Betabeers

Depuración y profiling : casos prácticos

- **Xhprof »** https://github.com/facebook/xhprof
- **Xdebug »** https://github.com/ludovicPelle/vim-xdebug
- **Vdebug »** https://github.com/joonty/vdebug

xhprof

```
$ pecl install channel://pecl.php.net/xhprof-0.9.2
$ cd /tmp
$ tar zxvf /tmp/buildd/php5-5.3.3/pear-build-download/xhprof-0.9.2.tgz
$ cd xhprof-0.9.2/
$ su -
$ chown -R root.root .
$ cp -ra xhprof_html/ /usr/share/php
$ cp -ra xhprof_lib/ /usr/share/php
$ cd extension/
$ phpize
$ ./configure
$ make && make install
```

xhprof

```
$ su -
# cat  << EOF > /etc/php5/apache2/xhprof.ini
[xhprof]
extension=xhprof.so
xhprof.output_dir=/tmp/xhprof
EOF
# mkdir /tmp/xhprof
# chown www-data.www-data /tmp/xhprof
# sudo apt-get install graphviz
# /etc/init.d/apache2 reload
```

Xdebug

```
$ su -
# apt-get install php5-xdebug vim-nox
# cd /tmp
# git clone https://github.com/ludovicPelle/vim-xdebug.git
# cd vim-xdebug/plugin
# cp debugger.* /usr/share/vim/vim72/plugin/
```

vdebug

```
$ su -
# apt-get install php5-xdebug vim-nox
# cd /tmp
# git clone https://github.com/joonty/vdebug.git
# cp -ra vdebug/plugin/. /usr/share/vim/vim72/plugin/
# cp -ra vdebug/syntax/. /usr/share/vim/vim72/syntax/
```

- **Objetivo: Inicializar la ejecución de xhprof y reconocer las métricas generadas por el profiler**

- **Prueba: Algunos tests para mostrar incrementos de picos de memoria, tiempo de cpu, tiempo de ejecución o llamadas a funciones**

- **Conclusión: Veremos como un buen profiling puede permitirnos diagnosticar graves problemas de rendimiento y aplicar soluciones extremadamente rápidas**

- **Objetivo: Descubrir problemas por exceso de tiempo de ejecución**

- **Prueba: Implementar una llamada sleep con el tiempo de espera que necesitemos para emular el comportamiento**

- **Conclusión: Funciones que tarden mucho tiempo en ser ejecutadas son fácilmente identificables**

- **Solución: Evitar esperas. Son una fuente de race conditions**

Ejemplo 2: HTTP Request test

Betabeers

- **Objetivo: Demostrar que las llamadas a recursos externos son peligrosas si no se controlan**

- **Prueba: Implementar una llamada drupal_http_request contra el callback que definimos antes para demostrarlo**

- **Conclusión: Las llamadas a servicios externos deben controlarse**

- **Solución: Controlar los tiempos máximos de respuesta en las llamadas (en curl: CURLOPT_CONNECTTIMEOUT)**

crononauta

Ejemplo 3: User list peak mem test

Betabeers

- **Objetivo: Descubrir problemas por exceso de consumo de memoria**

- **Prueba: Implementar una comprobación user_access en hook_user, con un bucle de lectura de todos los usuarios de la plataforma**

- **Conclusión: En general las llamadas node_load y/o user_load son peligrosas cuando se ejecutan sobre un listado de nodos y/o usuarios sin límite**

- **Solución: Evitar las llamadas a user_load / node_load al recorrer nodos / usuarios**

crononauta

Ejemplo 4: taxonomy_get_children memory leak

Betabeers

- **Objetivo: Descubrir problemas por exceso de consumo de memoria**

- **Prueba: Implementar una llamada a taxonomy_get_children pasando como argumento tid=0 con miles de términos de free tags creados**

- **Conclusión: Hay que controlar todos los casos de uso posibles antes de llamar a taxonomy_get_children**

- **Solución: Evitar hacer llamadas a taxonomy_get_children usando como parámetro tid = 0**

crononauta

- **Objetivo: Demostrar un ejemplo de uso de xdebug**

- **Prueba: Recorrer un listado de usuarios y comprobar cómo las condiciones se cumplen para salir del bucle**

- **Conclusión: Los depuradores también son útiles para los lenguajes interpretados**

¿Dudas? ¿Preguntas?

Trabaja con nosotros:
me.apunto@crononauta.com

http://crononauta.com/
http://al.quimia.net/

Javier Carranza
javier.carranza@crononauta.com
twitter: @trunks

crononauta