ART.framework 架构之谜

ART.framework 研究分析

2012/4/30

湘南学院计算机系
李博文

项目团队 ART.Studio

前言

自 Lady Lovelace 写下第一个程序以来，程序员已经写了 100 多年代码，比电子计算机还要早一百多年。1946 年，ENIAC 诞生了，计算机终于跨上了高速路，迅猛发展起来。在此之前，计算机的发展是非常缓慢的，在此之后，计算机硬件以摩尔定律或者超过摩尔定律发展。

可惜，软件并不是那么令人如愿，各种硬件平台上深深地隔阂使得软件并不能实现兼容各个平台。程序员们要付出巨大的努力才能使得程序在各个平台上运行，艺术家们利用快捷的开发模式涉入程序开发领域，传统的程序员受到冲击，成为一个优秀的程序员越来越难，人们对软件的需求却越来越大。这都是软件开发需要迫切解决的问题。

软件的兼容性使得软件的部署具有很明显的局部性，人们不得不投入更多的资源开发适合其他平台的软件，在 WPS Office 开发 23 年之后，才正式开始了 Linux 系统上的移植。Qt 是跨平台的图形用户界面应用程序开发框架，尽管 WPS Office On Linux 都是基于 Qt 开发的，要实现 Linux 上的 WPS Office 都存在很多的问题。为了布局 Linux，WPS，WPS2012 放弃了 delphi，也是使用 Qt 开发，尽管这样代码库一致，但是 Linux 版本任然要等待很久才能发布，现在还未进入 Beta 阶段。

很多跨平台的软件都不得不在源代码中添加额外的宏定义如：

```
#ifdef _WIN32
#include <xxxx>
#endif
```

也有一些库需要针对不同平台写一些平台相关代码，组织不同平台的二进制，或许还要针对不同平台设置编译环境。

这样下来程序员需要更多的精力去学习那些突兀的平台特性，虽然平台特性的学习尤为重要，但不得不说，这并不适合快速开发。对程序员来说这也非常消耗时间的。

ARM 在不愠不火的发展多年后，随着移动互联网，智能手机，平板电脑的发展，一下子爆火起来。ARM 是 ARM 系列处理器 IP 内核设计公司，ARM 处理器基于 RSIC（精简指令集）构架。而我们传统的 x86 处理器是基于 CISC（复杂指令集）构架的，CISC 指令多大几百条，而 RISC 指令一般都只有几十条（RISC-I 31 条，RISC-II 39 条）。32 位的 x86 指令是一种可变长的指令 1~12 字节不等，寻址方式种类多，RISC 指令长度固定，寻址方式种类少。除此之外，x86 处理器有 Ring0，Ring1，Ring2，Ring3 4 个特权级别模式，ARM 处理器的特权模式是不相同的，ARM 有七种工作模式，除用户模式外，其它模式均为特权模式（Privileged Modes）。ARM 内部寄存器和一些片内外设在硬件设计上只允许（或者可选为只允许）特权模式下访问。此外，特权模式可以自由的切换处理器模式，而用户模式不能直接切换到别的模式。

异常模式

特权模式中除系统（system）模式之外的其他 5 种 模式又统称为异常模式。它们除了可以通过在特权下的程序切换进入外，也可以由特定的异常进入。比如硬件产生中断信号进入中断异常模式，读取没有权限数据进 入中止异常模式，执行未定义指令时进入未定义指令中止异常模式。其中管理模式也称为超级用户

模式，是为操作系统提供软中断的特有模式，正是由于有了软中 断，用户程序才可以通过系统调用切换到管理模式。

处理器工作模式	特权模式	异常模式	说明
用户（user）模式			用户程序运行模式
系统（system）模式	该组模式下可以任意访问系统资源		运行特权级的操作系统任务
一般中断（IRQ）模式		通常由系统异常状态切换进该组模式	普通中断模式
快速中断（FIQ）模式			快速中断模式
管理（supervisor）模式			提供操作系统使用的一种保护模式，swi命令状态
中止（abort）模式			虚拟内存管理和内存数据访问保护
未定义指令终止（undefined）模式			支持通过软件仿真硬件的协处理

除了用户模式其他都是特权模式。

不同的处理器存在二进制不兼容，这由处理器指令集和处理器特性决定。而相同的处理器如果有同的操作系统一般也是不兼容到的，操作系统人为的造成了软件兼容性问题，问题何在，最开始可以发现是可执行格式的无法识别，无法识别,内核就不能将可执行程序代码加载到内存运行，而后还存在函数无法访问，诸多问题。软件兼容性问题一直都不是那么令人满意！

自去年五月以来我一直思考如何解决软件兼容性问题，也了解到很多项目，如：底层虚拟机（LLVM），.NET，Mono，JVM，wine，ReactOS，GCC，XMLVM，Boost，OpenGL，OpenCL，Qt，wxWidgets，GCC，Open WATCOM ，还有 Linux 兼容内核 Longene，GTK+，WTL，在反复审视后，最终提出了 ART.framework 的构想，ART 代表基本应用编程接口和基本运行时（Base API&Base Runtime），并且以开源的方式研究，并获得湘南学院计算机系支持。我将在正文详细介绍 ART.framework 详细构想，发展步骤，以及团队成员。

湘南学院 计算机系
2012.05.04
李博文

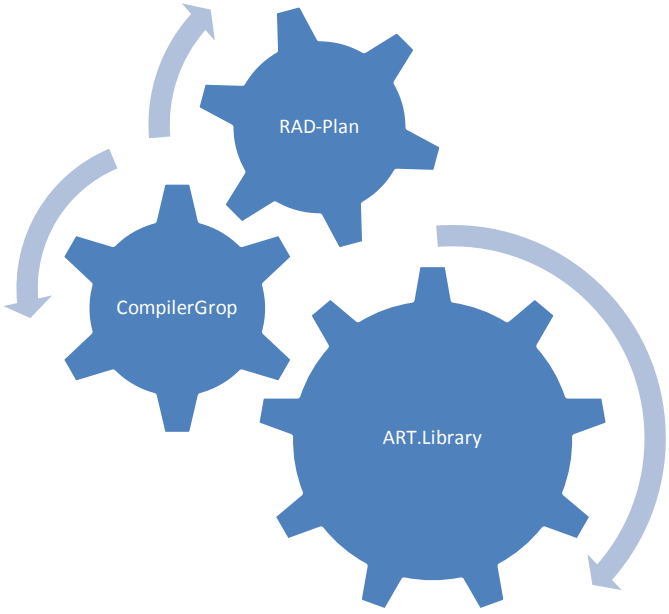
ART.framework 核心分析

ART.framework 即基本应用编程接口和基本运行时框架，你可能很简单的理解，ART 是.NET 的开源跨平台版本，但事实上并不是如此，ART.framework 的目标是：适用 ART-API 开发的应用程序适用于安装有 ART-API 库的操作系统，并且，在目标平台上是使用机器码运行，不需要虚拟机，最大限度的消除无关的条件，并且 RuntimeLibrary 提供安全保障，增强应用程序健壮性！而且，ART.framework 致力于解决软件开发存在的问题，降低软件开发的难度，发布的难度，部署的难度！

我们知道软件兼容性问题主要存在两个方面，一个是硬件平台，硬件平台的问题核心主要是处理器，其它部分的兼容性问题通过标准化基本可以得到解决，而软件代码由处理器执行，然而，处理器架构的巨大差别导致软件完全无法兼容，这个是问题的核心。二是操作系统，每个操作系统都有自己的方式加载程序，提供系统调用，这个方面，不同的操作系统也一般存在着软件兼容性问题。

对此，我们提出一个方案，两个分支，解决软件兼容性问题。一个是从处理器级别解决软件兼容性问题，另一个是从操作系统级别解决软件兼容性问题，这就是 ART---Base API & Base Runtime 项目。

要实现 ART 项目，我们提出了 ART.framework，即 ART 框架，利用 ART.framework 实现我们的构想。ART.framework 开发模块有 ART.Library(ART 库)，编译器组，快速开发计划。



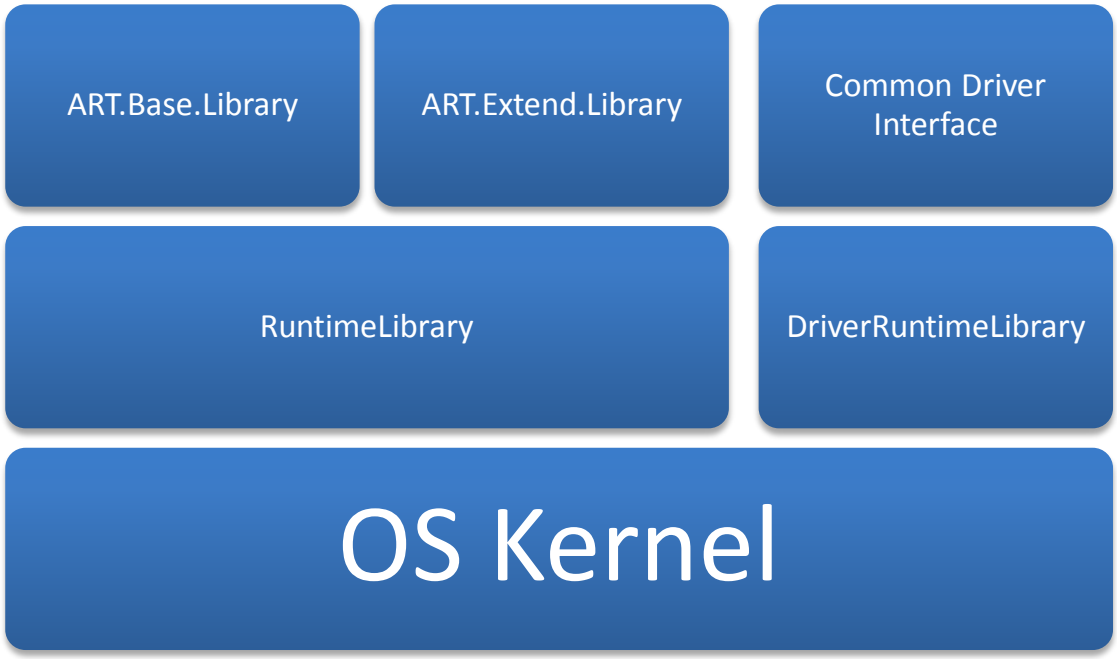
ART.framework 模块 图 T-1

可以看到，难度最大，而且最核心的是 ART.Library,其次是编译器组，最后是 RAD 计划，实现方面，我们要先实现 ART.Library，因为 RAD 是基于 ART.Library 实现的，而编译器组的实现要同步 ART.Library。

一个初步的构想是，所有基于 ART.Library 开发的应用程序都能运行在不同的平台上，这需要将 ART.Library 进行平台抽象和标准化，为不同的平台提供统一的接口，这样，库是和平台相关的，为了进一步实现平台抽象化，也是符合微内核的构想，我们只把一些系统底层的系统调用抽象标准化，如文件系统，硬件

资源访问，进程和线程创建，底层图形以及命令接口的抽象化。还要实现 C 与 C++语言的运行时库。这部分很基础，叫做 RuntimeLibrary，但仅仅这样是无法实现更多的功能和一些基本的应用开发，在 RuntimeLibrary 的基础上实现 ART.Base.Library 即 ART 基本库，这个库与平台无关，完全一样的代码适合任何平台，ART.Base.Library（ABL）实现一些比教常见的功能，如网络通信，图形界面，封装的进程和线程库，科学计算，图像，字处理，正则表达式。ART.Base.Library 由于其跨平台要求，必须保持精简和高效。

方便应用开发的立场是不能改变的，我们支持在 RuntimeLibrary 的基础上实现各种库，方便应用开发，但是由于 ART.Library 的设计要求，并不建议在 ABL 中实现各种库，因此我们提出了 ART.Extand.Library (AEL)，即 ART 扩展库，ART 项目组定期审查 ABL，AEL 决定 ABL 的某些模块是否降级为 AEL，或则 AEL 某些模块是否升级为 ABL，是否增加新的内容！



ART.Library 核心 图 T-2

没错，这就是 ART.Library 的核心，RTL 内陷到操作系统内核（一部分底层操作），对于宏内核的 Linux 等尤其需要这样做，对于微内核并没有多大关系。

DriverRuntimeLibrary 驱动运行时库，这是以后支持兼容模型开发驱动的一部分，在计划中，先期任务主要是普通应用程序开发。

RuntimeLibrary 的核心主要有 VSC Virtual System Call，VFS Virtual File System；RUN 程序运行相关的函数，Lib 基本的语言运行时库，这部分正如前文所说，这个是一个由具体平台到抽象平台的过程，RTL 基本上封装了每个平台的 API 抽象出一组基于 ART 的 API，事实上，完全可以利用 RTL 的接口开发比较底层的应用程序。

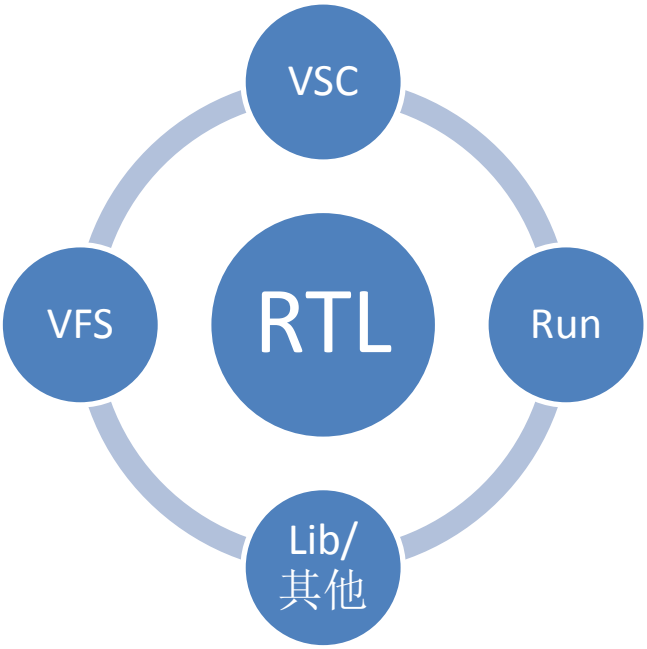
VSC 封装基本的系统调用，VFS 屏蔽不同的文件系统特性，使得任何文件系统的基本操作对于 RuntimeLibrary 外部程序而言都是一样的。

Run，为什么要单独列出来，因为最关键的是封装了进程线程函数后，我们还需要针对平台做不同的处理才能实现原生的进程和线程的最优。

Lib 依然是那些底层的语言运行时库。一些底层的其他组件并没有例如下图，

事实上，我们还是会在底层支持，如基于图形的 OpenGL，图形界面的 wxWidgets 的图形底端与系统有关的，还有 OpenCL 等。

RTL 还将实现代码安全，减少程序的错误。



RuntimeLibrary 模型 图 T-3

ABL 视图 T-4

ART.Base.Library 包含的基本组件有上述模块，这个也许是暂时的。但必要的，底层的平台有关的都要脱离，由 RTL 实现，上层的可以重用。

ABL 的开发离不开开源项目的支持，ART 项目本就要开源，合理的利用前辈的资源发展并不是一件坏事，软件业发展就是在巨人的肩膀上看的更远。

目前，已经计划研究的开源库有图形界面库 wxWidgets C++准标准库 Boost，Boost 提供的库真的比较强大，支援的领域也比较多，ABL 利用 Boost 库无疑会

获得优秀的性能和比较广泛的支持。不过 Boost 要迁移到 RTL 上来，研究压力也会很大。

ABL 开源支持库 T-5

一些其他的库的审查也将逐步展开。

AEL 一般而言只要开发在 RTL 基本上，并不存在很多压力。

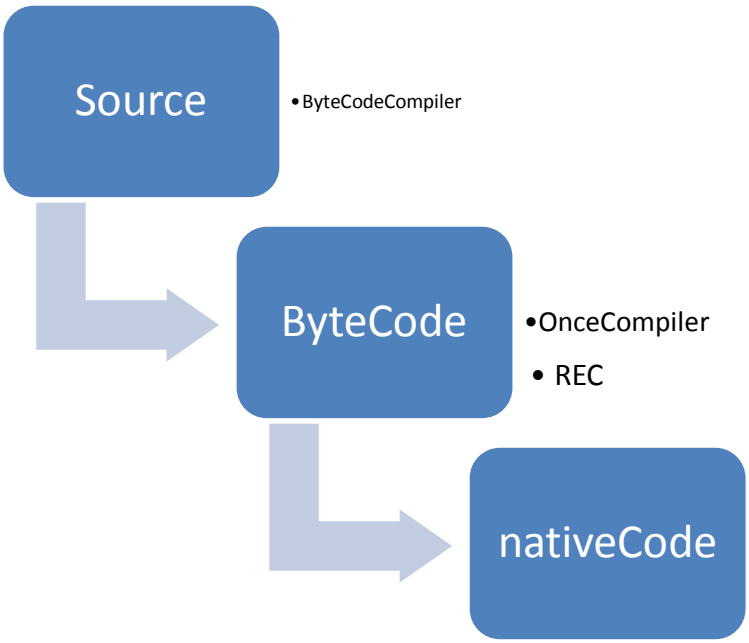
AEL T-6

AEL 的支持是开放的，所以项目组只会以 ABL 的标准标准化，具体实现是开放的。所以以数字代替。

单纯的从库上解决并不能实现我们的目标，只根本只是开始，没有编译器组，无法生成指定格式的原生程序，更谈不上程序的原生执行。

直接编译成原生程序并不是很合适，原因在于开发者要针对不同点的平台进行编译，开发任务繁重，ART 的设计思想是开发只需要程序员一次编译，客户端

一次编译，程序就可以运行。开发主机上编译如果直接编译要实现跨平台编译，并且很难针对特定平台进行深度优化，毕竟，同一个构架的不同处理器可能存在少量的指令差异，CISI 构架经常会出现如此状况。ART 的设计思想实现跨平台是不建议直接原生编译，我们采取的策略是：



所有的程序都是有源代码到可执行文件，最终执行的也是原生代码，Java 虚拟机中的所谓字节码也是有 JVM 虚拟机解释后自己执行的。ByteCode 字节码，ByteCode 的目标仍未确定，在方案中，有几种可选的格式可选，一个是使用 CISCx86 指令格式，其他平台只需要流水线扩充或者优化，在其他平台上编译很容易，在 x86 上也只要优化和链接就可以了。还一种虚拟指令格式，针对每一个平台都是一样的，VI 是一种理想的指令格式，所以对任何平台都是均权的。

不得不说的是 ByteCode 的格式由 COFF（COFF 格式定义在下文）扩展为 BOFF 格式，这是为了抽象到不同平台上二人为设定的格式，方便转化为不同平台上的格式。

这样我们思考一下代码编译的过程，第一步，在开发主机上程序员编写好之包含 ART 类型库的应用程序，第一次编译，这次编译为 ByteCode,不链接任何外部库，这时候可以将程序打包部署在任何安装有 ART.Core 的客户端上，当程序安装后运行首次，ART.Service 被激活，开始利用 ArtBuild 工具利用配置文件将基于 BOFF 格式的 ByteCode 编译成本机机器码，随后激活冗余消除编译器，对代码进行优化，随后部署到镜像目录，原生程序开始运行，下一次的时候，由于原生程序已经存在，所以系统直接运行原生程序，不需要再次编译，处理器的差异和简化开发者的开发而言，这时非常合适的选择！

为此，我们需要开发一组编译器，包含最基本的 3 类，字节码编译器，一次编译器，冗余消除编译器，还有一系列的相应套件。

对于编译的开发，一是利用现有的开源项目进行修改移植，而是重新开发，当然也要学习开源编译器项目。前一种也并不轻松，因为要修改的很多。无论如何，我们都要借鉴开源编译器项目的优点，这里编译器有 Open WATCOM；LLVM；GCC 等等。 事实上跨平台不错的有 GCC，OW 代码优化很不错，LLVM 的理念很优秀……

当然，ART 支持的语言不会仅仅只有 C/C++，还包括 C#，Java，以及其他的语言，其他语言也要严格按 ART 的规则实现。

ART 致力于平台的无关性，对于 C/C++程序，开发的程序入口是 mainART()，wmainART()，对于特殊平台的还提供 mainARTU()；代码封装如下：

一般程序主源文件：

```
#include <…>
Using namespace ….;
int mainART(int argc,char* argv[])
{
return 0;
}
```

内部封装为：

```
/*ARTOBJ*/
int mainART(int argc,char* argv[]);

int main(int argc,char *argv[])
{
    return mainART(argc,argv);
}
```

这样很容易实现内部封装。程序只有在 OC 的时候才会主动链接不同平台的 ARTOBJ，这只是小技巧而已。

附：COFF 格式如下：

```
/*Common Object File Format*/
// 2012 (C) Copyright ART Public License ;
//
/*-------------------------*/
/*
 *   1. 文件头（File Header）
      2. 可选头（Optional Header）
      3. 段落头（Section Header）
      4. 段落数据（Section Data）
      5. 重定位表（Relocation Directives）
      6. 行号表（Line Numbers）
      7. 符号表（Symbol Table）
      8. 字符串表（String Table）
 *
 * */
#ifndef _COFF_H
#define _COFF_H
#define MAG_I386 0x014c

#define VNull 0
#define BIN_COFF 0x010b
#define ROM_COFF 0x0107
```

```
/*COFF-header*/
typedef struct Filehdr{
    unsigned short usMagic;//魔法数字
    unsigned short usNumSec;//段落（Section）数
    unsigned long ulTime;//时间戳
    unsigned long ulSymbolOffset;//符号表偏移
    unsigned long ulNumSymbol;//符号数
    unsigned short usOptHdrSZ;//可选头长度
    unsigned short usFlags;//文件标记
}FILEHDR;
/*COFF-可选头*/
typedef struct Opthdr{
    unsigned short usMagic;//魔法数字
    unsigned short usVersion;//版本标识
    unsigned long ulTextSize;//正文（text）段大小
    unsigned long ulInitDataSZ;//已初始化数据段大小
    unsigned long ulUninitDataSZ;//为初始化数据段大小
    unsigned long ulEntry;//入口点
    unsigned long ulTextBase;//正文段基址
    unsigned long ulDataBase;//数据段基址（在 PE32 中才有）
}OPTHDR
/*COFF-段落头*/
typedef struct Sechdr{
    char cName[8];//段名
    unsigned long ulVSize;//虚拟大小
    unsigned long ulVAdde;//虚拟地址
    unsigned long ulSize;//段长度
    unsigned long ulSecOffset;//段数据偏移
    unsigned long ulRelOffset;//段重定位表偏移
    unsigned long ulLNOffset;//行号表偏移
    unsigned long ulNumRel;//重定位表长度
    unsigned long ulNumLN;//行号表长度
    unsigned long ulFlags;//段标识
}SECHDR;
/*COFF-段数据*/
typedef struct Reloc{
    unsigned long ulAddr;//定位偏移
    unsigned long ulSymbol;//符号
    unsigned short usType;//定位类型
}RELOC;
/*COFF-行号表*/
typedef struct Lineno{
    unsigned long ulAddrORSymbol;//代码地址或符号索引
    unsigned short usLineNo;//行号
```

```
}LINENO;
/*COFF-符号表*/
typedef struct Syment{
    union {
        char cName[8];//符号表标识
        struct{
            unsigned long ulZero;//字符表表示
            unsigned long ulOffset;//字符串偏移
        }e;
    }e;
    unsigned long ulValue;//符号值
    short iSection;//符号所在段
    unsigned short usType;//符号类型
    unsigned char usClass;//符号存储类型
    unsigned char usNumAux;//符号附加记录数
}SYMENT;
#endif
```

还需要实现一个 COFF 结构或者 COFF 类，

COFF 结构，C 语言：

```
typedef struct    cBinCoff{
    Filehdr BinFilehdr;
    Opthdr    BinOpthdr;
    Sechdr BinSechdr;
    Reloc BinReloc;
    Lineno BinLineno;
    Syment BinSyment;
}CBINCOFF;
```

如果是 C++，则使用类实现

```
class COFF{
private:
    Filehdr BinFilehdr;
    Opthdr    BinOpthdr;
    Sechdr BinSechdr;
    Reloc BinReloc;
    Lineno BinLineno;
    Syment BinSyment;
public:
                void InitCOFF();
};
void COFF::InitCOFF()
{
}
```

ART 在客户端上实现跨平台的核心部件是 ART.Core，ART 标准应用程序部署在客户端需要的核心就是 ART.Core，ART 提供了 RTL，ABL。OC，REC，ART.Service，ART.Control 其他的都不支持，这些包含最简的部分，保证客户端的 ART 最合适的体积和最合适的效率。ART 应用程序只需要这些就能运行。

ART.Service 和 ART.Control 同样要保证 ART 应用程序的正常运行，安装卸载，版本控制，所以，这也是必不可少的组件之一。

ART-RAD 计划：

快速开发计划也是 ART 的计划之一。ART 的原则是简化程序的开发，所以一个 RAD 计划也是项目组的规划之一，提供 RAD 还有一个重要的部分，UIDL，用户界面设计语言，这个语言基于 XML 或者是 HTML-5/CSS-3 当然后者的设计的界面文件可以直接使用浏览器调试。

目前常见的设计界面的有 XMAL，Resource Script ，QML，不过 Resource Script 是 Windows 上比较老的界面资源设计，更加底层，但是可视化设计并不是很好，而 XMAL 是 WPF 的界面设计语言可视化设计较好，支持的平台也多了 Windows Phone ，QML 支持的平台更多。很多软件开发商也在研究可视化设计快速开发模式。都是为了使得软件开发的难度降低，所以 ART 也是支持这个方向的。

ART 与有关项目的比较

ART 注定要发挥各种开源项目的优点，对于一些商业项目，事实上，ART 也需要借鉴他们的技术和特点，不过前提是，商业软件不得使用源代码，不得侵犯软件版权，开源软件使用源代码或者重写必须遵循原有的开源协议。一下是对 ART 有影响或者值得 ART 借鉴的项目，软件，架构的一个介绍和分析。

.NET，来自微软，我们指的.NET 是 Microsoft.NET framework 目前最高版本是 4.5beta，来自百度百科的完整介绍为：

.NET 框架是以一种采用系统虚拟机运行的编程平台，以通用语言运行库（Common Language Runtime）为基础，支持多种语言（C#、VB、C++、Python 等）的开发。.NET 也为应用程序接口（API）提供了新功能和开发工具。这些革新使得程序设计员可以同时进行 Windows 应用软件和网络应用软件以及组件和服务（web 服务）的开发。.NET 提供了一个新的反射性的且面向对象程序设计编程接口。.NET 设计得足够通用化从而使许多不同高级语言都得以被汇集。.NET Framework 中的所有语言都提供基类库(BCL)。

有些人说这是山寨版 Java/JVM，我们先不谈这些，我将.NET 放置在最开始是有缘由的，就 ART 全局模型来看，ART 更像.NET 的开源版本，多语言扩充版本……仔细分析.NET 的核心我们就会发现，ART 项目的意义。

诚然，.NET 只是微软的私家财产，微软决定它跨平台就跨平台，然而，微软的跨平台也仅仅限制在 Windows，Windows Phone，WindowsRT，微软是不会轻易的支持 Microsoft.NET 部署在 Linux，BSD，Mac OS …… 而 ART 的优势就在于 ART 支持大部分平台，当然，这也是一个过程，ART 的目标总是支持更多地平台。虽然.NET 支持生产原生映像（ngen.exe），但.NET 默认下支持 CLR，在客户端部署运行的时候是通过 JIT 实现的，显然微软并没有打算更进一步，在 Windows 系统目录”C:\Windows\assembly”目录下你会发现很多不一样的文件，这些文件的属性中有一栏处理器，在 x86 平台上的处理器为 x86，而 MSIL，则是微软中间语言，也就是 CLR 的代码格式，我们可以发现就只有很少的文件实现了原生代码，显然，微软准备在程序再次运行之前再次编译一次。

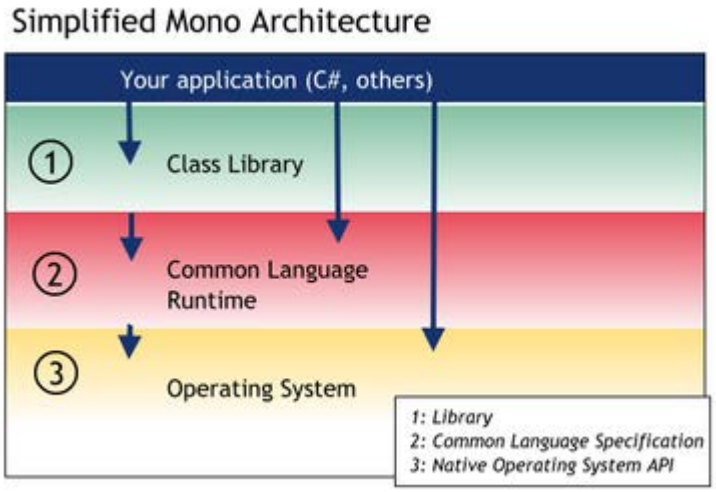
我们认为，外部存储器有足够的容量存储额外的原生映像的，而减少编译次数对提高基于 CLR 的程序的运行效率有无与伦比的诱惑力。微软并不情愿这么做，而 ART 显然决定要减少不必要的编译时间，所以 ART 与.NET 不同的另一点就是：

ART 只支持客户端程序使用原生代码直接运行！

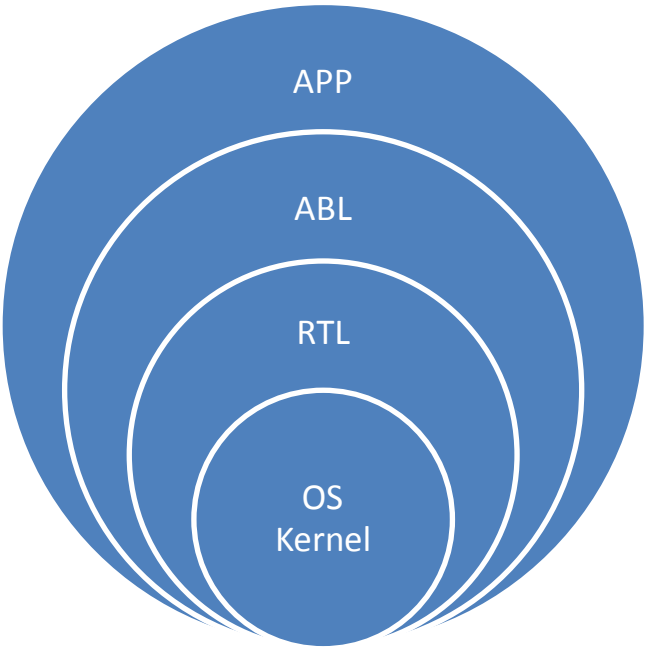
Mono.NET 是 Microsoft.NET 的跨平台和开源实现，也是 ART 值得学习的一个项目。事实上，我们发现 ART 的模型和 Mono 有很高的相似性。Mono 的官方网址为：http://www.mono-project.com/Main_Page ART 不可能是 Mono 的克隆版本，不过 Mono 的技术还是可以借鉴，虚拟机，类库，不同操作系统的移植，这些技术都是 ART 紧迫的，研究充满压力的部分，但是，基于 ART 的核心的抽象化，ART.Library 的平台抽象化，ART.Library 的库的原则也和 Mono 类库存在着很多的不同，Mono 作为一个 Microsoft.NET 的开源跨平台实现，基本上继承了.NET 的特性以保持对.NET 的高度兼容，所以在.NET 与 ART 的核心区别，适用于 ART 和 Mono 的区别。

Mono 的架构是操作系统→Common Language Runtime→Class Library→App；

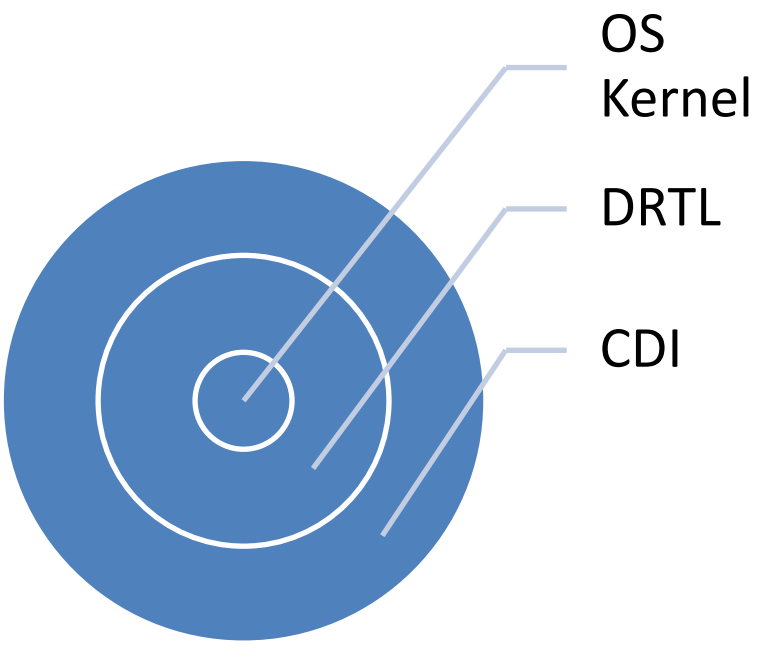
而 ART 的核心是，OS Kernel→RTL→ABL(AEL)；虽然存在相似之处，事实上，这些库都是本地化了的，原生支持系统的。比较底层实现，AEL 扩展性强。



MONO 体系结构



ART 体系结构



ART 标准驱动体系结构

我们注意到.NET 和 Mono 确实有实现类似的 OC，可能也有人怀疑 ART 是否有实现的必要，Mono 存在的目标只是实现.NET 的跨平台，提供的是 Runtime 实现跨平台，ART 是依据本地化和 RuntimeLibrary 实现跨平台，目标存在差别，实现上也存在差别，而 ART 的 RTL 也肩负着实现 VFS，VSC 的重任，很容易推测，完全基于 ART 技术很容易实现一个与其他操作系统和平台兼容的操作系统。ART 是剥离计算机不兼容部分实现统一接口的一个宏伟计划。ART 能够实现兼容驱动也是 Mono 达不到的。底层的开发往往依靠 C/C++和汇编，ART 在不损失 C/C++汇编代码效率的前提下实现跨平台拥有致命的诱惑力，封装 C/C++汇编的快速开发，使得这些语言的使用变得更加容易和高效，不只是程序员还有机器。

LLVM 是一个编译器基础项目（Low Level Virtual Machine）底层虚拟机，来自维基百科的介绍时：

它是一个编译器的基础建设，以 C++写成。它是为了任意一种编程语言写成的程式，利用虚拟技术，创造出编译时期，链结时期，执行时期以及“闲置时期”的最佳化。它最早是以 C/C++为实作对象，目前它支援了包括 Objective-C、Fortran、Ada、Haskell、Java bytecode、Python、Ruby、 ActionScript、GLSL 以及其他语言。

LLVM 也是 ART 技术的来源之一，这点毋庸置疑，LLVM 以其优异的技术折服了我。Mono+LLVM+wine+Cygwin+Boost+OpenGL+wxWidgets…这些是 ART 的技术来源，ART 除了重构整个框架，很多部分都是由这些模块的整合和优化，底层剥离和平台抽象无关化。而 ART 的编译器组很多要继承自 LLVM，并在格式和体系上进行相对于 ART 的优化。

LLVM 可以实现高效的 JIT，经过改造可以实现 ART 的 OC，具体步骤有些不同，ART 程序在跨平台的时候需要 OC，链接 再实现 REC 获得最优化的本地代码，并且受到 ART.Service 和 ART.Control 的监管。

LLVM 的多语言支持也是 ART 需要实现的，所有语言，除了脚本语言，在 ART 的眼中都只是编译性语言，都是最终会实现本地化的。最终在客户端运行都是直接的机器码运行。

LLVM 官方主页：http://www.llvm.org

还有一些项目值得我们借鉴，例如 wine，wine 这个软件可以使得 Linux 运行 Windows PE 程序，相当于一个兼容 API 层，还有 Linux 兼容内核 Longene 在内核上实现对 Windows 的兼容，同时能够运行 Linux 程序，还有 ReactOS 开源的 Windows NT 操作系统的实现，目的是完全兼容 Windows 程序，驱动，等等。这些项目在内核，API 层面上实现的 Windows 的兼容给了 ART 很好的启示，ART 的 RTL 构思也有一部分来自 wine，在 ART 实现 Windows NT 的抽象化可能需要得到这些项目的支持，还有 Cygwin 在 Windows 系统上实现 POSIX 调用，虽然相对于 Windows 的 POSIX 子系统而言，这个 Cygwin 并没有那么高效，但是由于其开源有利于 ART 的深入研究。ART 对平台的抽象化封装很难讲不利用这些现有的技术，现有的代码。当然，并不能说这个就是这些项目简单的合并发展，因为 ART 的核心是实现程序跨越操作系统，跨越硬件平台运行，这些项目并不能完全做到，所以还是存在差别。

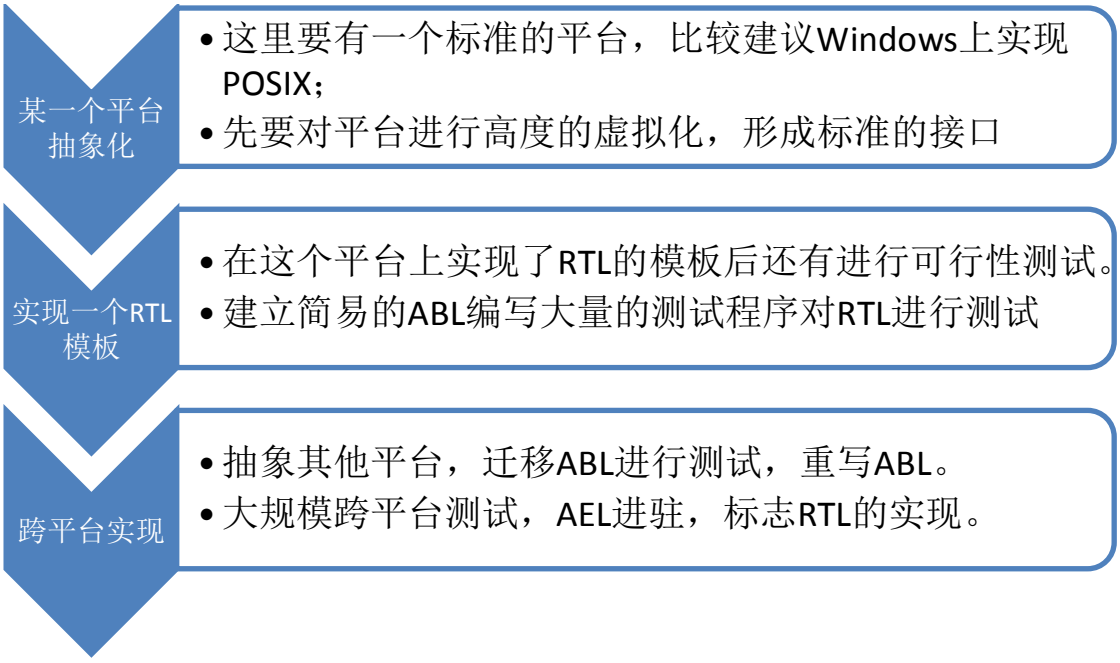
Java，我们并没有忽略它，Java 的低效率和甲骨文的垄断使得我并不是很情愿的提起这个跨平台的语言和工具集。Java 的跨平台是实现在 JVM 之上的，JVM 是个小丑，效率低下，虽然现在有所谓 JIT 编译成机器码运行，但是和静态编译的程序相比，JIT 并不能达到我们的期待，官方的 Java 并不是开源的，只有开源社区的 OpenJDK 才是开源的，甲骨文总是信誓旦旦的将 Java 是他们的“私有财产”，Java 语言是一个专利，这是很滑稽的，不可否认，Java 对原生程序的编译支持，官方完全没有当作一回事，而且 JIT 并不像.NET 的 JIT 那么高效，换而言之，.NET 超越了 Java，虽然.NET 不开源，不跨跟多的平台。不过 Java 的 API 平台封装和实现值得我们仔细研究，我们并不会使用甲骨文的 Java，而是使用 OpenJDK 来研究 Java API，毕竟很多平台都有实现了 Java API，对 Java API 的 C++ 转换也是我们要研究的问题之一。

还有很多的库会对 ART 产生积极的影响，这些项目有：OpenAL 开源声音库，OpenCL 开放运算语言 （实现跨 GPU 的 GPU 运算），OpenGL 开放图形库 绘制二维和三维图形，OpenSSL 开放安全套接层协议，XMLVM 交叉编译器工具链，还有一系列操作系统或者内核的研究如：BSD，Drawin，Linux，Singularity 微软研究中的操作系统，WindowsResearchKernel（Windows 研究内核 2003），Minix，Mach 等等。

一般而言上述项目都可以通过合法的途径获得标准，或者源代码，针对源代码和实例的研究是我们的重点。我们也相信通过对这些项目的研究能够为 ART 实现做出巨大贡献。

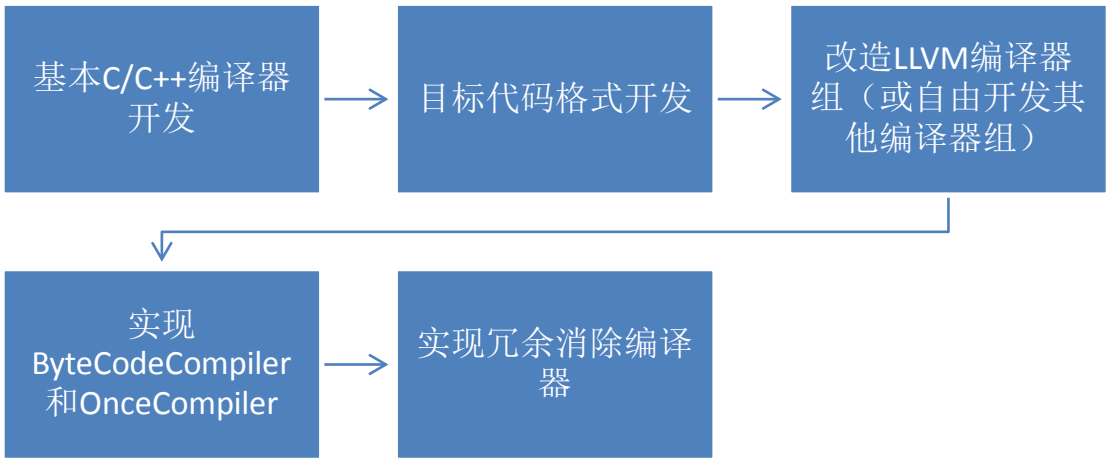
ART 项目规划

ART 项目的实现时一个长久的过程，最开始要启动 RTL 的研究开发，其次才能实现 ABL/AEL，而我们说过 DRTL 的复杂性使得 DRTL 将在以后才会去实现。实现 RTL 一般而言要进行的步骤是：



在实现 RTL 的后期也要开发 ABL 对 RTL 进行测试，可以发现这个过程是复杂而且反复的。

CG，CompilerGrop，编译器组的实现离不开开源的编译器，发展步骤如下：

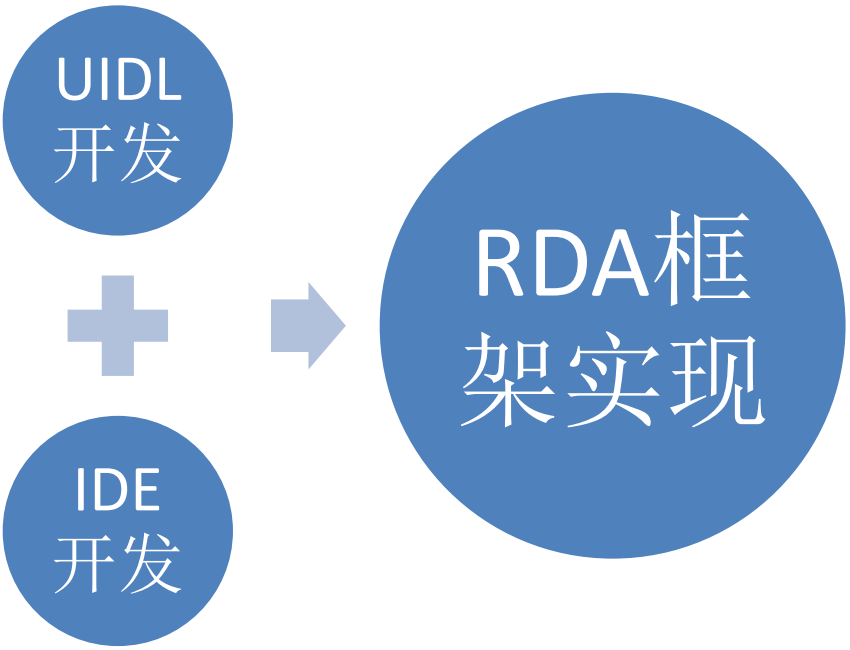


编译器组的开发与 RTL 同步，而且必须在 ABL 跨平台开发完成之前完成，因为项目需要使用编译器组的工具进行开发。

编译器组还包含一些工具，ArtBuild 自动构建工具，RCBuild 资源构建工具，ArtDebug 调试工具，ArtShell 命令控制台.....

但上述项目能够实现编译 ABL 程序时就可以开发 RAD 主体架构。

RAD 开发步骤如下：



UIDL 的设计是可以先期实现的，实现要靠后。IDE 的开发只能在 ABL 实现后才能开发，并且还要使用 AEL 才能实现更加强大的功能。所以 IDE 开发靠后，RDA 的合并完工要等到 IDE 和 UIDL 开发后，开发一系列工具后就可以了。

开发人员开发任务图：

开发人员	学校	主要开发任务	备注
李博文	湘南学院	ART.Library/CG/RAD/RAD-UIDL	
尹望峰	北京大学	CG	
胡杰	湘潭大学	Library	
李想	湘南学院	RAD-UIDL	
李栋	西南石油大学/空军航空大学	Library	
王恺敏	湘南学院	Tools	

项目主要负责人：李博文 联系电话：18711506975
E-mail:stnapoli@msn.cn

项目主页：未定。

2012 年 5 月 7 日星期一