文章编号:1001 - 9081(2007)06 - 00 - 0

基于 CAR 构件的 WebKit 本地扩展策略

蒋章概,陈 榕
(同济大学 基础软件工程中心,上海 200092)
(zkjiang163@163.com)

摘 要:通过浏览器来完成应用与用户的交互,能够简化应用程序的开发和移植,是当前研究热点。JIL、Palm WebOS、Android 都采取类似的解决方案。为强化对基于浏览器的应用程序的支持,Google 在 Chrome 的新版中引入了二进制的本地扩展解决方案。CAR 是一种二进制的构件技术,目标是嵌入式应用。使用 CAR 技术对开源的浏览器引擎 WebKit 进行本地扩展是本研究的重点。

关键词:WebKit;CAR 构件;JavaScriptCore;本地扩展;Elastos
中图分类号: TP316 **文献标志码**:A

Native extension strategy of WebKit based on CAR components

JIANG Zhang-gai, CHEN Rong
(*System Software Engineering Centre, Tongji University, Shanghai* 200092, *China*)

Abstract: Accomplishing the interaction of applications and users through a browser is one of the current research focus and it can simplify the development and transplantation of application programs. The shared resolution has been taken by JIL, Palm WebOS and Android. In order to intensify the support provided to browser-based application programs, Google import a kind of binary native extension resolution into the new version of Chrome. CAR is a kind of binary component technology and its aim is in embedded applications. Using CAR technology to natively extend the open source browser engine WebKit will be the crucial point of this research.

Key words: WebKit; CAR components; JavaScriptCore; native extension; Elastos

0 引言

WebKit 是一种开源的浏览器引擎,它应用广泛,Apple 的 Safari、Google 的 Chrome 和 Nokia 的 Series 60 都基于它进行了开发,Palm WebOS 更是融合了 WebKit 并将界面系统管理器构建于它之上。

WebKit 对 Web 资源进行解析排版,通过 JavaScript 和文档对象模型(Document Object Model,DOM)与用户进行交互。然而基于 WebKit 的 Web 应用程序受限于 WebKit 本身,在本地调用和执行性能上存在着障碍。现代的浏览器提供了扩展机制来加载本地代码以作为 Web 应用的一部分,比如 ActiveX 和 NPAPI 插件的方式[1]。Google 提出了 Native Client 技术[1]。本文基于 Elastos 嵌入式操作系统平台进行探索,基于 CAR(Component Assembly Runtime)构件技术提出了另外一种本地扩展策略,即改造 WebKit,将 JavaScript 对象本地化,实现 WebKit 与 CAR 构件互操作。

1 CAR 构件技术和 WebKit 概述

1.1 CAR 构件技术

Elastos 是一种具有灵活内核的构件化嵌入式操作系统。CAR 是 Elastos 上面向构件编程的编程模型,它表现为一组编程规范,它规定了一组构件间相互调用的标准,包括构件、类、对象、接口等定义与访问构件对象的规定[3]。CAR 构件能够自描述,能够在运行时动态链接。

对 WebKit 进行本地扩展时,本文将重点运用以下一些 CAR 的特性和机制:

1)元数据是描述数据的数据,元数据是一种数据,是对数据的抽象,它主要描述了数据的类型信息。CAR 把模块信息、接口信息、类信息等作为描述构件的元数据。这些信息由 CAR 文件编译而来,是 CAR 文件的二进制表述。

2)反射机制是 CAR 构件系统具备的特征之一,是 CAR 构件实现其动态性的关键。基于 CAR 构件的程序在运行时能够动态加载、探知 CAR 构件,获取 CAR 构件的信息,包括模块信息、类信息、接口信息、方法信息等。

3)CAR 还具有回调机制,简单的说它有一种接收器对象,客户程序可将自己实现的事件处理函数(回调函数)向接收器进行注册,构件对象在条件成熟时激发事件,如果客户注册了自己的回调接口方法,那么就会被调用,否则就调用接收器默认实现的回调接口方法。

CAR 构件采用 C/C++ 编写,携带元数据信息,元数据通过反射机制参与构件组装计算,生成的代码直接以目标平台的二进制代码运行,编译结果采用".dll"为规范的文件结尾格式,能够达到 C/C++ 的运行效率。

1.2 WebKit

在 Elastos 平台上,WebKit 主要包含一个网页引擎 WebCore、一个脚本引擎 JavaScriptCore 和 WebKit 外壳部分。WebCore 主要包含页面管理与资源加载、DOM 解析、编辑与排版以及脚本引擎与 DOM 树的绑定等部分。JavaScriptCore

收稿日期:;**修回日期**:。 **基金项目**:国家"863"计划项目(2001AA113400);国家移动通信产品研究开发专项项目,财政部(财建[2005]182 号)、信息产业部(信部请函[2005]297 号)。

作者简介:蒋章概(1985 -),男,浙江温州人,硕士研究生,主要研究方向:嵌入式操作系统、系统软件支撑技术; 陈榕(1957 -),男,北京人,教授,博士生导师,主要研究方向:嵌入式系统、构件技术。

主要包含 JavaScript 解析引擎、JavaScript 基础 API 的实现、与 DOM 绑定的部分以及 KDE 的 C++模板库 WTF 等。

2 扩展 WebKit 使其与 CAR 构件互操作模型

该模型是基于 Elastos 操作系统平台建立，如图 1 所示。WebKit 向上对基于它的应用程序提供支持，WebKit 外壳负责发起资源加载请求并接收外部事件和输入与用户进行交互，作为网页引擎的 WebCore 使用内部的 Page 模块对 WebKit 外壳的请求进行响应，使用 Loader 模块对资源进行加载和缓存，最后使用 HTML DOM、XML DOM 等对加载的资源进行解析，并将生成的 DOM 树与脚本引擎 JavaScriptCore 进行动态绑定。CAR 构件运行于 Elastos 平台之上，提供本地代码模块构件化的服务，它可以通过系统调用接口访问 Elastos 操作系统。符合一定规范的 CAR 构件对 WebKit 进行扩展，以支持基于 WebKit 的应用程序的本地调用，并通过正调和回调机制进行互相操作。

本文研究方法是通过自定义 JavaScript 对象并将其本地化，在运行时调用本地代码。通过比较，我们发现 JavaScriptCore 内的 JavaScript 类和方法信息与 CAR 构件内部的类和方法信息形式上具有一定程度的相似性，我们可以对它们作一一的映射，将 JavaScript 的类和方法一一对应到本地代码 CAR 构件的类和方法，以这种方式实现自定义 JavaScript 对象的本地化。

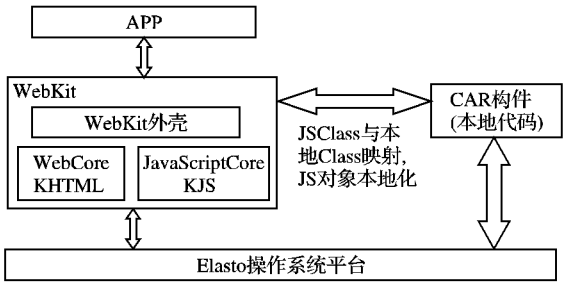

图 1 扩展 WebKit 使其与 CAR 构件互操作的模型

CAR 构件本身分两种形态：一种是静态的 Aspect 形态，生命期和运行环境由调用者管理，它所具有的行为由运行时环境决定；另一种是 Applet 形态，它自身具有运行环境，这个环境可以是 Context 构件（一种 Aspect），它有自己的线程模型。我们采用第一种形态对 CAR 构件进行引用，由 WebKit 负责加载并管理它的生命期和运行环境。

3 具体设计和实现

3.1 初始化 WebKit 加载 CAR 构件的上下文环境

WebKit 在加载资源并进行解析的时候，需要找到一个合适的时机去加载 CAR 构件。JavaScriptCore 与 DOM 树是动态绑定的，那么 JavaScript 上下文环境是与 DOM 树相关的，而 DOM 树是在 WebCore 的 HTML DOM 模块解析 HTML 文档时动态生成的，所以我们在 HTML DOM 使用 HTMLTokenizer 类对文档字符串进行解析的时候，如果遇到 Script 标签，那么就找到文档的框架 Frame 并开始执行 CAR 构件的加载，而 Frame 会将这个过程交给框架加载器 FrameLoader 来完成。为了保持网页引擎内核 WebCore 的相对的完整性和稳定性，我们把 CAR 的加载的具体实现放置在 WebKit 的外壳中进行。WebKit 外壳中的 WebFrame 继承于 WebCore 中的框架加载器 FrameLoaderClient，所以由它来完成 CAR 构件的具体加载。

在 WebFrame 中，我们做的事情主要有以下 2 件：

1）读取配置文件以获取所有要加载的 CAR 构件名称。配置文件中的构件名以".dll"结尾，使用";"分隔各构件名。

2）获取 JavaScript 全局的上下文环境和全局文档对象，并根据它们创建 CAR 构件加载器 CarLoader，最后加载器根据 1）中读取的 CAR 构件名称并对它们进行一一加载。

3.2 CAR 构件中的类与自定义的 JavaScript 类的一一映射

从参考文献[2]中我们可以知道 JavaScriptCore 提供 JSClassCreate 方法来创建 JavaScript 类，为了便于后文论述，我们根据参考文献[2]和开源的 WebKit 代码中的 JavaScriptCore 部分先列出以下一些关键定义：

```
JSClassRef JSClassCreate( const JSClassDefinition * definition)    (1)
```

其中 JSClassDefinition 的定义如下：

```
typedef struct{
  …
  const char  * className;
  const JSStaticFunction  * staticFunctions;
  JSObjectSetPropertyCallback setProperty;
  JSObjectCallAsConstructorCallback callAsConstructor;
  …
} JSClassDefinition;                                                (2)
```

其中 JSStaticFunction 的定义如下：

```
typedef struct{
  const char  * const name;
  JSObjectCallAsFunctionCallback callAsFunction;
  JSPropertyAttributes attributes;
} JSStaticFunction;                                                 (3)
```

其中 JSObjectCallAsFunctionCallback 的定义如下：

```
typedef JSValueRef ( * JSObjectCallAsFunctionCallback )
(JSContextRef ctx, JSObjectRef function, JSObjectRef thisObject, size
_t argumentCount, const JSValueRef arguments[ ], JSValueRef *
exception);                                                         (4)
```

另外再列出一个重要的结构体的定义：

```
struct StaticFunctionEntry {
  StaticFunctionEntry ( JSObjectCallAsFunctionCallback _
  callAsFunction,
  JSPropertyAttributes _ attributes ): callAsFunction ( _
  callAsFunction), attributes( _attributes)
  {}
  JSObjectCallAsFunctionCallback callAsFunction;
  JSPropertyAttributes attributes;
  };                                                                (5)
```

1）JavaScriptCore 的改造。

这里我们先改造一下 JavaScriptCore，主要是为下文的工作做铺垫。

a）将上述定义（3）和（5）的结构体 JSStaticFunction 和 StaticFunctionEntry 进行修改，都增加一个 void 指针类型的成员变量 vPrivate。

b）修改 JavaScriptCore 中的 JSCallbackFunction 类，增加一个 void 指针类型的私有成员变量 vPrivate 和相应的获取该成员变量的公有方法。

c）修改与 a，b 相关的实现部分，使 JavaScriptCore 保持正确性。

2）CAR 构件的元数据信息反射和 JSClassDefinition 的初始化。

在 CAR 构件加载器 CarLoader 加载 CAR 构件时，利用 1.1 节中提到的 CAR 的反射机制，遍历所有 CAR 构件，从元数

据(参看 1.1 节)信息中获取每一模块信息,通过每一个模块信息获取模块中的每一个类信息,从类信息中获取类名将其赋值给上述定义(2)的 JSClassDefinition 的成员变量 className,通过每一个类信息获取类中每一个方法信息,最后通过方法信息获取方法名并将其赋值给 JSStaticFunction 的成员变量 vPrivate。

3)创建自定义 JavaScript 对象。

自定义 JavaScript 对象的创建主要有两种方式。

a)注册全局对象的方式。

根据 2)中初始化好的 JSClassDefinition,使用定义(1)的方法 JSClassCreate 创建自定义的 JavaScript 类,根据 CAR 反射得到的类信息创建对象,然后根据自定义的 JavaScript 类和 CAR 的类信息和对象,使用方法 JSObjectMake 来创建 JavaScript 对象,最后将创建的自定对象设置为全局文档对象的属性,这样自定义的对象就可在全局上下文环境中使用。

b)以 new 方式构造对象。

这种方式指的是当 JavaScriptCore 解析 JavaScript 时遇到 new 方法时就构造自定义的对象的方式。创建的过程和方法与注册全局对象的方式类似,区别的地方在于创建的过程是在构造的回调函数中完成的,即将 JSClassDefinition 中的回调函数指针 callAsFunction 指向创建自定义对象的具体的函数实现部分。

3.3 JavaScript 正向调用 CAR 构件

JavaScript 对象在调用自身的方法时,会对定义(4)的函数指针指向的回调函数进行调用。我们知道回调函数的 thisObject 参数正是定义的 JavaScript 对象本身,我们可以使用 JSObjectGetPrivate 方法取出 CAR 构件的类信息和对象。此外,因为在 3.2 节中使用 JSClassCreate 创建 JavaScript 类时,会将 JSStaticFunction 中的成员变量 vPrivate 所携带的 CAR 构件的类的方法名进行传递,赋值给 JSCallbackFunction 和 StaticFunctionEntry 的成员变量 vPrivate。所以我们此时将回调函数的 function 转化为 JSCallbackFunction 类型并从中取出成员变量即 CAR 构件的类的方法名。这样我们就可以根据 CAR 构件的类信息和方法的名字得到方法信息,此时如果 CAR 构件的类中的方法如果无参数,我们就可以使用得到的 CAR 构件的类的对象和方法信息对方法进行调用。

如果 CAR 构件的类中的方法是有参数的,那么我们需要对 JavaScript 和 CAR 的参数类型进行相互转换,因为它们定义的基本数据类型是不一样的。定义(4)的回调函数的参数 argumentCount 和 arguments 是 JavaScript 方法输入的参数个数和参数值,我们需要做的是将 JavaScript 的参数类型转换为 CAR 的 in 参数(即输入参数)类型,将 CAR 的 out 参数(即输出参数)类型转换为 JavaScript 的参数类型。下面我们简单列出和说明目前已经实现的参数类型转换,如表 1 和表 2 所示。

表 1 JavaScript 参数类型到 CAR 参数类型的转换

方法名	方法说明
jsVal2Int16	将 JSValue 转换为 CAR 的 Int16 参数类型
jsVal2Int32	将 JSValue 转换为 CAR 的 Int32 参数类型
jsVal2Int64	将 JSValue 转换为 CAR 的 Int64 参数类型
jsVal2IntByte	将 JSValue 转换为 CAR 的 Byte 参数类型
jsVal2IntFloat	将 JSValue 转换为 CAR 的 Float 参数类型
jsVal2IntDouble	将 JSValue 转换为 CAR 的 Double 参数类型
jsVal2IntAChar	将 JSValue 转换为 CAR 的 AChar 参数类型
jsVal2IntWChar	将 JSValue 转换为 CAR 的 WChar 参数类型
jsVal2IntAString	将 JSValue 转换为 CAR 的 AString 参数类型
jsVal2IntWString	将 JSValue 转换为 CAR 的 WString 参数类型
jsVal2IntBoolean	将 JSValue 转换为 CAR 的 Boolean 参数类型

表 2 CAR 参数类型到 JavaScript 参数类型的转换

方法名	说明
JSValueMakeNumber	根据不同的参数,分别将 CAR 的 Int16、Int32、Int64、Float、Double、Enum 类型转换为 JSValue
JSValueMakeBoolean	将 CAR 的 Boolean 类型转换为 JSValue
JSValueMakeString	根据不同的参数,分别将 CAR 的 Byte、AChar、WChar、AStringBuf、WStringBuf 转换为 JSValue

3.4 CAR 构件回调 JavaScript

在 JavaScript 中,我们可以利用本文 1.1 节中提到的 CAR 的回调机制向 CAR 构件的代理注册回调函数,在 CAR 构件对象激发事件时,回调 JavaScript 函数从而实现 CAR 构件对 JavaScript 的操纵。具体的实现过程如下。

1)设置自定义 JavaScript 类的属性。

需要在 3.2 节中 JSClassDefinition 的初始化的时候赋值函数指针 setProperty 并实现它所指向的回调函数。判断所设置的属性是否为 CAR 构件的回调事件,如果是且所要注册的 JavaScript 函数不为空,则进入下一步。

2)代理服务创建和 JavaScript 回调函数注册。

使用 JSObjectGetPrivate 方法获取 JavaScript 对象中的 CAR 构件的类信息,并根据 1)中设置的属性名获取 CAR 构件的回调事件的信息,根据回调事件的信息创建代理服务,并将向其注册 JavaScript 回调函数。

3)函数调用和参数转换。

当 CAR 构件对象触发事件时,代理会调用所注册的 JavaScript 函数。如果是有参数的情况下,在这里我们同样也需要将 CAR 构件的参数类型转换为 JavaScript 的参数类型,转换的方式与 3.3 节中表 2 类似。

4 应用

在 Windows XP 上建立的 Elastos 操作系统中间件运行平台上,对本文扩展策略进行了验证,对 Elastos 平台上基于 HTML 的 Widgets 的本地调用进行了支持并兼容部分 BAE Widgets。另外,还在此平台上基于该技术开发了 Kortide Office Domain,它是一种集即时聊天、资源共享、Widgets 运行和传播等功能于一身的 Web 应用程序。

5 结语

本文提出了基于 CAR 构件的 WebKit 的本地扩展策略,设计和实现了 WebKit 与 CAR 构件互操作的模型,简单阐述了基于此技术进行开发的一些应用程序。WebKit 与 CAR 构件互操作的实现能够支撑更加强大的基于 WebKit 的应用,对于受限于 WebKit 且计算性能尤为重要的一些应用程序来说是具有重要作用的。这种模型使得基于 WebKit 的应用程序一方面可以很好的利用 WebKit 的排版能力加快 UI 的设计和开发,另一方面与桌面应用程序具有同等的本地调用的功能,提供更好的“富客户端”的用户体验,对于基于此研究的 WebOS 的设计和实现有着积极的意义。

参考文献:

[1] YEE B, SEHR D, DARDYK G, *et al*. Native client: A sandbox for portable, Untrusted x86 native code[EB/OL]. [2009 - 04 - 25]. http://nativeclient.googlecode.com/svn/data/docs_tarball/nacl/googleclient/native_client/documentation/nacl_paper.pdf.

[2] JavaScriptCore framework reference[EB/OL]. [2009 - 04 - 28]. http://developer.apple.com/documentation/Carbon/Reference/WebKit_JavaScriptCore_Ref/index.html #//apple_ref/doc/uid/TP40004754.

[3] Elastos 资料大全[Z]. 上海科泰世纪科技有限公司, 2008.

[4] The WebKit open source project[EB/OL]. [2009 - 04 - 28]. http://webkit.org/.