QTest 在 QEMU 中的简单应用

PLCT实验室 陈嘉炜

报告简介

QTest是什么?	• QEMU中的单元测试框架
QTest可以做什么?	• QTest测试哪些内容
如何进行测试?	• QTest测试命令使用说明
如何编写测试用例?	• QTest测试用例编写

QTest介绍

- QTest是QEMU单元测试中使用的内部框架。
- 基于QTest的测试可以启动一个或多个QEMU二进制文件，并通过socket控制测试进行。
- 通常使用QMP-socket运行二进制文件触发事件。

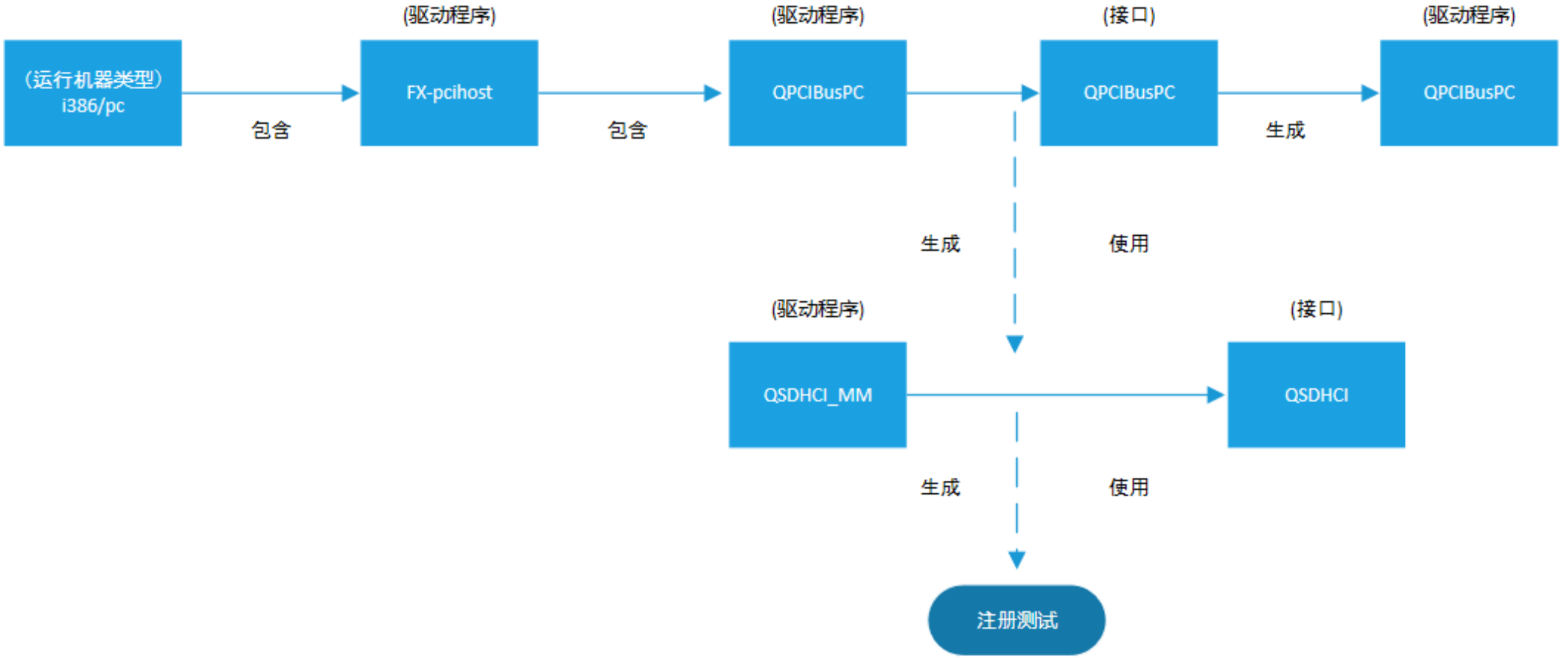
QTest驱动框架

- 通过驱动程序框架**libqos**：
  - QTest可以生成QEMU支持的计算机类型和一组驱动程序的描述。单元测试可以请求一个驱动程序，框架负责使用提供该驱动程序的选项来启动QEMU。

QTest驱动框架

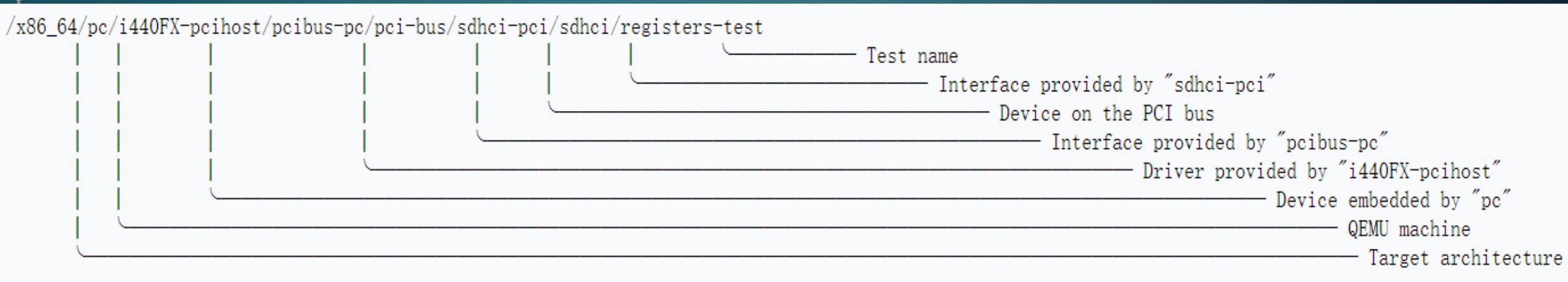
- 该框架可以提供如下对象的测试：

1. 设备的接口
2. 设备的驱动程序
3. QEMU设备映射关系
4. PCI总线的接口
5. PCI总线的驱动程序
6. QEMU“pc”计算机及其嵌入式设备


(运行机器类型)
i386/pc
包含
(驱动程序)
FX-pcihost
包含
(驱动程序)
QPCIBusPC
(接口)
QPCIBusPC
生成
(驱动程序)
QPCIBusPC
生成
使用
(驱动程序)
QSDHCI_MM
(接口)
QSDHCI
生成
使用
注册测试

QTest驱动框架

- 下面的路径可以表示一个类似“ / x86_64 / pc / i440FX-pcihost / pcibus-pci / sdhci-pci / sdhci / registers-test”的测试。
- 构建框架之后，通过遍历框架图来进行测试：要运行的每个测试都对应于从框架图的根到单元测试的路径。



QTest驱动框架

- 在运行时，框架中隐藏了很多步骤，这些步骤是通过遍历构建的路径来实现的：
- 所有驱动程序和测试都将自己注册到框架图中
- libqos运行QEMU以检测机器类型
- libqos采用它知道的机器类型，并将其添加到图中
- libqos遍历图形并为每个路径（从机器到测试）注册一个测试用例

QTest驱动框架

- 对于每个测试：
  - libqos遍历路径并构建QEMU命令行
  - libqos启动QEMU
  - libqos创建机器对象
  - 从机器对象开始，libqos遍历路径以获得测试功能所需的对象
  - libqos将接口传递给测试功能

QOS数据结构介绍

- **QOSGraphNode**
  - 包含驱动程序/机器/测试/接口的名称
- **QOSGraphEdge**
  - 以“生成”或“使用”关系连接两个QOSGraphNode
- **QOSGraphObject**
  - 在遍历图形路径期间创建的所有“实例”所共有的结构对象

QTest框架编写

- 如果我们要将自己的测试加入到qtest中，只需要使用libqos中的一些接口就可以轻松实现添加测试的功能

```
qos_add_test("register-test", "sdhci", sdhci_register_test_func);
                    |            |                 └──────────── Function invoked to the run test
                    |            └──────────────────────────────── Interface consumed by the test
                    └──────────────────────────────────────────────── Test name
```

```
qos_node_create_machine("arm/raspi4", qos_create_machine_arm_raspi4);
qos_node_contains("arm/raspi4", "i2c-bus");
qos_node_create_driver("i2c-bus", NULL); // created via "contains" only
qos_node_create_driver("i2c_timer", NULL);
qos_node_create_interface("timer_counter");
qos_node_produces("i2c-bus-timer", "i2c-bus");
qos_node_consumes("i2c-timer", "i2c-bus");   // "consumed by" edge from i2c-bus to timer-counter
```

QTest使用方法

- 下载qemu，并进行配置
  - ,$ git clone git@github.com:qemu/qemu.git
  - $ cd qemu
  - $ mkdir build
  - $ ../configure
- 在tests/qtest中编写QTest用例后进行测试(以riscv为例)
  - $ make check-qtest-riscv64 v=1

QTest使用方法

- 1.运行指定测试并查看结果
  - $ QTEST_QEMU_BINARY=riscv64-softmmu/qemu-system-riscv64 QTEST_QEMU_IMG=qemu-img MALLOC_PERTURB_=${MALLOC_PERTURB_:-$((RANDOM % 255 + 1))} gtester -k --verbose -m=quick tests tests/qom-test

QTest使用方法

- 2.查看LOG输出
  - $ QTEST_LOG=1

  QTEST_QEMU_BINARY=riscv64-softmmu/qemu-system-riscv64
  QTEST_QEMU_IMG=qemu-img

  MALLOC_PERTURB_=${MALLOC_PERTURB_:-$((RANDOM % 255 + 1))}

  gtester -k --verbose -m=quick tests/vhost-user-test -p /riscv64/qom-test

QTest使用方法

- 3. gdb调试模式运行
  - QTEST_QEMU_BINARY="xterm -e gdb --tty $(tty) --args riscv64-softmmu/qemu-system-riscv64" QTEST_QEMU_IMG=qemu-img gtester -k --verbose -m=quick tests/device-introspect-test
  - QTEST_QEMU_BINARY="valgrind --vgdb-error=1 --log-file=vg.log qemu-system-riscv64" QTEST_QEMU_IMG=qemu-img MALLOC_PERTURB_=${MALLOC_PERTURB_:-$((RANDOM % 255 + 1))} gtester -k --verbose -m=quick tests/device-introspect-test