QEMU internal APIs

How abstractions inside QEMU (don't) work together

Eduardo Habkost <ehabkost@redhat.com>

Contents

Not included:

- The *right way* to do something
- Solutions to issues
- Every single API in QEMU

Context

"QEMU is a generic and open source machine emulator and virtualizer."

— *http://qemu.org/*

External Interfaces

Command-line

```
$ qemu-system-x86_64 -cpu Nehalem -vga cirrus \
      -device e1000,mac=01:02:03:04:05:06     \
      -machine pc-i440fx-2.7,accel=kvm
```

Config files

```
[device]
  driver = "e1000"
  mac = "01:02:03:04:05:06"

[machine]
  type = "pc-i440fx-2.7"
  accel = "kvm"
```

Human Monitor (HMP)

```
QEMU 2.8.50 monitor - type 'help' for more information
(qemu) device_add e1000,mac=01:02:03:04:05:06
(qemu) info network
e1000.0: index=0,type=nic,model=e1000,macaddr=01:02:03:04:05:06
(qemu) info kvm
kvm support: enabled
(qemu) info cpus
* CPU #0: pc=0xffffffff8105ea06 (halted) thread_id=21209
(qemu)
```

Machine Monitor (QMP)

```
⇒ { "execute": "device_add",
     "arguments": { "mac": "01:02:03:04:05:06",
                    "driver": "e1000" } }
⇐ { "return": {} }
⇒ { "execute": "query-cpus",
     "arguments": {} }
⇐ { "return": [{ "halted": false, "pc": 133130950,
                  "current": true,
                  "qom_path": "/machine/unattached/device[0]",
                  "thread_id": 22230, "arch": "x86",
                  "CPU": 0 } ] }
⇒ { "execute": "query-kvm",
     "arguments": {} }
⇐ { "return": { "enabled": true, "present": true } }
```

QEMU Internals

Things to handle:

- Configuration options
- Monitor commands
- Device configuration
- Device state (including migration)
- Backend configuration
- *etc.*

Internal APIs

API: QemuOpts (2009)

- Handling of command-line and config file options
- Few basic data types
- Flat data model

QemuOpts usage

- ~~Most~~ Many command-line options
- Internal storage of config options
- Config file support (`-readconfig`, `-writeconfig`)

QemuOpts example

```
$ qemu-system-x86_64 -memory 2G,maxmem=4G
```

⇓

```
static QemuOptsList qemu_mem_opts = {
    .name = "memory",
    .implied_opt_name = "size",
    .head = QTAILQ_HEAD_INITIALIZER(qemu_mem_opts.head),
    .merge_lists = true,
    .desc = {
        { .name = "size",   .type = QEMU_OPT_SIZE, },
        { .name = "slots",  .type = QEMU_OPT_NUMBER, },
        { .name = "maxmem", .type = QEMU_OPT_SIZE, },
        { /* end of list */ }
    },
};
```

API: qdev (2009)

- Bus/device tree
- Single API to create, configure and plug devices
- Property system, introspection
- Rebuilt on top of QOM (2011)

qdev usage

- Every device emulated by QEMU
- External generic interfaces (e.g. `-device`, `device_add`)
- Introspection of device tree (e.g. `info qtree`)

qdev Example

```
$ qemu-system-x86_64 -device e1000,mac=12:34:56:78:9a:bc
```

⇓

```
#define DEFINE_NIC_PROPERTIES(_state, _conf)                \
    DEFINE_PROP_MACADDR("mac",   _state, _conf.macaddr), \
    DEFINE_PROP_VLAN("vlan",     _state, _conf.peers),   \
    DEFINE_PROP_NETDEV("netdev", _state, _conf.peers)

static Property e1000_properties[] = {
    DEFINE_NIC_PROPERTIES(E1000State, conf),
    DEFINE_PROP_BIT("autonegotiation", E1000State,
                    compat_flags, E1000_FLAG_AUTONEG_BIT, true),
    /* [...] */
};
```

qdev device tree

```
(qemu) info qtree
bus: main-system-bus
  type System
  dev: hpet, id ""
    gpio-in "" 2
    gpio-out "" 1
    gpio-out "sysbus-irq" 32
    timers = 3 (0x3)
    msi = false
    hpet-intcap = 4 (0x4)
    mmio 00000000fed00000/0000000000000400
  dev: kvm-ioapic, id ""
    gpio-in "" 24
    gsi_base = 0 (0x0)
    mmio 00000000fec00000/0000000000001000
```

API: QAPI (2011)

- Formal schema for interfaces
- Visitor API
- Generated code for:
  - C types
  - Serialization
  - Visitors
  - QMP commands and events
  - Interface introspection
  - Documentation

QAPI usage

- All QMP commands
- Some command-line options

QAPI Example: `chardev-add`

```
⇒ { "execute" : "chardev-add",
     "arguments" : {
         "id" : "bar",
         "backend" : { "type" : "file",
                       "data" : { "out" : "/tmp/bar.log" } } } }
⇐ { "return": {} }
```

chardev-add QAPI schema

```
{ 'command': 'chardev-add',
  'data': { 'id': 'str',
            'backend': 'ChardevBackend' },
  'returns': 'ChardevReturn' }

{ 'union': 'ChardevBackend',
  'data': { 'file': 'ChardevFile',
            'serial': 'ChardevHostdev',
            [...] } }

{ 'struct': 'ChardevFile',
  'data': { '*in' : 'str', 'out' : 'str', '*append': 'bool' },
  'base': 'ChardevCommon' }
```

⇓

```
ChardevReturn *qmp_chardev_add(const char *id,
                               ChardevBackend *backend,
                               Error **errp);
```

API: QOM (2011)

(Don't confuse with QObject)

- QEMU Object Model
- Type hierarchy
- Property system, introspection
- qdev rebuilt on top of it

QOM in action

```
$ qemu-system-x86_64 -device e1000,mac=12:34:56:78:9a:bc
```

```
$ qemu-system-x86_64 \
  -object memory-backend-file,size=512M,mem-path=/hugetlbfs \
  [...]
```

```
$ qemu-system-x86_64 -machine pc,accel=kvm
```

```
$ qemu-system-x86_64 -cpu Nehalem,+vmx,-nx,pmu=on
```

```
qemu_irq qemu_allocate_irq(...)
{
    irq = IRQ(object_new(TYPE_IRQ));
    [...]
}
```

```
void memory_region_init(...)
{
    object_initialize(mr, sizeof(*mr), TYPE_MEMORY_REGION);
    [...]
}
```

Mixing Abstractions

Example: –numa option

(QemuOpts + QAPI)

```
$ qemu-system-x86_64 -numa node,cpus=0-1,mem=2G \
                     -numa node,2-3,mem=2G
```

-numa QemuOptsList

```
QemuOptsList qemu_numa_opts = {
    .name = "numa",
    .implied_opt_name = "type",
    .head = QTAILQ_HEAD_INITIALIZER(qemu_numa_opts.head),
    .desc = { { 0 } }
};
```

–numa QAPI schema

```
{ 'union': 'NumaOptions',
  'data': {
    'node': 'NumaNodeOptions' } }

{ 'struct': 'NumaNodeOptions',
  'data': { '*nodeid': 'uint16',
            '*cpus':   ['uint16'],
            '*mem':    'size',
            '*memdev': 'str' } }
```

-numa glue

```
static int parse_numa(void *opaque, QemuOpts *opts, Error **errp)
{
    NumaOptions *object = NULL;
    Visitor *v = opts_visitor_new(opts);
    visit_type_NumaOptions(v, NULL, &object, &err);
    /* [...] */
}
```

Summary: `-numa`

- QAPI-based implementation
- QemuOpts-based interface
- All options documented in QAPI schema
- No duplication of QAPI schema info in the C code
- Glue code made possible by `OptsVisitor`
- Similar method used for: `-net`, `-netdev`, `-acpitable`, `-machine`

Example `object-add` QMP command

(QAPI + QOM)

```
⇒ { "execute": "object-add",
     "arguments": { "qom-type": "rng-random", "id": "rng1",
                    "props": { "filename": "/dev/hwrng" } } }
⇐ { "return": {} }
```

object-add: QOM properties

```
static void rng_random_init(Object *obj)
{
    RngRandom *s = RNG_RANDOM(obj);
    object_property_add_str(obj, "filename",
                            rng_random_get_filename,
                            rng_random_set_filename,
                            NULL);
    /* [...] */
}
```

object-add QAPI schema

```
{ 'command': 'object-add',
  'data': {'qom-type': 'str',
           'id': 'str',
           '*props': 'any' } }
```

Summary: `object-add`

- QOM-based implementation
- QAPI-based interface
- QAPI schema is incomplete
- Similar method used for: `device_add`

Example: −cpu option

(command-line + qdev/QOM)

```
$ qemu-system-x86_64 -cpu Nehalem,+vmx,-nx,pmu=on
```

-cpu: QOM properties

```
void x86_cpu_register_bit_prop(X86CPU *cpu,
                               const char *prop_name,
                               uint32_t *field, int bitnr)
{
    object_property_add(OBJECT(cpu), prop_name, "bool",
                        x86_cpu_get_bit_prop,
                        x86_cpu_set_bit_prop,
                        x86_cpu_release_bit_prop, fp,
                        &error_abort);
}
/* [...] */
static Property x86_cpu_properties[] = {
    DEFINE_PROP_BOOL("pmu", X86CPU, enable_pmu, false),
    /* [...] */
};
```

-cpu: glue code

```
static void x86_cpu_parse_featurestr(const char *typename,
                                     char *features,
                                     Error **errp)
{
    for (featurestr = strtok(features, ",");
         featurestr; featurestr = strtok(NULL, ",")) {
        /* [...] */
        prop->driver = typename;
        prop->property = g_strdup(name);
        prop->value = g_strdup(val);
        prop->errp = &error_fatal;
        qdev_prop_register_global(prop);
    }
}
```

Summary: `-cpu`

- qdev/QOM-based implementation
- command-line interface
- Glue based on qdev's `-global` properties
- Not described on QAPI schema
- Still not ported to QemuOpts

Example: `query-cpu-model-expansion`

(QAPI + QOM)

```
⇒ { "execute": "query-cpu-model-expansion",
     "arguments": { "type": "static",
                    "model": { "name": "Nehalem" } } }
⇐ {"return": { "model": {"name": "base",
                         "props": { "cmov": true, "ia64": false,
                                    "aes": false, "mmx": true,
                                    "rdpid": false,
                                    "arat": false,
                                    [...] } } } }
```

q-c-m-expansion: QAPI schema

```
{ 'command': 'query-cpu-model-expansion',
  'data': { 'type': 'CpuModelExpansionType',
            'model': 'CpuModelInfo' },
  'returns': 'CpuModelExpansionInfo' }

{ 'struct': 'CpuModelExpansionInfo',
  'data': { 'model': 'CpuModelInfo' } }

{ 'struct': 'CpuModelInfo',
  'data': { 'name': 'str',
            '*props': 'any' } }
```

Summary: `q-c-m-expansion`

- qdev/QOM-based implementation
- QAPI-based interface
- QAPI schema is incomplete
- Arch-specific glue code (currently)

Summary: QOM & the QAPI schema

- QOM classes and properties are registered at run time (`class_init` & `instance_init` methods)
- QAPI schema is a static file
- QOM class-specific info doesn't appear on QAPI schema

Conclusion

Please ask

Some practices are not well-documented.

When in doubt, ask developers & qemu-devel.

Questions?

Thank You

This slide deck:

https://habkost.net/talks/fosdem-2017/

Incomplete guide to QEMU APIs:

https://goo.gl/c8SzD7

Appendix

Interface documentation

- QAPI schema: **comprehensive**
- QemuOpts: **brief**
- QOM types and properties: **almost none**

Data types

Type	int	float	bool	string	enum	list	dict
QemuOpts	✔*		✔	✔		✔**	
qdev	✔*		✔	✔	✔		
QAPI	✔	✔	✔	✔	✔	✔	✔
QOM	✔	✔	✔	✔	✔	✔**	✔**

* Limited support

** Very limited support

Abstractions equivalency

QemuOpts	qdev	QOM	QObject	QAPI
QemuOptsList	type	class	-	schema struct
QemuOptDesc	property	property	-	schema field
option default	property default	property default	-	-
QemuOpts	device	instance	QDict	C struct
QemuOpt	property value	property value	QObject	C field

▨ static data ■ runtime data

QOM: internal vs. external

- Unclear:
  - What should be user-visible
  - What should be a stable interface
- *Types* can be hidden from the user (`no_user`)
- *Properties* can't be hidden
  - Today's (undocumented) convention: **"x-"** prefix

QOM tree manipulation

- QOM device/object tree can be manipulated through QMP
- Not very popular in practice

Not Covered

- Migration system (VMState, savevm handlers)
- Main loop
- Char devices
- Block layer
- Coroutines
- Many more

Interfaces *vs* internal abstractions

- QMP commands: built on top of **QAPI**
- (Many) Command-line options: handled using **QemuOpts**
- `-device`/`device_add`: built on top of **qdev**
- `-object`/`object-add`: built on top of **QOM**
- `-cpu`: built on top of **qdev**

Can translate:

- QAPI ⇔ QObject
- qdev ⇒ QOM (qdev *is* QOM)
- QemuOpts ⇒ QAPI structs
- QemuOpts ⇒ QOM

anything ⇒ QAPI schema

Not possible by definition

(QAPI schema is a static source code file)

anything ⇒ QemuOpts

- Not translated
- Limited QemuOpts data model
- Not a problem in practice

Other "schema" data

(QAPI schema, QOM type hierarchy, config groups)

- No mechanisms for translation
- QOM/QAPI dilemma when designing new interfaces
- Normally we choose QAPI
  - Exceptions: CPU config, `device_add` and `object-add` options
- Exception: a few QemuOpts config groups (property descriptions are optional)

Issue: Introspection & data availability

Translation issues:

- Incompatible data-types
- Data unavailable at the right time

Issue: overlap and duplication

- APIs providing similar features
- Some code is not shared

Duplication example:

- Parsing code

Overlap example:

Data representation: QemuOpts vs QOM vs QAPI

- OK when translation is possible
- Interface design dilemmas when translation is not possible
- Affects design of external interfaces

Steps

- Compile time (static)
- Runtime:
  - Event: Initialization (static)
    - static var
    - hardcoded at main()
    - QOM/qdev type registration
    - QOM/qdev class_init
    - QOM/qdev instantiation
  - Event: Monitor is available
  - Event: machine creation
  - Event: machine is running

Data items

- qdev type list
- QOM properties
- QemuOpts sections
- QAPI schema
- machine-type list
- machine-type defaults
- machine-type devices

Static data treated like dynamic data

- QOM type hierarchy
- QOM property lists
- machine-type default options
- machine-type default devices/buses

Dynamic data whose static defaults are hard to discover

- machine-type default options
- machine-type default devices/buses

Static data that never becomes available to the outside

- Some machine-type behavior