PB173 Perl

12 DBI, Mojolicious

Roman Lacko xlacko1@fi.muni.cz

2023-07-25

Obsah

Databázy

Databázy

Typické programy sú *bezastavové*; spracúvajú vstup a produkujú výstup nezávisle na tom, čo spracovali predtým (napr. ‹grep›).

Niekedy je však cieľom programu udržovať **stav** a pri ďalšom spustení s týmto stavom pokračovať ďalej.

Databázy: Úvod

Malé programy si typicky vystačia s perzistenciou nad súbormi:

- ‹Storable› [1]
- ‹JSON:PP›, ‹JSON›, ‹JSON::XS›, …
- ‹YAML::XS›, ‹YAML::PP›, ‹YAML::Syck›, ‹YAML›, …
- Ties
- DataBase Manager (DBM), ‹DB_File›, ‹AnyDBM_File›, …

[1] Core module

Databázy: ‹Storable›

```
use Storable;

sub load($fn = 'data.db') {
    -f $fn ? retrieve $fn : {};
}

sub save($data, $fn = 'data.db') {
    store $data, $fn;
}
```

Tento prístup sa dá výrazne vylepšiť pomocou RAII.

Databázy: ‹tie›

```
tie VARIABLE, CLASS, LIST
```

Zviaže premennú s triedou. Na rozdiel od ‹bless›

- ‹VARIABLE› nemusí byť referencia ani dokonca skalárna premenná,
- primitívne operácie na ‹VARIABLE› sa preložia na volania metód ‹CLASS›.

Zviazaným premenným sa tiež niekedy hovorí *enchanted variables*.

Databázy: ‹tie›

```
package Power;
use parent 'Tie::Array';

sub TIEARRAY($class, @) {
        return bless [], $class;
}

sub FETCH($, $index) {
        return $index ** 2;
}

tie my @power, 'Power';
say $power[5];          # ≈ Power::FETCH($self, 5) → 25
say scalar @power;      # ≈ Power::FETCHSIZE($self) → error
```

Databázy: DBM

DBM Files — Perzistencia asociatívnych polí na disku

V princípe podobné ako `Storable`, ale

- namiesto `load`, `modify` a `save` používajú `tie`,
- obsah súboru je možné čítať aj v iných jazykoch.

Databázy: DBM

```
use Fcntl;
use DB_File;

tie my %user, 'DB_File', 'user.db', O_RDWR | O_CREAT, 0600
    or die "user.db: $!";

say "Users in database: ", scalar keys %user;

if (!$user{root}) {
    $user{root} = {
        id => 0,
        password => Digest::SHA::sha256(RAW_PASSWORD),
        ...
    };
}
```

Databázy: DBM

Rôzne implementácie v štandardných moduloch Perlu:

`NDBM_File`	New DBM
`DB_File`	Berkeley DB
`GDBM_File`	GNU DBM
`ODBM_File`	Open DBM
`SDBM_File`	Substitute DBM
`AnyDBM_File`	Wrapper nad ostatnými

Častokrát vyžadujú nejakú natívnu knižnicu v systéme, napríklad `libgdbm.so`.

Databázy: RDBMS

Relational **D**ata**B**ase **M**anagement **S**ystem

Rôzne definíce podľa literatúry. V základe by mali poskytovať aspoň:

- Tabuľky (multimnožiny), kde riadky predstavujú záznamy.
- Vzťahy (relácie) medzi tabuľkami.

Databázy: RDBMS

Typicky so sebou prinášajú aj textové API:

Data Manipulation Language	Vyberanie, vkladanie
Data Query Language	Zisťovanie
Data Definition Language	Úpravy schém
Data Control Language	Práva, roly

Tento jazyk je skoro vždy SQL.

Databázy: RDBMS: DBI

‹DBI› je *jednotné* rozhranie medzi SQL databázou a Perlom.

Toto rozhranie implementuje *ovládač* v mennom priestore ‹DBD›,
napríklad ‹DBD::SQLite› alebo ‹DBD::Pg›.

```
use DBI;
my $db = DBI->connect(SOURCE, USER, AUTH, {ATTR...});
```

‹SOURCE› má formát ‹dbi:DRIVER:TEXT›. Význam ‹TEXT› určuje ovládač, napríklad:

```
dbi:SQLite:dbname=data.db
dbi:Oracle:host=is.muni.cz;sid=ISMU
```

Základné operácie s DBI

```
my $statement = $db->prepare(SQL_STATEMENT);
```

Preloží SQL príkaz. Môže používať ‹?› na neskoršie dosadenie hodnôt.

SQL Injection
Vyhýbajte sa interpolácii nevalidovaných reťazcov do SQL výrazu!

```
$statement->bind_param(NUMBER, VALUE);
```

Bezpečne dosadí do preloženého výrazu hodnotu na zadaný index.

```
$statement->execute([@BIND_VALUES])
```

Vykoná príkaz a vráti počet zmenených riadkov.
Môže tiež dočasne dosadiť hodnoty do výrazu.

`$s->bind_param(N, V)` dosadí `V` do výrazu `$s` permanentne.
`$s->execute(V)` dosadí `V` do výrazu len do skončenia príkazu.
Pripravené výrazy je možné používať opakovane.

```
$statement->fetchrow_array;
$statement->fetchrow_arrayref;
$statement->fetchrow_hashref;
```

Vrátia ďalší záznam z vykonaného výrazu.
Po poslednom vrátia ‹`undef`›.

```
$statement->fetchall_arrayref(SLICE, MAX_ROWS);
$statement->fetchall_hashref(KEY)
```

Vrátia *zostávajúce* riadky z vykonaného výrazu.

Spúšťanie jednorázových výrazov je možné zjednodušiť obaľujúcimi funkciami:

```
$db->do(SQL_STATEMENT, {ATTR}, BIND_VALUES…);
```

Vykoná jeden príkaz bez očakávania výsledku (‹INSERT›, ‹UPDATE›).

```
$db->selectrow_array(SQL_STATEMENT, {ATTR}, BIND_VALUES…);
$db->selectrow_arrayref(SQL_STATEMENT, {ATTR}, BIND_VALUES…);
$db->selectrow_hashref(SQL_STATEMENT, {ATTR}, BIND_VALUES…);
```

Preloží príkaz, spustí ho a vráti prvý riadok z výsledku.

```
$db->selectall_*(SQL_STATEMENT, {ATTR}, BIND_VALUES…);
```

Ako ‹selectrow_*›, ale vráti všetky riadky.

Vo východzom stave sa každá operácia vykoná naostro, ako keby bol za každým výrazom príkaz `commit`.

Toto je možné vypnúť príznakom `AutoCommit`, čím sa zmeny začnú vykonávať v jednej transakcii:

```
$db->{AutoCommit} = 0;
```

Prebiehajúca transakcia môže (v závislosti od databázového systému) zamknúť časti tabuliek alebo celé tabuľky.
Nemala by preto bežať zbytočne dlho.

Zapnutím `AutoCommit` sa prebiehajúca transakcia automaticky dokončí.

Čistejší spôsob zápisu transakcie je pomocou metód:

```
$db->begin_work
```

Začne transakciu vypnutím `AutoCommit`.

```
$db->commit
$db->rollback
```

Ukončí transakciu aplikovaním resp. zahodením zmien. Obnoví `AutoCommit`.

Toto je možné ešte obaliť RAII prístupom:

- Konštruktor začne transakciu.
- Deštruktor transakciu zahodí (kvôli výnimkám).
- `commit` sa vykoná explicitným zavolaním metódy.

Mojolicious

Mojolicious

Mojolicious je rozsiahly nástroj na jednoduchú tvorbu plnohodnotných webových aplikácií v Perli. Medzi jeho výhody patrí:

- Minimum kódu na obsluhu cieľového volania.
- Veľká sada zabudovaných utilít (koláčiky, sedenia, ...).
- Šablóny pre dynamicky generovaný obsah.
- WebSockets.

Okrem toho je to najpopulárnejší modul na CPAN.

- Modul, ktorý ešte viac zjednodušuje prácu s Mojolicious.
- Používa sa na prototypovanie, celá aplikácia môže byť obsiahnutá v jednom skripte.

```
use Mojolicious::Lite;

get '/' => sub ($c) {
    $c->render(text => "Hello there!");
};

app->start;
```

To je všetko. *Fakt!*

```
$ morbo hello.pl &
$ curl 'http://127.0.0.1:3000/'
Hello there!
```

Veľmi pohodlné spracovanie ciest (*routes*):

```
put '/create/:name' => sub ($c) {
    $c->render(text => "Created " . $c->param("name"));
};
```

```
$ curl -X PUT 'http://127.0.0.1/create/bender'
Created bender
$ curl -X PUT 'http://127.0.0.1/create/leela'
Created leela
```

Zabudovaný JSON výstup pre jednoduchú implementáciu RPC:

```
get '/api/v2/users' => sub ($c) {
    $c->render(json => $db->get_users);
};
```

Validácia parametrov:

```
del '/resource/:id' => sub ($c) {
    if (!$c->validation->required('id')->num(0, $max_id)->is_valid) {
        return $c->render(status => 400, text => 'Try again.');
    }

    # Delete resource.
};
```

Podpora viacerých formátov stránky:

```
get '/info' => [format => [qw(html txt)]] => sub ($c) {
    if ($c->stash eq 'html') {
        # Render HTML page.
    } else {
        # Render plaintext page.
    }
};
```

HTML stránky a šablóny môžu byť uložené

- na disku v ‹templates›,
- priamo v skripte v ‹__DATA__› pre jednoduchšie testovanie malých aplikácií.

Keď ‹Mojolicious::Lite› aplikácia dospeje, je vhodné ju upraviť na plnohodnotnú ‹Mojolicious› službu. Jej typická štruktúra je:

```
hello/
├── script/
│   └── hello              # The application CLI.
├── lib/                   # Application modules
│   ├── Hello.pm           # Web application module.
│   └── Hello/Controller/
│       └── Actions.pm     # Controller and routes endpoints.
├── public/                # Static files.
└── templates/             # Layouts and dynamic templates.
```

Podrobnosti viď ‹perldoc Mojolicious::Guides[::Growing]›.