大型复杂软件产品持续集成的实践与反思

作者： wuzhimin

持续集成作为一种敏捷软件开发实践，已经被越来越多的开发者所接受。持续集成倡导开发团队频繁地进行系统集成——通常一天一次到数次，每次集成都能被自动编译和测试验证，从而能在最短的时间内发现问题，缩短开发周期，提高软件质量。

笔者面对的是具有十多年开发维护历史，的 5 个相互依赖产品，每个产品均超过百万行代码的复杂系统。集成本身涉及很多烦琐的手工操作，很难实现过程自动化。在实施过程中，受困于软件系统的历史遗留问题，而通常市面上的持续集成工具又不能满足系统的需求，让我们不得不着手开发自己的集成系统。经过近一年的持续努力，终于完成了系统集成的自动化，将集成频度从数周甚至数月集成提高到日集成，大大提高了生产效率。

集成的困境

OSS 系统是我们一个具有十多年生命历程的复杂系统，多年来一直处于不断的开发和维护中，子产品的数目和系统的代码量也随着时间日益增长，集成的周期、发布周期也随之逐渐增长。OSS 系统包含有 5 个需要集成的产品：ComLIB 、DB-Com、Data Broker、DataGen、DataAgent，每个产品均有上百万行源代码和庞大的测试用例，产品可以单独构建，但运行时有相互依赖关系。OSS 产品栈如图 1 所示，上层产品对下层产品具有依赖，部分产品栈亦可以组成一个应用系统，如 DataGen+CommonLib 可以组成一个应用系统，DataBroker+ DBCom+CommonLib 也可以组成一个应用系统。

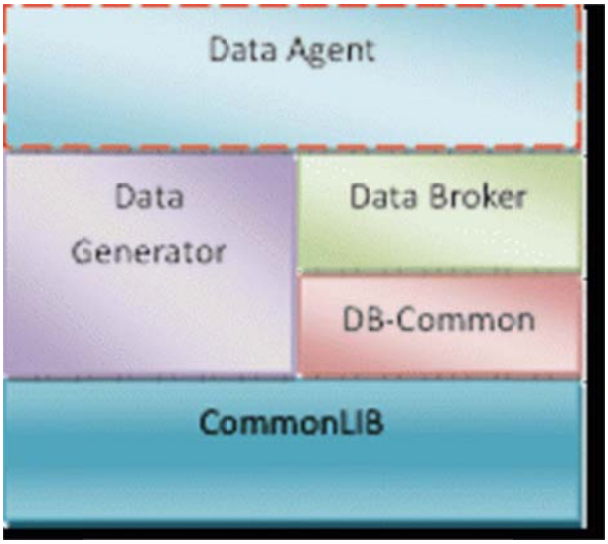

图 1 OSS 产品栈

同时，OSS 的每个产品又包含不同的发布单元，用于运行于不同功能服务器上：Control Server、SITE、PAP、Workstation、Application Server 和不同的操作系统环境中，产品在不同的服务器上安装单元如表 1 所示。

	SERVER	SITE(P)	Workstation	Application Server1
CommonLIB	CommonLIB-ENA CommonLIB-RUN	CommonLIB-ENA CommonLIB-RUN	CommonLIB-ENA CommonLIB-RUN	CommonLIB-ENA CommonLIB-RUN
DBC	DBC-ENA DBC-RUN	DBC-ENA DBC-RUN	DBC-ENA DBC-WS-RUN	DBC-ENA DBC-AP-RUN
Data Gen	DataGen-SERVER-RUN DataGen-ENA DataGen-VAR DataGen-LIB	DataGen-SITE-RUN DataGen-ENA DataGen-VAR DataGen-LIB	DataGen-WS-RUN DataGen-VAR-RUN DataGen-ENA-RUN	DataGen-AP1-RUN DataGen-VAR-RUN DataGen-LIB
Data Broker	DB-ENA DB-SERVER-RUN	DB-SITE-RUN DB-SITE-ENA DB-LIB	DB-WS-RUN DB-VAR-RUN DB-ENA-RUN	DB-AP1-RUN DB-VAR-RUN DB-LIB
Data Agent	DataAgent-SERVER-RUN DataAgent-ENA DataAgent-VAR DataAgent-LIB	DataAgent-SITE-RUN DataAgent-ENA DataAgent-VAR DataAgent-LIB	DataAgent-WS-RUN DataAgent-VAR-RUN DataAgent-ENA-RUN	DataAgent-AP1-RUN DataAgent-VAR-RUN DataAgent-LIB

表 1 OSS 产品在不同服务器上的安装单元

传统手工集成需要完成如下事情：

1．从代码库取出产品栈中每个产品需要集成的代码；

2．在 Build server 上对每个产品进行编译，单元测试（每个产品编译需要 2~3 小时，单元测试需要 4~5 小时）；

3．在 Package server 上对编译好的产品进行打包（每个产品打包需要约 1 小时）；

4．在 Lab 中卸载旧的产品；

5．在不同的服务器上（Control Server、Site、Workstation、APP Server）安装产品中相应的产品的相应部件，需要按照产品依赖关系进行安装　（2~3 小时）；

6．检查每个产品的每个部件在 Lab 环境中的安装情况，确认安装成功；

7．运行功能测试脚本做回归测试（10~12 小时）；

8．检查测试结果（3 小时）；

9．对系统进行卸载测试，检查卸载结果　（1~2 小时）。

如果在此过程中，一旦遇到问题，则需要花费更长的时间。如此昂贵的集成代价使得 OSS 系统通常数周甚至更长的时间才艰难地进行一次集成。而自动化还要解决多产品、多平台、多种安装和运行环境、多个操作需要人工干预的难题。

集成目标

持续集成需与自动化结合才能发挥其威力，为给 OSS 系统瘦身，我们设定了如下自动化持续集成目标：

1．在集成服务器上配置 cron job，每天晚上进行集成，第二天早上发布结果；

2．在集成服务器上配置集成方案，指定所需要的产品栈或子栈，指定所需产品的版本或分支，指定需要进行卸载、安装测试，功能测试的 Lab 以及 Lab 里的服务器；

3．只需修改少量参数配置，即可以实现不同的持续集成方案。

自动化实现

通过一系列自动化工具的开发和整合，我们逐步完成了多产品编译、测试、安装、卸载、功能测试等一系列步骤的自动化操作，为了缩短集成时间，我们广泛地采用并行工作，在编译、测试、安装、卸载等多个环节均采用并行，最后实现了成熟的集成系统。

1．自动化构建+并行编译

首先完成 5 个产品统一的 Daily build。对 5 个产品采用统一的编译、单元测试、打包接口，利用不同的编译服务器对 5 个产品同时进行编译，对同一产品的不同编译单元，也采用负荷分担的方式分发到多台服务器上进行并行编译、并行单元测试、并行打包，并对编译、单元测试结果进行统一管理。

2．自动化监视

通过监视编译、打包、测试过程的输出和进程状态等对编译、打包、测试过程进行监控，发现诸如 NFS 错误或者编译系统故障之类非代码错误及时取消并重启操作，确保 Daily build 不会因为编译服务器等异常发生中断。

3．自动部署安装

利用脚本工具打包好的系统进行自动并行安装，免除手动安装的烦琐操作。当所有待安装的产品都编译、打包完毕后，利用脚本工具将集成后的系统并行安装到多个 Lab（每个 Lab 由一套不同机器的设备组成，包含 Control Server、Site、WorkStation、Application Server），对于同一 Lab 中的不同机器亦进行并行部署，并对安装日志进行统一管理。

4．自动功能测试

卸载、安装操作成功后，即对选用的测试用例进行测试，测试用例经过精心挑选，根据功能分为不同的核心单元，周一到周五只运行核心的测试用例，周末时间再运行所有的用例。用例无法在一个 Lab 一个晚上运行时，则将用例分布配置在多个 Lab 里面，并行运行。

5．结果汇报及存档

定点（如早上 9:00）对前一天晚上进行的所有操作进行汇报，包括不同产品的编译、打包、测试，不同 Lab 上不同服务器的卸载、安装结果，不同 Lab 上功能测试用例通过情况等信息自动以邮件的方式通知开发和测试人员，并将结果存档方便随时查看，对其中的关键数据如测试用例通过率等写于数据库，便于跟踪分析。

自动测试统计结果如图 2 所示。

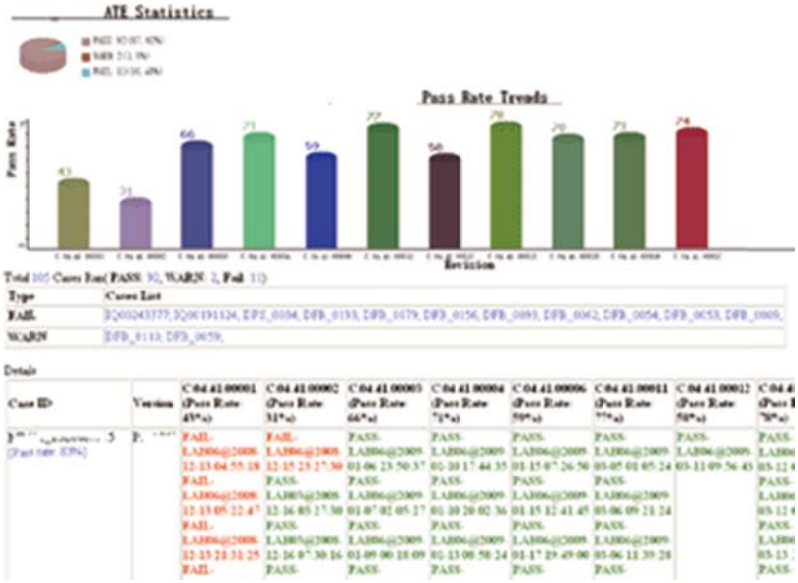

图 2 自动测试统计结果

6．流程整合和协作

流程 1~2、5 发生在编译服务器群上，流程 3~4 发生在运行环境 Lab 群上，需要将 1~5 流程集成自动化，并且流程间需要相互协作。我们采取集中控制的方式，将 1~5 所需要脚本工具和配置文件集中放置，这样便于管理和维护，其他服务器通过 NFS 共享访问脚本工具和配置文件，只需要在其他服务器上配置相关的 cron job，便能相互协作起来。协作流程如图 3 所示。

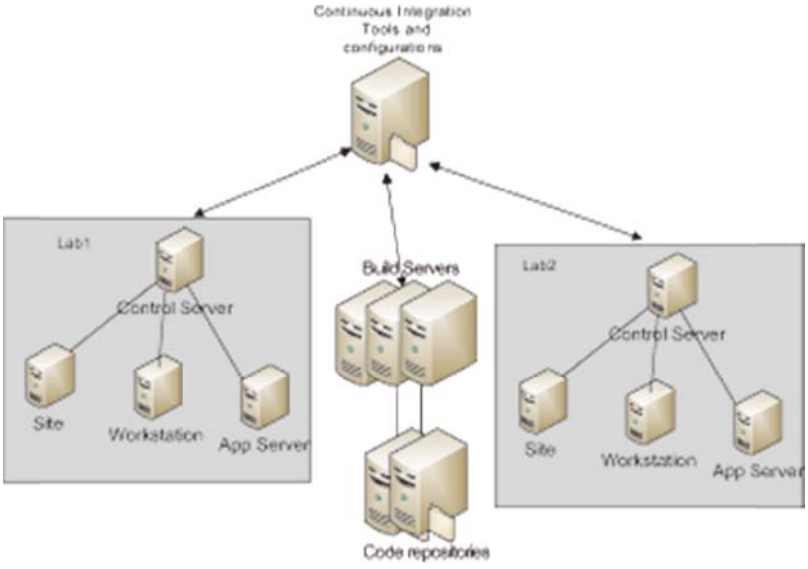

图 3 协作流程示意图

回顾与反思

经过近一年的完善，我们的 OSS 系统终于从之前的数周才能一次手工集成，达到了每天集成，许多人工操作和检查统计工作均自动化，不仅仅大大解放了 Build Manager 的工作，

而且避免了人工差错，提高了效率。对于像 OSS 这种历史悠久的复杂系统，持续集成是一个长期优化、完善的实践过程。毕竟持续集成不是一朝一夕就可以实现的，而应该成为一种持之以恒的习惯。

总之，一个完善的运行良好的持续集成，必须考虑如下问题：

1．持续集成的频度和粒度

每个开发人员提交代码更改之前，在自己的 branch 上需要确保编译、单元测试通过，避免不必要的集成失败，只有开发人员确认的可靠代码，才进行集成。对于小型系统可以做到每次提交都集成，对于像我们 OSS 这样的大型系统，可以做到每天一集成。

2．统一构建、并行构建

对于多产品的系统，采用统一的构建、打包、安装方式有助于灵活实现自动化。并行构建、并行测试、并行安装等可以充分利用服务器物理资源缩短集成时间。

3．协作整合

在并行的同时，也需要采用合适的同步协作的方法，以保证系统正确运行。如编译、打包、单元测试的进程在编译服务器群上，安装、卸载、功能测试等进程在运行系统的 Lab 服务器群上，而邮件通知则又需要将这些进程运行的结果整合起来。简单的文件或目录共享远程调用往往就能完成这些控制权的转移，实现协作和工具整合。

4．高性能 build server

高性能的编译服务器或服务器群也是持续集成中强有力的支撑。支持并行编译的话将更好。

5．完善的结果报告

完善的结果报告能缩短开发或测试人员发现、解决问题的时间。完整有效的报告应该包含软件版本信息，本次集成中所做的修改，单元测试和回归测试结果，卸载/安装结果。

6．持续集成工具

在持续集成实践过程中，一个完善的 SCM 系统（如 ClearCase、SVN、CVS），分支并行开发的策略几乎是必备的支撑。同时，如果在 Unix 或 Linux 环境下，亦无须迷信于专门的持续集成工具，有些时候，借助各种脚本工具与 cron job 完全可以满足这些需要。

7．自动化

持续集成必须跟自动化结合，才能真正的发挥其威力。