文章编号:1671-4598(2019)12-0024-05 DOI:10.16526/j.cnki.11—4762/tp.2019.12.006 中图分类号:TP311.56 文献标识码:A

基于智能云测试平台通信模块的设计

王 亮[1],谢锡海[2],尹成瑞[1]
(1.青海大学 信息化技术中心,西宁 810016;2.西安邮电大学 通信与信息工程学院,西安 710061)

摘要:根据智能云测试系统的特点,并结合实际中通用测试平台的通信机制;为实现快速有效的驱动执行机,在selenium框架基础上,引入了智能云测试系统通信模块的设计与实现;该智能云测试系统的通信机制,采用Carbon—Server作为框架,进行了智能云测试服务提供方、服务调用方通信细节的研究,采用其通信机制对大量单机版测试工具进行整合,形成了一个比较便捷的分布式开发架构,可以较好地模拟大规模组网;结果表明,与通用测试平台的通信机制相比,智能云测试通信系统在实际工程中能够更有效地提高通信效率。

关键词:智能云测试;通用测试平台;selenium;Carbon—Server

Design of Communication Module Based on Intelligent Cloud Test Platform

Wang Liang[1], Xie Xihai[2], Yin Chengrui[1]
(1. Information Technology Center, Qinghai University, Xining 810016, China; 2. College of Communication and Information Engineering, Xi'an University of Post and Telecommunication, Xi'an 710061, China)

Abstract: According to the characteristics of Intelligent Cloud test system and combining with communication mechanism of general testing platform in practice. In order to realize the fast and effective drive actuator, the design and implementation of the communication module of intelligent cloud test system was introduced on basis of the selenium framework. the communication module use the Carbon—Server as a framework, research intelligent cloud service providers and clients to the details of the communication, the communication mechanism integrated a large number of single version testing tools and formed a more convenient and distributed development framework. which can better simulate the large—scale network. The results show that the intelligent cloud test communication system have more communication efficiency than the universal test platform in practical engineering.

Keywords: intelligent cloud testing; universal test platform; selenium; Carbon—Server

0 引言

自动化测试技术[1-3]的发展经历了捕获回放[4]、单脚本执行、关键字驱动的测试集执行[5-6]这3个阶段的发展,已经逐步走向成熟,这三者的共同特点是基于单机版的测试平台执行[7]。对于单机版测试平台来说,其测试套中的脚本运行是串行化的,当测试套中脚本规模较大时,运行一次所需的时间比较长。

智能云测试系统[8-9]的目标是在实现测试资源管理自动化的基础上,提供更高效的自动化脚本执行平台,在智能云测试系统中,需要存在大量的执行机运行在独立的计算机或者虚拟机[10]上。这些执行机需要在统一的调度程序控制下,完成启动进程、打开相应的控制台、接受并运行指定脚本、反馈当前的执行状态、结束进程的动作。然而通用测试平台执行机的启动、运行及停止均是无人值守的。智能云测试系统内部通过命令来驱动执行机来完成,这个需求对测试平台的通信机制提出了新的挑战,并且这一通信机制对于后续测试工具及自动化的发展带来了新思路。

1 通用测试平台及通信机制

通用测试平台提供多种连接方式来连接设备,以TCL语言为基础,为测试人员提供了广泛的手工测试和自动化测试支持。通用测试平台实质是将多种不同的设备连接方式作了统一管理。避免了用户分别使用超级终端、SSH Client等不同的客户端来访问设备。所以通用测试平台需要支持同时以不同的连接方式连接设备,并与设备CLI接口进行交互。通用测试平台集成文本编辑器来支持TCL脚本的编写,使得通用测试平台集TCL脚本编辑、执行功能于一体,更加接近于TCL语言集成开发环境。

测试平台作为一个标准的Windows程序,其绝大部分功能是由主界面的各项菜单来驱动的。用户在通用测试平台主界面上的各项操作,包括点击各类菜单,编辑器中输入字符等操作,最终都会产生一系列的Windows消息,通用测试平台通过接收并处理这些消息来响应用户,并进行各项处理,大致的流程如图1所示。

现有通用测试平台系统并不支持任何分布式特性,面

收稿日期:2018-09-17; **修回日期**:2019-05-08。
基金项目:赛尔网络下一代互联网技术创新项目(NGII20160313);陕西省科技攻关资助项目(2014K05—20)。
作者简介:王 亮(1989-),男,陕西咸阳人,硕士,助教,主要从事数字信号处理、软件测试平台和计算机网络方向的研究。
谢锡海(1967-),男,陕西汉中人,硕士,研究员,主要从事数字信号处理方向的研究。

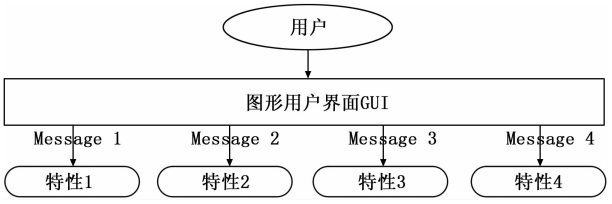

图 1　消息驱动

对来自智能云测试系统的新需求时，进行相应升级改造。智能云测试系统执行机的功能需求实际只是当前通用测试平台系统所实现功能的一个子集，通过将外部的驱动方式由 Windows 消息修改为特定命令格式后，可以满足智能云测试的实际应用，将具体操作理解为驱动模块，将通用测试平台的实际功能模块理解为业务模块。将驱动模块由 Windows 消息转换为其他特定命令，而业务模块即可透明的支持多种应用场景，如图 2 所示。

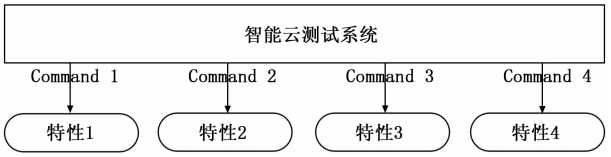

图 2　命令驱动

在实际实现时，考虑到通信机制的通用性，重点考虑单机版测试平台与分布式测试平台实现上的统一；通信机制需要有良好的扩展性来支撑未来需求。通过正常方式启动的通用测试平台不受任何影响，仍然是由用户通过 GUI 菜单产生的 Windows 消息来驱动；而在启动通用测试平台时给出特定命令行参数，通用测试平台会在启动时进行智能云测试服务注册，注册成功后会自动开启另外一条命令通道，接受并处理 Carbon 命令。

2　智能云测试系统

智能云测试[11]是在云计算的基础下一种新型测试方法。由云系统维护资源池，统一管理测试所需的环境：设备、仪器和 PC 机等。智能云测试系统以云为中心，将测试资源集中管理，按需动态分配，同时脚本集中在云端并发分布式执行。用户通过 web 接口向系统提交测试任务，设定被测设备型号，选定测试版本，测试用例，提交后系统会自动计算该任务所需要的环境，并动态分配测试组网、执行机来执行相应测试例，并及时给出结果反馈。

2.1　智能云测试系统通信机制

智能云测试通信机制本质是一种 Client－Server 通信机制的扩展，通过将消息报文采用标准的 HTTP 消息进行封装，来支持多种语言实现的服务端和客户端。也就是与编程语言无关的多客户端多服务端通信机制，如图 3 所示。

通信机制有如下特点：1）语言无关性：对于客户端和服务提供端来说，可以采用任意语言编程实现，且支持混

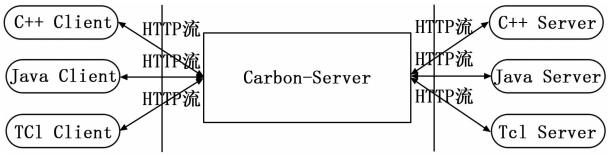

图 3　智能云测试从通信机制

合调用。TCL 语言客户端可以调用 C＋＋服务提供端的服务，反之亦然。客户端需要调用某服务时，不需要关心还服务是采用何种语言实现；2）平台无关性：客户端、Carbon－Server 和服务提供者可以运行在 Windows 及 Linux 操作系统，并且 Windows 系统上的客户端程序也可以向 Linux 系统下的某个服务发起请求并得到处理，整个过程对客户端完全透明；3）部署简单：客户端、Carbon－Server 和服务提供者在同一台物理机器，亦可以完全在不同的物理机器；4）可扩展性高：由于服务和客户程序的松耦合性，在本系统中添加新的服务是极为便利的事情。不需要更改任何现有的服务提供者和客户端，相应的是，某服务提供者提供更多的功能命令时，也只需要修改该服务本身，不对其他服务造成影响。

2.2　Carbon－Server 框架

Carbon－Server 以 C/S 模式实现一个简便的进程间通信平台。基于命令行形式的服务命令调用；简便的客户端/服务实现接口；多编程语言支持；选用 Selenium 作为基础框架进行开发，Selenium 是一款为实现 Web 应用程序开源的自动化测试框架[12-15]，其服务程序的开发和部署都相对容易，对操作系统和服务平台依赖小，可扩展性强。Selenium 的通信过程如图 4 所示。

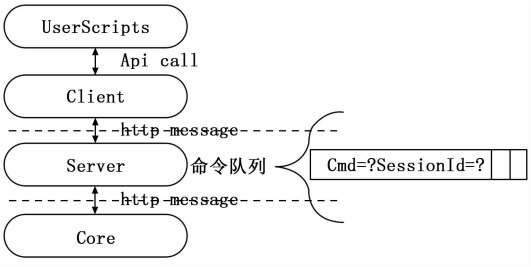

图 4　Selenium 通信结构

这个通信过程中，客户端接口将用户调用的服务器命令及参数封装在 HTTP 请求当中发送给 Selenium－Server，Selenium－Server 解析命令内容并构造 JSON 字符串写入到对应浏览器的命令队列。浏览器中运行的 Java－Script 服务程序（Core）定时向 Selenium－Server 发送 HTTP 响应（带有 POST 消息）。如果命令队列不为空，Selenium－Server 就把命令字符串通过 HTTP 响应发送给服务程序运行。服务程序执行命令并把结果封装在下一个 HTTP 响应当中返回给 Selenium－Server，最后 Selenium－Server 把命令结果封装在 HTTP 响应命令当中返回给客户端。要满足

Carbon－Server 的要求，Selenium－Server 需要删除其中的浏览器支持部分，而主要保留了其中的通信机制、命令队列、日志处理这 3 个部分。并对以下几个方面进行了改进，改进后 Carbon－Server 的通信结构如图 2 所示。

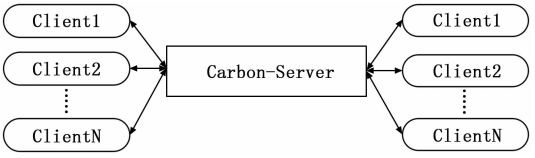

图 5 Carbon－Server 通信结构

2.2.1 注册/撤消服务

Carbon－Server 支持服务的注册和撤消。使用 Carbon－Server 提供的服务程序开发接口编写的服务程序，通过向 Carbon－Server 发送 registerService 命令来完成注册。Carbon－Server 接受到注册命令时为服务程序创建命令队列。注册之后，服务程序就可以定时从命令队列当中读取命令。相应的，unregisterService 命令撤消服务的注册。Carbon－Server 会删除服务对应的命令队列，并停止处理客户端对该服务的命令请求。

2.2.2 服务队列标识

服务程序在注册 Carbon－Server 时，在 register Service 命令中包含了服务的名称信息。Carbon－Server 使用服务队列管理类来维护服务名称到对应的命令队列的映射。每一个服务创建一个命令队列，并映射到一个唯一的服务名称标识。因此，在 Carbon－Server 上注册的服务不能存在重名的现象。客户端在请求某个服务的命令时也必须指定服务的 ServiceId。

2.2.3 并发访问

Carbon－Server 的应用环境中，通常存在两种情况的并发需求：多客户端并发访问同一个服务、多客户端并发访问不同的服务。Carbon－Server 使用 Apache 的 Http 包来提供 HTTP 服务。对于每一个 HTTP 请求（响应）都会创建一个处理线程。因此，并发访问实际上体现为多线程并发读写命令队列。实现时使用同步阻塞的方法来限制同时只有一个线程写入指定服务的命令队列并等待结果。

2.2.4 定时老化

Carbon－Server 为每一个注册成功的服务创建一个计数器，并使用一个专门的定时器线程对所有定时器进行刷新，定时器刷新线程每隔一定时间将所有计数器的值加 1。如果计数器的值达到限制的数值，则认为该服务已超时，需要删除其命令队列。另一方面，当 Carbon－Server 接受到服务程序发送的命令获取消息时，计数器设置为 0。

2.2.5 状态查询

Carbon－Server 提供 listAllService 命令来查询所有服务的状态。返回信息以换行符分隔，每一行对应一个服务的状态信息。行的内容又以逗号分隔为不同的列。

2.2.6 服务类型

Carbon－Server 的配置文件中可以定义两种类型的服务，Internal 和 External：对于 Internal 类型的服务，Carbon－Server 在启动时会自动根据配置文件的信息来启动服务程序；而对于 External 类型的服务，Carbon－Server 不会自动启动服务（用户可以手动启动服务或在客户端通过 stsrtService 命令启动）。当服务的定时器超时时，对于 Internal 的服务，Carbon－Server 会关闭原来的服务程序并重新启动，而对于 External 类型的服务则只是删除命令队列。

2.2.7 并发方式及命令封装

Carbon－Server 支持两种服务并发方式：基于线程的并发和基于进程的并发。如果服务程序支持基于线程的并发，则主线程只处理 getNewSession/releaseSession 命令。则当客户端发送 getNewSession 命令时，服务器程序需要启动一个新的服务线程并注册到 Carbon－Server。新线程的 ServiceId 包含一个随机的 GUID 和最初的服务名称。Carbon－Server 会把这个 ServiceId 返回为客户端。这个新的线程可以作为一个单独的服务来使用。处理结束时，客户端需要发送 releaseSession 来释放服务线程。如果服务程序支持基于进程的并发，则不管 Internal 还是 External 类型的服务，Carbon－Server 都不会自动服务程序。只有当接受到客户端的 startService 命令时才会通过命令行来启动服务程序。这类服务程序的第一个命令行参数必须为一个字符串参数，服务程序使用该字符串参数加上内置的服务名称作为 ServiceId 注册到 Carbon－Server。因此，Carbon－Server 在启动服务程序时需要生成一个随机的 GUID 作为服务启动参数。Carbon－Server 使用 XML 字符串来封装客户端发送给服务程序的命令信息。

2.2.8 服务与客户接口

Carbon－Server 提供 Java/C＋＋/TCL 的服务端开发接口。不管哪种语言形式，服务开发接口都需要提供 3 个功能：注册命令、启动服务、停止服务。注册命令接口至少包含两个参数：命令名称和命令执行对象。通过调用注册命令接口可以建立命令名称到命令执行对象之间的映射。服务程序根据这个映射关系来执行客户端的命令请求。首先生成一个 CarbonAgent 对象。构造函数的参数分别指定 Carbon－Server 的地址、服务名称。然后使用 register 接口注册一系列命令。所以命令类都需要实现接口 AbstractCarbonAgentCommand 的方法 execute。该方法传入一个字符串数组参数，返回结果也为一个字符串。Carbon－Server 提供 Java/C＋＋/TCL 的客户端开发接口。客户端接口实际上只需要把命令行封装在 HTTP 请求中发送即可。客户程序只需要调用 submit 接口把命令发送给服务器执行即可。

2.3 智能云测试系统服务提供方

在智能云测试通信系统中，服务提供方由业务模块、代理模块及监听服务器组成。其中业务模块与代理模块在

同一进程中，通过函数调用及回调的方式来交互。监听服务器一般在单独进程中实现，进程间采用 Socket 方式封装 HTTP 消息来实现。

对于执行机分布式执行的场景来说，业务模块代表通用测试平台进程，代理模块在 CCarbonAgent.dll 中实现，为通用测试平台进程提供业务注册，消息封装及解封装，保活操作；监听服务器则对应 Canbon－Server 进程。它们之间的关系如图 6 所示。

图 6　provider 组成

由于 Canbon－Server 服务器对外完全采用 HTTP 消息的方式通信，这样的方式决定了业务模块与代理模块的实现不受语言限制。一般情况下，对于不同形态的业务模块来说，分别提供不同的代理模块来配合，如图 7 所示。

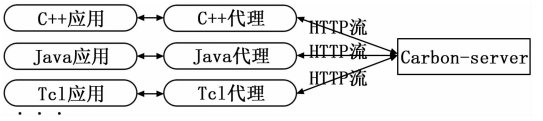

图 7　多代理框架

代理模块实现时，通常要遵循的原则是：实现上尽量独立，与业务模块通过接口集成，代理模块的实现要充分考虑不同语言的特点。对于 C/C＋＋代理程序来说，比较适合的实现是独立的标准 DLL。这样的话可以方便任意 C/C＋＋程序经过简单的改造演变成服务模块。对于 Java 程序来说，比较合适的实现是封装成独立的 Jar 包来完成。对于 TCL 代理来说，封装成 Lib 库是合适的选择。

2.3.1　服务提供方通信细节

在通用测试平台的实践中，代理模块向通用测试平台提供如下的接口来注册命令。

```
DLL_API int CarbonServer_InstallService(struct sCarbonServiceInfo * pService)
```

入参的数据结构定义如下（所有参数以字符串的形式统一传递）：

```
typedef struct sCarbonCmdList {
Std::string strName;
int ( * fCmdHandle) (std::string);
}sCarbonCmdList;
注册本地服务基本信息
typedef struct sCarbonServiceInfo {
DWORD dServerIpAddr;
UINT uServerPort;
Std::string strServiceName;
Std::vector<sCarbonCmdList * >vecCmdList;
}sCarbonServiceInfo;
```

其中，字符串变量 strServiceName 表示通用测试平台所提供的服务名称，向量 vecCmdList 包含了通用测试平台服务所提供的所有有效命令列表。对于每个命令结构，由 sCarbonCmdList 结构来记录命令的名称及回调函数指针。注册后，当监听服务器 Carbon－Server 收到该服务对应的命令请求时，会从以上命令列表中进行匹配，检查是合法命令时，则直接通过回调函数来调用通用测试平台服务触发相应操作。

2.4　智能云测试系统服务调用方

智能云测试通信系统中，调用方一般是由 Client、Agent 及 Carbon－Server 组成。其实现机制与服务提供方类似，如图 8 所示。

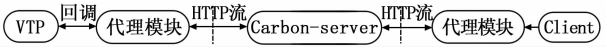

图 8　调用方通信机制

一般情况下，Carbon－Server 与服务提供方运行在一台服务器，并向外提供 TCP 端口服务。调用方通过代理模块向该服务发送命令请求，并接受运行结果。同一种语言的代理模块需要分别支持 Provider 服务和 Caller 客户端操作接口。

服务调用方通信细节：通用测试平台的实践中，通用测试平台主机注册就是采用向注册服务器请求，并由注册服务器记录到数据库中。注册程序是一个独立的 GUI 程序，内部通过调用代理模块提供的命令请求接口来实现通信。代理模块提供的请求接口如下：

```
DLL_API int CarbonClient_ExecCommand(bool bIsNewSession,Struct sExeRegInfo * pRegInfo,std::string &strOutput)
```

其中，入参中的 sExeRegInfo 结构记录了待请求的服务及命令，strOutput 变量为输出参数，记录了命令的执行结果。值得注意的是，该接口会触发请求服务的相应功能，不会立刻返回。sExeRegInfo 结构的定义如下：

```
Typedef struct sExeRegInfo{
DWORD dServerIpAddr;
UINT uServerPort;
std::string strServiceName;
Std::vector<std::string> vecCmdInfo;
}sExeRegInfo;
```

这里包含了带请求服务所注册的 Carbon－Server 地址信息，包含其 IP 地址和 TCP 端口号，以及具体服务的名称和命令名称、参数。

3　大规模混合性能测试平台

当前的性能测试主要以采用测试仪为主，外加单机版测试工具辅助的方法来进行，当测试工具的性能达不到要求时，极端情况下会采用多单机版工具同时运行的情况。但到指标相差太多时，此种方式基本无法适用。可以考虑基于智能云测试通信机制对大量单机版测试工具进行整合，在客户端以 TCL 脚本的形式来驱动其执行，如图 9 所示。

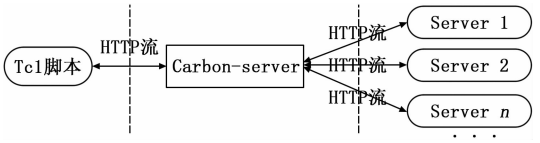

图 9 Tcl 驱动通信机制

Server1，Server2 等对应不同的性能测试模块，由客户端 TCL 脚本的运行来统一驱动。好处：1）支持大量相同的性能测试模块同时运行：通过 TCL 脚本，同时驱动大量的相同性能测试模块来运行，相当于极大增加了单机版性能测试模块的性能指标；2）支持大量不同的性能测试模块同时运行：可以较高的模拟大规模组网，使得测试平台能通过 PC 进行大规模混合流量测试。

4 实验结果与分析

该分布式通信系统在任意单机版程序中经过简单改造后均能提供分布式的统一服务，其充分利用现有软件资源及各类语言的特点来实现强大统一的工具平台，并且 Carbon－Server 在一个比较便捷的分布式开发架构下，其管理机制以及开发接口都比较简单，易于理解和实现，功能强大的优点。但在实际的应用中还存在不足之处，缺少 GUI 管理界面，命令执行超时处理可靠性不够，后续开发中应该解决这些问题。

参考文献：

[1] 田鸿运，刘 青，凯成杰，等. 一种面向高性能数值模拟软件的自动化测试平台 [J]. 计算机工程与科学，2017 (11)：1980－1985.
[2] 陈 晔. 关于软件设计可靠性自动化测试仿真研究 [J]. 计算机仿真，2017 (6)：281－284.
[3] 柏 莹. 基于 NET 平台下 Web 自动化测试的研究与设计 [D]. 西安：西安电子科技大学，2013：28－33.
[4] 刘 旭. 软件测试自动化的测试研究 [J]. 煤炭技术，2012，31 (7)：168－169.
[5] 朱 菊，王志坚，杨 雪，等. 基于数据驱动的软件自动化测试框架 [J]. 计算机技术与发展，2006，16 (5)：68－70.
[6] 接 卉，兰雨晴，骆 沛，等. 一种关键字驱动的自动化测试框架 [J]. 计算机应用研究，2009，26 (3)：927－929.
[7] 王 军，孟凡鹏. 基于关键字驱动的自动化测试研究与实现 [J]. 计算机工程与设计，2012，33 (9)：3652－3656.
[8] Leah R K，Ossit，Karis. Testing in the Cloud：Exploring the Practice [J]. IEEE Software，2012，29 (2)：46－51.
[9] 丁小盼，周 浩，贺 珊，等. 基于 OpenStack 的云测试平台及其性能分析研究 [J]. 软件学报，2015 (1)：6－10.
[10] 秦中元，沈日胜，张群芳，等. 虚拟机系统安全综述 [J]. 计算机应用研究，2012，29 (5)：1618－1620.
[11] 李 乔，何栋梁，王小林. 云测试研究现状综述 [J]. 现代计算机，2011 (23)：25－30.
[12] 张慧琳，李 威，佟秋利，等. 基于 Selenium 和 TestNG 的集成自动化测试平台设计 [J]. 实验技术与管理，2015，32 (9)：153－155.
[13] 高玉军. 面向对象分布式 Web 自动化实现 [J]. 软件，2013，34 (11)：86－89.
[14] 樊付星，黄大庆，周 未. 基于 Web 的自动化测试框架的研究与实现 [J]. 电子设计工程，2102，20 (20)：36－38.
[15] 周 娟，蒋外文. 基于 Web 的自动化测试框架 [J]. 计算机工程，2009，35 (18)：65－66.

(上接第 13 页)

恢复航天器正常工作状态起到了关键性作用。在工程实用性上具有明显优势，提炼总结 10 余项功能和算法，形成通用化构件，多维动态可配置的设计，使软件代码复用率达 90％以上，研制效率有效提高 3 倍，型号间通用化程度有效提升。该设计方案对提高航天器自主健康管理的具有较高的通用性设计意义。

后续需深入研究基于大数据分析的航天器故障诊断、预测、健康状态评估技术，为航天器任务管理和运行维护提供更高水平的系统健康状态分析能力。

参考文献：

[1] 曹国荣. 航天器在轨自主健康管理技术的研究和应用 [D]. 长沙：国防科学技术大学，2007.
[2] 梁 克，邓凯文，丁 锐，等. 载人航天器在轨自主健康管理系统体系结构及关键技术探讨 [J]. 载人航天，2014，20 (2)：117.
[3] 王 聪. 基于定性模型的航天器热控系统故障诊断方法的研究 [D]. 哈尔滨工业大学，2009.
[4] 李 晴，孙国江，李孝同. 基于星务管理系统的小航天器自主健康管理系统 [J]. 航天器环境工程，2012，29 (5)：574.
[5] 潘全文，李 天，李行善. 预测与健康管理系统体系结构研究 [A]. 2007 年全国测控、计量、仪器仪表学术年会 [C]. 2007.
[6] 彭 宇，刘大同. 数据驱动故障预测和健康管理综述 [J]. 仪器仪表学报，2014，35 (3)：482.
[7] Johnson S B，Gormley T J，Kessler S S，等. 系统健康管理及其在航空航天领域的应用 [M]. 景 博，杨 洲，池小泉，等译. 北京：国防工业出版社，2014.
[8] 张金晔，刘慧超，郭 蔚，等. 网络设备故障预测与健康管理系统设计 [J]. 软件导刊，2016，15 (4)：172.
[9] 王 妍，蔡 彪，程迎坤. 大型航天器控制系统健康管理 [J]. 空间控制技术与应用，2016，42 (5)：43－44.
[10] 张力元. 基于贝叶斯网络的航天器健康管理系统 [D]. 哈尔滨：哈尔滨工业大学，2015.