截标签信息。

[0082] (五)通过IDE插件，在有标识拦截标签信息的方法前后静态织入，即添加拦截装置代码，形成一种混合的代码。

[0083] 在使用IDE编译系统项目时，通过自定义开发的IDE插件，就会根据以上信息拦截标签信息或拦截配置文件，自动将拦截装置代码插入相应的位置。需要注意的是此Attribute拦截标签，在编译前只是用于判断添加的拦截装置代码在类中方法名位置，是否需要添加、拦截前还是后、采用同步还是异步，与反射reflection并无关系。

[0084] (六)再次检测项目中每个类每个方法，跳转至步骤四。

[0085] 上述实施例的方法利用动态脚本IDE插件模块、动态脚本代码拦截模块以及动态脚本代码请求通信模块完成，如图5所示，完成拦截标签的设置后，将拦截标签的设置信息通知到动态脚本代码拦截模块服务器端，进行记载。

[0086] 添加拦截标签以及业务逻辑拦截代码添加之后，继续进行业务逻辑拦截代码、编译运行就能实现系统功能动态拦截扩展，下面对业务逻辑拦截代码添加、编译运行的步骤进行详细说明。

[0087] 业务逻辑拦截代码添加具体包括如下步骤：

[0088] 如图2所示，包括如下步骤：

[0089] (一)根据业务系统模块功能点Xml文件在线添加/编辑需要扩展拦截的业务逻辑脚本代码，即业务逻辑拦截代码。添加的信息有：扩展拦截语言；扩展拦截代码段；扩展拦截代码的作用描述；扩展拦截方式：同步拦截、异步拦截；拦截代码执行超时时间；拦截代码执行错误处理方式：中断、继续运行；拦截级别：系统级拦截(及作用于系统中所有客户)、客户级拦截(只作用于使用系统的客户)；此扩展拦截代码所作用的客户Id(每个客户所需要的个性化业务逻辑是不同的)。

[0090] (二)将业务逻辑脚本代码进行虚拟类包装，并进行唯一性哈希、生成一个36位MD5字符串，能够唯一标识此脚本。唯一性哈希信息除了有业务逻辑脚本代码，还有系统标识、客户标识(系统购买者)，类方法名标识、所用语言。

[0091] (三)根据此唯一性哈希值判断NoSql脚本存储仓库(Redis)中是否已经存在。

[0092] (四)如果NoSql脚本存储仓库中不存在，则跳转到步骤(六)

[0093] (五)如果NoSql脚本存储仓库中存在，则跳转到(八)。

[0094] (六)如果没有编译语法错误，则业务逻辑脚本代码保存到NoSql脚本存储仓库。

[0095] (七)如果有编译语法错误，则跳转到步骤(八)

[0096] (八)根据上下文提示脚本添加者，已经存在于NoSql脚本存储仓库中或编译存在语法错误或添加编辑成功，跳转到步骤(一)，再次操作新的，对于自定义扩展拦截业务逻辑行为，在模块功能方法Method的前后，允许插入无限个拦截、拦截执行的顺序根据配置的顺序依次执行，也可以随时在添加后的扩展拦截列表中随时调整执行顺序或对之进行开启或停用、或再编辑替换。

[0097] 例如添加的业务逻辑拦截代码用于实现Rest服务，添加界面如图12所示，可将步骤简便快捷。添加业务逻辑拦截代码的界面如图13所示，添加业务逻辑拦截代码的数据源代码界面如图14所示。

[0098] 如图10A、10B所示，相比于传统的应用程序系统的固定的实现方式，即各个功能模块所提供的接口和实现都是之前定义好的模块，不能在程序的运行过程中根据一些条件的判断来动态地改变程序的行为或扩展添加新的功能。本发明能够在系统的运行时刻根据运行环境上下文来判断是否需要改变或扩展程序的行为。而在动态脚本引擎体系结构的程序中，动态脚本引擎管理程序和动态脚本之间的划分是非常清楚的，动态脚本引擎管理程序的动态脚本功能模板可以完全独立开发或第三方公司开发，这有利于在大规模软件开发过程中，使开发能够并行进行，提高开发的效率和质量。并且在开发和调试过程中，当部分动态脚本模块发生问题的时候，由于各个动态脚本插件模块之间相对独立，并不会影响到其他模块，将有利于软件测试工作的开展和进行，便于在测试和调试的过程中发现问题解决问题。另外，对部分动态脚本的修改，不会影响到其他动态脚本，避免了因为改正错误而引出新问题的情况，而且系统在运行时刻可以动态地替换组件，根据运行环境的变化或某些条件而引起新的行为，从而动态的提高软件系统的自适应性。

[0099] 业务逻辑拦截代码添加是利用动态脚本添加模块以及脚本代码扩展代码处理模块完成，如图5所示，符合要求的代码由动态脚本代码仓库模块存储入动态脚本代码仓库。

[0100] 拦截标签以及业务逻辑拦截代码添加好后可形成多语言动态脚本代码列表，如图

11所示。

[0101] 需要说明的是，拦截装置代码与业务逻辑拦截代码存在本质上的区别，业务逻辑拦截代码用于实现系统扩展的功能，可实时进行更改，通过远程服务器进行云编译来实现运行；而拦截装置代码只是一个壳，只是一段织入程序，具体来说是引导到需要远程扩展的方法上，进行本地编译，真正的编写的逻辑代码，即业务逻辑拦截代码不包括在这个拦截装置代码里面。例如这里的拦截装置代码如：RunDynamicScript(“SendSms”,“13011199191”,“你好这是短信”)；但发送短信的实际功能代码(即业务逻辑拦截代码)是在远程服务器。

[0102] 编译运行具体包括如下步骤，如图3所示：

[0103] 1)业务调用某模块功能方法，系统根据业务系统模块功能点扩展拦截参数从高速缓存Redis中判断是否存在扩展拦截并且为启用状态(只取系统级业务逻辑扩展和当前此客户的个性化业务逻辑扩展)。传递的扩展拦截参数主要有：业务逻辑脚本代码标识、系统标识、客户标识(系统购买者)，类方法名标识以及方法参数等上下文信息。

[0104] 2)如果检测存在扩展拦截，将系统标识、客户标识(系统购买者)，类方法名标识等信息传递，根据这些标识从脚本NoSql存储仓库读取扩展拦截脚本代码等信息，并做脚本代码唯一性哈希。这些信息包括：扩展拦截语言；扩展拦截代码；扩展拦截方式：同步拦截、异步拦截；拦截代码执行超时时间；拦截代码执行错误处理方式：中断、继续运行；拦截级别：系统级拦截、客户级拦截；

[0105] 3)根据扩展拦截脚本代码唯一性哈希，从检测内存是否已经编译好的脚本代码实例(不存在已经编译的实例，则有两可能，第一、首次运行、只编译一次，第二、扩展拦截脚本代码已经被编辑发生变化需要被替换)。动态脚本载入到框架系统并创建实例所遵守的原则是：在运行的时候再进行载入，不需要或者需要替换就把脚本模块卸载，这样可以减少内存的占用。

[0106] 4)如果内存中存在此扩展拦截脚本代码实例，则跳转6。

[0107] 5)如果内存中不存在此扩展拦截脚本代码实例，则跳转11。

[0108] 6)从内存读取定位已编译的内存实例。

[0109] 7)动态脚本执行引擎根据类方法名标识以及方法参数、所采用的语言等上下文信息，由多语言动态脚本执行引擎驱动运行实例同步运行指定函数方法(如果是异步运行，则在后台由其它线程队列处理运行)，传递协议采用thrift。

[0110] 8)如果运行出现异常或超过指定的时间，则跳转至9。

[0111] 9)系统异步记录运行日志&性能、耗时情况，异常信息返回。

[0112] 10)正常运行结果返回，跳转到12。

[0113] 11)动态脚本实例把计数器加1，并更新最后调用时间。

[0114] 12)系统重新检测是否存在扩展或拦截，跳转至1。多个扩展拦截脚本代码的执行结果最后一个直接返回，之前及中间的执行结果可以添加到系统特别提供的ApplicationContext当前执行线程数据槽(ApplicationContextDataSlot)，可以在进程中或远程来回中传递获取。

[0115] 上述执行过程适用于远程业务扩展，同时适用于本地业务扩展，如图7、8所示，由多语言动态脚本代码编译引擎装置和执行引擎装置完成编译和执行，其中运行实例存储在内存中，运行在脚本存在在脚本仓库中。扩展拦截装置用于实现拦截标签以及拦截代码的

添加。

[0116] 其中生成扩展拦截脚本代码实例包括以下步骤：

[0117] (1)从NoSql脚本存储仓库读取扩展拦截脚本代码，其中脚本仓库的构造如图4所示。

[0118] (2)将扩展拦截脚本代码载入多语言动态脚本编译引擎。不同的语言，将载入到不同的编译引擎中进行编译。

[0119] (3)加载扩展拦截语言所需要的外部关联模块和资源进行编译。复杂的业务逻辑，通常一段脚本代码是无法完成的，需要依赖于外部dll或jar等资源参与共同编译。

[0120] (4)将编译后的内存字节放置于内存并实例化做相应的初始化操作。

[0121] 上述编译运行方法由动态脚本代码拦截模块服务器端、动态语言管理引擎模块以及动态脚本实例运行调用模块配合完成，如图5所示，运行超时时间由动态脚本异常超时模块进行相应的处理。

[0122] 对动态脚本实例的调用有两种方式：事件调用和自动调用。事件调用指将某种特定的事件消息被触发，如用户的鼠标在点击某个功能点时，将此操作后面所调用的方法等扩展拦截信息传递给系统框架，系统判断调用相对应的动态脚本实例处理运行。自动调用则是系统可以根据系统本身的需要或用户的需要，在达到某个时间点时或某个条件时、或按一定的时间间隔(每秒/分钟/小时/周/月)，系统框架自动触发对动态扩展拦截脚本代码实例进行调用或自动在后台运行。系统框架对动态扩展拦截脚本代码实例的调用过程是：首先执行动态脚本实例，然后判断是否存在下一个扩展拦截点，如果扩展中又存在扩展点或有并行的其它扩展点，则继续执行下去，这样这些扩展拦截点就形成了一条的有机会扩展链条，从而扩展完成系统与客户所需要的不同功能。

[0123] 客户的个性化定制业务逻辑有时可能比较复杂，利用扩展拦截脚本代码业务逻辑的卸载机制，可以在一定程度上大大减少内存资源占用，具体卸载机制包括如下步骤：

[0124] 1)为每个动态脚本实例设置计数器、以及最后调用时间。

[0125] 2)调用一次动态脚本实例即把计数器加1，并更新最后调用时间。

[0126] 3)根据配置在指定的周期内定时检测计数器和最后调用时间，将小于指定调用次数的且大于指定时间一直未调用的动态脚本实例添加去除标识。

[0127] 4)系统将有去除的标识的动态脚本实例从内存中动态卸载，其中此步骤由动态脚本实例卸载模块或动态脚本实例替换模块完成，如图5所示。

[0128] 如图6所示，本发明的业务系统通过拦截标签的设置进行业务扩展点定义，之后在相应的扩展点添加业务逻辑拦截代码，并存储进脚本仓库。在运行时，由多语言脚本编译/执行引擎从脚本仓库下载脚本并控制关联语言外部关联模块进行多语言代码的编译并运行，完成业务系统功能的在线扩展。

[0129] 动态脚本引擎体系代表的是一个横向的关系，如果说“系统”是一个空心的圆柱体，其中封装的是对象的属性和行为；那么动态脚本引擎，就仿佛一把利刃，将这些空心圆柱体剖开，以获得其内部的消息。然后它又巧妙地将这些剖开的切面恢复原状。这种特性提高了开发效率，使变化造成的影响局部化，代码向模块化方向发展，进而降低耦合，达到软件开发维护简易化，软件迭代简化的目标。

[0130] 与现有技术相比，本发明的技术方案具有以下优点：(1)即时生效：由于是开线开

发、在线编译，所以开发编译后可立即生成服务，供应用系统直接立即调用，所有不需要开发人员重新编译，热部署无需重启系统；(2)增强代码可维护性：代码在线开发、在线调试、在线运行，如果遇到问题，可以在线修复，快速升级扩展或替换、启用或停用，实现动态可插拔。(3)降低业务的复杂度，迫使去合理划分系统边界，合理设计领域模型，使迭代的周期更短，可以多版本，灰度升级，服务降级，服务分布式跟踪等。(4)代码重用率高：动态脚本引擎把每个方面实现为独立的模块，模块之间是松散耦合的。松散耦合的实现通常意味着更好的代码重用性，可以将其移值到其它系统中直接调用，从而降低了软件的开发成本和难度，一次开发，即可在任何地方调用(远程/本地)。(5)性能高：动态脚本框架根据上下文调用需要，即时将脚本进行编译，编译后的实例放置于内存中，就像直接引用对象一样方便且高效，且有同步、异步(队列非阻塞)、定时多种调用处理方式。(6)多种语言支持：对于应用系统，如果需要扩展添加新的功能，不仅可以用现有语言开发，也可以用其它语言进行开发，并且可以跨平台运行。(7)易于管理：每个开发者都可以单独开发、集成、测试、部署、完成上线流程，有效减少运维成本和管理成本，提高开发效率。(8)技术创新：容易尝试技术创新，甚至每个功能扩展点都可以采用不同的编程语言编写。(9)稳定性增强：系统稳定性增强，单个服务的失效不会影响其他服务，可以一定程度实现服务降级。(10)易于开发：每个动态脚本扩展关注点都使用最小耦合来进行处理，将系统中的模块进一步细化，降低了模块之间的耦合度，不同专家和团队负责开发不同的模块，彼此间互不影响。实现快速敏捷开发。(11)代码集中易于理解。使用动态脚本引擎技术可以将需要多次使用的公用代码集中到一处实现，这样代码的修改范围就可以得到严格的控制，减少代码修改对系统稳定性的影响，减少代码的冗余度和耦合度，增强可读性，提高软件质量，解决了面向对象编程跨模块造成的代码混乱和代码分散问题。(12)系统容易扩展。很容易通过建立新的动态脚本加入新的功能来扩展业务逻辑规则。当向系统中加入新的模块功能时，已有的动态脚本自动横切进来，使系统易于扩展；在系统发布后可在不必重新编译原有系统的前提下按需对系统功能进行扩充，这样就可以开发扩展更多动态脚本，快速响应客户个性化需求变化。(13)分布式：实现将动态在线扩展的业务逻辑可以同时部署到多台服务器上，实现业务逻辑的负载均衡横向扩展及分布式部署，从而适应多用户、高并发请求。

[0131] 本发明使用微软新一代编译系统(Roslyn)，这是一个更为开放式的编译器，与以往不透明的编译过程不同，开发者可以在编译过程中访问和分析编译数据如解析代码，语义分析，本发明对此封装并进行再次扩展，借助编译器进行解析编译源代码文件、动态为编程语言增加功能、自定义编译功能动作等操作，扩展后并使之更加方便易用。与之相比所采用的Roslyn则是从更从底托管代码的基础上建立起来的，在同时编译多个脚本时，CPU占用率会相对更少，编译过程更稳定高效，性能更高，并且在编译过程中可以做更多的控制和扩展操作。

[0132] 本发明通过动态在线即时添加代码编译代码的方式可以更加快捷的实现，具体方案为可以只使用Controller的一个动作Action，在此Action中根据视图上传递过程的数据源关键参数，去动态触发执行各种不同关系数据库/非关系型NoSql数据源的脚本代码，获取数据并且还可以二次过滤加工，然后将最终的数据赋值到实体Bean上返回给视图呈现。

[0133] 本发明提供了一种全新的、更加高效的、即时的、快捷的在线编程编译、动态扩展拦截系统功能的可插拔的方法装置，当系统中某个功能在原有基础上需要扩展业务逻辑、

触发新的动作或新增加扩展新的功能时，可以在线编程、即时编译，动态扩展系统功能，这种功能扩展和拦截不仅可以是系统级别，也可以是客户级别扩展和拦截，另外无需线下重新开发上传，实现系统功能和资源的快速动态扩展和有效利用，并使之互相连接、相互支撑、相互协作，从而为客户提供价值，另外本发明还可以在线修正软件中的故障、调试、测试，快速生效，快速应用，给原有系统带来极大的灵活性、扩展性以及可维护性。

[0134] 本发明中动态脚本引擎技术，是把客户的每个需求(与业务相关的或无关逻辑)都映射为一段动态脚本代码，对系统中的扩展点进行动态扩展，在初次调用时，编译器把它翻译成可执行代码放置于本地/远程(远程动态脚本微服务)内存当中(这段代码可以用常见的编程语言C#、VB.NET、JAVA、Python等编写)，以此解决代码集中耦合的问题，提高软件重用性，降低核心模块的代码复杂度和重复代码的问题，进而降低项目的开发费用和成本；应用动态脚本的软件开发方法，还可以使系统在开发阶段各模块的职能更加明确，功能模板之间的耦合更为松散，开发的系统更具有可扩展性、维护性。构建在线编程即时编译引擎，可以动态的扩展增加原有系统的功能。当发现当前系统不能满足目前的需求时，只需要在线增加一个相应的动态脚本，不停止现有系统运行的情况下，对现有模块在线进行功能扩展或补充即可，而不用重新在线下开发，再次构造整个系统重新部署，从而实现敏捷业务扩展开发，即时满足客户个性化定制需求。

[0135] 动态脚本引擎体系对于软件运行时的扩展提供了很好的平台，业务动态脚本逻辑的指令集可以支持常见语言(C#/VB.NET/JAVA/Python)，语言编译器通过框架可以生成低级指令的对象代码，整个系统在字节码在首次运行时一次性织入，所织入的代码被动态编译成高效的原始代码，并放置于内存之中，大大提高了系统的执行速度。

[0136] 动态脚本引擎体系把软件开发中的模块功能细分，使开发者可以使用不同语言在线开发，动态部署、动态织入，动态运行的方案，以减少重复代码编写，然后在系统运行需要时编译、加载、运行时并织入原有系统的框架，这套框架体系可以提高开发效率的编程思想。在线动态可扩展脚本模块甚至可以交给第三方公司开发，提高行业内的合作，达到优势互补的目的。在开发完成后，脚本模块功能也可被其他程序系统所调用，能提高软件的可重用性。因此，本发明提出的动态可扩展结构对于软件体系结构发展也具有广泛的实用意义。

[0137] 在体系结构方面，传统的软件体系结构不利于大规模的集群化开发、部署，对现代软件产业的发展造成了极大的限制。采用动态可扩展的在线编程即时编译引擎体系结构，可以使各个功能模块具有更大的独立性，改善了软件的可测试性和可维护性。与传统的软件体系结构相比，在动态可扩展结构中，主程序完成各个动态可扩展模块之间的相互协调工作，而各个新扩展的模块之间是相对独立的，这使得程序的结构简单，各部分可以完全独立开发，独立部署、独立运行，有利于在大规模软件开发过程中提高开发的并行性和效率。动态脚本引擎体系结构将大部分扩展的功能放在外部远程服务器中，由于这种方法的方便和灵活性，能够通过插入动态脚本和改变外部动态脚本来实现软件功能的扩充、改进和提高，这也是动态脚本引擎体系结构开发方法的优势所在。

[0138] 发明为开发者提供了一种利用动态脚本快速增加扩展系统功能的方法，它允许自定义程序代码执行顺序的关系和执行逻辑，每个动态可扩展脚本都可以单独在线编程调试、升级维护、配置，可以根据需要动态编译、安装、载入、运行、停止、启用、停用、卸载，既不影响其已有功能模块的工作，又具有极大的灵活性。

[0139] 本发明并不局限于应用软件所用的这一种语言，可以实现.Net、VB.NET、Java、Python在线编译服务，不需要像传统方式一样，使用开发软件VisualStudio或Eclipse在电脑本地进行开发，而是直接在线开发，在线动态编译、实现编译即服务并通过预留接口动态切入当前系统，实现原有功能的快速扩展。

[0140] 本发明并不局限于对原有系统的标准模块功能方法的系统级与客户级的动态脚本扩展，甚至也可以是用户级，也可以脱离单独提供动态脚本远程RPC服务供需要的系统直接调用，也可以生成在线Restful服务，供前后端开发人员直接调用(系统级指适用于所有客户的所有用户、客户级指只适用于当前客户的所有用户，用户级指客户级中个别或部分指定用户)。

[0141] 本发明并不局限于在标准的业务系统已有的功能方法上做本地动态扩展，如果您的业务系统是基于SOA/ESB/MicroService的构建模式，这样的动态扩展也可以直接移植融合到远程业务接口或添加到您需要扩展的地方，不同的动态扩展之间，可以互相调用，也可以调用外部其它接口，从而构造复合的动态组件，使之更加灵活多变。

[0142] 以上各实施例仅用以说明本发明的技术方案，而非对其限制；尽管参照前述各实施例对本发明进行了详细的说明，本领域的普通技术人员应当理解：其依然可以对前述各实施例所记载的技术方案进行修改，或者对其中部分或者全部技术特征进行等同替换；而这些修改或者替换，并不使相应技术方案的本质脱离本发明各实施例技术方案的范围，其均应涵盖在本发明的权利要求和说明书的范围当中。

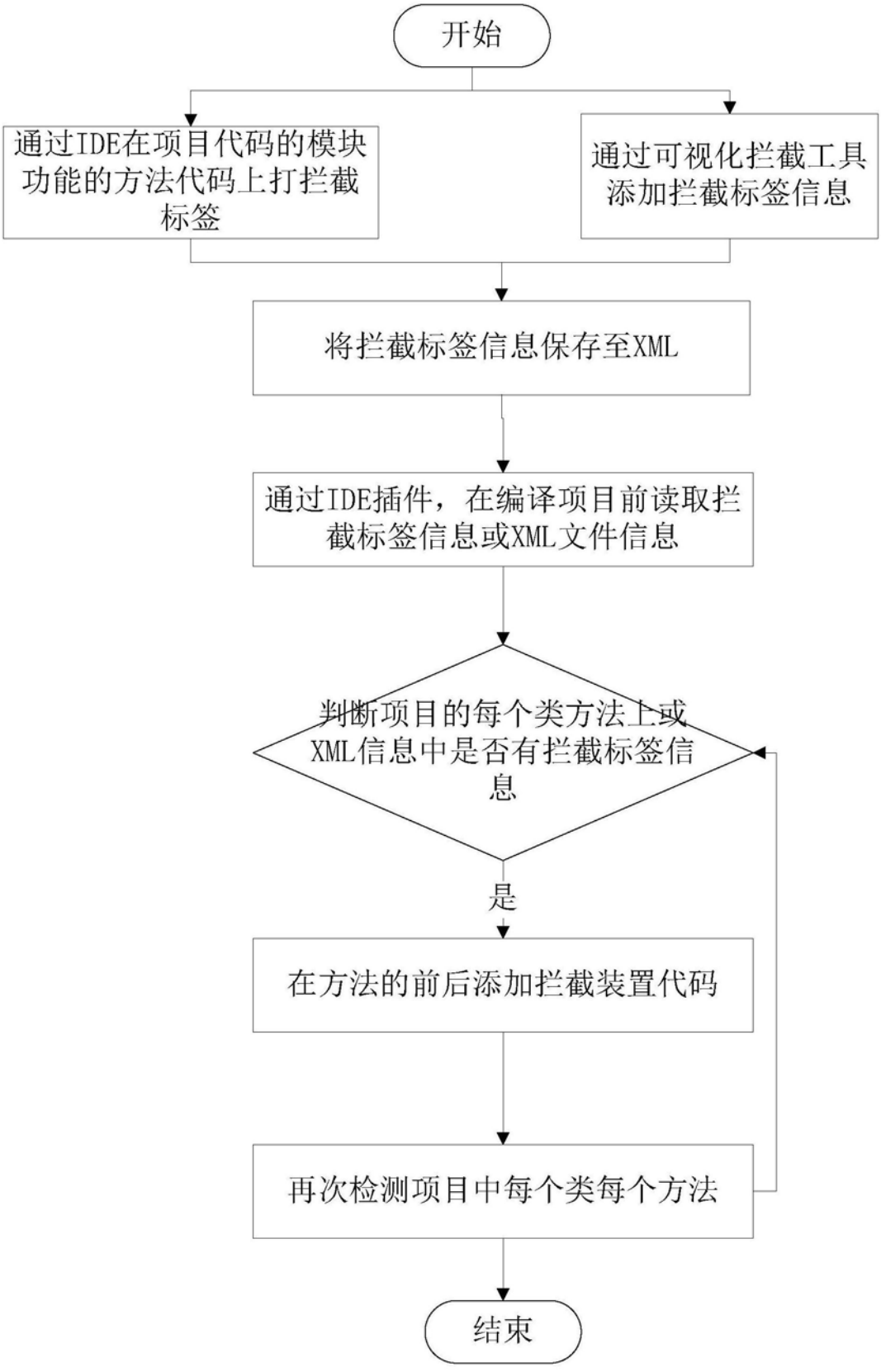

开始
通过IDE在项目代码的模块功能的方法代码上打拦截标签
通过可视化拦截工具添加拦截标签信息
将拦截标签信息保存至XML
通过IDE插件，在编译项目前读取拦截标签信息或XML文件信息
判断项目的每个类方法上或XML信息中是否有拦截标签信息
是
在方法的前后添加拦截装置代码
再次检测项目中每个类每个方法
结束

图1

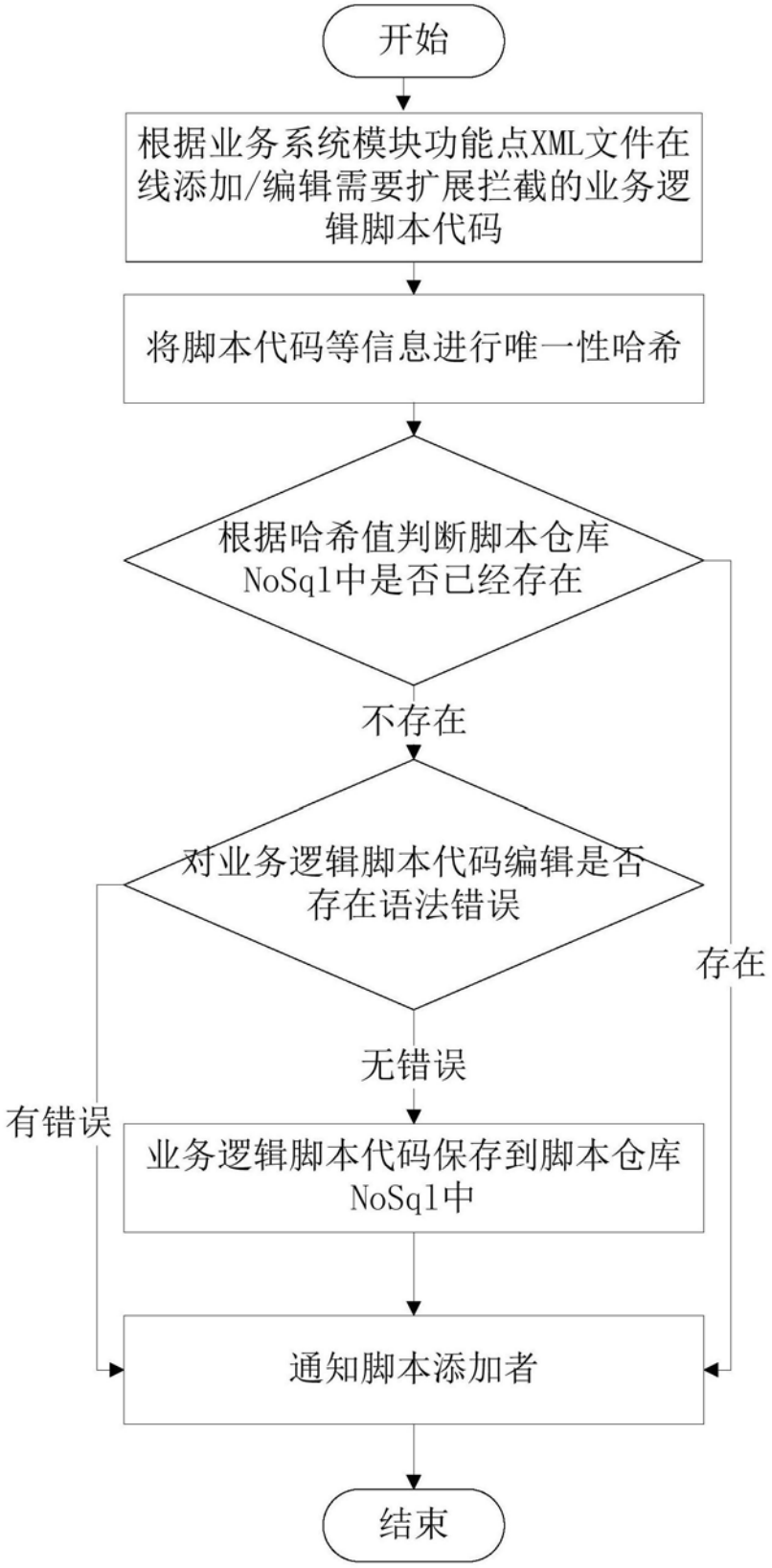

开始
根据业务系统模块功能点XML文件在线添加/编辑需要扩展拦截的业务逻辑脚本代码
将脚本代码等信息进行唯一性哈希
根据哈希值判断脚本仓库NoSql中是否已经存在
不存在
存在
对业务逻辑脚本代码编辑是否存在语法错误
无错误
有错误
业务逻辑脚本代码保存到脚本仓库NoSql中
通知脚本添加者
结束

图2

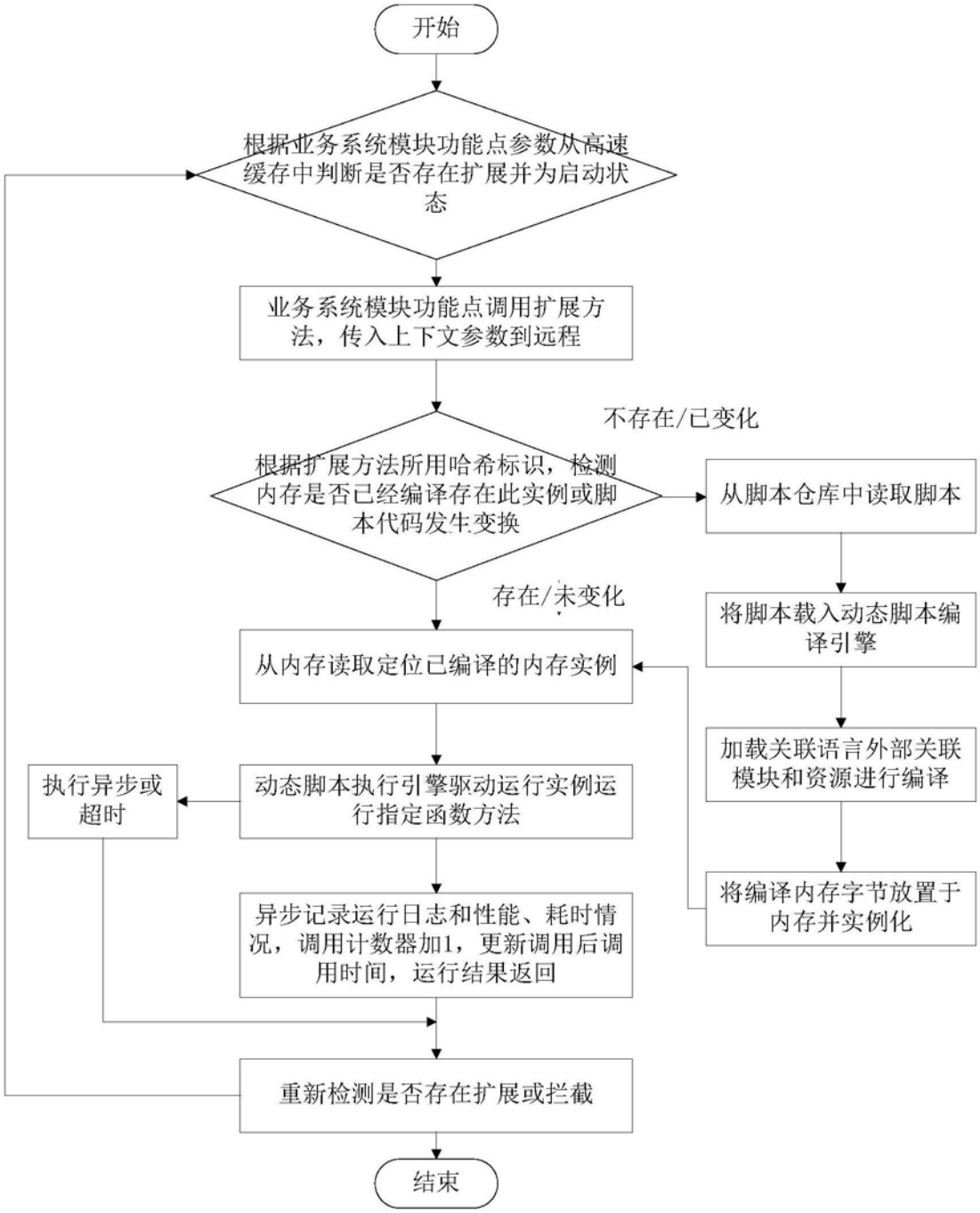

开始
根据业务系统模块功能点参数从高速缓存中判断是否存在扩展并为启动状态
业务系统模块功能点调用扩展方法，传入上下文参数到远程
不存在/已变化
根据扩展方法所用哈希标识，检测内存是否已经编译存在此实例或脚本代码发生变换
从脚本仓库中读取脚本
存在/未变化
将脚本载入动态脚本编译引擎
从内存读取定位已编译的内存实例
加载关联语言外部关联模块和资源进行编译
执行异步或超时
动态脚本执行引擎驱动运行实例运行指定函数方法
将编译内存字节放置于内存并实例化
异步记录运行日志和性能、耗时情况，调用计数器加1，更新调用后调用时间，运行结果返回
重新检测是否存在扩展或拦截
结束

图3


脚本仓库
业务系统A
业务系统B
系统级-扩展拦截
扩展拦截代码信息
扩展拦截代码信息
扩展拦截代码信息 n...
客户级-客户X扩展拦截
扩展拦截代码信息
扩展拦截代码信息
扩展拦截代码信息 n...
客户Y扩展拦截脚本代码信息
客户Z扩展拦截脚本代码信息
系统级-扩展拦截
扩展拦截代码信息
扩展拦截代码信息
扩展拦截代码信息 n...
客户级-客户X扩展拦截
扩展拦截代码信息
扩展拦截代码信息
扩展拦截代码信息 n...
客户Y扩展拦截脚本代码信息
客户Z扩展拦截脚本代码信息

图4

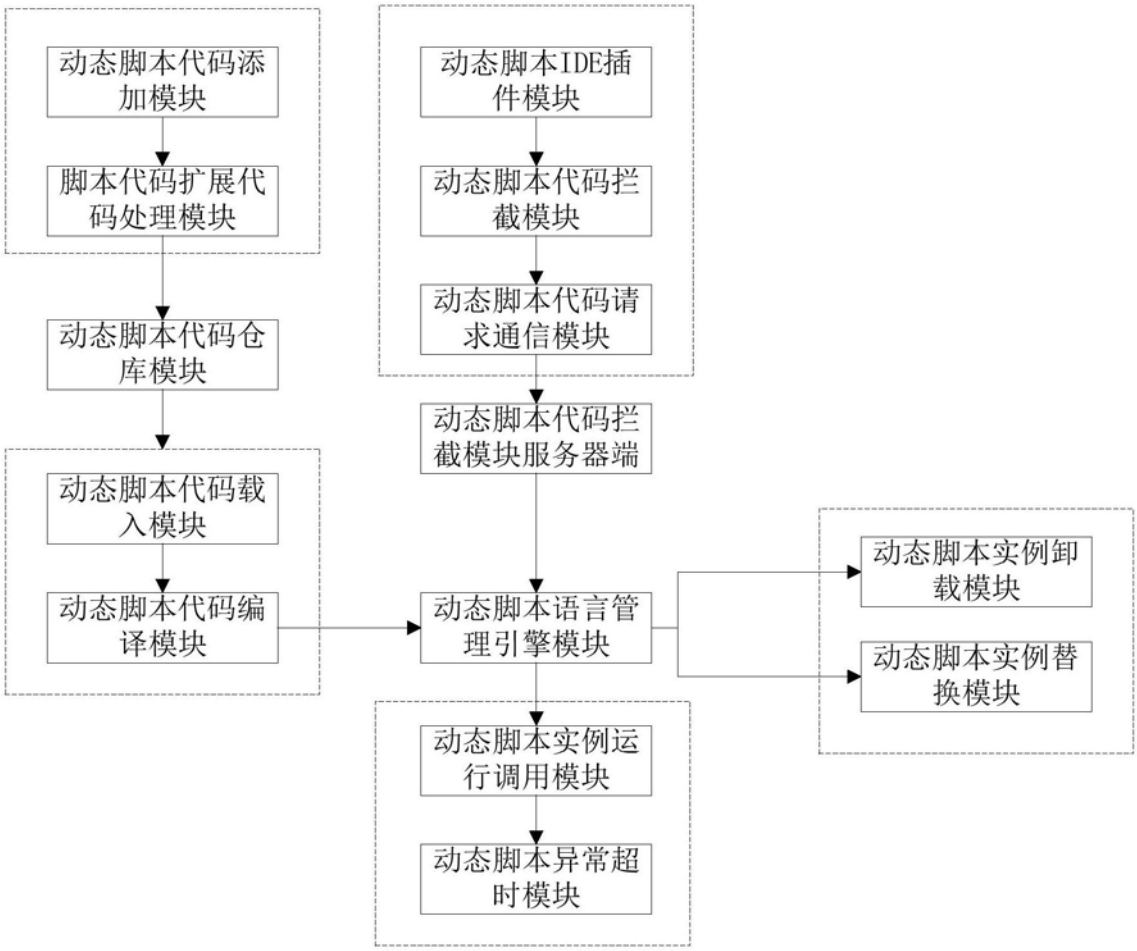

动态脚本代码添加模块
脚本代码扩展代码处理模块
动态脚本代码仓库模块
动态脚本代码载入模块
动态脚本代码编译模块
动态脚本IDE插件模块
动态脚本代码拦截模块
动态脚本代码请求通信模块
动态脚本代码拦截模块服务器端
动态脚本语言管理引擎模块
动态脚本实例卸载模块
动态脚本实例替换模块
动态脚本实例运行调用模块
动态脚本异常超时模块

图5

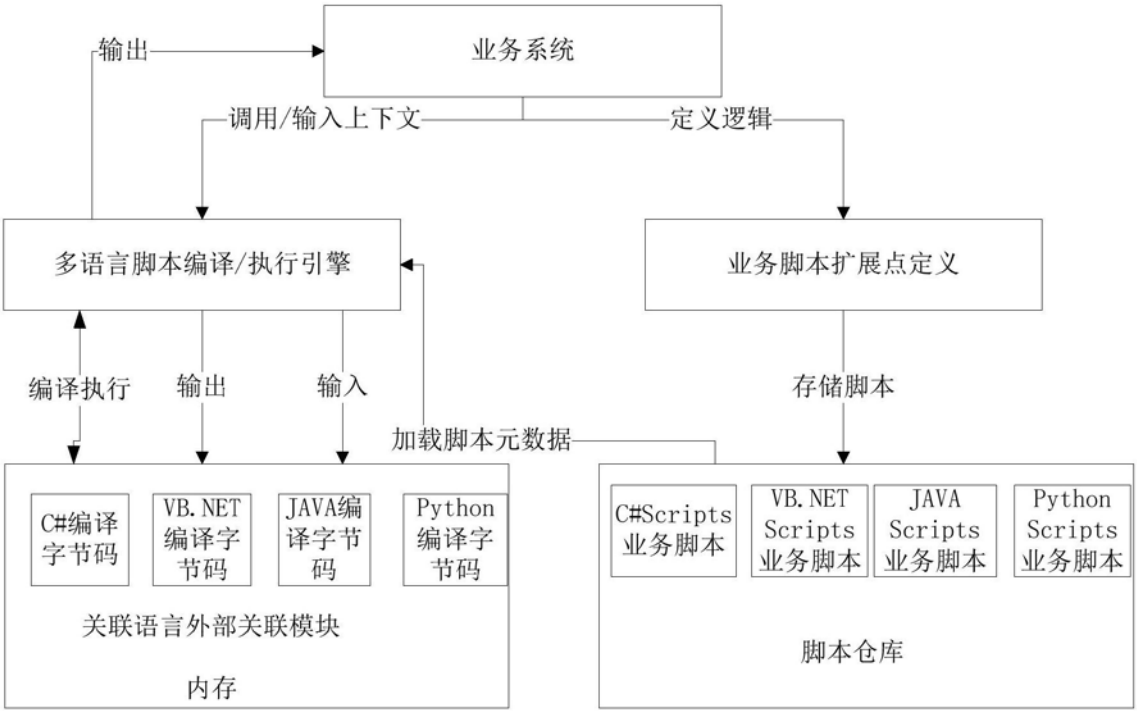

输出
业务系统
调用/输入上下文
定义逻辑
多语言脚本编译/执行引擎
业务脚本扩展点定义
编译执行
输出
输入
加载脚本元数据
存储脚本
C#编译字节码
VB.NET编译字节码
JAVA编译字节码
Python编译字节码
关联语言外部关联模块
内存
C#Scripts业务脚本
VB.NET Scripts业务脚本
JAVA Scripts业务脚本
Python Scripts业务脚本
脚本仓库

图6

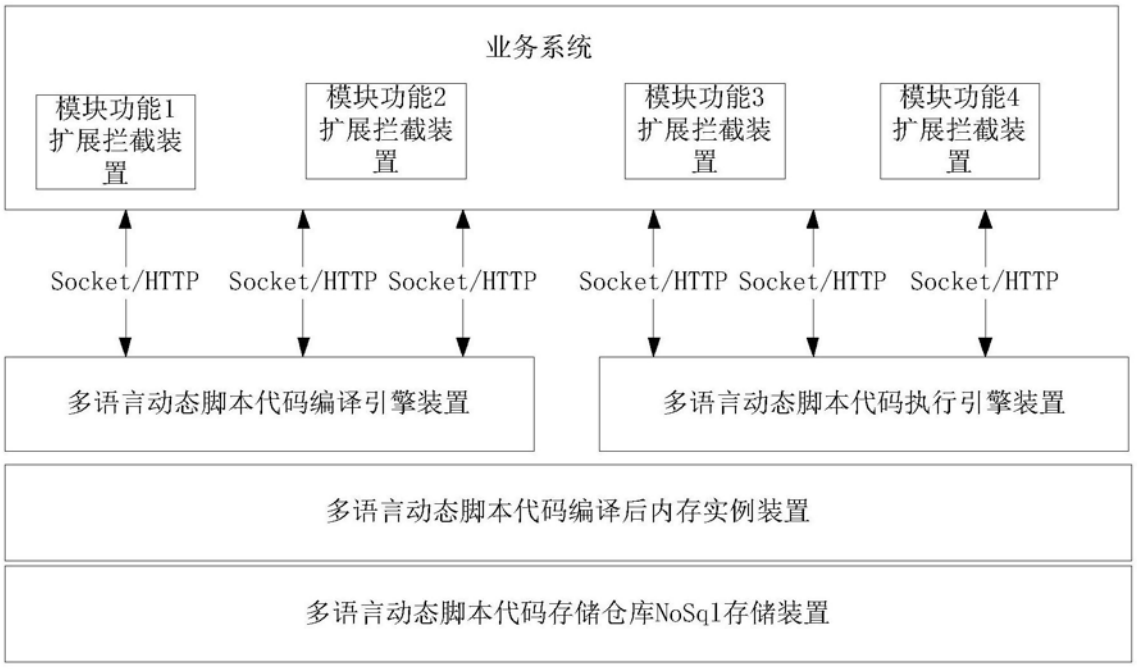

业务系统
模块功能1扩展拦截装置
模块功能2扩展拦截装置
模块功能3扩展拦截装置
模块功能4扩展拦截装置
Socket/HTTP
Socket/HTTP
Socket/HTTP
Socket/HTTP
Socket/HTTP
Socket/HTTP
多语言动态脚本代码编译引擎装置
多语言动态脚本代码执行引擎装置
多语言动态脚本代码编译后内存实例装置
多语言动态脚本代码存储仓库NoSql存储装置

图7

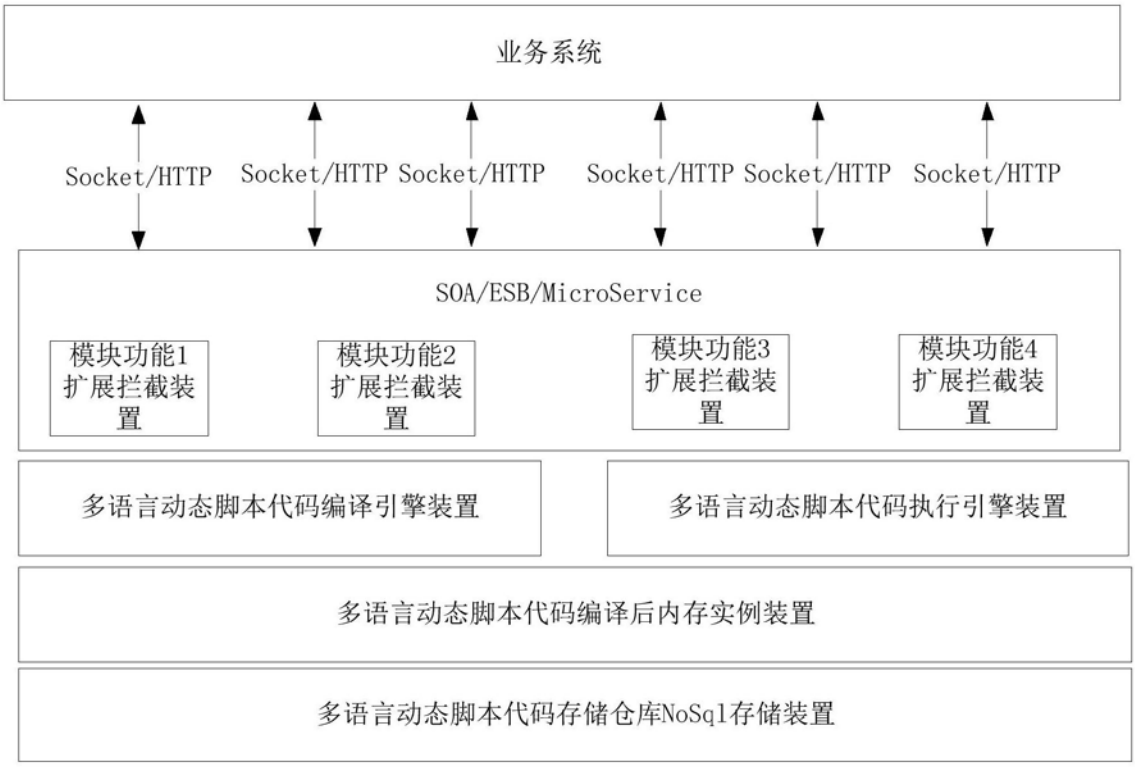

业务系统
Socket/HTTP
Socket/HTTP
Socket/HTTP
Socket/HTTP
Socket/HTTP
Socket/HTTP
SOA/ESB/MicroService
模块功能1 扩展拦截装置
模块功能2 扩展拦截装置
模块功能3 扩展拦截装置
模块功能4 扩展拦截装置
多语言动态脚本代码编译引擎装置
多语言动态脚本代码执行引擎装置
多语言动态脚本代码编译后内存实例装置
多语言动态脚本代码存储仓库NoSql存储装置

图8

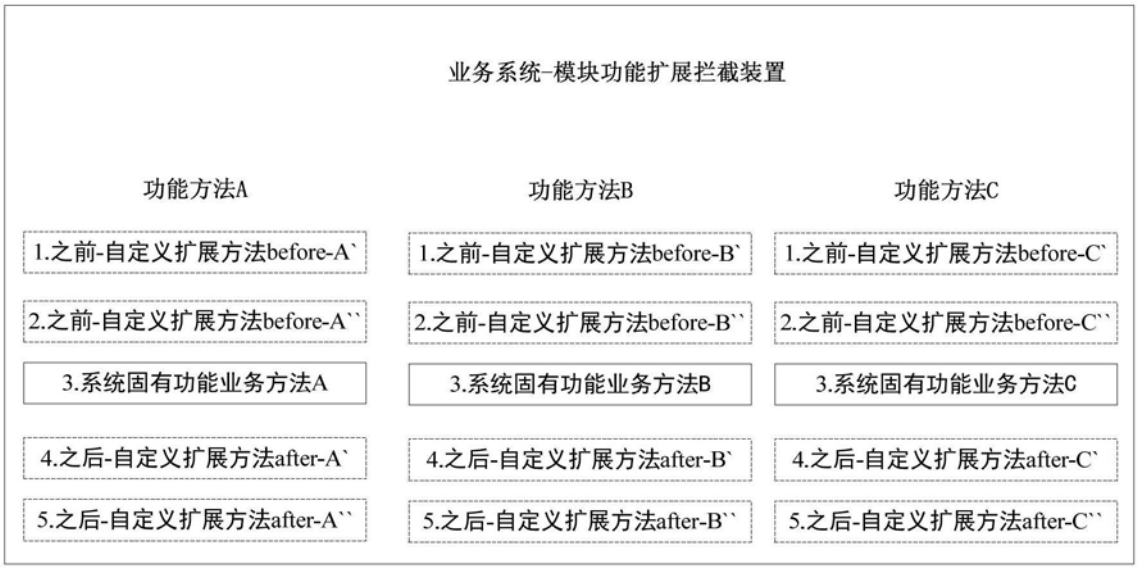

业务系统-模块功能扩展拦截装置
功能方法A
功能方法B
功能方法C
1.之前-自定义扩展方法before-A`
1.之前-自定义扩展方法before-B`
1.之前-自定义扩展方法before-C`
2.之前-自定义扩展方法before-A``
2.之前-自定义扩展方法before-B``
2.之前-自定义扩展方法before-C``
3.系统固有功能业务方法A
3.系统固有功能业务方法B
3.系统固有功能业务方法C
4.之后-自定义扩展方法after-A`
4.之后-自定义扩展方法after-B`
4.之后-自定义扩展方法after-C`
5.之后-自定义扩展方法after-A``
5.之后-自定义扩展方法after-B``
5.之后-自定义扩展方法after-C``

图9

图10A

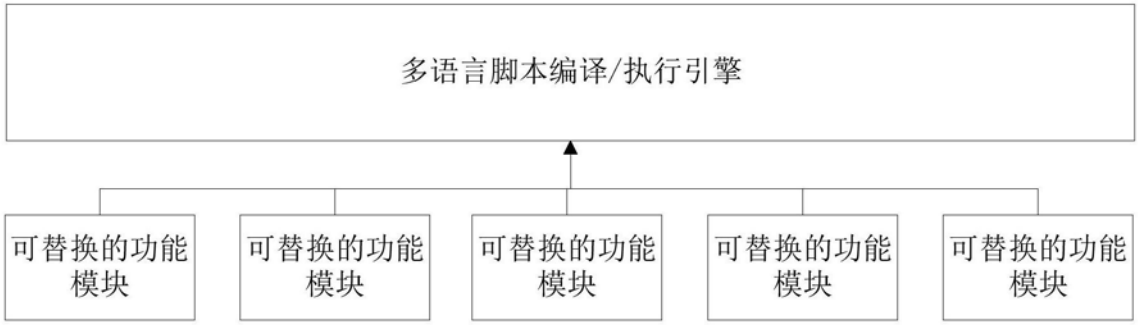


图10B

图11

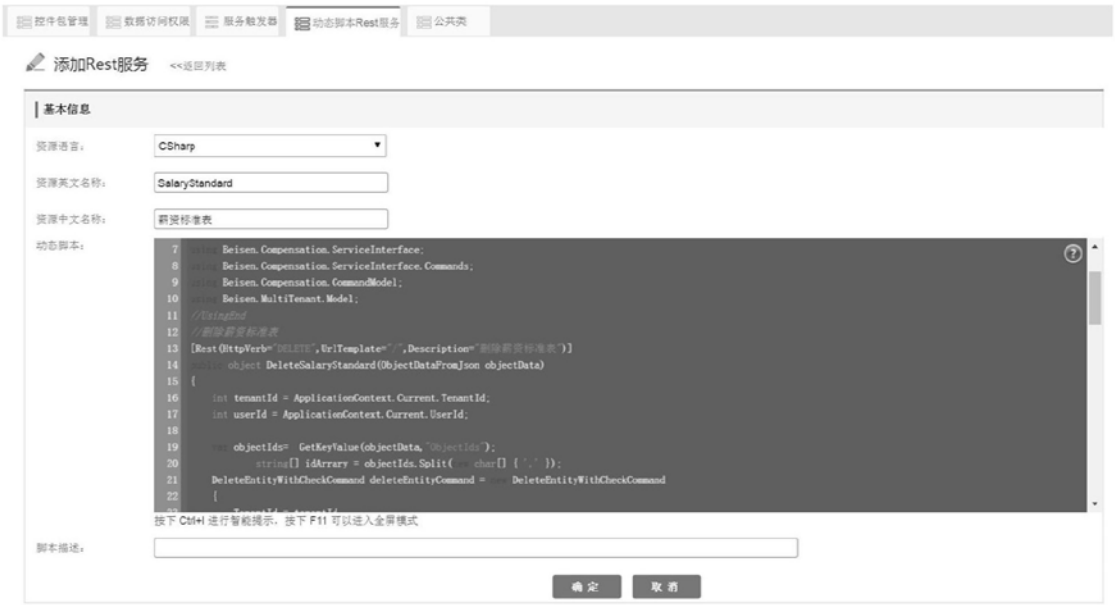

控件包管理
数据访问权限
服务触发器
动态脚本Rest服务
公共类
添加Rest服务
<<返回列表
基本信息
资源语言：
CSharp
资源英文名称：
SalaryStandard
资源中文名称：
薪资标准表
动态脚本：
using Beisen.Compensation.ServiceInterface;
using Beisen.Compensation.ServiceInterface.Commands;
using Beisen.Compensation.CommandModel;
using Beisen.MultiTenant.Model;
//UsingEnd
//删除薪资标准表
[Rest(HttpVerb="DELETE",UriTemplate="/",Description="删除薪资标准表")]
public object DeleteSalaryStandard(ObjectDataFromJson objectData)
{
int tenantId = ApplicationContext.Current.TenantId;
int userId = ApplicationContext.Current.UserId;
var objectIds= GetKeyValue(objectData,"ObjectIds");
string[] idArrary = objectIds.Split(new char[] {','});
DeleteEntityWithCheckCommand deleteEntityCommand = new DeleteEntityWithCheckCommand
{
按下 Ctrl+I 进行智能提示，按下 F11 可以进入全屏模式
脚本描述：
确 定
取 消

图12

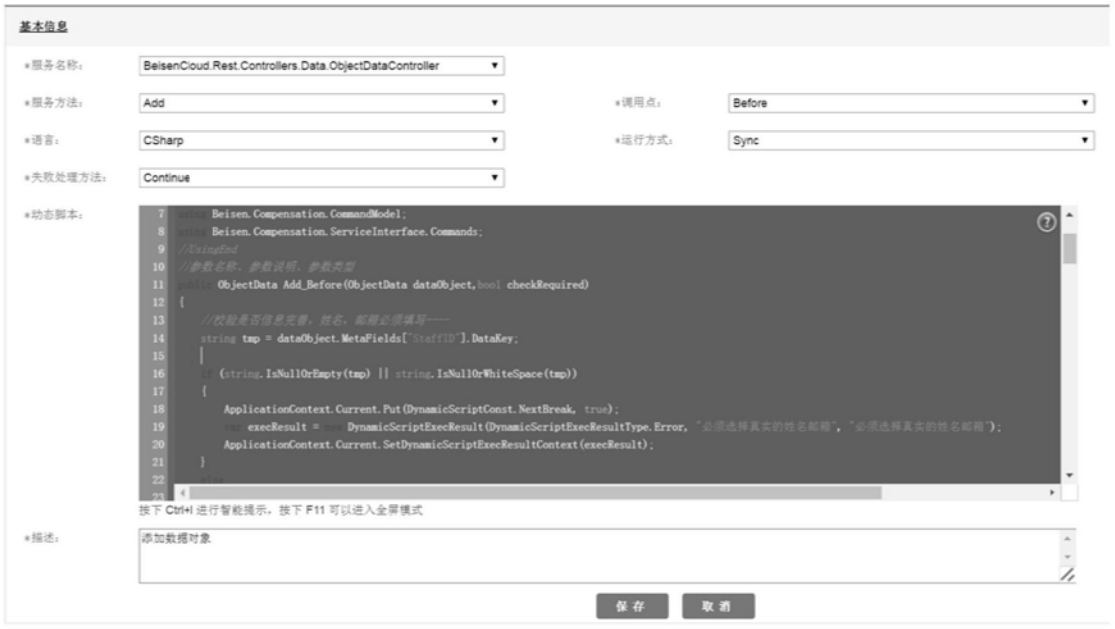

基本信息
*服务名称：
BeisenCloud.Rest.Controllers.Data.ObjectDataController
*服务方法：
Add
*调用点：
Before
*语言：
CSharp
*运行方式：
Sync
*失败处理方法：
Continue
*动态脚本：
using Beisen.Compensation.CommandModel;
using Beisen.Compensation.ServiceInterface.Commands;
//UsingEnd
//参数名称，参数说明，参数类型
public ObjectData Add_Before(ObjectData dataObject,bool checkRequired)
{
//校验是否信息完善，姓名，邮箱必须填写~~~
string tmp = dataObject.MetaFields["StaffID"].DataKey;
if (string.IsNullOrEmpty(tmp) || string.IsNullOrWhiteSpace(tmp))
{
ApplicationContext.Current.Put(DynamicScriptConst.NextBreak, true);
var execResult = new DynamicScriptExecResult(DynamicScriptExecResultType.Error, "必须选择真实的姓名邮箱", "必须选择真实的姓名邮箱");
ApplicationContext.Current.SetDynamicScriptExecResultContext(execResult);
}
按下 Ctrl+I 进行智能提示，按下 F11 可以进入全屏模式
*描述：
添加数据对象
保 存
取 消

图13

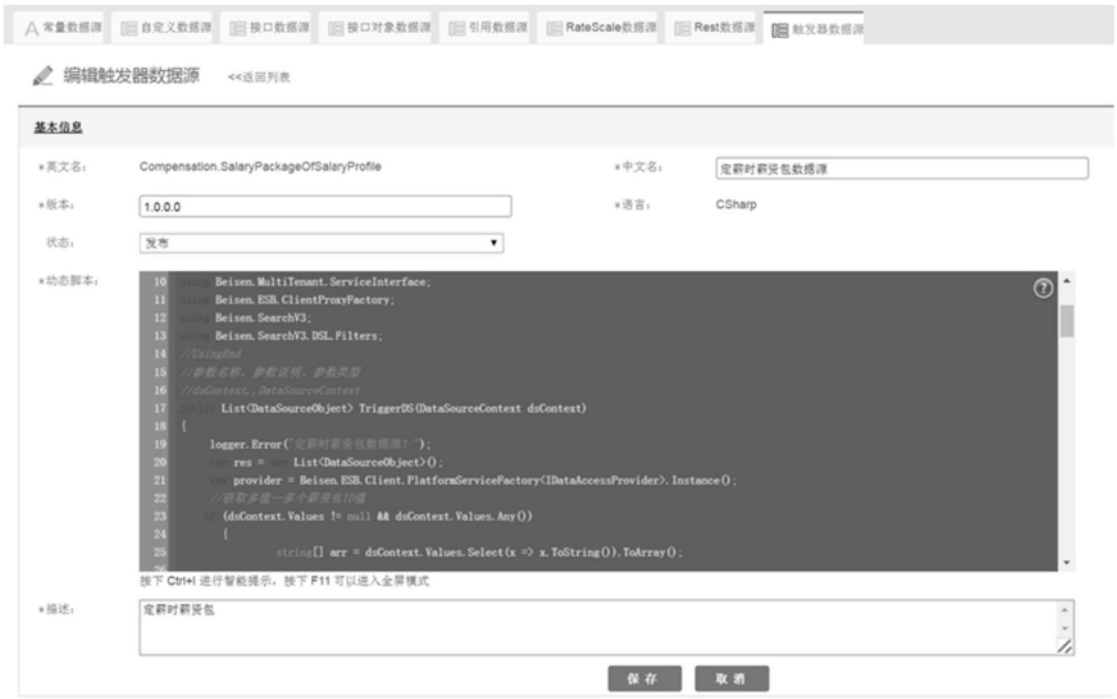

常量数据源
自定义数据源
接口数据源
接口对象数据源
引用数据源
RateScale数据源
Rest数据源
触发器数据源
编辑触发器数据源
<<返回列表
基本信息
*英文名:
Compensation.SalaryPackageOfSalaryProfile
*中文名:
定薪时薪资包数据源
*版本:
1.0.0.0
*语言:
CSharp
状态:
发布
*动态脚本:
using Beisen.MultiTenant.ServiceInterface;
using Beisen.ESB.ClientProxyFactory;
using Beisen.SearchV3;
using Beisen.SearchV3.DSL.Filters;
//UsingEnd
//参数名称、参数说明、参数类型
//dsContext,,DataSourceContext
public List<DataSourceObject> TriggerDS(DataSourceContext dsContext)
{
logger.Error("定薪时薪资包数据源!");
var res = new List<DataSourceObject>();
var provider = Beisen.ESB.Client.PlatformServiceFactory<IDataAccessProvider>.Instance();
//获取多值一多个薪资包ID值
if (dsContext.Values != null && dsContext.Values.Any())
{
string[] arr = dsContext.Values.Select(x => x.ToString()).ToArray();
按下 Ctrl+I 进行智能提示，按下 F11 可以进入全屏模式
*描述:
定薪时薪资包
保 存
取 消

图14