NAME

perlmroapi - Perl method resolution plugin interface

DESCRIPTION

As of Perl 5.10.1 there is a new interface for plugging and using method resolution orders other than the default (linear depth first search). The C3 method resolution order added in 5.10.0 has been re-implemented as a plugin, without changing its Perl-space interface.

Each plugin should register itself by providing the following structure

```
struct mro_alg {
    AV *(*resolve)(pTHX_ HV *stash, U32 level);
    const char *name;
    U16 length;
    U16 kflags;
    U32 hash;
};
```

and calling "Perl_mro_register":

```
Perl_mro_register(aTHX_ &my_mro_alg);
```

resolve
    Pointer to the linearisation function, described below.

name
    Name of the MRO, either in ISO-8859-1 or UTF-8.

length
    Length of the name.

kflags
    If the name is given in UTF-8, set this to "HVhek_UTF8". The value is passed direct as the
    parameter *kflags* to "hv_common()".

hash
    A precomputed hash value for the MRO's name, or 0.

Callbacks

The "resolve" function is called to generate a linearised ISA for the given stash, using this MRO. It is

called with a pointer to the stash, and a *level* of 0. The core always sets *level* to 0 when it calls your function - the parameter is provided to allow your implementation to track depth if it needs to recurse.

The function should return a reference to an array containing the parent classes in order. The names of the classes should be the result of calling "HvENAME()" on the stash. In those cases where "HvENAME()" returns null, "HvNAME()" should be used instead.

The caller is responsible for incrementing the reference count of the array returned if it wants to keep the structure. Hence, if you have created a temporary value that you keep no pointer to, "sv_2mortal()" to ensure that it is disposed of correctly. If you have cached your return value, then return a pointer to it without changing the reference count.

Caching

Computing MROs can be expensive. The implementation provides a cache, in which you can store a single "SV *", or anything that can be cast to "SV *", such as "AV *". To read your private value, use the macro "MRO_GET_PRIVATE_DATA()", passing it the "mro_meta" structure from the stash, and a pointer to your "mro_alg" structure:

```
meta = HvMROMETA(stash);
private_sv = MRO_GET_PRIVATE_DATA(meta, &my_mro_alg);
```

To set your private value, call "Perl_mro_set_private_data()":

```
Perl_mro_set_private_data(aTHX_ meta, &c3_alg, private_sv);
```

The private data cache will take ownership of a reference to private_sv, much the same way that "hv_store()" takes ownership of a reference to the value that you pass it.

Examples

For examples of MRO implementations, see "S_mro_get_linear_isa_c3()" and the "BOOT:" section of *ext/mro/mro.xs*, and "S_mro_get_linear_isa_dfs()" in *mro_core.c*

AUTHORS

The implementation of the C3 MRO and switchable MROs within the perl core was written by Brandon L Black. Nicholas Clark created the pluggable interface, refactored Brandon's implementation to work with it, and wrote this document.