MarvinScratch

MarvinHumphrey's scratch pad for Lucy-related topics

Should Clownfish objects extend Perl's SV type?

Is it possible to extend Perl object SVs the same way we would extend a Python class?

```
    struct cfish_Obj {
        void *sv_any;
        U32   sv_refcnt;
        U32   sv_flags;
        union { void *ptr; IV iv; } sv_union;
        cfish_VTable *vtable;
    };
```

```
cfish_Obj*
cfish_VTable_make_obj(cfish_VTable *self) {

    // Allocate an empty, undef SV but with a greater size.
    char *raw;
    Newxz(raw, self->obj_alloc_size, char);  // Newxz to memzero object.
    SV *obj_as_sv       = (SV*)raw;
    SvREFCNT(obj_as_sv) = 1;
    SvANY(obj_as_sv)    = 0;
    SvFLAGS(obj_as_sv)  = 0;

    // Make the object into an inner Perl object SV.
    SvOBJECT_on(obj_as_sv);
    PL_sv_objcount++;
    SvUPGRADE(obj_as_sv, SVt_PVMG);
    sv_setiv(obj_as_sv, PTR2IV(obj_as_sv));  // is circular ref a problem?

    // Connect class association.
    // TODO: cache stash pointer as a member of the VTable.
    cfish_CharBuf *class_name = Cfish_VTable_Get_Name(self);
    HV *stash = gv_stashpvn((char*)Cfish_CB_Get_Ptr8(class_name),
                            Cfish_CB_Get_Size(class_name), TRUE);
    SvSTASH_set(obj_as_sv, (HV*)SvREFCNT_inc(stash));

    // Add Clownfish VTable pointer.
    cfish_Obj *obj = (cfish_Obj*)obj_as_sv;
    obj->vtable = self;

    return obj;
}

uint32_t
cfish_Obj_get_refcount(cfish_Obj *self) {
    return SvREFCNT((SV*)self);
}

cfish_Obj*
cfish_Obj_inc_refcount(cfish_Obj *self) {
    SvREFCNT_inc_simple_void_NN((SV*)self);
    return self;
}

uint32_t
cfish_Obj_dec_refcount(cfish_Obj *self) {
    modified_refcount = SvREFCNT((SV*)self->ref.host_obj) - 1;
    // If the SV's refcount falls to 0, DESTROY will be invoked from
    // Perl-space.
    SvREFCNT_dec((SV*)self->ref.host_obj);
    return modified_refcount;
}

void*
cfish_Obj_to_host(cfish_Obj *self) {
    return newRV_inc((SV*)self);
}

void
cfish_Obj_destroy(cfish_Obj *self) {
    // no-op, because Perl is going to take care of freeing `self`.
}
```

Tuning for specific processors

Lucy's build should be tuned for the most specific processor type possible by default. Organizations which distribute code which must run on multiple processors will have to override this setting to produce lowest-common-denominator code, but that's OK.