基于 OSGi 的分布式系统集中日志管理方案①

王宇飞, 刘 丹, 吴嘉生

(国网信息通信产业集团有限公司 北京中电普华信息技术有限公司, 北京 100192)

摘 要: 目前分布式业务应用的日志多存储在各分布式服务器节点本地日志文件中, 没有集中存储和管理, 导致业务系统问题定位速度慢, 解决问题效率低. 本文提供一种基于 OSGi 的分布式日志收集与分析技术方案. 该方案单独设计了集中的日志存储服务器用于存储日志, 并提供一套通用日志模型, 业务应用分布式节点向该设备发送基于该模型的日志数据, 日志存储服务器接收到各节点的日志数据后进行统一存储和界面化分析展示, 帮助开发人员快速定位和分析问题. 该方案以 OSGi 插件形式部署到应用系统, 应用卸载该插件后则以原有方式存储日志. 应用结果表明, 采用该日志管理方案对 1000 并发下记录日志的业务应用访问性能平均提升 2 秒, 并且没有日志数据丢失. 开发人员反馈, 错误日志更加一目了然, 定位问题的时间明显短于普通的日志存储方式.

关键词: OSGi; 分布式; 日志管理

Centralized Log Management Scheme for Distributed System Based on OSGi

WANG Yu-Fei, LIU Dan, WU Jia-Sheng

(Beijing China-Power Information Technology Co. Ltd., State Grid Information & Telecommunication Group, Beijing 100192, China)

Abstract: At present, the distributed business application logs are stored in the local log files on distributed servers, and there is no centralized storage and management, which leads to slow positioning speed for business system problems , and low efficiency in solving problems. This paper provides a distributed log collection and analysis scheme based on OSGi technology. It uses a centralized log storage server to store the log, and provides a set of general log model, so that distributed business application nodes can send the log data to the servers, based on the model. The log storage server receives log data of each node and then unifies storage and interface of analysis display, helping developers to quickly locate and analyze the problem. The scheme is deployed to the application system in the form of the OSGi plug-in, and log is stored in the original way after unloading the OSGi plug-in. Application results show that when the log management scheme is applied to 1000 concurrent business applications which perform logging, the access performance is improved to 2 seconds, and there is no log data loss. According to the developers' feedback, the error logs are clearer, the time of locating problems is obviously shorter than ordinary log storage.

Key words: OSGi; distributed; log management

目前大部分的应用系统为了提升自身性能, 都会采用分布式或者集群方式进行多节点部署, 各个节点都单独记录日志数据到本地日志文件. 一旦应用出现问题, 需要在各个节点逐一检查日志文件中的信息. 有一定规模的应用往往多达几十个节点, 排查问题困难. 而且一般的第三方日志存储系统都采用与应用程序同步的方式输出和存储日志, 日志存储完成后, 应用程序才能继续往下执行, 影响了应用程序本身的性能. 本文提供的日志管理方案设计了集中的存储日志的服务器, 各节点通过异步和缓冲区技术发送日志到该服务器, 减小了日志存储对应用程序的影响.

本文所述的日志管理方案实现包含 log-collecter

① 基金项目:国家电网公司 2016 年科技项目

收稿时间:2016-09-28;收到修改稿时间:2016-10-27 [doi:10.15888/j.cnki.csa.005794]

(日志收集器)、log-channel(日志通道)和 log-sender(日志发送器)三个 OSGi 模块. 其中 log-collecter 和 log-sender 是必选模块, log-buffer 是可选模块, log-channel 提供一段缓冲区缓冲日志数据, 还没来得及发送的日志则先缓冲到该缓冲区. 如果应用系统并发较小, 日志数据较少, 不需要缓冲区, 可以卸载掉 log-buffer 模块.

一个分布式架构的应用系统可以采用这种集中日志管理方案进行分布式节点的日志统一收集, 并且对于多个应用系统也可以采用一套集中日志管理系统进行日志统一收集, 通过在日志模型中增加应用编码来区分各个应用系统即可.

1 相关技术

1.1 RPC

日志从部署节点到日志存储服务器采用 RPC[1](Remote Procedure Call)协议.

RPC 协议是一种通过网络从远程计算机程序上请求服务, 而不需要了解底层网络技术的协议. RPC 协议假定某些传输协议的存在, 如 TCP 或 UDP, 为通信程序之间携带信息数据. 在 OSI 网络通信模型中, RPC 跨越了传输层和应用层. RPC 使得开发包括网络分布式多程序在内的应用程序更加容易.

RPC 采用客户机/服务器模式. 请求程序就是一个客户机, 而服务提供程序就是一个服务器. 首先, 客户机调用进程发送一个有进程参数的调用信息到服务进程, 然后等待应答信息. 在服务器端, 进程保持睡眠状态直到调用信息的到达为止. 当一个调用信息到达, 服务器获得进程参数, 计算结果, 发送答复信息, 然后等待下一个调用信息, 最后, 客户端调用进程接收答复信息, 获得进程结果, 然后调用执行继续进行.

1.2 OSGi

OSGi[2](Open Service Gateway Initiative), 一个基于 Java 语言的服务(业务)规范, 定义了一个优雅、完整和动态的组件模型. OSGi 环境的应用程序都是以 Bundle 的形式存在, 各个 Bundle 之间物理隔离. Bundle 可以在系统运行期被远程安装、启动、升级和卸载(其中 Java 包/类的管理被详细定义), 每个 Bundle 都可以被单独管理.

Bundle 之间的依赖分为 2 种方式, 一种是通过 Export 和 Import 包的方式. 另一种是通过 OSGi 服务的方式, 一个 Bundle 作为服务提供者, 对外提供服务, 使用者可以查找到这个服务, 并使用这个服务.

2 总体设计

2.1 总体架构

在本文所述的集中日志管理的情况下, 各个应用系统的各台服务器会统一向日志存储服务器发送日志, 日志存储服务器收到各节点的日志数据后, 存储到数据库中. 如果日志记录频繁, 日志存储并发大, 可以部署多台日志存储服务器, 各个应用系统可以发送到不同的日志存储服务器.

日志分析服务器会读取日志存储服务器中的日志数据, 开发人员通过自己电脑连接日志分析服务器, 在界面中对日志数据进行分析. 如果应用系统出现错误, 开发人员能够通过日志分析界面快速查找出错误日志. 总体的部署拓扑图如图 1.

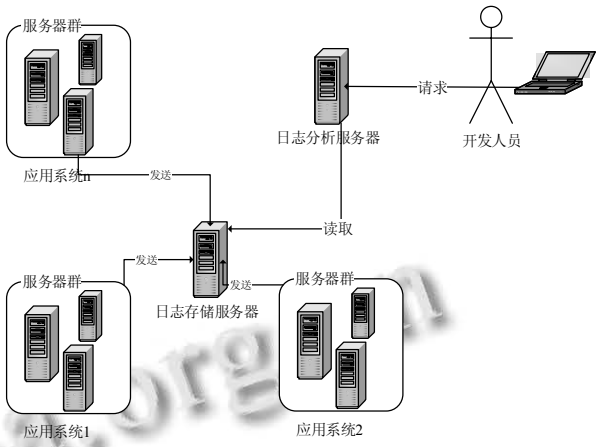

图 1　分布式日志收集与分析部署拓扑图

这种部署方式让日志数据能够统一收集、统一管理、统一分析, 改变了传统的日志管理方式.

2.2 技术架构

考虑到集中日志管理系统会产生大量日志数据, 需要通过内存缓冲技术降低磁盘 IO 压力, 考虑到内存占有量大, 所以把日志缓冲组件单独部署在应用系统服务器中, 与应用程序不在一个进程[3]. 应用程序调用记录日志程序, 通过 RPC 方式把日志数据传输给本地的日志收集组件. 本地日志收集组件接收到日志数据后, 再通过 RPC 方式将日志发送给集中日志存储服务器上的日志收集组件, 为了保证日志数据不丢失, 该日志收集组件会对日志数据进行缓冲处理, 之后通

过队列方式把日志存储到分布式文件系统 HDFS 中, 为日志集中分析提供数据依据.

日志分析服务器相对简单, 实现了一套通用的日志分析图形化界面. 使用者可以自己实现或者在该基础上进行扩展.

日志收集和分析组件技术架构图如图 2, 下面的章节会说明具体的实现方式.

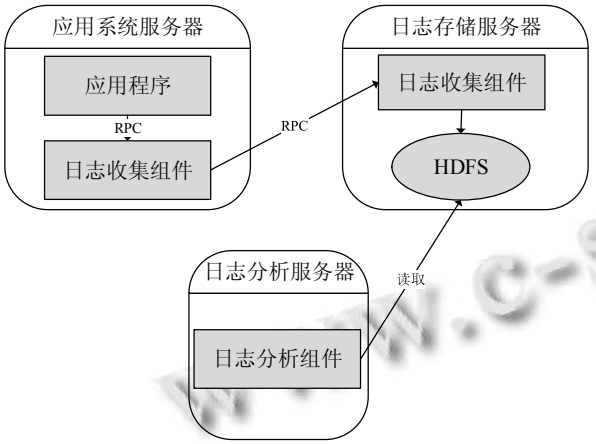

图 2 分布式系统集中日志收集与分析技术

3 分项设计

3.1 日志通用模型

分布式环境下的日志模型要比普通环境的日志模型多很多属性, 普通的日志模型应该含有日志编码、日志级别、日志主要内容、日志详情和创建时间. 在分布式环境下为了识别应用和服务器, 模型属性中应该增加应用名称和服务器 IP. 具体见图 3.

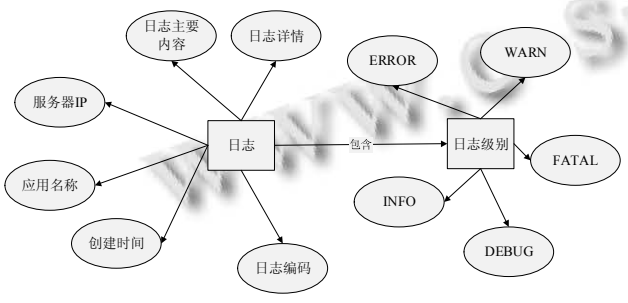

图 3 日志通用模型图

日志数据都会有日志级别, 用来表明输出该日志的问题严重性. 日志级别从低到高一般最少包含以下五个:

DEBUG: 用于调式程序, 记录程序运行过程, 级别最低;

INFO: 用于记录程序运行过程中比较重要的信息;

WARN: 对潜在危险做出警告, 程序仍然经常运行;

ERROR: 程序出现错误, 但不影响系统正常运行;

FATAL: 程序出现致命错误, 导致系统停止运行.

日志元数据按照上述模型进行封装后再传输到日志存储服务器, 存储之前先把元数据的属性都解析出来, 存放到数据库中.

3.2 日志收集组件

日志收集组件主要包含 log-collector、log-channel 和 log-sender 三个 OSGi 模块, 这三个模块都正常运行, 才能保证日志高效稳定的传输到日志存储服务器. 日志收集的基本架构图如图 4.

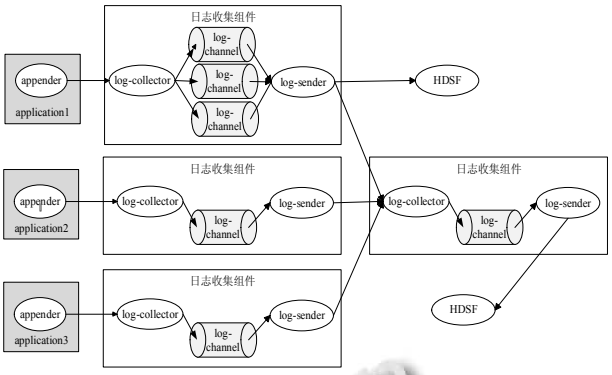

图 4 日志收集组件架构图

Appender 是应用程序在输出日志时触发日志输出事件而调用的一段程序, 程序会读取日志的输出级别、输出格式、哪些包需要输出日志、应用名称、当前服务器 IP 等信息, 把这些信息和日志内容一起封装起来传给 log-collector 模块.

Log-collector 为日志收集器, 可以收集来自另一个日志发送器发送的日志数据. 也可以同时收集多个日志发送器发送的数据, 并进行数据汇总[4].

Log-channel 为一个数据存储池, 用来缓冲来没来得及发送出去的日志. Memory Channel, Database Channel , File Channel. Memory Channel 用内存的方式存储日志数据, 可以实现高效的吞吐, 但是稳定性差, 一旦自身进程死掉, 数据会全部丢失. Database Channel 以数据库为存储介质, 效率不如 Memory Channel, 稳定性强, 不会造成数据丢失. File Channel 以文件为存储介质, 只要硬盘空间够, 数据就可以存

储进来. 一个日志收集器收集到的日志可以缓冲到多个 Channel 中.

Log-sender 为日志发送器, 可以直接存储日志数据到数据库, 可以传给其他消息传输中间件,如 JMS, 也可以把数据发送到下一个日志收集器. 基于这种实现方式, 可以把日志收集组件串联起来, 这样可以减轻单个日志收集组件的压力.

3.3 日志分析组件

日志分析组件相对简单, 实现了一套默认的日志分析图形化界面. 主要包含 3 个功能模块:

实时展现模块. 以图表方式对日志数据进行实时展现, 定时刷新日志数据. 可以自己定义展现规则.

预警模块. 当单位时间出现 ERROR 级别以上的日志条数超过阀值时, 给运维人员发出警告.

智能分析模块. 日志列表页面, 可以按照日志元数据的各个字段的值进行查询, 能够快速定位到想查看的日志.

用户可以根据自己的需求开发日志分析页面, 也可以在日志分析组件的基础上扩展. 日志分析组件的架构图如图 5.

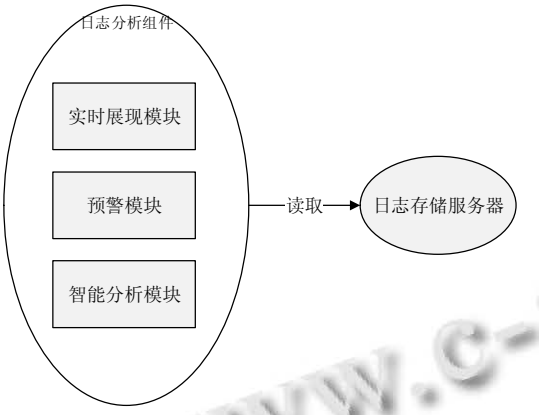

图 5　日志分析组件架构图

4　系统实现

4.1 日志收集组件

日志收集组件主要包含 Collector 类、Queue 类、Sender 类和 ConfigProperty 类. 静态 UML 结构图如图 6.

Collector 类负责收集日志, Queue 类负责提供日志缓冲队列, Sender 类负责发送日志. ConfigProperty 类负责解析各种配置, 见 ConfigProperty 类的属性. 主要包含收集器类型(序列化类型)、收集器端口、收集器绑定地址、缓冲队列类型、缓冲区大小、发送器发送地址、端口、类型等. 这些配置为日志传输过程提供重要参数.

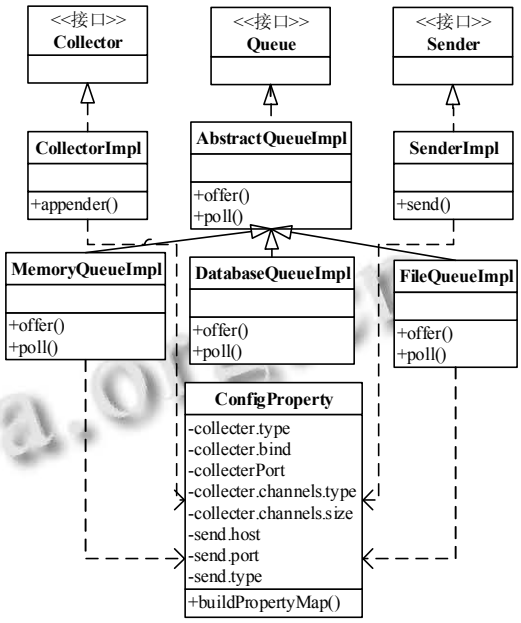

图 6　日志收集组件 UML 静态结构图

日志收集组件序列图如图 7.

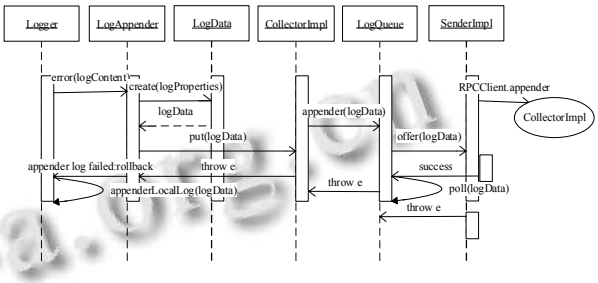

图 7　日志收集组件序列图

应用程序调用 Logger 对象的 error 方法, 触发日志输出事件, 触发事件后, LogAppender 调用 put 方法向 CollectorImpl 类发送一条日志数据. 日志数据除了普通日志数据中的日志内容、日志创建时间、日志级别外, 还有服务器 IP 和应用名称. 日志数据由 LogData 的构造方法返回. CollectorImpl 接收到日志数据后, 并不会马上把日志数据发送给 SenderImpl, 而是会先缓冲到 LogQueue 中, LogQueue 为一个数据存储队列, 先放入队列的日志会被先发送给 SenderImpl(参见图 7). LogQueue 调用 offer 方法向日志数据队列中插入数据. 一旦 SenderImpl 空闲, 队列会按先进先出的原则给

SenderImpl 发送一条日志数据. SenderImpl 收到数据后调用 RPCClient 把日志发送到下一个 CollectorImpl, 也可以直接存储数据到分布式文件系统 HDFS.

日志收集过程中某一个环节出现问题, 导致日志存储失败, 做出日志回滚操作, 调用 Logger 的 appenderLocalLog 方法把日志存储到本地文件中.

LogQueue 为 log-channel 的主要部分, LogQueue 默认把数据存放在内存中. 当把 channel 的类型配置成 Database 或者 File 时, 会把日志缓冲到数据库或者文件中.

在日志输出数据量大的情况下, 日志缓冲区的数据量会非常大. SenderImpl 单线程无法保证缓冲区的数据及时发送出去. 如图 8 所示, 初始化一个线程池 SendThreadPool, 线程池中存放多个 SenderImpl 线程同时处理日志数据, 可以有效缓解日志输出高峰期时日志缓冲区的挤压, 让日志高效稳定的传输到下一个环节.

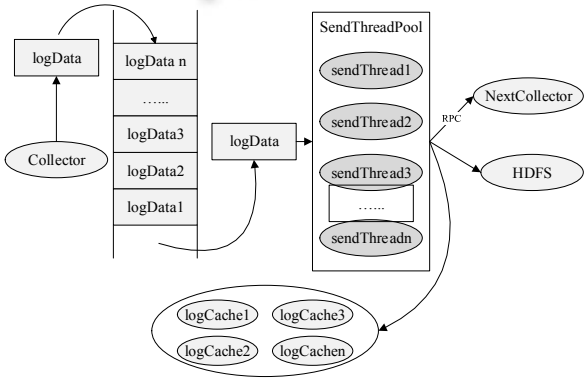

图 8 日志收集组件流程图

日志发送是通过 RPCClient 的 appender 方法完成, RPCClient 发送的数据, RPCServer 在接收完成并成功处理之后, 会告诉 RPCClient 数据发送成功. SenderImpl 在发送日志数据之后, 如果不做任何处理, RPCServer 接收日志失败会造成数据丢失. 为了防止数据丢失, 如图 8 所示, 在日志发送后, 先把日志数据缓存起来, 在 RPCServer 返回发送成功的标识之后, 再把缓存数据删除. 否则, 把缓存日志数据存储到本地日志文件中.

分布式环境下, 日志的数据量远远超出普通环境日志数据量, 需要分布式文件系统来存储. RPCServer 在接收到日志数据之后, 会把数据交给分布式文件系统来处理, 如目前非常流行的 Hadoop. 也可以把日志数据发送给下一个日志收集组件的 Collector.

4.2 日志分析组件

日志分析组件默认提供了一些展现分析组件, 包含实时展现组件、预警组件和智能分析组件. 其 UML 静态结构图如图 9.

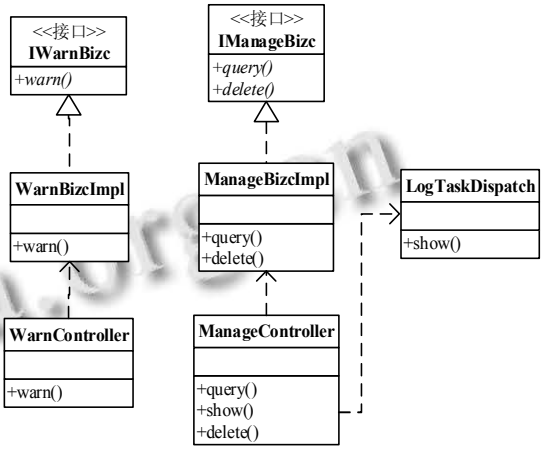

图 9 日志分析组件 UML 静态结构图

实时展现组件为一个定时任务组件, 由 LogTaskDispatch 类实现, 平均每 5 秒取一次日志数据, 刷新一下展现页面.

ManageController 为实时展现组件和智能分析组件提供数据访问入口. 在页面上访问实时展现组件和智能分析组件, 请求会进入 ManageController 来获取数据, ManageController 调用逻辑组件 ManageBizcImpl 和 LogTaskDispatch 来获取日志数据, 该两个逻辑组件再去日志存储服务器中获取数据. 页面点击预警功能, 请求会进入 WarnController 中, WarnController 在调用逻辑组件 WarnBizcImpl, WarnBizcImpl 从日志存储服务器获取日志数据.

日志分析组件是基于 OSGi 实现的, 利用 OSGi 扩展模块的特性可以方便的对分析组件的功能进行扩展.

5 实验结果

5.1 性能测试

本设计的核心为日志集中收集, 按方案实现了一套集中日志管理系统, 针对日志收集功能进行了性能测试. 对分布式日志收集和普通日志收集方式进行了对比. 测试服务器配置如表 1, 测试软件为 Loadrunner.

表 1 服务器环境

CPU	Intel i5-3230M CPU 2.60GHz
操作系统	Win7 64 位
内存	4.00G
应用服务器	Weblogic10.3.6
数据存储	Hadoop

测试环境定义了一个简单的业务场景, 在该业务场景下每次请求输出 10 条日志. 采用 Loadrunner 对该业务场景分别进行 300、500 和 1000 并发量测试, 每次压力测试进行 10 次求平均值, 测试结果如表 2.

表 2 集中日志管理性能测试结果

并发	普通日志收集耗时(秒)	分布式系统集中日志收集耗时(秒)
300	3.35	2.97
500	5.35	4.40
1000	8.23	6.28

由测试结果可以看出, 分布式日志收集的效率高于普通日志收集的效率, 并且并发数越高, 优势越明显. 在 1000 并发下访问的平均时间提高了 2 秒. 并且分布式环境下日志条数和正确性都和普通环境一样, 不存在数据丢失的问题.

5.2 应用案例

基于上述测试的良好结果, 该方案结合国网公司的网络大学系统进行了试运行. 网络大学系统为国家电网公司全部员工提供在线学习、在线考试等功能, 系统采用一级部署, 有用户访问量大、访问高峰集中等特点. 网络大学系统生产环境由 53 个节点组成. 在部署集中日志管理系统之前, 系统出现问题, 需要逐个节点排查问题, 效率较低; 部署之后, 系统运维人员反馈, 一旦系统出现问题, 排查和定位问题的效率明显高于以前.

6 结语

本文研究与实现了一套基于 OSGi 的分布式系统集中日志管理方案, 可以对日志集中收集和分析. 日志存储过程中采用了缓冲区和多线程发送技术, 日志收集性能高于普通的日志收集方式. 由于多台服务器中的日志数据被集中存储到一台服务器上, 管理起来也更加方便, 定位和分析问题的效率更高. 该集中日志管理采用 OSGi 扩展模块的方式进行扩展, 可扩展性强. 由于分布式环境下日志数据传输环境复杂, 传输过程中容易造成数据丢失. 下一步工作是加强各种突发情况下的日志收集数据完整性测试. 比如在突然断网的情况下, 是否能做到分布式日志收集和普通日志收集的无缝切换. 这些是下一步工作的重点.

参考文献

1 冯新扬,沈建京.REST 和 RPC:两种 Web 服务架构风格比较分析.小型微型计算机系统,2010,31(7):1393–1395.
2 刘丹,王宇飞,杨宁.一种基于 OSGi 的 Web 应用模块化架构设计.计算机系统应用,2014,23(1):62–67.
3 张煜.一种使用 Node.js 构建的分布式数据流日志服务系统.计算机系统应用,2013,22(2):68–71.
4 刘必雄,魏连,许榕生.基于 Agent 技术的多源日志采集系统的设计与实现.计算机系统应用,2008,17(2):71–74.