

title: Alios pagemanagerd与seed通信 date: 2018-06-4 categories: - Alios 源码分析 tags: - Alios源码分析 - pagemanagerd - seed

## pagemanagerd和seed简介

Page是Alios的最核心元素，是Alios界面元素的管理单元，而与之对应的有两个非常重要的native进程pagemanagerd和seed

- pagemanagerd管理系统的domain和page，这其中包含大家熟知的DPMS和SPMS(类似于Android的AMS和PMS)
- seed则为所有page的孵化进程  
这两个进程一个用于**管理Page**，一个用于**创建销毁Page**，两者必然需要频繁通信，因此了解二者的通信实现方可知道Page的创建流程。
- 二者皆在开机时启动，开机流程：BootLoader-->kernel-->systemd-->native services-->runtime-->Key apps  
pagemanagerd和seed为native services，二者相互联系且依赖，下面我们来看看二者如何通信的。

## pagemanagerd和seed通过socket通信

首先要明确，pagemanagerd和seed IPC通信是直接通过socket实现的，要知道alios进程通信主要采用的是ubus，即封装好的socket通信。至于为什么直接用socket，原因必然是socket更合适，更高效。ubus kernel层调用的是kdbus，kdbus是可以传输大块数据的，中间应该用了共享缓存机制，pagemanagerd和seed的通信仅为少量消息的传递，且需要相对较高的实时性，所以直接采取socket方式。

需要说明的是这里的socket通信不同于TCP流式的网络socket，本地socket实际上是client端读写设备文件，services端监听设备文件从而达到消息传递。具体实现关键库为**SockClient**，**SockLisener**和**SockServer**，使用时需要引入socket/Socket.h。

pagemanagerd和seed对应的设备文件如下

```
static const String kSeedSocketName("/dev/socket/seed");//实际的设备文件名需要根据systemd去确定
static const String kDPMSocketName("/dev/socket/dpms");
```

pagemanagerd和seed关键代码：

framework\nativeservice\seed\src\Seed.cpp

framework\nativeservice\pagemgrd\dpms\src\ipc\SocketMessageCenter.cpp

framework\nativeservice\pagemgrd\dpms\src\DynamicPageManagerService.cpp

## seed发送消息给pagemanagerd

framework\nativeservice\seed\src\Seed.cpp

```
bool Seed::sendMessageToDPMS(const String& msg) {
    LOG_D("sendMessageToDPMS: %s", msg.c_str());
    if (!mDPMSSockClient) {
        mDPMSSockClient = YUNOS_NEW(SockClient, kDPMSSocket); //kDPMSSocket == "/dev/socket/dpms"
        if (!mDPMSSockClient) {
            YUNOS_HANDLE_OOM();
            return false;
        }
    }
    return mDPMSSockClient-->sendMessage(msg);
}
```

Seed中用SockClient发消息给pagemanagerd的设备节点 `/dev/socket/dpms`

## seed监听到至pagemanagerd的消息

实现socket监听要实现Socklisener，然后用SockServer绑定对应设备文件，并监听该Socklisener，当设备文件有新写入时socket机制会回调Socklisener的onReceived函数去做具体的消息处理。Socklisener实现如下

```
class Listener : public SockListener {
public:
    explicit Listener(Seed* seed) : mSeed(seed) {
    }
    virtual ~Listener() {
    }

    virtual void onReceived(const String& inMessage, String& outMessage) {
        if (mSeed) {
            mSeed-->handleMessage(inMessage);
        }
    }

private:
    Seed* mSeed;
};
```

SockServer的实例化和绑定在Seed.cpp的run函数中，如下是代码片段

```

int socketFd = ProcessUtil::retrieveSeedSocketFd();//获取"/dev/socket/seed_xxx"设备节点
.....
mSeedSockServer = YUNOS_NEW(SockServer, socketFd);//初始化SockServer并绑定设备节点
mListener = YUNOS_NEW(Listener, this);//初始化Listener并传递Seed本身作为回调句柄
.....
mSeedSockServer-->setMainLoop(env-->event_loop());
ret = mSeedSockServer-->listen(mListener);//开始监听

```

当pagemanagerd向seed设备节点发消息后，Listener的onReceived会被调用，然后回调Seed.cpp的handleMessage函数，handleMessage中有3个消息，"pagemanagerd start"，"homeshell ready"，page的创建销毁，详细如下代码片段

```

void Seed::handleMessage(const String& msg) {
    .....
    if (msg.find("pagemanagerd start") == 0) { //pagemanagerd启动后通知seed
        .....
        sendMessageToDPMS(String("seed ready"));//发消息告诉pagemanagerd seed已经准备好了
        .....
    } else if (msg.find("homeshell ready") == 0) {
        invokeHomeshellReadyCB();
    } else { //其他消息则为进程的创建和销毁消息
        SharedPtr<ProcessInfo> info = YUNOS_NEW(ProcessInfo);
        if (!info.pointer()) {
            YUNOS_ABORT_OOM();
            return;
        }
        if (!info-->parseJson(msg)) { //解析msg中的信息到ProcessInfo
            LOG_E("Seed Socket Failed to parse domain infomation from json string");
        };
        return;
    }
    if (info-->action() == "fork") {
        forkDomainProcess(info);//创建进程
    } else if (info-->action() == "kill") {
        killDomainProcess(info);//销毁进程
    }
}
}
}

```

## pagemanagerd监听seed的消息

来着seed的第一个消息为seed初始化的时候，在Seed.cpp的run函数中

```

bool ret = sendMessageToDPMS(String("seed start"));

```

pagemanagerd中socket监听同样也是实现SockListener，用SockServer绑定设备节点并监听，具体实现代码 framework\nativeservice\pagemgrd\dpms\src\ipc\SocketMessageCenter.cpp

```
class Listener : public SockListener {
public:
    explicit Listener(SocketMessageCenter* center) : mCenter(center) {
        assert(mCenter);
    }

    virtual ~Listener() {}

    virtual void onReceived(const String& inMessage, String& outMessage) {
        if (mCenter) {
            mCenter-->handleMessage(inMessage);
        }
    }

private:
    SocketMessageCenter* mCenter;
};
```

SocketMessageCenter中 SockServer绑定和监听放在线程ServerThread中，直接看其run()函数

```
virtual int run() {
    .....
    mServer = YUNOS_NEW(SockServer, kDPMSocketName); //kDPMSocketName == "/dev/socket/dpms"
    if (!mServer) {
        YUNOS_ABORT_OOM();
    }
    mServer-->listen(mListener); //mListener是Listener实例
}
```

当seed发送的seed start消息到来时，首先调用Listener的onReceived,然后回调给SocketMessageCenter的handleMessage函数：`mCenter-->handleMessage(inMessage)`，handleMessage通过loop发消息回调，最后调用handleMessageCB，而handleMessageCB又调用handleMessageInner

```

virtual void onReceived(const String& inMessage, String& outMessage) {
    if (mCenter) {
        mCenter-->handleMessage(inMessage);
    }
}
.....
void SocketMessageCenter::handleMessage(const String& inMessage) {
    if (mMainLooper) {
        mMainLooper-->sendTask(Task(handleMessageCB, this, inMessage));
    }

    void SocketMessageCenter::handleMessageCB(SocketMessageCenter* self, String in
Message) {
        assert(self);
        self-->handleMessageInner(inMessage);
    }
}

```

故最终我们只需要关注SocketMessageCenter的handleMessageInner函数即可

```

void SocketMessageCenter::handleMessageInner(const String& inMessage) {
    if (inMessage.endsWith("seed start")) {
        .....
        notifySeedOnStart(client); //发送pagemanagerd start消息给seed
        .....
    }
    if (inMessage.endsWith("seed ready")) {

    }

    .....
}

```

## pagemanagerd向seed发送消息

发送消息的调用实例如下

```

void DynamicPageManagerService::updatePageStatus(const String& pageId,
    const String& status, int32_t uid) {
    .....
    sendRawMessageToSeed(String("homeshell ready"), seeds[i]);
    .....
}

```

sendRawMessageToSeed的代码:

```
bool DynamicPageManagerService::sendRawMessageToSeed(const String& message, const
String& seed) {
    if (!mSocketMessageCenter) {
        LOG_E("%s: mSocketMessageCenter is nullptr", __FUNCTION__);
        return false;
    }
    return mSocketMessageCenter-->sendRawMessageToSeed(message, seed);
}
```

DynamicPageManagerService发消息给seed还是调到SocketMessageCenter的sendRawMessageToSeed, 具体调用栈如下

```

bool SocketMessageCenter::sendRawMessageToSeed(const String& message, const String
& seed) {
    SeedSocketClient* client = getSocketClient(seed); //获取对应的SocketClient
    if (!client-->mIsSeedReady) {
        client-->mPendingMessageList.push_back(message);
        return true;
    }

    return sendRawMessageToSeedInner(message, client);
}
.....

bool SocketMessageCenter::sendRawMessageToSeedInner(const String& message,
    SeedSocketClient* client) {
    LOG_D("Sending message to seed: %s", message.c_str());
    return client-->mSeedClient.sendMessage(message);
}
.....
SocketMessageCenter::SeedSocketClient* SocketMessageCenter::getSocketClient(
    const String& seed) {
    SeedSocketClient* client = mSeedClients[seed];
    if (client == NULL) {
        String socketName = kSeedSocketName; //"/dev/socket/seed"
        if (!seed.isNull()) {
            socketName.append("_");
            socketName.append(seed);
        }
        client = YUNOS_NEW(SeedSocketClient, socketName);
        if (!client) {
            YUNOS_HANDLE_OOM();
        }
        mSeedClients[seed] = client;
    }

    return client;
}

```

通过上面代码分析，seed和pagemanagerd通信基本打通，二者互为socket client和socket service。二者启动后会建立socket连接，从而相互通信。



## page的创建

一个page的创建通常是通过page.sendLink开始,API调用流程一笔带过

framework\npm\caf2\src\page\Page.js : sendLink

framework\npm\pageapi\PageImpl.js : sendLink

framework\npm\pageapi\PageInstance.js : sendLink

framework\npm\pageapi\internal\PageIPC.js : DPMSProxy : sendLink

yidl

framework\libs\page\manager\src\ipc\DPMS.cpp : sendLink

IPC(UBus) -->

/framework\libs\page\manager\src\ipc\DPMSServer.cpp : onMessage

/framework\nativeservice\pagemgrd\dpms\src\ipc\MessageAdaptor.cpp : onRequestPageLink

framework\nativeservice\pagemgrd\dpms\src\DynamicPageManagerService.cpp : requestPageLink

(以上api调用和IPC调用留待下一篇讲解)

下面我们来看本地服务中DynamicPageManagerService : requestPageLink如何创建Page的



```
void DynamicPageManagerService::requestPageLink(
    SharedPtr<PageLink>& link, const SharedPtr<DMessage>& msg, int userId) {
    rememberAccessHistory(userId, link);
    int ret = requestPageLinkInner(link, msg, userId);
    if (ret == SUCCESS) {
        LOG_I("requestPageLink success, send reply");
        sendReply(msg);
    } else if (RETURN_FAIL(ret)) {
        LOG_E("requestPageLink fail(error = %d), send error reply", ret);
        sendErrorReply(msg, ret);
    } else if (ret == SUCCESS_BUT_UNFINISHED) {
        LOG_D("requestPageLink SUCCESS_BUT_UNFINISHED");
    }
}
```

requestPageLinkInner中继续处理

```

int DynamicPageManagerService::requestPageLinkInner(
    SharedPtr<PageLink>& link, const SharedPtr<DMessage>& msg, int userId) {

    .....
    // if load System bootup
    .....
    // if load homeshell
    .....
    // this is a broadcast
    if (uri.domain() == reserved_domain::kBroadcast) {
        ret = broadcastPageLink(link, msg);
    } else if (uri.domain() == reserved_domain::kPluginCompatible) { //这是一个插件
        LOG_D("PageLink is a intent, data: %s", link-->getData().c_str());
        GET_PLUGIN_CLIENT(pluginClient);
        if (pluginClient) {
            ret = pluginClient-->startPluginTask(link, msg);
        } else {
            return ERR_START_ACTIIVTY_FAILED;
        }
    } else { // 这是一个Page
        const String targetId = link-->getPageId();
        if (targetId.isNull()) { //如果不存在则创建
            ret = loadPage(uri.toString(), link, msg, userId);
        } else { //如果存在则启动对应page
            LOG_D("sendLink to specific target: %s", targetId.c_str());
            ret = sendLinkToSpecificTarget(link, msg);
        }
    }

    return ret;
}

```

requestPageLinkInner中有较多逻辑处理，为各种page初始化link参数然后loadPage

```

int DynamicPageManagerService::loadPage(
    const String& uri, const SharedPtr<PageLink>& link,
    const SharedPtr<DMessage>& msg, int userId) {
    if (!link) {
        LOG_E("%s error: link is NULL", __FUNCTION__);
        return ERR_INVALID_LINK;
    }
    Uri uriType(uri);
    String pureUri = uriType.getLeftPart();
    PageRequest* data = YUNOS_NEW(PageRequest, link, msg);
    if (!data) {
        YUNOS_ABORT_OOM();
    }
    TaskCallback callback = makeCallback(&DynamicPageManagerService::onPageInfoGot
,
        dpmsMAIN, static_cast<void*>(data));
    mStaticPageManager-->getPageInfo(link, callback, userId);
    return SUCCESS_BUT_UNFINISHED;
}

```

framework/nativeservice/pagemgrd/dpms/src/DynamicPageManagerService.cpp

--> onPageInfoGot --> handlePageInfoReceived --> loadPageInner

loadPageInner 中 `getOrCreateDomainRecord` 会先创建DomainRecord

-->

framework/nativeservice/pagemgrd/dpms/src/manager/StackManager.cpp : loadPage

-->

framework/nativeservice/pagemgrd/dpms/src/model/PageStack.cpp : loadPage

-->

framework/nativeservice/pagemgrd/dpms/src/model/TaskRecord.cpp : loadPage

```
PageRecord* pageRecord = dpmsFTY-->createPageRecord(pageInfo, link, msg, this);
```

--> framework/nativeservice/pagemgrd/dpms/src/DPMSFactoryImpl.cpp

```

PageRecord* DPMSFactoryImpl::createPageRecord(
    const SharedPtr<PageInfo>& pageInfo, const SharedPtr<PageLink>& link,
    const SharedPtr<DMessage>& msg, TaskRecord* task) {
    PageRecord* page = YUNOS_NEW(PageRecord, pageInfo, link, msg, task); //new 一个PageRecord
    ABORT_IF_NULL(page)
    return page;
}

```

--> framework/nativeservice/pagemgrd/dpms/src/model/PageRecord.cpp

```

PageRecord::PageRecord(const SharedPtr<PageInfo>& pageInfo, const SharedPtr<PageLink>& link,
    const SharedPtr<DMessage>& msg, TaskRecord* task) :Identity(PAGE_TYPE),
    mPageInfo(pageInfo), mOriginalPageLink(link),
    mTask(task), mEnvFilter(kDefaultEnvFilter), mUserId(task-->getUserId()) {
    init(link, msg, pageInfo); //初始化
    LOG_D("%s CREATED", __FUNCTION__);
    DPMSUtil::logTraceEvent(getpid(), getPageUri(), getUniqueId(),
        String("PAGE"), String("CREATE"), 0);
}

void PageRecord::init(const SharedPtr<PageLink>& link, const SharedPtr<DMessage>& msg,
    const SharedPtr<PageInfo>& pageInfo) {
    .....
    mProcess = dpmsFTY-->getProcessManager()-->allocateProcess(
        this, link-->getOption().isInGroup());
    .....
}

```

--> framework/nativeservice/pagemgrd/dpms/src/manager/ProcessManager.cpp

```

Process* ProcessManager::allocateProcess(PageRecord* page, bool inGroup) {
    if (!page) {
        LOG_E("%s: failed, page is null", __FUNCTION__);
        return nullptr;
    }
    SharedPtr<PageInfo> pageInfo = page-->getPageInfo();
    String title = getProcessTitle(pageInfo);
    int uid = pageInfo-->getAppPermissions().mUid;
    int32_t taskId = inGroup ? page-->getTask()-->getTaskId() : -1;
    LOG_D("%s: title=%s, inGroupTaskId=%d, uid=%d", __FUNCTION__, title.c_str(), taskId, uid);
    Process* process = tryToAllocateRunningProcess(title, taskId, uid);
    if (!process) { //创建Process
        process = createProcess(page-->getPageUri(), title, pageInfo-->getDomain(),
            taskId,
            pageInfo-->getSeed());
    }
    process-->attachPageRecord(page); //Process绑定Page
    return process;
}

```

framework/nativeservice/pagemgrd/dpms/src/model/Process.cpp

```

void Process::attachPageRecord(PageRecord* page) {
    if (!page) {
        LOG_E("%s: failed, page is null", __FUNCTION__);
        return;
    }
    LOG_D("%s: %s(%s)", __FUNCTION__, page-->getPageUri().c_str(), page-->getUnique
eId().c_str());
    mPageMap[page-->getUniqueId()] = page;
    if (mPageMap.size() == 1) {
        // Fork a new process when first page attached.
        startProcessViaSeed(page); //如果是第一个page, 则创建进程
    } else {
        // createPage in the same process
        realLoadPage(page);
    }
}

.....

void Process::startProcessViaSeed(PageRecord* page) {
    .....
    String msg = String::format(kSeedForkMessageTemplate, pageInfo-->getDomain().c
_str(),
        mTitle.c_str(), getUniqueId().c_str(), 0, appPermission.mUid + delta,
        appPermission.mGid, appPermission.mGids.c_str(), appPermission.mCap,
        appPermission.mMount, debugStr.c_str(), extraStr.c_str());
    if (!dpmsMAIN-->sendRawMessageToSeed(msg, mSeed)) { //msg.action = ACTION_FORK
}
}

```

这里的 `dpmsMAIN-->sendRawMessageToSeed` 即为前面讲的DynamicPageManagerService的 `sendRawMessageToSeed`，消息会发送给Seed并创建进程。

## 总结

这里总结下Page的创建过程涉及到几个概念Process, Domain, Stack及与之对应的管理者DomainManager, ProcessManager, StackManager, 与之对应的数据model有(PageRecord, DomainRecord), Process, PageStack, TaskStack。

一个TaskStack包含多个PageStack, sendLink一个PageInfo会首先创建一个PageStack放在对应的TaskStack, 然后创建PageRecord, PageRecord中包含Process, Process负责发消息给seed创建实际的Page。如果PageRecord是其对应的Domain中第一个Page, 则先创建一个DomainRecord。

详细page管理后续DPMS专题继续分析。