一、事件处理

emitter实现了通用的事件注册、注销和分发功能，widget对此做了进一步包装，使用起来非常方便。

1.1 注册控件事件的处理函数

使用widget_on来注册事件处理函数：

```
/**
 * @method widget_on
 * 注册指定事件的处理函数。
 * @scriptable custom
 * @param {widget_t*} widget 控件对象。
 * @param {event_type_t} type 事件类型。
 * @param {event_func_t} on_event 事件处理函数。
 * @param {void*} ctx 事件处理函数上下文。
 *
 * @return {uint32_t} 返回id，用于widget_off。
 */
uint32_t widget_on(widget_t* widget, event_type_t type, event_func_t on_event, void*
ctx);
```

示例：

```
static ret_t on_inc(void* ctx, event_t* e) {
  widget_t* progress_bar = (widget_t*)ctx;
  uint8_t value = (PROGRESS_BAR(progress_bar)->value + 10) % 100;
  progress_bar_set_value(progress_bar, value);
  (void)e;
  return RET_OK;
}

...

progress_bar = progress_bar_create(win, 10, 80, 168, 30);
widget_set_value(progress_bar, 40);

ok = button_create(win, 10, 5, 80, 30);
widget_set_text(ok, L"Inc");
widget_on(ok, EVT_CLICK, on_inc, progress_bar);
...
```

在上面这个例子中，我们创建了一个进度条和一个按钮，并为按钮的『点击』事件注册一个处理函数，当按钮被点击时，增加进度条的值。

1.2 注销控件事件的处理函数

由于窗口关闭时会销毁其中所有控件，所以一般不需要手工去注销。如果确实需要，可以使用widget_off:

```
/**
 * @method widget_off
 * 注销指定事件的处理函数。
 * @scriptable custom
 * @param {widget_t*} widget 控件对象。
 * @param {uint32_t} id widget_on返回的ID。
 *
 * @return {ret_t} 返回RET_OK表示成功，否则表示失败。
 */
ret_t widget_off(widget_t* widget, uint32_t id);
```

1.3 控件分发事件

一般只有在控件内部实现中，或者需要向控件注入事件时，才需要调用事件分发函数。如果需要分发事件，可以调用widget_dispatch：

```
/**
 * @method widget_dispatch
 * 分发一个事件。
 * @param {widget_t*} widget 控件对象。
 * @param {event_t*} e 事件。
 *
 * @return {ret_t} 返回RET_OK表示成功，否则表示失败。
 */
ret_t widget_dispatch(widget_t* widget, event_t* e);
```