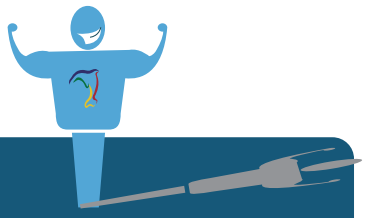
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

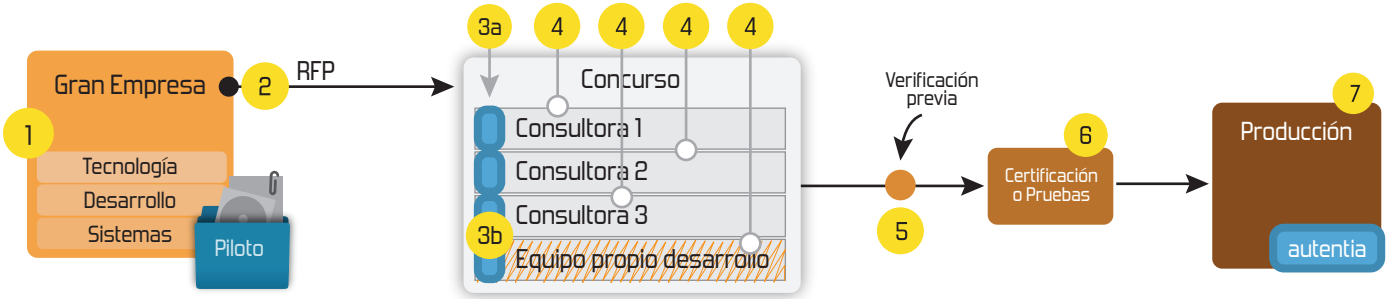
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

DESARROLLADO POR:
Jordi Monné Miranda

Ingeniero Superior en Informática por la Universidad de Lleida.

Co-fundador de jobsket

http://jordim.tumblr.com

Fecha de publicación del tutorial: 2011-01-13

2.048

Introducción a Apache James

0. Índice de contenidos.

1. Introducción

En este tutorial vamos a ver una pequeña introducción de Apache James, qué es y en qué nos puede ayudar. Mostraremos una pequeña guía de cómo configurar Apache James y jugaremos con él creando unos componentes necesarios como son los Mailets y los Matchers. Este pequeño artículo no pretende ser una guía detallada de cómo configurar ni utilizar James sino una pequeña introducción a los aspectos más generales e importantes de este servidor.

2. Entorno.

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook' (2.4 GHz Intel Core 2 Duo).
- Sistema Operativo: Mac OS X Snow Leopard 10.6.6
- JDK 1.6.0_22
- Apache James 2.3.2

3. ¿Qué es Apache James?.

Apache James es un proyecto de la fundación Apache que porporciona un servidor de correo basado íntegramente en Java. La elección de James como servidor de correo para nuestras aplicaciones nos dota de una gran flexibilidad a la hora de tratar con los correos que mueve nuestro dominio, permitiendo tener un control total sobre lo que está pasando en nuestro servidor de correo como por ejemplo el decidir cuáles són las reglas a aplicar sobre e-mails que recibe el servidor, monitorizar todo el flujo de correos, decidir cuáles deberían prohibirse, poder definir nuestra propia política de anti-spam, entre otras.

4. Instalación.

Para instalar James hay que ir a esta página y descargarse el binario (a la hora de escribir este tutorial, la versión estable de James era la 2.3.2).

El siguiente paso es descomprimir el fichero en algún directorio de vuestra máquina y deberíais ver una estructura de directorios como la que se muestra a continuación:

```
macbook-de-jordi-monne-miranda:james-2.3.2 jordi$ ls
LICENSE.txt             RELEASE_NOTES.txt       bin                     lib                     tools
NOTICE.txt              UPGRADE.txt             conf                    logs                    work
README.txt              apps                    ext                     temp
```

Una vez tenemos instalado James, el siguiente paso es arrancarlo para empezar a trabajar con él. Para ello hay que dirigirse al directorio **bin** y ejecutar el archivo **run.sh** o **run.bat** en función del sistema operativo. Una vez arrancado se debería ver esto:

Catálogo de servicios Autentia

Get ADOBE FLASH PLAYER

Últimas Noticias

- VII Autentia Cycling Day
- Autentia patrocina la charla sobre Java SE 7 en Madrid
- Alfresco Day 2011
- XVII Charla Autentia - Grails - Vídeos y Material
- ¡¡¡ 15 millones de descargas de tutoriales !!!

Histórico de NOTICIAS

Últimos Tutoriales

- Spring MVC: acceder a las propiedades de un fichero desde una JSP con Expression Language (EL)
- Framework Scala liftweb
- Trabajando con JAXB y Eclipse
- Configurar Spring Security 3.1 para autenticarse contra un Active Directory
- Migración a ICEfaces 2.0

Síguenos a través de:

Últimas ofertas de empleo

2011-07-06 Otras Sin catalogar - LUGO.

2011-06-20 Comercial - Ventas - SEVILLA.

2011-05-24 Contabilidad - Expecialista Contable - BARCELONA.

2011-05-14 Comercial - Ventas - TARRAGONA.

```
Using PHOENIX_HOME:   /Users/jordi/Programacio/llibreries/james-2.3.2
Using PHOENIX_TMPDIR: /Users/jordi/Programacio/llibreries/james-2.3.2/temp
Using JAVA_HOME:      /System/Library/Frameworks/JavaVM.framework/Versions/1.6/Home
Running Phoenix:
Listening for transport dt_socket at address: 8000

Phoenix 4.2

James Mail Server 2.3.2
Remote Manager Service started plain:4555
POP3 Service started plain:110
SMTP Service started plain:25
NNTP Service started plain:119
FetchMail Disabled
```

Desconozco cómo es en Windows pero sobre plataformas Unix tendréis que arrancar James como *root* ya que por defecto se trabaja con los puertos 110, 22 y 119, puertos reservados para el usuario *root* al ser inferiores a 1024. Para cambiar estos puertos hay que dirigirse al fichero de configuración principal que se encuentra en *james-2.3.2/apps/james/SAR-INF/config.xml.*

5. Mailets y Matchers

Hay dos pilares fundamentales a entender dentro de Apache James, los *matchers* y los *mailets* componentes sobre los que gira en gran parte todo el servidor.

¿Qué es un Matcher?

Un *matcher* es un agente de seguridad, permite a James decidir si un e-mail debe ser procesado o si por el contrario no se debe permitir su procesamiento posterior. Un *matcher* es el segurata del sistema.

James proporciona una serie de *matchers* programados por defecto. Se pueden ver aquí. Por ejemplo si somos los administradores de una lista de correo de adictosaltrabajo.com podemos definir el *matcher SubjectStartsWith* para especificarle a James que todos esos correos que no tienen en el asunto "Adictos al trabajo:" que los envie a la basura ya que no cumplen con las directrices de la lista. Por el contrario si el correo empieza con ese texto, el *matcher* dejará pasar el correo para poder ser tratado en fases posteriores.

¿Qué es un Mailet?

Un *mailet* es un agente para procesar e-mails.

Si un servlet es invocado cuando una conexión HTTP se lanza contra un contenedor de servlets, en este caso y salvando las distancias un *mailet* es ejecutado cuando entra un e-mail dentro de James para ser procesado. Antes de llegar al *mailet*, el correo debe haber pasado por los "seguratas" de James, los *matchers*.

Todo *mailet* tiene un ciclo de vida establecido, se inicia, se procesa y se destruye mediante los métodos:

view plain print ?

```
01. init(MailetConfig config)
02. service(Mail mail)
03. destroy()
```

Apache James proporciona por defecto una serie de *mailets* para ser utilizados dentro del contenedor de correos como por ejemplo un *mailet* para procesar un e-mail que entra al contenedor de tal forma que devuelva el mismo correo pero sólo la parte que contiene texto eliminándolo.

Apache James proporciona por defecto una serie de *mailets* que se pueden ver aquí.

Por ejemplo y para intentar de acabar de asimilar el concepto de *mailet*, James proporciona un *mailet* llamado *ReplaceContent* que permite reemplazar cadenas de texto del asunto y/o contenido mediante expresiones regulares. Un ejemplo de este *mailet* podra ser utilizado para un servidor de correos de una red social para niños. El jefazo de la red social no quiere que haya una sola palabra malsonante en los correos así que configura James para que substituya todas las palabrotas por asteriscos.

A grandes rasgos el flujo sería:

- 1 - Niño A envía correo a usuario niño B.
- 2 - James recibe correo de niño A.
- 3 - James dirige esos correos hacia los *matchers* (para simplificar, sólo hay uno y deja pasar a todos los correos).
- 4 - James observa que el e-mail cumple con todos los *matchers* así que les permite continuar.
- 5 - James ejecuta el *mailet* configurado para esos *matchers*.
  - 5.1 - James modifica todas las palabrotas que niño A le escribirá en el e-mail a niño B.
- 6 - James deja proceder con el correo.
- 7 - Niño B ve asteriscos.

6. Ejemplo de Mailet y Matcher personalizado.

En el punto anterior hemos visto por encima que James nos proporciona una buena lista de *mailets* y *matchers* que se adaptan bastante bien a muchos escenarios. No obstante el conjunto de posibles escenarios es imilitado así que James nos proporciona una forma de crear nustros propios *mailets* y *matchers*.

Como ejemplo vamos a poner un escenario en el que en una empresa existen dos correos de atención al cliente, *cuenta1@empresa.com* y *cuenta2@empresa.com*. Si bien puede resultar extraño nos será bastante útil para mostrar el ejemplo. La idea es que el jefe quiere que cuando se estén enviando (por X razones) más correos a una cuenta en particular, que le envie un correo avisándole de este desbalanceo de trabajo para poner solución.

Lo primero que vamos a hacer es crear un *matcher* que sólo invoque el *mailet* que posteriormente vamos a escribir sólo y exclusivamente si las direcciones a las que se dirige el correo son *cuenta1@empresa.com* y *cuenta2@empresa.com*.

Para crear un *matcher* personalizado sólo hay que crear una clase que implemente la interfaz *Matcher*.

view plain print ?

```
01. public BlockEmailMatcher implements Matcher {
```

El código para decidir cuales són las direcciones de correo que queremos dejar pasar y enviar hacia un *mailet* va dentro del método *match(Mail mail)*.

view plain print ?

```
01.  public Collection match(Mail mail) throws MessagingException {
02.
03.          Collection<MailAddress> recipients = new ArrayList<MailAddress>();
04.          Iterator<MailAddress> it = mail.getRecipients().iterator();
05.          while(it.hasNext()) {
06.              MailAddress to = (MailAddress)it.next();
07.              if(customerService.validEmail(to)) {
08.                  recipients.add(to);
09.              }
10.          }
11.          return recipients;
12.      }
```

customerService es una instancia de un objecto que nos permite conocer si la dirección a la que se dirige el correo es *cuenta1@empresa.com* o *cuenta2@empresa.com*. Las direcciones que estén dentro de la colección de devolvemos van a ser recibidas por el *mailet* que se asocia a este *matcher* así que si en un *matcher* lo que queremos es filtrar, tenemos que devolver una colección vacía.

Para crear un *mailet* personalizado sólo hay que crear una clase que implemente la interfaz *Mailet*.

```
view plain  print  ?
01.  public class BlockEmailMailet implements Mailet {
```

Anteriormente hemos comentado que un *mailet* ejecuta una lógica cuando se recibe el contenedor recibe un correo y pasa el *matcher*. Esta lógica está contenida dentro del método *service(Mail mail)*:

```
view plain  print  ?
01.  @Override
02.      public void service(Mail mail) throws MessagingException {
03.
04.          MailAddress to = null;
05.          Iterator<MailAddress> it = mail.getRecipients().iterator();
06.          while(it.hasNext()) {
07.              to = (MailAddress)it.next();
08.              if(workers.uncompensatedWork(to)) {
09.                  mailetConfig.getMailetContext().log("Too much work! sending an email to the
10.                  mail.send(UNBALANCED_WORK);
11.              }
12.          }
13.      }
```

Cuando llega un correo se ejecuta el método de arriba y si hay una desbalanza de trabajo entre los trabajadores responsables de la atención al cliente, se envia un correo al gerente.
mail es una instancia de un objecto para enviar correos.
workers es una instancia de un objeto que nos devuelve *true* o *false* si un trabajador está más ocioso que otro.

Para configurar que *BlockEmailMailet* debe estar relacionado con *BlockEmailMatcher* hay que añadir esta configuración en *james-2.3.2/apps/james/SAR-INF/config.xml.*

```
view plain  print  ?
01.  <mailet match="BlockEmailMatcher" class="com.adictosaltrabajo.james.mailet.BlockEmailMailet"
02.
03.  <mailetpackages>
04.        <mailetpackage>org.apache.james.transport.mailets</mailetpackage>
05.        <mailetpackage>org.apache.james.transport.mailets.smime</mailetpackage>
06.        <mailetpackage>com.adictosaltrabajo.james.mailet</mailetpackage>
07.     </mailetpackages>
08.     <matcherpackages>
09.        <matcherpackage>org.apache.james.transport.matchers</matcherpackage>
10.        <matcherpackage>org.apache.james.transport.matchers.smime</matcherpackage>
11.        <matcherpackage>com.adictosaltrabajo.james.matcher</matcherpackage>
12.    </matcherpackages>
13.
```

Cuando se reciba un correo se invocará *BlockEmailMatcher* conjuntamente con todos los *matchers* configurados y si pasa ese *matcher* concreto, se ejecutará el código escrito en *BlockEmailMailet*.

7. Conclusiones.

Apache James nos brinda un interesante servidor de correos para poder crear aplicaciones que procesen los correos electrónicos a nuestra entera disposición trabajando en Java.

Un saludo.

Jordi

jmonne@gmail.com

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda, podemos ofrecerte un mejor servicio.

Enviar comentario

(Sólo para usuarios registrados)

COMENTARIOS

jcarmonaloeches *2011-01-18 - 11:00:49*

Que alegría ver a una persona de Jobsket publicando cositas... seguro que interesantes.