流媒体技术笔记（**DarwinStreamingServer** 相关）

简介

Darwin Streaming Server 简称 DSS。DSS 是 Apple 公司提供的开源实时流媒体播放服务器程序。整个程序使用 C++编写，在设计上遵循高性能，简单，模块化等程序设计原则，务求做到程序高效，可扩充性好。并且 DSS 是一个开放源代码的，基于标准的流媒体服务器，可以运行在 Windows NT 和 Windows 2000，以及几个 UNIX 实现上，包括 Mac OS X，Linux，FreeBSD，和 Solaris 操作系统上的。
网址：http://dss.macosforge.org/
源码：http://dss.macosforge.org/downloads/DarwinStreamingSrvr6.0.3-Source.tar

特性

支持 MP4、3GPP 等文件格式；
支持 MPEG-4、H.264 等视频编解码格式；
支持 RTSP 流控协议，支持 HTTP 协议，支持 RTP 流媒体传输协议；
支持基于 Web 的管理；
支持单播和组播；
具有完备的日志功能。
此外，该服务器版本提供了一个基于模块的扩展方法。利用 DSS 提供的 API 就可以很方便地编写静态或动态的模块，对 DSS 进行扩展，使其支持其它文件格式、协议或者功能。

DSS 服务器架构

设计模式相关

1、并发设计模式

阻塞的 IO 方式效率极低，这里不予讨论。非阻塞的 IO 分成两种，分别是非阻塞同步 IO 和非阻塞异步 IO，对应的，Reactor 和 Proactor 是高性能并发服务器设计中常见的两种模式，Reactor 用于同步 IO，Proactor 用于异步 IO。无论是 Reator 还是 Proactor，都包含了时间分离器和时间处理器。

2、Darwin 使用的设计模式

Darwin Streaming Server 从设计模式上看，采用了 Reactor 的并发服务器设计模式。Reactor 模式是典型的事件触发模式，当有事件发生时则完成相应的 Task，Task 的完成是通过调用相应的 handle 来实现的，对于 handle 的调用是由有限个数的 Thread 来完成的。

在 DSS 的各类线程中，事件线程充当 Reactor 模式中的事件分离器，任务线程充当 Reactor 模式中的事件处理器。

主框架

DSS 的核心服务器部分是由一个父进程所 fork 出的一个子进程构成，该父进程就构成了整合流媒体服务器。父进程会等待子进程的退出，如果在运行的时候子进程产生了错误从而退出，那么父进程就会 fork 出一个新的子进程。

服务器的作用是充当网络客户和服务器模块的接口，其中网络客户使用 RTP 和 RTSP 协议来发送请求和接收响应，而服务器模块则负责处理请求和向客户端发送数据包。核心流媒体服务通过创建四种类型的线程来完成自己的工作，具体如下：

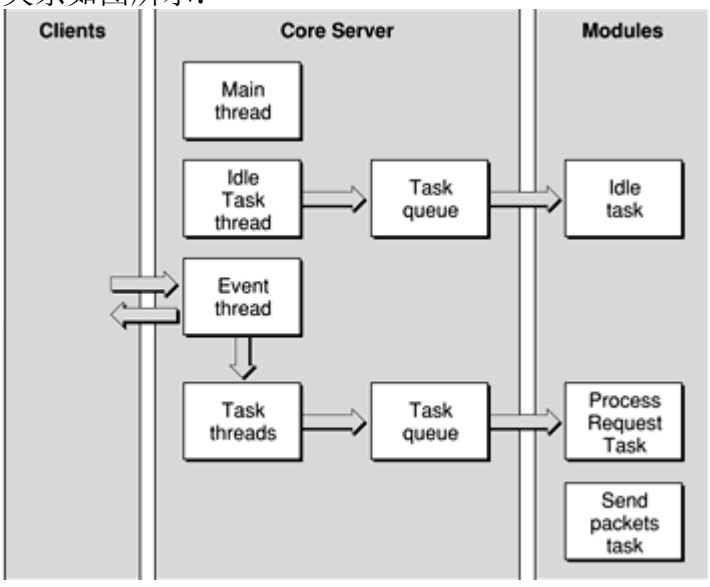
服务器主线程（**Main Thread**）：这个线程负责检查服务器是否需要关闭，记录状态信息，或者打印统计信息。

空闲任务线程（**Idle Task Thread**）：空闲任务线程管理一个周期性的任务队列。该任务队列有两种类型：超时任务和 Socket 任务。

事件线程（**Event Thread**）：负责监听套接口相关事件，当有 RTSP 请求或者收到 RTP 数据包时，事件线程就会把这些实践交给任务线程来处理。

任务线程（**Task Thread**）： 任务线程会把事件从事件线程中取出，并把处理请求传递到对应的服务器模块进行处理，比如把数据包发送给客户端的模块，在默认情况下，核心服务器会为每个处理器核创建一个任务线程。

关系如图所示：



DSS 参数说明

-v　　：　　显示使用方法，即参数列表

-d : 在前台运行程序
-D : 在前台运行程序，并显示性能数据（服务器统计信息）
-Z xxx : 设定 debug 级别
-p xxx : 指定默认的 RTSP 服务器监听端口
-S n : 每隔 n 秒在控制台中显示一次服务器统计
-c myconfigpath.xml : 指定一个配置文件
-o myconfigpath.conf: 指定一个 DSS 1.x/2.x 的配置文件来生成 xml
-x : 强制建立一个新的 xml 配置文件并退出
-I : 以空闲状态启动服务器

模块

媒体服务器使用模块来响应各种请求及完成任务。有三种类型的模块：

1. 内容管理模块

内容管理模块负责管理与媒体源相关的 RTSP 请求和响应，比如一个文件或者一个广播。每个模块负责解释客户的请求，读取和解析它们的支持文件或者网络源，并且以 RTSP 和 RTP 的方式进行响应。在某些情况下，比如流化 mp3 的模块，使用的则是 HTTP。

内容管理模块包括：

QTSSFileModule

QTSSReflectorModule

QTSSRelayModule

QTSSMP3StreamingModule

2. 服务器支持模块

服务器支持模块执行服务器数据的收集和记录功能。

服务器模块包括：

QTSSErrorLogModule

QTSSAccessLogModule

QTSSWebStatsModule

QTSSWebDebugModule

QTSSAdminModule

QTSSPOSIXFileSystemModule

3. 访问控制模块

访问控制模块提供鉴权和授权功能，以及操作 URL 路径提供支持。

访问控制模块包括：

QTSSAccessModule

QTSSHomeDirectoryModule

QTSSHttpFileModule

QTSSSpamDefenseModule

协议

DSS 支持以下协议：

RTSP over TCP

RTP over UDP

RTP over Apple's Reliable UDP

RTSP/RTP in HTTP (tunneled)

RTP over RTSP (RTP over TCP)

以下模块用 http 实现：

QTSSAdminModule

QTSSMP3StreamingModule

QTSSWebStatsModule

QTSSHTTPStreamingModule

QTSSRefMovieModule

QTSSWebStats

QTSSWebDebugModule

数据访问接口

当一个模块需要访问客户请求的 RTSP 报头时，可以通过 QTSS.h 这个 API 头文件中定义的请求对象来访问相应的请求信息。举例来说，RTSPRequestInterface 类实现了 API 字典元素，这些元素可以通过 API 来进行访问。名称是以"Interface"结尾的对象，比如 RTSPRequestInterface，RTSPSessionInterface，和 QTSServerInterface，则用于实现模块的 API。

下面是重要的接口类：

- QTSServerInterface — 这是内部数据的存储对象，在 API 中标识为 QTSS_ServerObject。在 API 中的每一个 QTSS_ServerAttributes 都在基类中声明和实现。
- RTSPSessionInterace — 这是内部数据的存储对象，在 API 中标识为 qtssRTSPSessionObjectType。在 API 中的每一个 QTSS_RTSPSessionAttributes 都在基类中声明和实现。
- RTPSessionInterface — 这是内部数据的存储对象，在 API 中标识为 QTSS_ClientSessionObject。在 API 中的每一个 QTSS_ClientSessionAttributes 都在基类中声明和实现。
- RTSPRequestInterface — 这是内部数据的存储对象，在 API 中标识为 QTSS_RTSPRequestObject。在 API 中的每一个 QTSS_RTSPRequestAttributes 都在基类中声明和实现。

主体类

DSS 源代码完全采用标准 C++语言写成，编程风格非常优秀，每一个 C++类都对应着一对和类同名的.h/.cpp 文件。整个服务器包括多个子系统，分别存放在独立的工程内，其中，最为重要的是基础功能类库（Common Utilities Lib）和流化服务器（Streaming Server）两个工程。

1、基础功能类库（Common Utilities Lib）

OS 类	执行与系统平台相关的数据结构。
Socket 类	执行与平台相关的 Socket 函数。
Tasks 类	主要包含使服务器异步运行模式的类。
RTCP Utilities Lib	主要包含对 RTCP 封包作解码或编码及产生封包的类。

2、核心功能库（Server Core）

RTSP 子系统	负责解析和处理 RTSP 请求。
RTP 子系统	负责媒体数据包的发送，根据 RTCP 的反馈进行服务质量控制。
公共服务子系统	负责服务器的启动/关闭、初始化参数设置以及为 module 机制、跨平台的多线程和事件机制等提供支持。

源代码组织

Server.tproj

这个目录包含核心服务器（core server）的代码，可以分成三个子系统：

- 服务器内核。这个子系统中的类都有一个 QTSS 前缀。QTSServer 负责处理服务器的启动和关闭。QTSServerInterface 负责保存服务器全局变量，以及收集服务器的各种统计信息。QTSSPrefs 是存储服务器偏好设定的地方。QTSSModule，QTSSModuleInterface，和 QTSSCallbacks 类的唯一目的就是支持 QTSS 的模块 API。
- RTSP 子系统。这些类负责解析和处理 RTSP 请求，以及实现 QTSS 模块 API 的 RTSP 部分。其中的几个类直接对应 QTSS API 的一些元素（比如，RTSPRequestInterface 类就是对应于 QTSS_RTSPRequestObject 对象）。每个 RTSP TCP 连接都有一个 RTSP 会话对象与之相对应。RTSPSession 对象是一个 Task 对象，负责处理与 RTSP 相关的事件。
- RTP 子系统。这些类处理媒体数据的发送。RTPSession 对象包含与所有 RTSP 会话 ID 相关联的数据。每个 RTPSession 都是一个 Task 对象，可以接受核心服务器的调度来进行 RTP 数据包的发送。RTPStream 对象代表一个单独的 RTP 流，一个 RTPSession 对象可以和任何数目的 RTPStream 对象相关联。这两个对象实现了 QTSS 模块 API 中的针对 RTP 的部分。

CommonUtilitiesLib

这个目录含有一个工具箱，包括线程管理，数据结构，网络，和文本解析工具。Darwin 流媒体服务器及其相关工具通过这些类对类似或者相同的任务进行抽象，以减少重复代码；这些类的封装简化了较高层次的代码；借助这些类还分离了专用于不同平台的代码。下面是对目录下的各个类的简短描述：

- OS 类。这些类在时间，条件变量，互斥锁，和线程方面提供了专用于不同平台的代码抽象。这些类包括 OS，OSCond，OSMutex，OSThread，和 OSFileSource；数据结构则包括 OSQueue，OSHashTable，OSHeap，和 OSRef。
- 套接口类（Sockets）。这些类为 TCP 和 UDP 网络通讯方面提供了专用于不同平台的代码抽象。通常情况下，套接口类是异步的（或者说是非阻塞的），可以发送事件给 Task 对象。这些类有：
  EventContext，Socket，UDPSocket，UDPDemuxer，UDPSocketPool，TCPSocket，和 TCPListenerSocket。
- 解析工具。这些类负责解析和格式化文本。包括
  StringParser，StringFormatter，StrPtrLen，和 StringTranslator。

- Task（任务）：这些类实现了服务器的异步事件机制。

QTFileLib

流媒体服务器的一个主要特性就是它能够将索引完成（hinted）的 QuickTime 电影文件通过 RTSP 和 RTP 协议提供给客户。这个目录包含 QTFile 库的源代码，包括负责解析索引完成的 QuickTime 文件的代码。服务器的 RTPFileModule 通过调用 QTFile 库来从索引过的 QuickTime 文件中取得数据包和元数据。QTFile 库可以解析下面几种文件类型：.mov，.mp4（.mov 的一种修改版本），和.3gpp（.mov 的一种修改版本）。

APICommonCode

这个目录包含与 API 相关的类的源代码，比如 moduletils，或者诸如记录文件的管理这样的公共模块函数。

APIModules

这个目录包含流媒体服务器模块目录，每个模块都有一个目录。

RTSPClientLib

这个目录包含实现 RTSP 客户端的源代码，这些代码可以用于连接服务器，只要该连接协议被支持。

RTCPUtilitiesLib

这个目录包含解析 RTCP 请求的源代码。

APIStubLib

这个目录包含 API 的定义和支持文件。

HTTPUtilitiesLib

这个目录包含解析 HTTP 请求的源代码。

二次开发模块添加的要求

每个 DSS 模块必须实现两个函数：一个是 Main 函数，服务器在启动时将调用这个函数进行必要的初始化。另一个是 Dispatch 函数，通过实现此函数，服务器可调用 DSS 模块并完成特定处理。对于编译到服务器里面的模块，其主函数的地址必须传递到服务器的模块 Main 函数中。

具体实现时，Main 函数必须命名为 MyModule_Main，其中 MyModule 是模块的文件名。此函数的实现通常如下所示：

```
QTSS_Error MyModule_Main(void* inPrivateArgs)
{
  return _stublibrary_main(inPrivateArgs, MyModuleDispatch);
}
```

每个 DSS 模块都必须提供一个 Dispatch 函数。服务器为了特定的目的需要使用某个模块时，是通过调用该模块的 Dispatch 函数来实现的，调用时必须将任务的名称及相应的参数传递给该函数。在 DSS 中，使用角色(Role)这个术语来描述特定的任务。Dispatch 函数的格式如下所示：

```
void MyModuleDispatch(QTSS_Role inRole，QTSS_RoleParamPtr inParams)；
```

其中 MyModuleDispatch 是 Dispatch 函数的名称；MyModule 是模块的文件名；inRole 是角色的名称，只有注册了该角色的模块才会被调用；inParams 则是一个结构体，可用于传递相应的参数。