Workshop xDebug mit Derick Rethans

https://phpinternals.news
https://github.com/xdebug/xdebug/
https://xdebug.org/docs/

xDebug macht die Verbindung zur IDE, nicht umgekehrt
-> PHP Server mit xDebug braucht physische Verbindung zur IDE
konfiguriert via via remote_host + remote_port

`remote_connect_back=1` -> xDebug ermittelt über Http-Header die anfragende IP

meisten xDebug Probleme sind Network-Probleme
Port 9000 wird von anderen genutzt, u.a. PHP-FPM

Log möglich -> `xdebug.remote_log`
nur für Debugging-Zwecke
nutzt (demnächst) systemd private tmp Verz. (ändert sich bei jedem Neustart)

Rückgabe zur IDE ist in XML

> Why not XML both ways? Quelle
> The primary reason is to avoid the requirement that a debugger engine has an XML parser available. XML is easy to generate, but requires additional libraries for parsing.

Demo-App: Whisky-Tasting-Tracker Dram :)
dazu gabs noch nix

Pfad-Mapping wenn Verz. auf Server nicht dem lokal entspricht (z.B. für Docker)

Start mit GET/POST/Cookie `XDEBUG_SESSION_START=1` (Inhalt ist egal)

Test via "Web Server Debug Validation" - funktioniert nur wenn keine HTTP HEader Rewrite aktiv sind

Suspend Breakpoint? k.A. was das macht :)

PhpStorm Features: Exception Breakpoints, Method Start Breakpoint
was noch fehlt: Method Exit Breakpoint (kann xDebug, aber PhpStorm noch nicht)

Remote Log

Start Debugging CLI

```
Log opened at 2019-10-04 09:54:38
I: Connecting to configured address/port: 127.0.0.1:9100.
I: Connected to client. :-)
-> <init xmlns="urn:debugger_protocol_v1" xmlns:xdebug="http://xdebug.org/dbgp/xdebug"
```

```
fileuri="file:///Users/r.hartenstein/bin/phpunit-6.phar" language="PHP"
xdebug:language_version="7.0.33" protocol_version="1.0" appid="5854" idekey="14559">
<engine version="2.6.1"><![CDATA[Xdebug]]></engine><author><![CDATA[Derick Rethans]]>
</author><url><![CDATA[http://xdebug.org]]></url><copyright><![CDATA[Copyright (c)
2002-2018 by Derick Rethans]]></copyright></init>

<- feature_set -i 1 -n show_hidden -v 1
-> <response xmlns="urn:debugger_protocol_v1"
xmlns:xdebug="http://xdebug.org/dbgp/xdebug" command="feature_set" transaction_id="1"
feature="show_hidden" success="1"></response>

<- feature_set -i 2 -n max_depth -v 1
-> <response xmlns="urn:debugger_protocol_v1"
xmlns:xdebug="http://xdebug.org/dbgp/xdebug" command="feature_set" transaction_id="2"
feature="max_depth" success="1"></response>

..

<- status -i 6
-> <response xmlns="urn:debugger_protocol_v1"
xmlns:xdebug="http://xdebug.org/dbgp/xdebug" command="status" transaction_id="6"
status="starting" reason="ok"></response>

<- step_into -i 7
-> <response xmlns="urn:debugger_protocol_v1"
xmlns:xdebug="http://xdebug.org/dbgp/xdebug" command="step_into" transaction_id="7"
status="break" reason="ok"><xdebug:message
filename="file:///Users/r.hartenstein/bin/phpunit-6.phar" lineno="3"></xdebug:message>
</response>

<- breakpoint_set -i 8 -t line -f file:///Users/r.hartenstein/dev/pfm-
webscraper/tests/bootstrap.php -n 3
-> <response xmlns="urn:debugger_protocol_v1"
xmlns:xdebug="http://xdebug.org/dbgp/xdebug" command="breakpoint_set"
transaction_id="8" id="58540001"></response>

..

<- stack_get -i 11
-> <response xmlns="urn:debugger_protocol_v1"
xmlns:xdebug="http://xdebug.org/dbgp/xdebug" command="stack_get" transaction_id="11">
<stack where="{main}" level="0" type="file"
filename="file:///Users/r.hartenstein/bin/phpunit-6.phar" lineno="3"></stack>
</response>

<- run -i 12
-> <response xmlns="urn:debugger_protocol_v1"
xmlns:xdebug="http://xdebug.org/dbgp/xdebug" command="run" transaction_id="12"
status="break" reason="ok"><xdebug:message
filename="file:///Users/r.hartenstein/dev/pfm-webscraper/tests/bootstrap.php"
lineno="3"></xdebug:message></response>
```

Watch ausführen

```
<- eval -i 97 --
JEdMT0JBTFNbJ0lERV9FVkFMX0NBQ0hFJ11bJzI3ODAzYzExLWU3ZTAtNDAwZC05MzIxLWUzNWZmMjJmOGU5ZSddP

-> <response xmlns="urn:debugger_protocol_v1"
xmlns:xdebug="http://xdebug.org/dbgp/xdebug" command="eval" transaction_id="97">
<property type="int"><![CDATA[2]]></property></response>
```

```
<- breakpoint_remove -i 101 -d 52640002
-> <response xmlns="urn:debugger_protocol_v1"
xmlns:xdebug="http://xdebug.org/dbgp/xdebug" command="breakpoint_remove"
transaction_id="101"><breakpoint type="line"
filename="file:///Users/r.hartenstein/dev/pfm-webscraper/vendor/symfony/browser-
kit/Client.php" lineno="333" state="enabled" hit_count="2" hit_value="0"
id="52640002"></breakpoint></response>
```

Erfolgreiches Ende

```
<- run -i 102
-> <response xmlns="urn:debugger_protocol_v1"
xmlns:xdebug="http://xdebug.org/dbgp/xdebug" command="run" transaction_id="102"
status="stopping" reason="ok"></response>

<- run -i 103
Log closed at 2019-10-04 09:51:38
```

Debug via CLI (über PHAR)

`phpunit.phar -dxdebug.remote_autostart=1`
Besser via CLI, braucht man PHPStorm nicht konfigurieren
-> ist aber nur problematisch wenn man viele PHP Versionen verwendet

"remote_enable" sollte "debug_enable" heißen

zwei Varianten:
`export XDEBUG_CONFIG="idekey=PHPSTORM; php phpunit.phar`
idekey: PhpStorm ignoriert, Netbeans und Eclipse prüfen exakt

Proxy: pydbgpproxy -- großes Pythonscript
https://www.jetbrains.com/help/phpstorm/xdebug-proxy.html
IDE verbindet sich mit Proxy, Script verbindet sich mit Proxy ..
nicht oft verwendet, besser SSH-Tunnel via Reverse-Port-Forward nutzen

Trace

auto_trace=1 ?
kann man mit der CI laufen lassen, bei Fehlern kann man reinschauen
mit return vals: xdebug.collect_return=1

mit argumenten: xdebug.collect_params=1
XT-Dateien: xdebug.trace_format=1

XT Syntax-Highlight mit VIM
https://github.com/xdebug/xdebug/blob/master/contrib/xt.vim

Trace ist cool bei verschlüsselten Code - Reverse Engineering
ioncube ist schwieriger
wann sinnvoll? bei Seg Faults oder unterdrückten Fehlermeldungen, vergessene die ()

Analyse von XT Files -> https://github.com/xdebug/xdebug/blob/master/contrib/tracefile-analyser.php

Memory Profiling macht kaum Sinn, weil der GC random dazwischenschießt

xdebug.trace_format=2 -> HTML-Format, eher nutzlos, wird wohl mit xDebug 3 gekickt

xdebug.file_link_format -> Links aus Dateien machen

xdebug_start_trace() & xdebug_stop_trace() - wenn man nur kleine Teile tracen will

Trace File zu FlameGraph -> https://github.com/pounard/xtrace2fg
wandelt XT zu FG komp. Datei um, muss mit FG gebaut werden
https://github.com/brendangregg/FlameGraph

Profiler

PhpStorm -> Tools -> Analyse Xdebug Profile Snapshot
QCacheGrind

alles mit "php::" sind interne Fkts
Ausführzeit > 100% - Rundungsfehler, Exit-Handler..

besser relative (%) statt absolute (ms) Ausführzeit anschauen

gibt derzeit keine Möglichkeit zu filtern, bei PHPUnit-Läufen z.B. das PHPUnit Framework..

"cycle" -> QCacheGrind aggregiert zyklische Aufrufe

Profiling macht Ausführung langsamer, weil "flush" nach jedem Funktionsaufruf

Pro-Tipp: beim profilen remote_enable=0 und trace_enable=0

Coverage

`--coverage-html=[pfad]`

langsam: xdebug überläd die internen PHP Opcodes -> coverage_enable steuert das
Whitelist: sammelt trotzdem alle Infos, schreibt sie aber nicht weg
Filterung vorher ist schwierig

HTML Generierung ist schwieriger Code. original vor 15 Jahren begonnen, wurde immer wieder angepasst

manchmal sind Zeilen nicht als ausführbar gekennzeichnet
xdebug hört nicht auf alle Opcodes, z.B. INIT_FCALL derzeit
Analyse: `php -dvld.active=1 script.php`
https://rancoud.com/read-phps-opcode/
nutzt Vulcan Logic Disassembler

100% Coverage sagt nicht dass man jeden Pfad im Code geprüft hat
xdebug_start_code_coverage() und *end* ... nur für wirklich krass kritische Business-Logik, analysiert jeden möglichen IF Pfad

Aktivierungs-Schalter

default_enable=1 -> schaltet schöne Stacktraces an für tägl. Gebrauch
coverage_enable=1 -> überläd Opcode-Aufrufe -> :exclamation_mark: *ist bei uns an. müssen wir mal prüfen was das Ausschalten für einen Impact bringt*
profiler_enable=1 -> Profiler aktivieren
auto_trace=1 -> Tracer aktivieren

Alternative Debugger

pcov -> wäre cool wenn der Autor auch zu xdebug contributet hätte (bestehender Code wäre zu schwierig für Autor zu verstehen gewesen) + gibt keine Visualisierung dafür
phpgdb -> gibt nicht die richtige Antwort / Protokoll

Ausklang

Side-fact:
composer: wenn Xdebug noch läuft, ruft sich Composer nochmal ohne auf
(macht es mit proc_open() und stream_set_blocking(..,0) vorher)
PHPUnit macht das nicht, Sebastian hat dagegen entschieden

Side-fact:
Xdebug Bug Reports nutzt Mantis :)
https://bugs.xdebug.org/view_all_bug_page.php

Support: https://xdebug.org/support
(auch mit Rechnung)
https://xdebug.org/log