АНУФРИЕВ Владимир Натанович

Шлюз Domino – Tomcat

Москва
2007

Vladimir Anufriev

Tomcat 5.0 and Lotus Domino 6.5.5 integration

Comments may be addressed to: avn@avnsite.com

Введение

В версии 6.0 компания IBM[1] удалила из своего сервера приложений Domino[2] рудиментарный собственный обработчик JSP[3] и предложила разработчикам, применяющим эту технологию, использовать либо сервер приложений Web Sphere[4] либо приложения других поставщиков. При этом IBM поддерживает собственный набор тегов для упрощения написания JSP-сервлетов, обращающихся за информацией к серверу Domino.

Вместе с тем, Domino содержит HTTP/1.1-совместимый сервер с поддержкой SSL, инструментальные средства разработки приложений (HTML, XML, Lotus Script, @-формулы, Java, JavaScript) на основе GUI и позволяет на одной вычислительной установке достаточной производительности поддерживать протоколы NOTES (почта и сервер приложений), SMTP, POP3, IMAP, LDAP, HTTP, а также их варианты со встроенной криптографической защитой. Такое совмещение функций позволяет максимально снизить издержки на системное администрирование и содержание аппаратного парка при достаточной масштабируемости для предприятия среднего размера[5].

Таким образом, применение развитой и привычной технологии JSP при использовании Domino HTTP-сервера требует решение проблемы интеграции с серверами Java-приложений.

Обзор существующих решений

В стек включенного в Domino HTTP-сервера IBM встроила так называемые *внешние фильтры* (DSAPI filter files), которые позволяют разработчикам применять собственные приложения на языке Си для изменения стандартных результатов поэтапной обработки HTTP-запросов или подмены встроенных обработчиков. Приложения пишутся с использованием интерфейса DSAPI и оформляются в виде библиотек функций с поздним связыванием (dynamic link libraries). DSAPI входит в состав Notes C API, который может быть загружен с сайта IBM[6]. Большая часть примеров, которые можно извлечь из документации к DSAPI и непосредственно с сайта IBM посвящены подмене процедуры аутентификации для HTTP-приложений в среде Domino. Со своей стороны, отметим, что данные примеры носят скорее академический характер, так как сервер Domino обладает довольно развитыми аутентификационными и авторизационными возможностями. Скорее, интересен ответ на вопрос, возможна ли аутентификация внешних приложений с помощью механизмов само**го** сервера Domino, который, в частности, содержит встроенный LDAP, поддерживает единый вход (Single SignOn – SSO), кластеризацию и распределение нагрузки.

Единственным открытым решением, использующим DSAPI для интеграции с сервером приложений Tomcat[7], является шлюз DSAPI plugin for Lotus Domino (другое наименование - Domino Tomcat Redirector) от Andy Armstrong (andy@tagish.com). Данный шлюз до последнего времени входил в состав дистрибутивов Tomcat. Спасибо Andy за то, что он, поискав в свое

1 http://www.ibm.com
2 http://www-142.ibm.com/software/sw-lotus/products/product4.nsf/wdocs/dominohomepage
3 http://java.sun.com/products/jsp/
4 http://www-306.ibm.com/software/websphere/
5 Один x86-сервер стоимостью до $20000 в масштабе цен и аппаратных возможностей 2001 года, сопровождаемый одним системным программистом, на практике позволяет обслуживать до 2000 пользователей в корпоративной сети.
6 http://www-128.ibm.com/developerworks/lotus/downloads/toolkits.html – Notes C Api
7 http://tomcat.apache.org/

время готовое решение (переписка до сих пор сохранилась в некоторых форумах) и не найдя его, создал эту программу, которая стала эталонным примером интеграции Domino и свободно распространяемого сервера приложений. Шлюз преобразует протокол вызовов внешних фильтров при обработке HTTP-запроса сервером Domino, доступный ему через интерфейс DSAPI, в протокол AJP 1.3[8], который использует так называемый *коннектор* (не будем его называть шлюзом, чтобы не путать с основным предметом данной статьи) сервера приложений Tomcat. Общая схема работы описываемых здесь приложений приведена на рисунке 1.

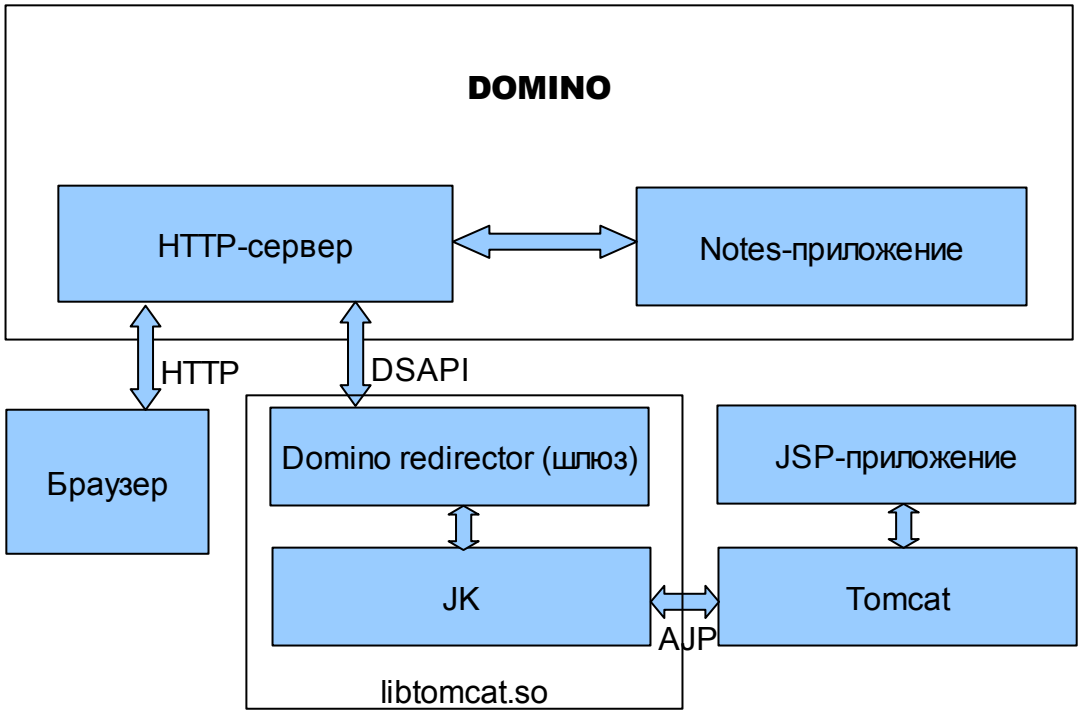

Рис. 1. Взаимодействие программ

В Интернете можно найти множество старых неработающих или несовместимых с современными версиями Tomcat вариантов интересующей нас программы с разбросом в номерах версий от 1.0 до 2.0.1 под Windows[9] и Linux под различными именами. В частности, на сайте проекта Jakarta можно найти ссылки на вариант Domino Tomcat Redirector, который компилируется с веткой JK2 шлюза Tomcat-Apache[10], ранее известного как mod_jk, а в последующем как JK. Поддержка ветки JK2 в настоящий момент прекращена[11] и приходится искать вариант для JK.

Оригинальный сайт Andy[12] дает ссылки на первоначальные древние варианты шлюза, которые практически невозможно применить в современном окружении. Множество ссылок на различные модификации шлюза с названиями как dsapi так и domino, наличие большого количества версий без пояснения отличий в них, а также тот факт, что разработчик отлаживал и применял свой шлюз в среде Microsoft Windows, чрезвычайно затрудняют поиск версии,

8 http://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html
9 Работоспособность версии 2.0.1 распространяемой в исполнимых кодах под Windows без исходных текстов мы не проверяли
10 http://www.jajakarta.org/tomcat/tomcat-jk2/en/docs/index.html
11 С 15 ноября 2004 года JK2 более не поддерживается, сообщение об этом факте можно прочесть на: http://tomcat.apache.org/connectors-doc/news/20041100.html
12 http://free.tagish.net/domino-tomcat/

подходящей для применения в Linux.

Работоспособный вариант шлюза, причем как в варианте с интерфейсом JK так и JK2, нашелся в составе дистрибутива Tomcat 5.0, поставляемого с SuSE Linux версий 9.2-10.x[13]. Мы проверили варианты Tomcat 5.0.27 из SuSE 9.2 и Tomcat 5.0.30 из SuSE 10.0. Шлюз из вышеупомянутой версии Tomcat 5.0.27 компилируется и работает в заявленном режиме без каких-либо существенных изменений. Версии идентифицируются в тексте программы на языке Си как 1.13 для коннектора JK и 1.14 для JK2.

Основными недостатками программы Andy являются:

- применение только одного события (вызова процедуры внешнего фильтра) DSAPI kFilterParsedRequest для полной обработки HTTP-запроса, что не позволяет

13 Версия SuSE Linux для исследовательских целей доступна с сайта разработчика: http://www.novell.com/linux/

До последнего времени дистрибутивы Linux от компании SuSE (приобретена Novell в 2006 году) были самыми «вылизанными» среди всех альтернатив. Версию Tomcat с исходными текстами шлюзов (они называются, соответственно, domino для версии JK и dsapi для версии JK2) можно взять здесь: ftp://ftp.suse.com/pub/suse/i386/9.2/suse/src/tomcat5-5.0.27-9.1.src.rpm К сожалению смена хозяина компании негативным образом отразилась на качестве сборки дистрибутива, и настройка Tomcat 5.0.30 из поставки SuSE 10.0 представляет собой довольно тягостный процесс, так как его административное приложение сразу после установки функционирует некорректно.

В вышеуказанной версии Tomcat не устранена хорошо известная ошибка http://issues.apache.org/bugzilla/show_bug.cgi?id=32381. К сожалению, приложенные на странице bugzilla патчи касаются версии Tomcat 5.5, поэтому в нашем случае патчи будут выглядеть так (длинные строки в приведенном ниже тексте свернуты по ограничениям ширины страницы):

```
       --- jakarta-tomcat-5.0.30-src/jakarta-tomcat-catalina/webapps/admin/WEB-
INF/classes/org/apache/webapp/admin/TreeControlTag.java 2004-11-24 19:55:28.000000000 +0300
       +++ jakarta-tomcat-5.0.30-src/jakarta-tomcat-catalina/webapps/admin/WEB-
INF/classes/org/apache/webapp/admin/TreeControlTag.java 2007-02-03 23:23:06.000000000 +0300
       @@ -88,7 +88,7 @@

            /**
             * The hyperlink to be used for submitting requests to expand and
       -     * contract tree nodes.  The placeholder "<code>${name}</code>" will
       +     * contract tree nodes.  The placeholder "<code>{name}</code>" will
             * be replaced by the <code>name</code> property of the current
             * tree node.
             */
       @@ -350,11 +350,11 @@
                // character in parameter values.
                String encodedNodeName = URLEncoder.encode(node.getName());

       -        String action = replace(getAction(), "${name}", encodedNodeName);
       +        String action = replace(getAction(), "{name}", encodedNodeName);


                String updateTreeAction =
       -            replace(getAction(), "tree=${name}", "select=" + encodedNodeName);
       +            replace(getAction(), "tree={name}", "select=" + encodedNodeName);
                updateTreeAction =
                    ((HttpServletResponse) pageContext.getResponse()).
                    encodeURL(updateTreeAction);
       --- jakarta-tomcat-5.0.30-src/jakarta-tomcat-catalina/webapps/admin/tree-control-test.jsp
24 Jul 2002 20:57:25 -0000      1.2
       +++ jakarta-tomcat-5.0.30-src/jakarta-tomcat-catalina/webapps/admin/tree-control-test.jsp
1 Dec 2004 22:30:30 -0000
       @@ -20,7 +20,7 @@

        <td width="200">
          <controls:tree tree="treeControlTest"
       -                 action="treeControlTest.do?tree=${name}"
       +                 action="treeControlTest.do?tree={name}"
                          style="tree-control"
                 styleSelected="tree-control-selected"
               styleUnselected="tree-control-unselected"
```

расширять шлюз с целью интеграции с аутентификационными и авторизационными механизмами сервера Domino;

- большое количество лишнего кода, который пытается обработать ожидаемую от libhttpstack.so информацию, в частности значения переменных окружения, которые в современных версиях Domino не устанавливаются;
- имя аутентифицированного Domino пользователя протоколу AJP не передается, поэтому невозможно реализовать SSO между Tomcat и Domino;
- все запросы, попадающие в шлюз, обрабатываются Tomcat, что не позволяет интегрировать в рамках одного виртуального www-сервера приложения Domino (базы данных NSF) и JSP;
- шлюз использует разделенные статические данные, изменяемые вне критических секций, что может приводить к непредсказуемым последствиям в нагруженных приложениях;
- поддержка шлюза и сами исходные тексты исключены из текущих поставок Tomcat (в частности, версий JK 1.19, 1.20);
- программа изначально создавалась, отлаживалась и применялась в среде Microsoft Windows, поэтому, в частности, Makefile требует существенной доработки.

Таким образом, в целях применения в корпоративной среде на серверах с операционныой системой Linux существующий шлюз было необходимо переработать с целью устранения хотя бы части вышеперечисленных недостатков.

Интерфейс DSAPI

Интерфейс DSAPI свободно доступен с сайта IBM по ссылкам, приведенным ранее, поэтому не будем вслед за Andy расстраиваться из-за невозможности легально распространять необходимые файлы заголовков совместно с текстом Domino-шлюза. Не будем здесь заниматься и переводом документации, каждый желающий может ее прочитать непосредственно в Интернете[14].

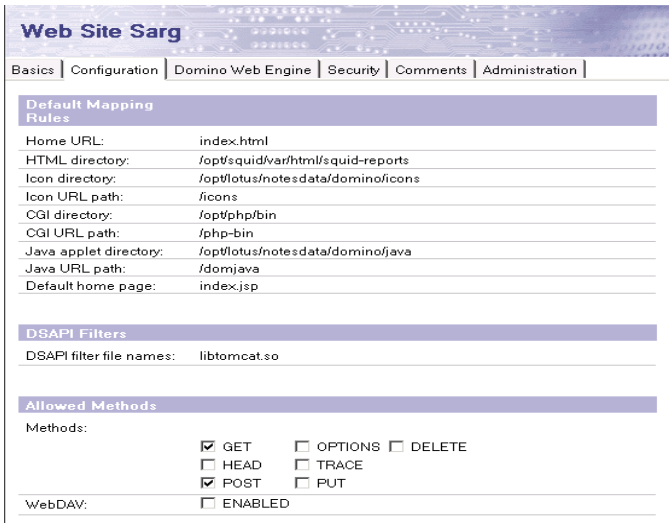

Рис. 2. Настройка DSAPI фильтра

В процессе обработки HTTP-запроса сервер Domino делает ряд вызовов процедур

14 http://www-12.lotus.com/ldd/doc/tools/c/6.5/api65ug.nsf – Notes C API User's Guide
http://doc.notes.net/tools/c/6.5/api65ref.nsf – Notes C API Reference

указанного в настройках внешнего фильтра (см. рис. 2) в определенной последовательности, которая, с точки зрения IBM, отражает внутренние этапы выполнения запроса.

Один вызов делается в момент инициализации шлюза (загрузки HTTP-сервера, автоматически или по команде с консоли Domino 'load http'). При этом из внешней библиотеки вызывается функция, прототип которой приводится ниже:

```
/*--- * filter initialization */
unsigned int FilterInit( FilterInitData *pFilterInitData );
```

Данный вызов предполагает, что шлюз должен сделать внутренние инициализации и передать HTTP-серверу информацию о том, какие события, возникающие в процессе обработки HTTP-запроса, он будет обрабатывать. Или, другими словами, на каких этапах обработки HTTP-запроса сервер Domino должен вызывать нижеописанную функцию шлюза:

```
/*--- * filter notification handling */
unsigned int HttpFilterProc( FilterContext *pContext,
                             unsigned int eventType,
                             void *pEventData );
```

Данная функция получает от сервера Domino информацию через указатель pEventData, характер и состав этой информации зависит от этапа обработки запроса. Указатель pContext на протяжении обработки всего запроса не меняется, позволяя однозначно идентифицировать запрос в многопоточной среде. В поле eventType передается номер этапа обработки запроса. Последовательность этапов описана ниже.

По окончании обработки запроса сервер вызывает процедуру TerminateFilter, в которой шлюз освобождает все ресурсы, занятые в ходе обработки данного запроса.

```
/*--- * filter termination */
unsigned int TerminateFilter( unsigned int reserved );
```

К сожалению, этапы обработки HTTP-запросов у сервера Domino и у Tomcat совершенно разнятся, что создает существенные трудности при организации взаимодействия этих программ. В эталонном шлюзе проблема решается простейшим образом, используется только один этап обработки, на котором производятся все инициализации, полностью выполняется запрос и результат возвращается HTTP-серверу для пересылки пользователю. События отличные от kFilterParsedRequest не обрабатываются. Данный подход имеет одно неоспоримое достоинство – предельную простоту написания и отладки обработчика. Однако целый ряд существенных данных сервер Domino передает шлюзу только на последующих этапах. Главным образом, это касается аутентификационной информации.

В общем случае обработка HTTP-запроса выполняется сервером Domino в следующей последовательности (здесь этапы именуются в соответствии с именами соответствующих констант):

1. kFilterStartRequest
2. kFilterRawRequest
3. kParsedRequest
4. kFilterRewriteURL
5. kFilterAuthenticate
6. kFilterUserNameList
7. kMapURL

8. kFilterTranslateRequest
9. kFilterPostTranslate
10. kFilterAuthorized
11. kFilterProcessRequest
12. kFilterEndRequest

На каждом этапе шлюз может изменить стандартный результат, возвращаемый клиенту. Большая часть этапов относится к подготовительной стадии обработки запроса. Основная стадия (формирование HTTP-кадра для отправки клиенту) проводится на этапе kFilterProcessRequest. Но для того, чтобы сервер обратился к шлюзу для выполнения этой основной стадии, на одном из первых этапов шлюз должен указать серверу, что основная стадия данного запроса будет обрабатываться им и никем больше[15]. В нашем случае это делается на этапе kParsedRequest вызовом функции сервера:

```
...
ok = context->ServerSupport(context, kOwnsRequest, NULL, NULL, 0, &errID);
...
```

К сожалению, необходимость принятия решения об обработке запроса шлюзом ранее прохождения авторизационного этапа сервером Domino (стандартный обработчик выполняет свою работу после вызовов всех, прописанных в стеке внешних фильтров) и отсутствие промежуточных этапов между kFilterAuthorized и kFilterProcessRequest не позволяют использовать встроенные механизмы авторизации Domino-сервера. Авторизацию Domino, в связи с этим, можно использовать только как дублирующее решение, что в одном приложении вряд ли приемлемо. Если в настройках (см. рис. 3) задать правила авторизации для URI, то сервер Domino заставит вас ввести аутентификационную информацию через свой механизм независимо от сервера Tomcat.

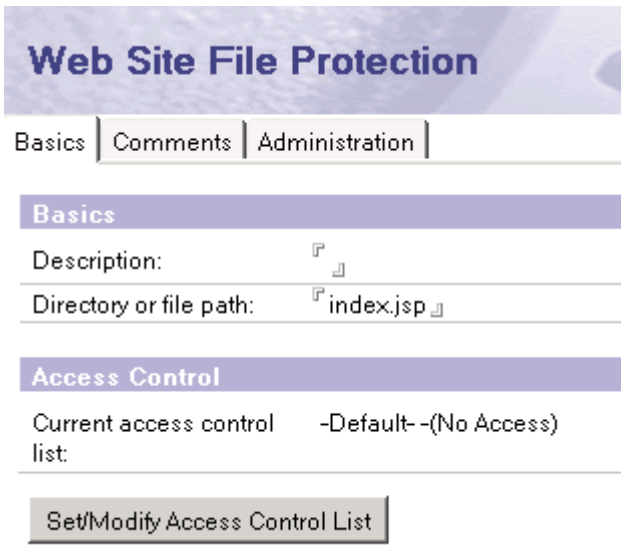

Рис. 3. Настройки правил авторизации для URI

15 Помимо своего встроенного обработчика HTTP-запросов Domino позволяет задать для каждого виртуального сайта до 256 внешних фильтров, которые последовательно обрабатывают все заявленные ими в процедуре FilterInit этапы исполнения запроса (события). Трудно представить себе приложение, которому была бы необходима такая возможность. Логику разработчиков Domino вообще довольно часто невозможно понять. Несогласованность, если не сказать случайность, некоторых технических решений очевидным образом проявляется в именованиях объектов, например, процедур FilterInit и TerminateFilter.

События kFilterResponse и kFilterRawWrite происходят в момент, когда сервер сформировал (получил от шлюза) ответ для клиента. Эти два события в наибольшей степени соответствуют обратным вызовам обработчика JK:

```
static int JK_METHOD StartResponse( jk_ws_service_t *s, int status,
                                    const char *reason,
                                    const char *const *hdrNames,
                                    const char *const *hdrValues,
                                    unsigned hdrCount );
/* Write() is called by Tomcat to send data back to the client */
static int JK_METHOD Write( jk_ws_service_t * s, const void *b, unsigned l );
```

Однако в связи с тем, что вышепреведенные методы вызываются JK, а не шлюзом, и в ходе только одного этапа обработки запроса, полезное применение событий kFilterResponse и kFilterRawWrite пока нам не представляется возможным.

В нашей реализации шлюза используются события (этапы) kFilterStartRequest, kFilterParsedRequest, kFilterProcessRequest, kFilterAuthorized и kFilterEndRequest. Желающие могут доработать их функциональность.

Особенности аутентификации в DSAPI

Несмотря на неоправданную сложность, интерфейс DSAPI не обладает достаточной гибкостью для простого решения задачи интеграции с серверами приложений сторонних производителей. Как уже отмечалось выше, в последовательности генерации событий наличествует скрытая совершенно не очевидная логика.

Если вы, программируя свой внешний фильтр (шлюз), указали, что он будет обрабатывать события kFilterAuthenticate, то вся работа по аутентификации пользователей перекладывается на этот фильтр. Стандартный обработчик больше не будет участвовать в этом процессе, даже если ваша HttpFilterProc будет в ответ на соответствующий вызов возвращать kFilterNotHandled.

С другой стороны, если kFilterAuthenticate не обрабатывается, то при Basic-аутентификации (например, в приложении manager из пакета Tomcat) после kFilterAuthorized, даже в случае если ваша HttpFilterProc возвращает kFilterHandledEvent, а при обработке kFilterParsedRequest вы дали понять HTTP-серверу, что этот запрос ваш (kOwnsRequest), стандартный метод аутентификации Lotus Notes выполнится уже после авторизации в вашем приложении (см рис. 4).

Рис. 4. Двойная аутентификация Tomcat Manager Application и Lotus Notes

При этом если вы авторизовались в обоих приложениях, то при запросе kGetAuthenticatedUserInfo[16] в качестве канонического имени будет возвращено ваше полное имя в Lotus Notes, а в качестве kWebUserName и kUserPassword – те значения, что вы ввели в окне аутентификации Tomcat.

Поскольку Form-аутентификация (приложение admin) является по отношению к HTTP-серверу Domino совершенно посторонней операцией, то она проходит для него прозрачно. При этом cookie, в котором хранится идентификатор сессии у Tomcat имеет имя JSESSIONID, а в Domino – DomAuthSessId.

Новая реализация шлюза Domino-JK

Наша реализация шлюза Domino-Tomcat для коннектора JK написана и оттестирована на следующем составе программных средств:

- SuSE Linux 9.2 (10.0) Desktop Professional[17]
- Tomcat 5.0.27-9.1 для SuSE 9.2 и Tomcat 5.0.30-6 для SuSE 10.0 (установлен в каталог /usr/share/tomcat5)
- Domino Enterprise 6.5.5FP2 для Linux x86[18] (установлен в каталог /opt/lotus)
- Lotus C API for Notes/Domino 6.5.4 (установлен в каталог /opt/lotus/notesapi)
- JDK 1.4.2.13-01 (rpm-модули java-1_4_2-sun-1.4.2.13-0.1, java-1_4_2-sun-plugin-1.4.2.13-0.1, javamail-1.3.1-7, java-1_4_2-sun-jdbc-1.4.2.13-0.1, java-1_4_2-sun-devel-1.4.2.13-0.1) (установлена в каталог /usr/lib/java)
- GNU C 4.0.2_20050901-3

Сервер Domino запускается под пользователем **notes** с соответствующими правами

16 ...
```
uinfo.fieldFlags = kCannonicalUserName | kWebUserName | kUserPassword;
ok = context->ServerSupport( context, kGetAuthenticatedUserInfo, &uinfo, NULL, 0, &errID );
```
...

17 Этот вариант Linux не сертифицирован IBM для эксплуатации Domino, но Domino в этой среде работает не хуже, чем в других. Требуются, однако, определенные усилия по исправлению nsd.sh и замена pstack, которая не входит в состав дистрибутива SuSE, а приложенная к Domino не работает. Nsd.sh из поставки IBM необходимо обработать приблизительно таким скриптом:

```
#!/bin/bash
TMPFILE=/tmp/fixnsd.tmp
echo -n $1 "processing... "
sed -e "s/\(\<tail\>\|\<head\>\) \-1/\1 \-n 1/g; s/-o user=userwithverylongname/-o euser/g"\
nsd.sh > $TMPFILE
cp $TMPFILE $1
rm -f $TMPFILE
```

Программу pstack можно найти по адресу http://ftp.suse.com/pub/projects/pstack/sles8-i386/pstack-1.1.7.IBM-1.src.rpm. Для того чтобы она работала под SuSE 9.2-10.0 нужно пропатчить исходный текст и пересобрать исполняемый модуль:

```
--- pstack.c    2005-03-31 14:04:19.000000000 +0400
+++ pstack.c    2007-01-15 13:40:00.000000000 +0300
@@ -546,6 +555,8 @@

   for ( ; lm.l_next; ) {
     readLinkMap(pid, (Elf32_Addr) lm.l_next, &lm, buf, sizeof(buf));
+    if( buf[0] == '\0' )
+      continue;
     if (!(syms = loadSyms(buf))) {
        printf("(No symbols found in %s)\n", buf);
        continue;
```

18 http://www-10.lotus.com/ldd/r5fixlist.nsf/(Progress)/655 FP2

как и предусматривается установочными инструкциями IBM. Domino использует собственную Java-машину, в среде которой исполняются агенты и сервлеты, написанные на Java. Чтобы системные настройки Java не влияли на Domino, файл ~notes/.bash_profile в домашнем каталоге пользователя **notes** выглядит так:

```
# .bash_profile
#
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=/bin:/usr/bin:/usr/X11R6/bin
export PATH
unset USERNAME

if [ -z DISPLAY ]; then export DISPLAY=:0.0; fi
declare -x CLASSPATH=
declare -x JAVA_BINDIR="/opt/lotus/notes/latest/linux/jvm/bin"
declare -x JAVA_HOME="/opt/lotus/notes/latest/linux/jvm/"
declare -x JRE_HOME="/opt/lotus/notes/latest/linux/jvm/"
declare -x JDK_HOME="/opt/lotus/notes/latest/linux/jvm/"
declare -x SDK_HOME="/opt/lotus/notes/latest/linux/jvm/"
declare -x JAVA_ROOT="/opt/lotus/notes/latest/linux/jvm/"

if [[ -z "$CATALINA_HOME" && -f /etc/sysconfig/j2ee ]]; then
    . /etc/sysconfig/j2ee
    export CATALINA_HOME
    export CATALINA_BASE
fi
export TOMCAT_HOME=$CATALINA_HOME

echo "To start Domino server enter server"
```

Сервер приложений Tomcat устанавливается SuSE в каталог /usr/share/tomcat5. На самом деле различные файлы этого довольно большого продукта разбросаны по системе в соответствии с общепринятыми в Unix соглашениями, например, конфигурационные файлы хранятся в подкаталоге /etc/tomcat5. В /usr/share/tomcat5 для удобства все части продукта собраны в одном месте с помощью символических ссылок.

Tomcat запускается с системной виртуальной Java-машиной (JVM), в нашем случае 1.4.2. Поскольку в системе может быть установлено множество версий Java, то важно удостовериться, что Tomcat работает в среде соответствующей JVM. Автоматический поиск системной JVM проводится скриптом bin/catalina.sh. Этот скрипт требует маленькой правки для того, чтобы мы были убеждены в том, что он находит нужную JVM:

строка

```
TOMCAT_CFG="@@@TCCONF@@@/tomcat5.conf"
```

должна быть заменена на две

```
TOMCAT_CFG="$CATALINA_HOME/tomcat5.conf"
unset JAVA_HOME
```

Для того, чтобы указать шлюзу, что определенные URI обрабатываются сервером Domino, а не Tomcat, мы ввели фиктивный обработчик с именем domino. Примерный файл uriworkermap.properties для совместной работы встроенного HTTP-сервера и Tomcat в рамках одного виртуального www-сервера будет выглядеть несложно, так как нужно указать только исключения из правил:

```
# *********** Begin uriworkermap.properties ***
# Mount the Servlet context to the ajp13 worker
# Fake domino worker used as mark for URI ignored by Tomcat and served by Domino
/icons/*=domino
/domjava/*=domino
# ************* End uriworkermap.properties ****
```

По-умолчанию шлюз считает, что все URI, которые ссылаются на базы данных Notes (файлы с расширением .nsf), обрабатываются Domino, остальные – Tomcat.

В общем случае для сборки шлюза вам необходимы файлы:

Makefile
config.h
inifile.h
inifile.c
jk_dsapi_plugin.c
mkini.sh
uriworkermap.properties
workers.properties

Для желающих запускать Tomcat в отладчике Jswat прикладывается примерный стартовый файл: runjswat.

Если ваш дистрибутив вы скопируете в каталог jakarta-tomcat-5.0.xx-src/jakarta-tomcat-connectors/jk/native/domino из соответствующей поставки Tomcat, предварительно затерев или скопировав куда-нибудь все, что в нем лежит, то достаточно настроить с помощью текстового редактора, например, jedit, пути к установленным продуктам в Makefile и запустить make и make install.

Заключение

На данном этапе мы не проверили работу шлюза с SSL, а также не изучили возможность совместного использования аутентификационной информации (имени пользователя и его членства в группах, пароля, сертификата x.509, DomAuthSess cookie, SSO cookie) Domino и Tomcat для организации единого входа в эти системы при задействовании аутентификационного механизма Domino (решение обратной задачи приведено в документации по DSAPI).

Владимир Ануфриев
2007-02-07