图层管理

图层管理模块是用来管理不同图层间的显示问题。

**哪些情况需要用到图层管理？**

比如你想轻松地把GUI内容从屏幕1切换到屏幕2上

或者你想显示某块窗口，或者把窗口显示在屏幕的任意区域等等。

**图层管理主要分为**：Display、Layer、Surface。

Display即屏幕。

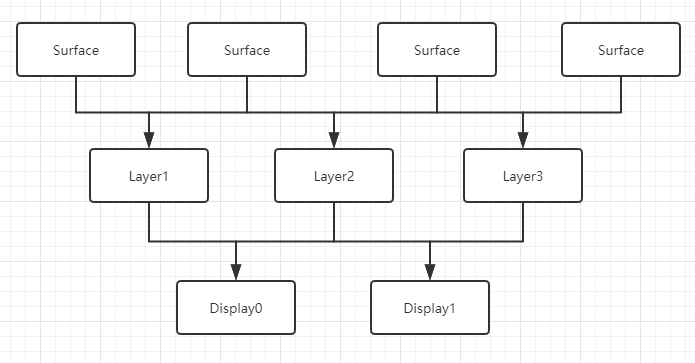
Layer是在屏幕之上虚拟出来的概念，如同GUI在屏幕上的显示有上下层级之分，即叠加的概念。

Surface是具体在Layer上渲染图像的一个面，理解为一张画布。

**它们之间的关系为**:

1. 一块Display上可以有多个Layer，Layer存在上下关系。
2. Layer表示图层位置关系，但如果需要在某个Layer显示图像，则需要在Layer上有Surface，用来供GUI进行渲染画图。
3. 一个Layer上可以有多个Surface（也可以理解为窗口，即window），但是Surface之间是互斥的，也就是说在一个Layer上只能同时显示一个Surface。(win32下除外)

可以用一张图来简单概括它们的关系：



**如何使用？**

**首先在kanzi显示进程端：**

以cluster举例：

1. 在cmakeList里增加对libuit\_screenfw.so的链接
2. 在main.cpp里面增加如下红色部分代码：

#define USE\_OZONE

#include "uit/screenfw/ScreenListener.h"

#include <kanzi/core.ui/platform/windowing/common/kzs\_desktop.h>

#include <kanzi/core.ui/platform/windowing/common/kzs\_window.h>

#include <kanzi/core.ui/platform/windowing/kzs\_window\_native.hpp>

#include <kanzi/core.ui/platform/windowing/kzs\_display\_native.hpp>

#include <kanzi/core.ui/platform/windowing/kzs\_desktop\_native.hpp>

#include <kanzi/core.ui/platform/graphics\_backend/gl/gl\_graphics\_output.hpp>

virtual void onProjectLoaded() KZ\_OVERRIDE

{

#ifdef WIN32

HWND handle=kzsWindowNativeGetHandle(kzsWindowGetNative(dynamic\_pointer\_cast<

GlGraphicsOutput>(getGraphicsOutput())->getWindow()));

bool ret = m\_listerner.connect((int)handle);

uit::Log::info(DISPLAY\_NAME, "ScreenListener connect{%d}", ret);

#else //QNX

screen\_window\_t\* screenWin = kzsWindowNativeGetScreenWindow(kzsWindowGetNative(

dynamic\_pointer\_cast<GlGraphicsOutput>(getGraphicsOutput())->getWindow()));

screen\_context\_t\* screen\_ctx = kzsDesktopNativeGetScreenContext(

kzsDesktopGetNative(dynamic\_pointer\_cast<GlGraphicsOutput>(getGraphicsOutput())->getDesktop()));

//下面connect()函数的第一个参数，是对应显示进程的windowID.参考配置文件说明：

bool ret = m\_listerner.connect(1001, \*screen\_ctx, \*screenWin);

uit::Log::info(DISPLAY\_NAME, "ScreenListener connect{%d}", ret);

#endif

//如果显示进程端希望窗口出现变化后能收到通知(比如显示隐藏，改变位置和大小等)，可以添加以下事件。当外部有控制到窗口时，显示进程端都会收到通知。如果不需要关心相关通知则不用添加如下代码：

void OnVisibleChanged(const uit::screenfw::ScreenListener::

VisibleChangedArgs &param) {}

void OnPositionChanged(const uit::screenfw::ScreenListener::

PositionChangedArgs &param) {}

void OnClipPositionChanged(const uit::screenfw::ScreenListener::

ClipPositionChangedArgs &param) {}

void OnSizeChanged(const uit::screenfw::ScreenListener::

SizeChangedArgs &param) {}

void OnClipSizeChanged(const uit::screenfw::ScreenListener::

ClipSizeChangedArgs &param) {}

void OnRectChanged(const uit::screenfw::ScreenListener::

RectChangedArgs &param) {}

void OnClipRectChanged(const uit::screenfw::ScreenListener::

ClipRectChangedArgs &param) {}

void OnFullScreenChanged(const uit::screenfw::ScreenListener::

FullScreenChangedArgs &param) {}

m\_listerner.VisibleChangedEvent +=

std::bind(&cluster::OnVisibleChanged, this,std::placeholders::\_1);

m\_listerner.PositionChangedEvent +=

std::bind(&cluster::OnPositionChanged, this, std::placeholders::\_1);

m\_listerner.ClipPositionChangedEvent +=

std::bind(&cluster::OnClipPositionChanged, this, std::placeholders::\_1);

m\_listerner.SizeChangedEvent +=

std::bind(&cluster::OnSizeChanged, this, std::placeholders::\_1);

m\_listerner.ClipSizeChangedEvent +=

std::bind(&cluster::OnClipSizeChanged, this, std::placeholders::\_1);

m\_listerner.RectChangedEvent +=

std::bind(&cluster::OnRectChanged, this, std::placeholders::\_1);

m\_listerner.ClipRectChangedEvent +=

std::bind(&cluster::OnClipRectChanged, this, std::placeholders::\_1);

m\_listerner.FullScreenChangedEvent +=

std::bind(&cluster::OnFullScreenChanged, this, std::placeholders::\_1);

}

**然后在控制端：**

可以理解为app部分，参考如下代码：

在windows端：

ScreenController::connect();

boo vis = true;

HWND handler = FindWindowA(nullptr, "Kanzi");//"kanzi"指的是指要控制的窗口的名称。

ScreenController::setVisibility((int)handler, vis);

在QNX端：

ScreenController::connect();

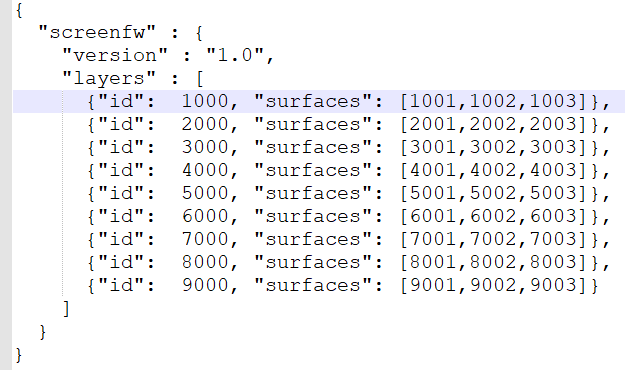
bool vis = true;

ScreenController::setVisibility(1001, vis);

//即可控制windowid=1001所对应的显示进程的显示了。

**配置文件**

这里会提供一个统一的配置文件uit.screen.conf(win32环境下开发可以忽略)，一般会发布到uit的dist目录，内容如下图：



配置文件是由显示进程（比如kanzi）和app开发人员之间来约定的。既可以进行修改。

比如仪表的kanzi显示进程如果选择surfaceid=1001，那么app端如果需要控制仪表的图层时，则只要在调用相关函数时，第一个参数选择surfaceid =1001即可.