Getting Started with PHP-FFI

Thomas Bley, June 2021 @ ipc

About me

- Senior PHP Developer
- Linux, PHP, MySQL since 2001
- studied at TU München
- working for Bringmeister in Berlin

What is PHP-FFI?

- PHP Foreign Function Interface
- PHP extension
- requires PHP 7.4
- currently experimental
- allows
  - loading of **shared libraries** (.so)
  - **calling C functions** and accessing C data structures
  - **in pure PHP** during runtime
- requires converting PHP types to C types

source:
php.net/manual/en/intro.ffi.php
phpconference.com/blog/php-ffi-and-what-it-can-do-for-you/

PHP-FFI use cases

- call C/C++ as shared libraries
  - allows direct hardware access (e.g. webcams, GPU)
  - gives more performance (e.g. image or video rendering)
  - integrate databases that have no PHP integration (e.g. DuckDB)
- call Go / Rust compiled as shared libraries
- easier than writing PHP extensions

source: github.com/gabrielrcouto/awesome-php-ffi

What is Go?

- programming language
- designed by Google
- statically typed
- memory safety, fast garbage collection
- more performance and concurrency
- can be compiled as a shared library

➔ combining PHP and Go gives a lot of new opportunities

source: en.wikipedia.org/wiki/Go_(programming_language)

PHP-Example: Ackermann function

```php
<?php
function ackermann(int $n, int $m): int {
    if ($n == 0) {
        return $m + 1;
    } else if ($m == 0) {
        return ackermann($n - 1, 1);
    } else {
        return ackermann($n - 1, ackermann($n, $m - 1));
    }
}
echo ackermann(3, 11);

// time php -dopcache.enable_cli=0 ackermann.php # 12.8s

// time php -dopcache.enable_cli=1 ackermann.php # 4.8s

// time php -dopcache.enable_cli=1 -dopcache.jit_buffer_size=32M ackermann.php # 3.2s
```

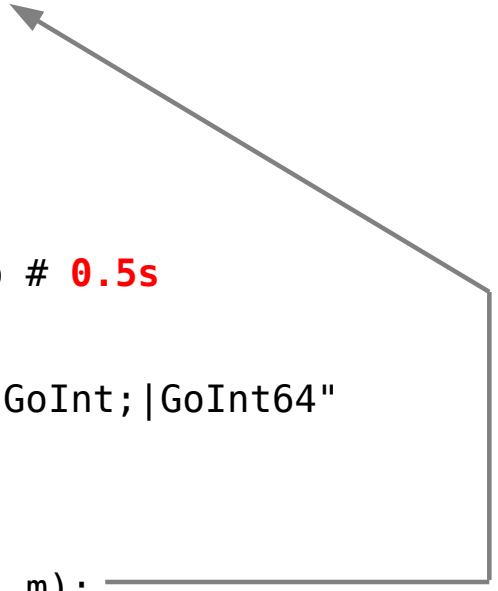
source: en.wikipedia.org/wiki/Ackermann_function
rosettacode.org/wiki/Ackermann_function

Go-Example: Ackermann function

```go
package main
import "C"

func main() {
    println(ackermann(3, 11));
}

//export ackermann
func ackermann(n int, m int) int {
    if n == 0 {
        return m + 1
    } else if m == 0 {
        return ackermann(n-1, 1)
    }
    return ackermann(n-1, ackermann(n, m-1))
}

// time go run ackermann.go # 0.7s
// go build ackermann.go && time ./ackermann # 0.5s (C/C++/Rust: 0.2s)
```

Calling Go from PHP

```
// create ackermann.so and ackermann.h:
// go build -o ackermann.so -buildmode=c-shared ackermann.go

<?php
error_reporting(E_ALL);

$ffi = FFI::cdef('long ackermann(long n, long m);', __DIR__ . '/ackermann.so');

echo $ffi->ackermann(3, 11);
unset($ffi);

// time php -dffi.enable=1 ackermann.php # 0.5s

// cat ackermann.h | grep -E "ackermann|GoInt;|GoInt64"
// typedef long long GoInt64;
// typedef GoInt64 GoInt;
// extern GoInt ackermann(GoInt n, GoInt m);
```



Using complex data types

```php
// php
$ffi = FFI::cdef('long ackermann(char* input);', __DIR__ . '/ackermann.so');
$result = $ffi->ackermann_json(json_encode([3, 11]));
echo json_decode(FFI::string($result));
FFI::free($result); // avoid memory leak
unset($ffi);
```

```go
// go
import "encoding/json"
//export ackermann_json
func ackermann_json(input *C.char) *C.char {
    var params [2]int
    err := json.Unmarshal([]byte(C.GoString(input)), &params)
    if err != nil { return nil }

    data, err := json.Marshal(ackermann(params[0], params[1]))
    if err != nil { return nil }

    return C.CString(string(data))
}
```

Checking memory leaks

```
// php
$ffi = FFI::cdef('long ackermann(char* input);', __DIR__ . '/ackermann.so');

for ($i=0; $i < 10000000) {
    $result = $ffi->ackermann_json(json_encode([0, 1]));
    FFI::free($result);
}
unset($ffi);

php -dffi.enable=1 ackermann.php &

while :; do pmap -x `pgrep php` | grep total; sleep 1; done
total kB         1489596   44040   16884
total kB         1489596   43864   16708
total kB         1489596   44568   17412
total kB         1489596   44048   16892
total kB         1489596   43896   16740
...
```

What is DuckDB?

- embedded database (similar to SQLite)
- optimized for analytical queries (OLAP)
- optimized for large tables, large resultsets
- column storage, 1 database = 1 single file
- vectorized query execution engine
- open source
- very fast

  but: no PHP extension available

source: duckdb.org/docs/why_duckdb

Using DuckDB

```
./duckdb
v0.2.6 8295e245d

D .open test.db

D PRAGMA enable_profiling;

D CREATE TABLE test1 AS SELECT * FROM read_csv_auto('test.csv.gz');
| Total Time: 8.63s |

D SELECT count(*) FROM test1;
| Total Time: 0.0166s |
| count_star() |
| 50000000     |

D DESCRIBE test1;
|    Field    |  Type   | Null | Key | Default | Extra |
| customer_id | INTEGER | YES  |     |         |       |
| product_id  | INTEGER | YES  |     |         |       |
```

Getting started with DuckDB

- download libduckdb-linux-amd64.zip
- extract duckdb.h and libduckdb.so
- C preprocessor directives are not supported in header files
  - remove “#ifdef __cplusplus...#endif” from duckdb.h
  - remove “DUCKDB_API ” from duckdb.h
  - remove lines with “duckdb_bind_uint8”, “duckdb_bind_uint16”
- FFI::cdef($header, $soLibrary) — Creates a new FFI object
- FFI::new($type) — Creates a C data structure
- FFI::addr($ptr) — Creates an unmanaged pointer to C data

How to integrate DuckDB?

master — duckdb / examples / embedded-c / main.c

48 lines (46 sloc) | 1.38 KB

```c
#include "duckdb.h"
#include <stdio.h>

int main() {
    duckdb_database db = NULL;
    duckdb_connection con = NULL;
    duckdb_result result;

    if (duckdb_open(NULL, &db) == DuckDBError) {
        fprintf(stderr, "Failed to open database\n");
        goto cleanup;
    }
    if (duckdb_connect(db, &con) == DuckDBError) {
        fprintf(stderr, "Failed to open connection\n");
        goto cleanup;
    }
    if (duckdb_query(con, "CREATE TABLE integers(i INTEGER, j INTEGER)
        fprintf(stderr, "Failed to query database\n");
        goto cleanup;
    }
    if (duckdb_query(con, "INSERT INTO integers VALUES (3, 4), (5, 6),
        fprintf(stderr, "Failed to query database\n");
        goto cleanup;
    }
    if (duckdb_query(con, "SELECT * FROM integers", &result) == DuckDB
        fprintf(stderr, "Failed to query database\n");
        goto cleanup;
```

```php
<?php
$ffi = FFI::cdef(
  file_get_contents(__DIR__ . '/duckdb.h'),
  __DIR__ . '/libduckdb.so');

$db = $ffi->new('duckdb_database');
$con = $ffi->new('duckdb_connection');
$result = $ffi->new('duckdb_result');

$ffi->duckdb_open(null, FFI::addr($db));
$ffi->duckdb_connect($db, FFI::addr($con));

$query = 'select current_date;';
$ffi->duckdb_query(
  $con, $query, FFI::addr($result));

$val = $ffi->duckdb_value_varchar(
  FFI::addr($result), 0, 0);
echo FFI::string($val);
```

Cleanup resources

```c
    if (duckdb_query(con, "SELECT * FROM integers", &result) == D
        fprintf(stderr, "Failed to query database\n");
        goto cleanup;
    }
    // print the names of the result
    for (size_t i = 0; i < result.column_count; i++) {
        printf("%s ", result.columns[i].name);
    }
    printf("\n");
    // print the data of the result
    for (size_t row_idx = 0; row_idx < result.row_count; row_idx+
        for (size_t col_idx = 0; col_idx < result.column_count; c
            char *val = duckdb_value_varchar(&result, col_idx, ro
            printf("%s ", val);
            duckdb_free(val);
        }
        printf("\n");
    }
    // duckdb_print_result(result);
cleanup:
    duckdb_destroy_result(&result);
    duckdb_disconnect(&con);
    duckdb_close(&db);
}
```

```php
<?php

FFI::free($val);

$ffi->duckdb_destroy_result(
    FFI::addr($result)
);

$ffi->duckdb_disconnect(FFI::addr($con));
FFI::free($con);

$ffi->duckdb_close(FFI::addr($db));
FFI::free($db);

unset($ffi);
```

DEMO

Thanks for listening!

Questions?

slides and sources:

github.com/thomasbley/php-go-integration

github.com/thomasbley/php-duckdb-integration