Writing Efficient XS

Sergey Aleynikov

YAPC::EU 2018

Optimizing?

Optimizing?

- Profiling

Optimizing?

- Profiling
- Profiling

Optimizing?

- Profiling
- Profiling
- Algorithm choice

perf top

- hv_common
- sv_setsv_flags
- pp_method_named

- gv_init_*
- gv_stash_*
- gv_fetchmeth_*
- hv_fetch_*

```
time perl -e '$a{foooooooooo}++ for (1..1_000_000)'
time perl -e 'my $x = "foooooooooo"; $a{$x}++ for (1..1_000_000)'
time perl -e '$a{foooooooooo}=1; my ($x) = keys %a; $a{$x}++ for (1..1_000_000)'
```

```
Foo*
Foo::new ()
CODE:
    RETVAL = new Foo();
OUTPUT:
    RETVAL
```

```
XS_EUPXS(XS_Foo_new)
{
    char *  CLASS = (char *)SvPV_nolen(ST(0));
    Foo *   RETVAL;;
    RETVAL = new Foo();
    {
        SV * RETVALSV;
        RETVALSV = sv_newmortal();
        sv_setref_pv( RETVALSV, CLASS, (void*)RETVAL );
        ST(0) = RETVALSV;
    }
    }
    XSRETURN(1);
}
```

```
static SV* foo_key;

BOOT:
{
    foo_key = newSVpvn_share("fooooooo", strlen("fooooooo"), 0);
}
```

```
static SV* foo_key;

BOOT:
{
    foo_key = newSVpvn_share("foooooooo", strlen("foooooooo"), 0);
}

# Not thread safe - needs CLONE section
```

```c
while ((entry = hv_iternext(hv))) {
    char* key = HeKEY(entry);
    if (strEQ(key, "magic_value") {...}
}
```

```c
while ((entry = hv_iternext(hv))) {
    char* key = HeKEY(entry);
    if (strEQ(key, "magic_value") {...}
}
```

$\Downarrow$

```c
while ((entry = hv_iternext(hv))) {
    if (HeKEY(entry) == magic_value_sv) {...}
}
```

```
hv_store_ent(hash, keysv, newSV(), 0);
hv_store_ent(hash, keysv, newSV(), 0);
```

$\Downarrow$

```
hv_store_ent(hash, keysv, &PL_sv_undef, 0);
```

```
while (HE* he = hv_iternext(hv)) {
    SV* key_sv = hv_iterkeysv(he);
    int keyval = SvUV(key_sv);
}
```

```perl
my $bar = exists $foo{bar} ? $foo{bar} : ($foo{bar} = 42);
my $bar = exists $foo{bar} ? $foo{bar} : ($foo{bar} = 42);

HE* he = hv_fetch_ent(hash, key_sv, 1, 0);
if (SvOK(HeVAL(he)) {
    HeVAL(he) = newSViv(42);
}
```

```
my $bar = exists $foo{bar} ? $foo{bar} : ($foo{bar} = 42);

HE* he = hv_fetch_ent(hash, key_sv, 1, 0);
if (SvOK(HeVAL(he)) {
    SvREFCNT_dec(HeVAL(he));
    HeVAL(he) = newSViv(42);
}
```

```perl
my $bar = exists $foo{bar} ? $foo{bar} : ($foo{bar} = 42);
```

```c
HE* he = hv_common(hash, key_sv, NULL, 0, 0,
    HV_FETCH_EMPTY_HE | HV_FETCH_LVALUE, NULL, 0);
if (!HeVAL(he)) {
    HeVAL(he) = newSViv(42);
}
```

```c
void
oldfoo (SV* self, SV* arg) {
    ...; // do stuff
}

void
foo (SV* self, SV* arg) {
    PUSHMARK(SP);
    XPUSHs(self);
    XPUSHs(arg);
    PUTBACK; ENTER; SAVETMPS;
    call_method("oldfoo", G_DISCARD);
    FREETMPS; LEAVE;
}
```

```c
void
oldfoo (SV* self, SV* arg) {
    ...; // do stuff
}

void
foo (SV* self, SV* arg) {
    PUSHMARK(SP);
    EXTEND(SP, 2);
    PUSHs(self);
    PUSHs(arg);
    PUTBACK; ENTER; SAVETMPS;
    call_method("oldfoo", G_SCALAR);
    SPAGAIN; POPs; PUTBACK;
    FREETMPS; LEAVE;
    // new stuff
}
```

```c
void
oldfoo (SV* self, SV* arg) {
    ...; // do stuff
}

void
foo (SV* self, SV* arg) {
    PUSHMARK(SP);
    EXTEND(SP, 2);
    PUSHs(self);
    PUSHs(arg);
    PUTBACK; ENTER; SAVETMPS;
    call_method("oldfoo", G_SCALAR);
    SPAGAIN; POPs; PUTBACK;
    FREETMPS; LEAVE;
    // new stuff
}
```

$\Downarrow$

```c
void
oldfoo (SV* self, SV* arg) {
    oldfoo(self, arg);
}
void
foo (SV* self, SV* arg) {
    oldfoo(self, arg);
    newstuff();
}
```

```
for (int i = 0; i < items; ++i) {
    process(ST(i));
}
```

```
for (int i = 0; i < items; ++i) {
    process(ST(i));
}
```

$\Downarrow$

```
for (int i = 0; i < items; ++i) {
    process(++SP);
}
```

```
class Foo {SV* val;};

void
Foo::bar(SV* arg) {
    THIS->val(arg);
}
```

```
class Foo {SV* val;};

void
Foo::bar(SV* arg) {
    THIS->val(arg);
}
```

$\Downarrow$

```
void
Foo::bar(SV* arg) {
    sv_setsv(THIS->val, arg);
}
```

```
class Foo {SV* val;};

void
Foo::bar(SV* arg) {
    THIS->val(arg);
}
```

$\Downarrow$

```
void
Foo::bar(SV* arg) {
    sv_setsv(THIS->val, arg);
}
```

$\Downarrow$

```
void
Foo::bar(SV* arg) {
    THIS->val(arg);
    SvREFCNT_inc_simple_NN(arg);
}
```

```
size_t len = AvFILL(av);
for (size_t i = 0; i<= len; ++i) {
    SV* elem = *av_fetch(av);
    ...;
}
```

```c
size_t len = AvFILL(av);
SV** elems = AvARRAY(av);
for (size_t i = 0; i<= len; ++i) {
    SV* elem = *(elems++);
    ...;
}
```

General directions

- Avoid tie()

General directions

- Avoid tie()
- Avoid overload

General directions

- Avoid tie()
- Avoid overload
- Look at opcodes for simple subs

Questions?

Writing XS in plain C

Ref::Util - more than you ever wanted to know

https://github.com/dur-randir/yapc-eu-2018