

Optimización de librerías Java: Manual de usuario de SmartLinkerConfGenerator

Introducción:

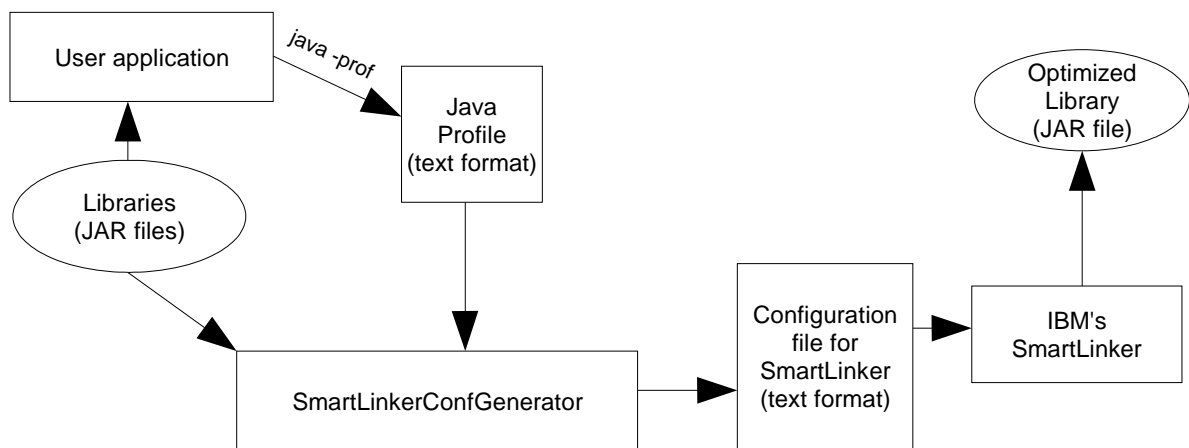
SmartLinkerConfGenerator es un programa Java para línea de comandos, diseñado para facilitar la optimización de librerías Java respecto a una aplicación Java determinada, la cual denominaremos "aplicación usuaria", ya que es la que utiliza dichas librerías.

Concretamente la función de SmartLinkerConfGenerator es, como su nombre indica, generar ficheros de configuración para SmartLinker de IBM, mediante el análisis de las librerías sujetas a la optimización y una serie de consultas al usuario. Por su parte, SmartLinker es una aplicación de IBM incluida en productos como IBS WebSphere Device Developer (WSDD) que sirve para optimizar aplicaciones destinadas a ejecutarse en entornos de recursos restringidos, como por ejemplo J2ME con sus diferentes perfiles (CLDC, MIDP, etc). Existe una versión de prueba descargable de la web de IBM (www.ibm.com).

Para realizar la optimización de las librerías con SmartLinkerConfGenerator, se requieren siguientes elementos básicos e indispensables:

- El perfil de una ejecución de la aplicación respecto a la cual se optimiza (aplicación usuaria), obtenido mediante "java -prof".
- Las librerías (ficheros .jar) que van a ser optimizadas.
- La aplicación SmartLinker de IBM (jxelink), que finalmente se ejecutará sobre el fichero de configuración obtenido de SmartLinkerConfGenerator, para reducir las librerías.

La siguiente figura ilustra el funcionamiento del sistema:



Debe hacerse notar que las librerías obtenidas tras todo el proceso de optimización que a continuación se detallará, son intrínsecamente no fiables cien por ciento. Ello es debido a que se optimizan respecto a una ejecución concreta de la aplicación, por lo que no se puede asegurar que funcionen para ejecuciones diferentes. Por ello, como veremos más adelante, es muy importante que al obtener el perfil, la ejecución de la aplicación usuaria sea lo más completa posible, es decir, que se desarrolle por todos los caminos de ejecución posibles, lo cual, por otra parte, normalmente es imposible.

Utilización:

Para ejecutar SmartLinkerConfGenerator, basta con invocar el intérprete Java sobre la clase con el mismo nombre, o, si se suministra empaquetada en un fichero JAR, ejecutarlo con la opción "-jar". En este punto es importante remarcar que la ejecución debe hacerse exactamente con el mismo *classpath* con el que se ejecutó la aplicación usuaria al obtener su perfil de ejecución.

Su sintaxis genérica es la siguiente:

```
SmartLinkerConfGenerator [-o outputfile] library1.jar [library2.jar]...
                           java.prof_to_parse
```

Siendo:

- *configuration_outputfile* - la ruta completa deseada, donde se creará el fichero de configuración. Si se omite, se creará un fichero llamado "jxelink.jxeLinkOptions" en el directorio actual.
- *libraryx.jar* - la ruta a la(s) librería(s) que se desea(n) optimizar. Es posible optimizar más de una librería al mismo tiempo, pero hay que tener en cuenta que el resultado generará una única librería con la unión de las librerías de entrada optimizadas.
- *java.prof_to_parse* - la ruta completa al perfil de ejecución de la aplicación usuaria.

A continuación realizaremos un ejemplo ficticio (con la sintaxis de un sistema Linux) para ilustrar el proceso completo de optimización de librerías Java.

Paso 1: Perfil de la aplicación usuaria

En este paso trataremos de obtener el perfil de ejecución de la aplicación usuaria. Para ello, debemos de ejecutar aplicación usuaria en "modo perfilado" (java -prof).

Por ejemplo, supongamos que la clase principal de nuestra aplicación usuaria se llama "BarkIDS", y que requiere de cuatro librerías (cuyas rutas ahora veremos). Entonces, la invocación sería de la forma:

```
java -prof -cp /usr/lib/java/axis.jar:/usr/lib/java/jmf.jar:\
/usr/lib/java/jain-sip.jar:/usr/lib/java/jamon.jar BarkIDS
```

Para realizar un perfil lo más completo posible, es necesario que la ejecución de la aplicación pase por el mayor número de caminos de ejecución posible, es decir, que se recorran todas las características o casos de uso que la aplicación usuaria posea. De esta manera las librerías resultantes del proceso de optimización serán más fiables.

Al terminar la ejecución, dispondremos de un fichero llamado "java.prof" (situado en el directorio actual) que contiene el perfil de la ejecución de la aplicación usuaria.

Paso 2: SmartLinkerConfGenerator

En este paso utilizaremos SmartLinkerConfGenerator para obtener el fichero de configuración para SmartLinker adaptado a nuestras necesidades. Para ello, basta con ejecutar ejecutar SmartLinkerConfGenerator sobre el fichero obtenido en el paso anterior.

Supongamos que deseamos optimizar la librería "/usr/lib/java/axis.jar", y que el fichero de perfil obtenido en el paso anterior se encuentra en "/home/yo/java.prof" y, por ejemplo, deseamos que SmartLinkerConfGenerator genere el fichero de configuración en:

"/home/yo/axisOpt.jxeLinkOptions". De esta manera, la invocación sería de la forma:

```
java -cp /usr/lib/java/axis.jar:/usr/lib/java/jmf.jar:/usr/lib/java/jain-sip.jar:\
/usr/lib/java/jamon.jar SmartLinkerConfGenerator \
-o /home/yo/axisOpt.jxeLinkOptions /usr/lib/java/axis.jar \
/home/yo/java.prof
```

Nótese que el classpath entregado a la máquina virtual ha de ser el mismo que el que se definió al ejecutar la aplicación usuaria (ver paso 1).

Una vez ejecutado SmartLinkerConfGenerator, éste nos realizará varias consultas:

1) Tras mostrar las opciones básicas que el programa posee por defecto, nos consultará si se desean modificar, y si este es el caso, preguntará por la ruta del fichero (de texto) que contiene las opciones básicas personalizadas.

Para nuestro ejemplo respondemos que no deseamos modificarlas.

2) Ruta deseada para la librería (fichero .jar) optimizada generada por SmartLinker (jxelink). Por ejemplo, introducimos `"/home/yo/axis-optimizado.jar"`.

3) ¿Desea que se incluyan todos los recursos de las librerías automáticamente?. Esta pregunta se refiere a si deseamos incluir en el fichero JAR final (la librería optimizada) todos los ficheros que no sean clases java contenidos en las librerías a optimizar. Podrían ser recursos por ejemplo los ficheros de propiedades (ej. `org/apache/axis/axis.properties`), o manifiestos (ej. `META-INF/MANIFEST.MF`).

- Es recomendable responder afirmativamente, así que introducimos "y".
- Si respondiéramos negativamente, el programa nos preguntaría si deseamos introducir manualmente algún recurso, para lo cual se le proporcionaría la ruta del mismo dentro de la librería (ej. `org/apache/axis/axis.properties`).

4) ¿Desea incluir explícitamente clases o paquetes de las librerías?. Esta pregunta se refiere a si deseamos forzar a introducir algunas clases (o paquetes enteros si usamos el asterisco), en la librería final.

Ya que SmartLinkerConfGenerator y SmartLinker pueden omitir algunas clases que posteriormente son necesarias para la ejecución de la aplicación usuaria (ver apartado de Consideraciones Técnicas), seguramente habrá que volver a realizar el paso 2 completo varias veces, incluyendo en esta pregunta las clases problemáticas. Por ello, lo usual es responder los nombres de las clases que nos requería la aplicación usuaria cuando falló (lanzando una `ClassNotFoundException`).

Para facilitar la introducción de estos nombres de clases, SmartLinkerConfGenerator nos da la posibilidad de cargarlos desde un fichero de texto externo (con una clase o paquete por línea).

En nuestro ejemplo, responderemos afirmativamente e introduciremos `"org.apache.axis.*"` de manera que obligamos a SmartLinker a que incluya todas las clases del paquete `"org.apache.axis"`.

5) Tras estas consultas, SmartLinkerConfGenerator comenzará a trabajar, hasta que genere el fichero de configuración requerido.

Paso 3: SmartLinker (jxelink)

Una vez obtenido el fichero de configuración del paso anterior, basta con ejecutar SmartLinker sobre él, para que este genere la librería optimizada como nosotros deseamos.

En nuestro ejemplo, la invocación sería de la forma:

```
jxelink @/home/yo/axisOpt.jxeLinkOptions
```

Con ello, tras varios mensajes de aviso y estado, SmartLinker nos habrá generado la librería optimizada en `"/home/yo/axis-optimizado.jar"`.

Paso 4: Pruebas

En este punto únicamente falta probar la aplicación usuaria con la librería optimizada y verificar que no se ha omitido ninguna clase esencial.

Para ello, ejecutamos la aplicación usuaria tal y como lo hicimos en el paso 1, pero desactivando el modo de perfilado y sustituyendo en el *classpath* la(s) librería(s) antigua(s) por la nueva y optimizada.

En nuestro ejemplo:

```
java -cp /home/yo/axis-optimizado.jar:/usr/lib/java/jmf.jar:\
/usr/lib/java/jain-sip.jar:/usr/lib/java/jamon.jar BarkIDS
```

Si la aplicación funciona correctamente, ya tendremos disponible la librería optimizada para nuestra aplicación. En caso contrario, habría que repetir el procedimiento desde el paso 2, introduciendo los nombres de las clases problemáticas cuando se requiera.

CONSIDERACIONES TÉCNICAS

Funcionamiento de SmartLinkerConfGenerator:

El funcionamiento interno de SmartLinkerConfGenerator se basa en la concatenación de los textos generados consecutivamente a lo largo de tres pasos, para formar finalmente el que será el fichero de texto de configuración para SmartLinker:

- 1º Opciones básicas de SmartLinker: se pueden utilizar las implementadas por defecto en el programa, u otras cargadas desde un fichero del usuario.
- 2º Opciones generadas dinámicamente en base a los parámetros de la invocación, tales como el classpath para SmartLinker, el fichero de salida de SmartLinker (librería optimizada), y los recursos de las librerías incluidos.
- 3º Lista de las clases necesarias para la ejecución de la aplicación usuaria. Esta lista se obtiene de analizar el perfil de ejecución de la aplicación usuaria con la aplicación "ProfileParser", que a continuación se detalla.

Funcionamiento de ProfileParser

ProfileParser es la aplicación encargada de extraer todos los nombres de clase que intervienen en el perfil, siempre y cuando pertenezcan a los paquetes dados.

Aunque esta aplicación se usa internamente desde SmartLinkerConfGenerator, también es posible su utilización desde línea de comandos. Su sintaxis es la siguiente:

```
ProfileParser [-smartlinker] [-o output_file] [-vector] prof_to_parse [package1]..
```

Siendo las opciones:

- *smartlinker*: activa la sintaxis de SmartLinker, de forma que devuelve los nombres de las clases precedidos de la cadena "-includeWholeClass".
- *output_file*: fichero de salida. Si no se especifica, se utilizará la salida estándar.
- *vector*: especifica al programa que se tomarán los datos internamente, desde otra aplicación, mediante el método `returnResult()`, por lo que no debe dar salida ninguna, sea por pantalla o por fichero.
- *prof_to_parse*: fichero de perfil a ser procesado.
- *packageN*: lista de los paquetes que se quieren estudiar, es decir, sólo se mostrarán los nombres de clase que pertenezcan a alguno de estos paquetes. Si no se introduce ninguno, se mostrarán todas las clases encontradas en el perfil.

Al ejecutarse, ProfileParser toma cada clase que aparece en el perfil, obtiene el nombre de su superclase y de sus interfaces, actuando recursivamente de la misma manera con ellas. Así obtiene finalmente los nombres de todas las clases que intervienen en el perfil, y sus relacionadas por herencia o implementación.

Omisión de clases necesarias:

La omisión de clases necesarias para la ejecución de la aplicación usuaria es un efecto indeseado que se produce cuando el nombre de la clase no aparece en el perfil de ejecución (lo cual ocurre, por ejemplo, cuando se carga dicha clase, pero no se llega a instanciar), ni está relacionada con ninguna de las que aparece, ni SmartLinker es capaz de incluirla a pesar del análisis del código que realiza.

En estos casos la única solución posible es ejecutar la aplicación usuaria, anotando el nombre de la clase que echa en falta (cuando aparecen excepciones del tipo `ClassNotFoundException`), y volver a realizar la optimización, repitiendo este procedimiento hasta que la aplicación funcione correctamente.

EXPERIENCIAS REALES

Optimización de las librerías de AXIS para la aplicación usuario "IDS"

Invocación:

```
java -cp $CPATH SmartLinkerConfGenerator $* \  
  /usr/java/axis-1_1/lib/axis.jar \  
  /usr/java/axis-1_1/lib/commons-discovery.jar \  
  /usr/java/axis-1_1/lib/commons-logging.jar \  
  /usr/java/axis-1_1/lib/jaxrpc.jar \  
  /usr/java/axis-1_1/lib/saaj.jar \  
  /usr/java/axis-1_1/lib/wsdl4j.jar \  
  /home/rafaelb/PFC/Profiles/ids-oscar-standalone-todosusos.prof
```

Siendo la variable CPATH el classpath de la aplicación usuario al obtener el perfil:
"/home/rafaelb/PFC/Profiles/ids-oscar-standalone-todosusos.prof"

Clases y paquetes introducidos manualmente (omitidos inicialmente):

```
org.apache.axis.encoding.ser.*  
org.apache.axis.types.*  
org.apache.axis.NoEndPointException  
org.apache.axis.transport.java.*  
org.apache.axis.transport.http.*  
org.apache.axis.transport.local.*  
org.apache.axis.components.net.*
```

Resultados:

Antes: suma total: 1.467KB
Después: 1.100KB
Ahorro: 25%

Optimización de las librerías de NIST-SIP para la aplicación usuario "IDS"

Invocación:

```
java -cp $CPATH SmartLinkerConfGenerator $* \  
  /usr/java/jain-sip/JainSipApil.1.jar \  
  /usr/java/jain-sip/nist-sip-1.2.jar \  
  /usr/java/jain-sip/nist-sdp-1.0.jar \  
  /home/rafaelb/PFC/Profiles/ids-oscar-standalone-todosusos.prof
```

Clases y paquetes introducidos manualmente (omitidos inicialmente):

```
gov.nist.javax.sip.parser.*  
gov.nist.javax.sip.header.*  
gov.nist.javax.sdp.parser.*
```

Resultados:

Antes: suma total: 507KB
Después: 421KB
Ahorro: 17%