PB173 Perl

11 Pokročilé možnosti

Roman Lacko <xlacko1@fi.muni.cz>

Obsah

Udalosťami riadené programovanie

Procedurálne programovanie

- Program má „lineárny” priebeh, v ktorom sa na rôznych miestach rozhoduje, ktorým smerom bude pokračovať (`if`, `while`, ...).
- Model, ktorý sme používali doteraz.

Udalosťami riadené programovanie

- Model vhodný na aplikácie, ktoré často interagujú s používateľom alebo inými aplikáciami.
- Abstrakcia nad predchádzajúcim modelom — v programe rozoznávame „udalosti“ a reakcie na nich.

Udalosťami riadené programovanie

Udalosť (event)

Všeobecný pojem pre vstup udalosťami riadeného programu (URP).
Napr. klik na grafický prvok, dáta v rúre, paket, signál, ...

Obsluha udalosti (event handler)

Kód, ktorý sa spustí v prípade, že nastane nejaká udalosť.
Rôzne udalosti môžu spustiť rôzne obsluhy, prípadne žiadne.

Cyklus udalostí (event loop)

Jadro URP, ktoré čaká na udalosti a spúšťa reakcie na nich.

Udalosťami riadené programovanie

Hoci cyklus udalostí nemusí byť nutne skutočný cyklus,
obvyklý vzor URP vyzerá napríklad takto:

```
my $event_loop = SomeEventLoopEngine->new(…);

while (my $event = $event_loop->wait) {
    if ($even->type eq 'signal') {
        handle_signal_event($event);
    } elsif ($event->type eq 'io') {
        handle_io_event($event);
    } …
}
```

`select(3)`, `poll(3)`

- Systémové volania POSIX.
- Riadenie vstupno-výstupných udalostí.

Udalosti:

- Je možné čítať deskriptor (rúru, soket, ...).
- Je možné zapisovať do deskriptora (voľný buffer).
- Na deskriptore nastala chyba (odpojený druhý koniec rúry).

Perl:

- Moduly `IO::Select` a `IO::Poll`.

Perl má aj **veľmi** nízkoúrovňové volanie `select` so štyroma argumentami. Preferujte však modul, ktorý je oveľa pohodlnejší.

`IO::Handle`

Pri práci s `IO::Select` a `IO::Poll` môže nastať niekoľko problémov.

Ako vieme, koľko bajtov môžeme prečítať alebo zapísať?

- Deskriptor je vo východzom stave *blokujúci*, tj. `read(3)` a `write(3)` sa môžu zablokovať, ak nie je dosť dát resp. je plná vyrovnávacia pamäť.
- Pre *neblokujúci* deskriptor tieto operácie skončia s chybou.
  - V C `open(3)` alebo `fcntl(3)` s príznakom `O_NONBLOCK`.
  - V Perli je každý deskriptor `IO::Handle` s metódou `blocking()`:

```
STDIN->blocking(0);     # fcntl(0, F_SETFL, O_NONBLOCK);

if (sysread(STDIN, $buffer, 4096) == 0
        && ($! == EAGAIN || $! == EWOULDBLOCK)) {
    # No data available yet.
}
```

IO::Handle

Ako dáta prečítať?

- `readline`, `read` atď. sú operácie s vyrovnávacou pamäťou, na neblokujúce deskriptory nie sú úplne pripravené.
- `sysread` a `syswrite` sú nízkoúrovňové funkcie (tiež dostupné aj ako metódy `IO::Handle`), ktoré obchádzajú vyrovnávacie pamäte.

```
my ($chunk, $bytes);

1 while ($bytes = sysread $handle, $chunk, 4096, length $chunk) > 0;

return "End of File"
    if $bytes == 0;
return "No more data for now"
    if $! == EAGAIN || $! == EWOULDBLOCK;
die "Error on handle: $!";
```

IO::Select

```
use IO::Select;

my $s = IO::Select->new;

$s->add(\*STDIN);
$s->add($pipe);

while ($s->handles) {
    foreach my $handle ($s->can_read) {
        # Process input from $handle.
    }
}
```

IO::Poll

Oproti `IO::Select` umožňuje naraz získať všetky typy udalostí.

```
use IO::Poll qw(POLLIN POLLOUT POLLHUP);

my $poll = IO::Poll->new;
$poll->mask($handle => POLLIN);         # Reading.
$poll->mask($pipe => POLLOUT);          # Writing.

while ($poll->handles > 0) {
    die "poll: $!" if !$poll->poll;

    foreach my $handle ($poll->handles) {
        my $ev = $poll->events($handle);
        # $ev & POLLIN  ≈ OK to read
        # $ev & POLLOUT ≈ OK to write
    }
}
```

AnyEvent

Jednotné rozhranie pre rôzne mechanizmy URP.

- IO
- Signály
- Časovače
- Podprocesy

Navyše je ľahko rozšíriteľný pomocou modulov i o ďalšie mechanizmy:

- Sieťová komunikácia (TCP, HTTP, IRC, FTP, Discord, ...)
- Grafické rozhrania (Qt, Tickit, i3, Sway, ...)
- Systémové rozhrania (DBus)
- Databázy (DBI)

AnyEvent

Spracovanie udalostí sa nastavuje pomocou „monitorov“ (*watchers*).

- Monitor môže popisovať reakcie na udalosti.
- *Neobsahuje* cyklus udalostí!
- Zničením monitora skončí aj spracovanie danej udalosti (RAII).

```
my $w_io;  # Separate declaration is necessary if we want to access
           # ‹$w_io› in handler. ‹my $w_io = AnyEvent->…› will not work.

$w_io = AnyEvent->io(fh => \*STDIN, poll => 'r', cb => sub {
    if (!handle_input(STDIN)) {
        # No more data will arrive.
        $w_io = undef;
    }
});
```

AnyEvent

Cyklus udalostí skrýva *podmienková premenná*.

- `$cv->recv` spustí cyklus udalostí, kým sa nesplní podmienková premenná.
- `$cv->send` splní podmienkovú premennú.

Alternatívne `$cv->begin` a `$cv->end` pre „transakcie“ alebo počítanie zdrojov.

```
my $cv = AnyEvent->condvar;

my $w = AnyEvent->signal(signal => 'USR1', cb => sub {
    $cv->send;
});

$cv->recv;
```

AnyEvent::Handle

Rozšírenie pre AnyEvent na neblokujúce čítanie vstupov.

```
my $w = AnyEvent::Handle->new(fh => \*STDIN,
    on_read => sub ($self) { … },
    on_eof => sub ($self) { … },
    on_error => sub ($self, $fatal, $message) { … },
);
```

Spracovanie zložitejšieho formátu pomocou *read queue*:

```
# Start with a line.
$w->push_read(line => sub ($, $line, $eol) { … });
# Then, 8 bytes must follow.
$w->push_read(chunk => 8, sub ($, $chunk) { … });
# Next, read everything up to ‹END›.
$w->push_read(regex => qr/END/, sub ($, $data) { … });
```

AnyEvent::Socket

Rozšírenie na prácu s TCP a UNIX soketmi.

```
use AnyEvent::Socket;

my $w = tcp_server($S_HOST, $S_PORT, sub ($socket, $c_host, $c_port) {
    $log->info("Client $c_host:$c_port connected");
    ...
    push @watchers, AnyEvent::Handle->new(fh => $socket, …);
});
```

Podobne klient ako `tcp_connect`.

Mojolicious

Mojolicious je rozsiahly nástroj na jednoduchú tvorbu plnohodnotných webových aplikácií v Perli.

- Minimum kódu na obsluhu koncového bodu.
- Veľká sada zabudovaných utilít (cookies, sessions, ...).
- Šablóny pre dynamicky generovaný obsah.
- Websockets.

Okrem toho je to najpopulárnejší modul na CPAN.

Mojolicious::Lite

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

```
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!
```

Mojolicious::Lite

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
```

Mojolicious::Lite

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.
};
```

Mojolicious::Lite

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.

Mojolicious

Keď `Mojolicious::Lite` aplikácia dospeje, je vhodné ju upraviť na plnohodnotnú `Mojolicious` službu.

Dospelý `Mojolicious` projekt má obvykle štruktúru:

```
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]`.