



ZAIN

Actualización Smartwrapper

19/11/2009

© Izenpe s.a. 2009

Este documento es propiedad de Izenpe, s.a. y su contenido es confidencial. Este documento no puede ser reproducido, en su totalidad o parcialmente, ni mostrado a otros, ni utilizado para otros propósitos que los que han originado su entrega, sin el previo permiso escrito de Izenpe, s.a.. En el caso de ser entregado en virtud de un contrato, su utilización estará limitada a lo expresamente autorizado en dicho contrato. Izenpe, s.a. no podrá ser considerada responsable de eventuales errores u omisiones en la edición del documento.



Control de documentación

Título documento: ZAIN – Actualizacion Smartwrapper

Histórico de versiones

Código	Versión	Fecha	Resumen de los cambios producidos
	1.0	19/11/2009	Primera versión.

Cambios producidos desde la última versión

Versión 1: Versión inicial



CONTENIDO

1. Introducción	4
2. INSTALACIÓN	5
3. CONFIGURACIÓN.....	6
4. CONFIGURACION MEDIANTE FICHERO.....	10
5. CONFIGURACION PROGRAMATICA	11



1. Introducción

El presente documento amplia explica los cambios que deben llevarse a cabo en los ejemplos distribuidos dentro del Kit de bienvenida de Smartwrapper, para la instalación de las nuevas librerías que permiten una configuración programática de las llamadas a ZAIN.



2. INSTALACIÓN

La instalación de las librerías comprende los siguientes pasos:

1. Eliminación de las siguientes librerías existentes en la carpeta *lib* del proyecto con los ejemplos de uso de *smartwrapper*.
 - a. *smartwrapper.jar*
 - b. *trustedx-client-axis-stub.jar*
 - c. *trustedx-client-axis.jar*
 - d. *trustedx-provider.jar*
2. Descomprimir el fichero *librerías-smartwrapper.zip* que acompaña al presente documento en la carpeta *lib* anterior. Se instalará los siguientes *jars*:
 - a. *smartwrapper.jar*
 - b. *trustedx-client-axis-stub.jar*
 - c. *zain-client-axis-1.0.jar*
 - d. *trustedx-provider.jar*
3. Actualizar el classpath del proyecto para reflejar los cambios introducidos por las nuevas librerías.



3. CONFIGURACIÓN

En este apartado se explicarán todos y cada uno de los parámetros más relevantes que conforman la configuración y las distintas maneras que tenemos de realizar la configuración.

Los parámetros aquí descritos, son utilizados por el smartwrapper para configurar determinados aspectos de la comunicación con los servicios web de ZAIN.

A continuación se describen los distintos parámetros de configuración existentes:

Truststore.active

true: para la conexión HTTP se utilizará el almacén de certificados definido en el parámetro Truststore.path.

false (valor por defecto): se utilizará el almacén de certificados configurado en la máquina virtual de Java (si lo hay).

Truststore.path

Almacén de certificados que se utilizará en la conexión HTTP (si el valor de Truststore.active es true).

Truststore.password

Contraseña del almacén de certificados definido en Truststore.path.

Keystore.active

true: para la conexión HTTPS se utilizará el almacén de claves definido en el parámetro Keystore.path.

false (valor por defecto): se utilizará el almacén de claves configurado en la máquina virtual de Java (si lo hay).

Keystore.path

Almacén de claves utilizado en la conexión HTTPS (si el valor de Keystore.active es true).

Keystore.password

Contraseña del almacén de claves definido en Keystore.path.

Keystore.type

Tipo del almacén de claves definido en Keystore.path. PKCS12 por ejemplo



 Proxy.active

true: para la conexión HTTP se utilizará un servidor Proxy.

false: no se utilizará un servidor proxy.

 Proxy.host

Servidor proxy utilizado (si el valor de Proxy.active es true).

 Proxy.port

Puerto del servidor proxy.

 Proxy.username

Nombre de usuario para acceder al servidor proxy (si requiere autenticación HTTP básica).

 Proxy.password

Contraseña para acceder al servidor proxy (si requiere autenticación HTTP básica).

 Timeout

Tiempo de espera de la conexión HTTP (en milisegundos).

 Ssl.allowCriticalExts

true: durante la conexión SSL se permiten certificados de servidor con cualquier extensión marcada como crítica.

false (valor por defecto): durante la conexión SSL no se permiten certificados de servidor con extensiones críticas.

 Ssl.noValidation

true: se anula la validación del certificado del servidor durante las conexiones SSL.

false (valor por defecto): se valida el certificado del servidor durante las conexiones SSL.

 Ssl.validation



El algoritmo de validación del certificado del servidor que se empleará durante una conexión SSL. Si se usa, debe ser un algoritmo implementado por un proveedor criptográfico registrado.

✚ Request.loadPath

El directorio base de los archivos creados para enviar en las peticiones cuando se va a tratar con datos de gran tamaño.

✚ Response.savePath

El directorio base de los archivos creados (si así se indica en la petición) al recibir las respuestas cuando se va a tratar con datos de gran tamaño.

✚ reg-log.active

true: se guardarán los archivos de log en las peticiones SOAP enviadas a ZAIN.

false (valor por defecto): no se guardarán los archivos de log en las peticiones SOAP enviadas a ZAIN.

✚ reg-log.savePath

El directorio en el que se guardarán los archivos de log con las peticiones enviadas a ZAIN si el parámetro anterior, req-log.active, vale true.

✚ res-log.active

true: se guardarán los archivos de log en las respuestas enviadas por ZAIN.

false (valor por defecto): no se guardarán los archivos de log en las respuestas enviadas por ZAIN.

✚ res-log.savePath

El directorio en el que se guardarán los archivos de log con las respuestas enviadas por ZAIN si el parámetro anterior, res-log.active, vale true.

✚ authN.policy



La política de autenticación solicitada (opcional). Tiene como valor esperado el identificador la política de autenticación que se solicita en la petición de servicio. Habrá que seleccionar una política en concreto, definida por Izenpe.

4. CONFIGURACION MEDIANTE FICHERO

En el siguiente apartado detallaremos como realizar la configuración mediante un fichero de propiedades, en concreto, mediante el fichero `smartwrapper.properties`. De esta forma, para que se procesen los parámetros de configuración anteriormente descritos, es necesario que este fichero se halle disponible en el classpath de la aplicación.

Por lo tanto, una vez descritos los parámetros, no nos quedaría mas que realizar la configuración del `smartwrapper.properties`, de manera que a cada parámetro le demos el valor requerido.

```
Truststore.active=true

#Almacén de certificados que se utilizará en la conexión
#HTTP (si el valor de Truststore.active es true).
Truststore.path=resources/application.truststore

#Contraseña del almacén de certificados definido en Truststore.pa
Truststore.password=trustedx

#-----#

#true: para la conexión HTTPS se utilizará el almacén de
#claves definido en el parámetro Keystore.path.
#false (valor por defecto): se utilizará el almacén de
#claves configurado en la máquina virtual de Java (si lo
#hay).
Keystore.active=true

#Almacén de claves utilizado en la conexión HTTPS (si el
#valor de Keystore.active es true).
Keystore.path=##RUTA AL KEYSTORE##

#Contraseña del almacén de claves definido en Keystore.path.
Keystore.password=##CONTRASEÑA DEL KEYSTORE##

#Tipo de keystore utilizado (JKS o PKCS12)
Keystore.type=PKCS12

#-----#

#true: para la conexión HTTP se utilizará un servidor Proxy.
#false: no se utilizará un servidor proxy.
Proxy.active=false

#Servidor proxy utilizado (si el valor de Proxy.active es
#true).
Proxy.host=##HOSTPROXY##

#Puerto del servidor proxy.
Proxy.port=8080

#Nombre de usuario para acceder al servidor proxy (si
#requiere autenticación HTTP básica).
Proxy.username=##USERPROXY##

#Contraseña para acceder al servidor proxy (si requiere
#autenticación HTTP básica).
Proxy.password=##PASSWORDPROXY##
```



5. CONFIGURACION PROGRAMATICA

A continuación se explica como realizar la configuración del API de Zain de una forma programática.

Importante: La configuración programática, extiende la configuración mediante fichero (*smartwrapper.properties*), lo cual permite que ambos sistemas puedan convivir. En caso de que el *API de Zain* detecte un objeto de tipo *ZainConfig* no nulo, los valores definidos en él, prevalecerán sobre los valores definidos en el fichero *smartwrapper.properties*, incluso aunque estos no hayan sido asignados a través del correspondiente *setter*. Si por el contrario, el API de Zain obtiene un objeto *ZainConfig* nulo, se aplicarán los valores contenidos en el fichero *smartwrapper.properties*. Al menos uno de los sistemas de configuración debe aplicarse.

El primer paso consistirá instanciar la clase *ZainConfig*:

```
ZainConfig config = new ZainConfig();
```

Los objetos de la clase *ZainConfig* se componen de una serie de variables de instancia (accesibles mediante sus correspondientes *getter* y *setter*), que trasladan las propiedades definidas en el fichero *smartwrapper.properties*, a un paradigma de orientación a objetos.



```
String endpoint = "https://psf2.izenpe.com:8443/trustedx-gw/SoapGateway";
SmartVerifyRequest smartVerifyReq = new SmartVerifyRequest(endpoint);

ZainConfig config = new ZainConfig();

// Rellenamos el config con la configuración
// correspondiente a nuestros valores
config.setTruststoreActive(true);
config.setTruststorePath("ssl/trustedx-des-truststore.jks");
config.setTruststorePassword("trustedx");

config.setKeystoreActive(true);
config.setKeystorePath("ssl/izenpe-pruebas-des.p12");
config.setKeystorePassword("admin");
config.setKeystoreType("PKCS12");

config.setProxyActive(false);
config.setProxyHost("");
config.setProxyPort("");
config.setProxyUsername("");
config.setProxyPassword("");

config.setTimeout("9500");

config.setAllowCriticalExts(false);
config.setNoValidation(true);

config.setRequestLogActive(true);
config.setRequestLogSavePath("c:\\logs");
config.setResponseLogActive(false);
config.setResponseLogSavePath("c:\\logs");
//
```

Imagen correspondiente a la configuración de los parámetros en una clase Java.

Los objetos de la clase *ZainConfig*, tienen como objetivo servir como contenedores de la información. En el siguiente paso veremos como hacer que el API de Zain “lea” las propiedades recién definidas, y haga uso de ellas. Para ello, se ha definido un método estático en la propia clase *ZainConfig*, que permite establecer la configuración a aplicar en el *thread* en ejecución (para más información consultar clase *ThreadLocal* del API de Java).

```
ZainConfig.setCurrent(config);
```

La asignación de la configuración a través del método *setCurrent()*, debe llevarse a cabo antes de cada llamada a la función *send()* del API de Zain. Esto es debido, a que el API de Zain, tras leer los valores asignados, borra la configuración en curso. Se ha optado por esta solución, para garantizar que en entornos multithread, no haya problemas de compartición de información por un olvido del programador, a la hora de borrar la configuración en curso. De esta forma, tanto la clase *ZainConfig*, como el sistema de configuración programática, se consideran *ThreadSafe*.



```
ZainConfig.setCurrent(config);

SmartVerifyResponse smartVerifyResp = smartVerifyReq.send();

// La configuración asignada se ha borrado, ya que el API de Zain,
// tras recuperar la configuración de la variable asociada al Thread
// en ejecución, la asigna a null. Esto es debido a que en las
// aplicaciones web hay un pool de Threads limitado. Sería posible
// que una vez se ha hecho el set de la configuración
// (ZainConfig.setCurrent(config)), en el futuro
// ese mismo Thread fuese reutilizado para hacer otra llamada a
// ZAIN para la que no se haya hecho la correspondiente asignación.
// Esto conllevaría que la configuración se mezclase entre diferentes ejecuciones.
```

[...]

```
ZainConfig.setCurrent(config);

SmartVerifyResponse smartVerifyResp2 = smartVerifyReq2.send();
```

Imagen correspondiente al orden de ejecución de la asignación de la configuración y llamada de los métodos.