SPICE源码分析

一、 spice server

qemu通过spice\_server\_new创建一个reds对象，将该reds传给spice\_server\_init进行初始化，具体的初始化过程在do\_spice\_init(core)中完成。core是qemu中的一个结构对象。

reds = spice\_new0(RedsState, 1);

spice\_server\_init(reds, core)

do\_spice\_init(core);

main\_dispatcher\_init(core); //初始化main\_dispatcher

reds\_init\_net()

main\_channel\_init();

inputs\_init();

main\_dispatcher\_init实现main\_dispatcher的初始化，将main\_dispatcher的recv\_fd添加到core的监听队列中，由qemu监听该文件描述符，并执行dispatcher\_handle\_read回调函数。最后注册3个消息处理函数。

main\_dispatcher\_init()

dispatcher\_init()

core->watch\_add()

dispatcher\_register\_handler()

网络secket的初始化，并把相应的socket添加到qemu的监听队列中。

reds\_init\_net()

reds\_init\_socket() //创建socket

core->watch\_add() //添加对socket的监听，回调函数reds\_accept()

---------------------------------------------------------------------------------------------------------------------------------

reds\_accept()

socket = accept()

spice\_server\_add\_client(reds, socket, 0) //添加一个client

添加新的client连接

spice\_server\_add\_client()

reds\_init\_client\_connection(socket)

reds\_handle\_new\_link(link);

---------------------------------------------------------------------------------------------------------------------------------

reds\_init\_client\_connection(socket)

//创建RedsStream \*stream; 并进行初始化

reds\_stream\_push\_channel\_event(stream, SPICE\_CHANNEL\_EVENT\_CONNECTED);

main\_dispatcher\_channel\_event(event, s->info);

dispatcher\_send\_message()

二、 spicec

[spice 鼠标移动代码分析](http://blog.csdn.net/zhuriyuxiao/article/details/8845122)

spice工作过程，相当繁琐。其中经过组件依次为  
1.客户端（例如spicec,spicy）  
2.spice-server  
3.qemu-kvm  
4.guest

下面分析spice鼠标移动事件工作过程，spice鼠标相对其他是比较简单的。

1.客户端spice mouse鼠标的移动操作  
<1>鼠标在Linux spice客户端进行移动时  
<2>X11引擎捕获该事件  
<3>发送鼠标移动信息给spice server.  
<4>客户端进行鼠标设置  
  
void RedWindow\_p::win\_proc(XEvent& event)  
{  
 switch (event.type) {  
    case MotionNotify: {  
        SpicePoint size = red\_window->get\_size();  
        if (event.xmotion.x >= 0 && event.xmotion.y >= 0 &&  
            event.xmotion.x < size.x && event.xmotion.y < size.y) {  
            SpicePoint origin = red\_window->get\_origin();  
            red\_window->get\_listener().on\_pointer\_motion(event.xmotion.x - origin.x,  
                                                         event.xmotion.y - origin.y,  
                                                         to\_red\_buttons\_state(event.xmotion.state));  
        }  
        break;  
    }  
  
}  
void RedScreen::on\_pointer\_motion(int x, int y, unsigned int buttons\_state)  
{                                                          
    if (\_mouse\_captured) {                                 
        on\_mouse\_motion(x, y, buttons\_state);  
        return;  
    }     
      
    \_pointer\_pos.x = x;  
    \_pointer\_pos.y = y;  
    \_mouse\_botton\_state = buttons\_state;  
                          
    if (update\_pointer\_layer() || !\_pointer\_layer) {  
        return;  
    }  
          
    \_pointer\_layer->on\_pointer\_motion(x, y, buttons\_state);  
}   
  
void RedScreen::on\_mouse\_motion(int x, int y, unsigned int buttons\_state)  
{     
    if (x != \_mouse\_anchor\_point.x || y != \_mouse\_anchor\_point.y) {  
        \_owner.on\_mouse\_motion(x - \_mouse\_anchor\_point.x,  
                               y - \_mouse\_anchor\_point.y,  
                               buttons\_state);  
        reset\_mouse\_pos();  
    }  
}  
  
void Application::on\_mouse\_motion(int dx, int dy, int buttons\_state)  
{  
    \_mouse\_handler->on\_mouse\_motion(dx, dy, buttons\_state);  
}  
  
void InputsChannel::on\_mouse\_motion(int dx, int dy, int buttons\_state)  
{  
    Lock lock(\_motion\_lock);  
    \_mouse\_buttons\_state = buttons\_state;  
    \_mouse\_dx += dx;  
    \_mouse\_dy += dy;  
    if (!\_active\_motion && \_motion\_count < SPICE\_INPUT\_MOTION\_ACK\_BUNCH \* 2) {  
        \_active\_motion = true;  
        \_motion\_count++;  
        post\_message(new MotionMessage(\*this));  
    }     
}  
  
void RedScreen::reset\_mouse\_pos()  
{  
    \_window.set\_mouse\_position(\_mouse\_anchor\_point.x, \_mouse\_anchor\_point.y);  
}  
  
void RedWindow::set\_mouse\_position(int x, int y)  
{  
    XWarpPointer(x\_display, None, \_win, 0, 0, 0, 0, x + get\_origin().x, y + get\_origin().y);  
}  
  
2.服务器端 spice mouse鼠标的移动信息处理  
<1> 按照spice protocal协议解析客户端放送消息  
<2>通过VDI接口调用，进入qemu-kvm  
static int inputs\_channel\_handle\_parsed(RedChannelClient \*rcc, uint32\_t size, uint16\_t type, void \*message)  
{  
    case SPICE\_MSGC\_INPUTS\_MOUSE\_MOTION: {  
        SpiceMsgcMouseMotion \*mouse\_motion = (SpiceMsgcMouseMotion \*)buf;  
  
        if (++icc->motion\_count % SPICE\_INPUT\_MOTION\_ACK\_BUNCH == 0 &&  
            !g\_inputs\_channel->src\_during\_migrate) {  
            red\_channel\_client\_pipe\_add\_type(rcc, PIPE\_ITEM\_MOUSE\_MOTION\_ACK);  
            icc->motion\_count = 0;  
        }  
        if (mouse && reds\_get\_mouse\_mode() == SPICE\_MOUSE\_MODE\_SERVER) {  
            SpiceMouseInterface \*sif;  
            sif = SPICE\_CONTAINEROF(mouse->base.sif, SpiceMouseInterface, base);  
            sif->motion(mouse,  
                        mouse\_motion->dx, mouse\_motion->dy, 0,  
                        RED\_MOUSE\_STATE\_TO\_LOCAL(mouse\_motion->buttons\_state));  
        }  
        break;  
    }  
  
qemu-kvm 提供VDI接口  
static const SpiceMouseInterface mouse\_interface = {  
    .base.type          = SPICE\_INTERFACE\_MOUSE,  
    .base.description   = "mouse",  
    .base.major\_version = SPICE\_INTERFACE\_MOUSE\_MAJOR,  
    .base.minor\_version = SPICE\_INTERFACE\_MOUSE\_MINOR,  
    .motion             = mouse\_motion,  
    .buttons            = mouse\_buttons,  
};  
  
static void mouse\_motion(SpiceMouseInstance \*sin, int dx, int dy, int dz,  
                         uint32\_t buttons\_state)  
{  
    kbd\_mouse\_event(dx, dy, dz, map\_buttons(buttons\_state));  
}  
  
void kbd\_mouse\_event(int dx, int dy, int dz, int buttons\_state)  
{                          
    QEMUPutMouseEntry \*entry;  
    QEMUPutMouseEvent \*mouse\_event;  
    void \*mouse\_event\_opaque;  
    int width;  
  
    if (QTAILQ\_EMPTY(&mouse\_handlers)) {  
        return;  
    }  
  
    entry = QTAILQ\_FIRST(&mouse\_handlers);  
      
    mouse\_event = entry->qemu\_put\_mouse\_event;  
    mouse\_event\_opaque = entry->qemu\_put\_mouse\_event\_opaque;  
      
    if (mouse\_event) {    
        if (graphic\_rotate) {  
            if (entry->qemu\_put\_mouse\_event\_absolute)  
                width = 0x7fff;  
            else  
                width = graphic\_width - 1;  
            mouse\_event(mouse\_event\_opaque,  
                        width - dy, dx, dz, buttons\_state);  
        } else  
            mouse\_event(mouse\_event\_opaque,  
                        dx, dy, dz, buttons\_state);  
    }  
}  
  
3.qemu-kvm 提供ps2 mouse鼠标硬件的模拟  
<1>VDI接口是虚拟设备的接口  
<2>该接口操从虚拟设备，从而影响客户机操作系统鼠标设备驱动。  
qemu\_add\_mouse\_event\_handler(ps2\_mouse\_event, s, 0, "QEMU PS/2 Mouse");  
qemu\_add\_mouse\_event\_handler(usb\_pointer\_event, s,  
static void ps2\_mouse\_event(void \*opaque,  
                            int dx, int dy, int dz, int buttons\_state)  
{  
    PS2MouseState \*s = opaque;  
  
    /\* check if deltas are recorded when disabled \*/  
    if (!(s->mouse\_status & MOUSE\_STATUS\_ENABLED))  
        return;  
  
    s->mouse\_dx += dx;  
    s->mouse\_dy -= dy;  
    s->mouse\_dz += dz;  
    /\* XXX: SDL sometimes generates nul events: we delete them \*/  
    if (s->mouse\_dx == 0 && s->mouse\_dy == 0 && s->mouse\_dz == 0 &&  
        s->mouse\_buttons == buttons\_state)  
        return;  
    s->mouse\_buttons = buttons\_state;  
  
    if (!(s->mouse\_status & MOUSE\_STATUS\_REMOTE) &&  
        (s->common.queue.count < (PS2\_QUEUE\_SIZE - 16))) {  
        for(;;) {  
            /\* if not remote, send event. Multiple events are sent if  
               too big deltas \*/  
            ps2\_mouse\_send\_packet(s);  
            if (s->mouse\_dx == 0 && s->mouse\_dy == 0 && s->mouse\_dz == 0)  
                break;  
        }  
    }  
}