Aplicativos móveis

FirefoxOS

A Mozilla disponibilizará seu sistema para tablets e smartphones no Brasil em 2013, gerando oportunidades para desenvolvedores. Neste artigo veremos os primeiros passos para criação de novos aplicativos.

por Alessandro de Oliveira Faria (Cabelo)

A Mozilla entrará no mercado de sistema móveis e competirá diretamente com Android, iOS e Windows Phone. O FirefoxOS, também conhecido pelo codinome B2G (*Boot to Gecko*), é totalmente baseado na Internet, então podemos deduzir que o esforço para portar aplicativos não exigirá "magia negra" em programação. O conceito é muito similar ao Chrome OS para celulares, sendo mais preciso – o sistema trabalha exatamente a partir de aplicativos online.

Um ponto muito importante é que o Brasil será o primeiro país a receber esta tecnologia de código aberto e totalmente isenta de royalties em rede móvel Telefonica Digital (operadora Vivo). A plataforma da B2G da Mozilla é baseada na tecnologia HTML5, ou seja, comprovadamente online. Não precisamos de grandes estudos para entender que smartphones com esta tecnologia serão mais baratos e deverão levar a uma difusão mercadológica na América Latina. "De acordo com Carlos Domingo, diretor de desenvolvimento de produtos e inovação da Telefonica Digital, o potencial de um aparelho B2G seria o de custar dez vezes menos que um iPhone. Ou seja: no Brasil, sairia por volta de R$ 180" [1].

O FirefoxOS é uma plataforma online de padrão aberto que proporciona o controle completo dos aplicativos, muito similar ao funcionamento de um navegador. Conta também com todos os recursos do aparelho, como mensagens, chamadas, buscas online e jogos que serão desenvolvidos em HTML5 e executados por meio do navegador. E claro, de acordo com a Mozilla, alguns recursos irão funcionar mesmo sem acesso à Internet. Acredito que esta tecnologia mudará conceitos, pois atualmente a melhor maneira de navegar na Internet com um dispositivo móvel é através de aplicativos, portanto, teremos Internet no bolso de todos.

O B2G é baseado no kernel Linux, com modificações do Android para suportar os dispositivos móveis baseados na arquitetura ARM. No lugar dos daemons/serviços entram em ação um runtime do Gecko. Este runtime nada mais é do que um componente de renderização de HTML do Firefox (ou seja, nada de aplicativos nativos). A utilização da Mozilla Web APIs garante a inclusão de padrões de APIs JavaScript para o acesso a recursos dos dispositivos móveis, como GPS, acelerômetro, SMS e câmera, suporte a multitoque e a tecnologia NFC (WebNFC).

Mão na massa, ao código fonte

Um fator muito importante é que diferentemente de qualquer SDK móvel (Android, iOS, Windows Phone e outros), todo o ambiente de desenvolvimento foi construído a partir do código-fonte. Isto demonstra que o projeto é todo em código aberto, conforme mencionado em diversas notícias espalhadas pela Internet.

Como no Android, precisamos de um emulador para executar a interface gráfica (no FirefoxOS denominado *GAIA*) e testar os aplicativos, uma vez que ainda não possuímos celulares disponíveis no mercado. Embora seja possível compilar o FirefoxOS para alguns modelos de celulares com Android, decidi embasar este artigo somente no emulador para atender a todos os leitores, independentemente do aparelho adquirido.

GAIA, Emulador e Desktop client

Para efeito de sincronismo e nomenclatura, GAIA é a interface do FirefoxOS, ou seja, toda a interface visual é desenhada pela GAIA, desde o bloqueio de tela até

discador e aplicativos. A interface é toda escrita em HTML5, CSS e JavaScript, projetada para executar sobre o FirefoxOS através do padrão WebAPI.

O B2G *desktop client* é um aplicativo que permite a execução e testes de aplicativos na interface GAIA e online no ambiente B2G. Não devemos entender este aplicativo como um emulador, por consequência não é um substituto para testes em hardware real.

O B2G emulador é, na verdade, a principal ferramenta para desenvolvedores que desejam testar o ambiente desta nova tecnologia. Podemos praticamente gerar dois modelos de binários, o emulador x86 e o ARM. Tanto a versão desktop quanto a versão emulador são baseadas no *qemu*.

Requisitos

Para compilar o ambiente de desenvolvimento ou desktop, além de muito café, também precisamos instalar "algumas" dependências, ou seja, os pacotes necessários para a compilação. Esta tarefa envolve desde a instalação do `gcc` até as bibliotecas básicas para compilação e execução do B2G. As **listagens de 1** a **5** contêm as instruções de instalação de dependências nas principais distribuições.

Após a instalação de todos os requisitos, o próximo passo é a compilação e instalação do B2G desktop client ou emulator.

Compilação e instalação do B2G desktop client

Antes de iniciar o download, crie a pasta `b2g` e efetue o download dos respectivos códigos-fontes utilizando as ferramentas de controle de versão Github e Mercurial.

```
$ mkdir b2g
$ cd b2g
$ hg clone http://hg.mozilla.org/
➥ mozilla-central src
$ git clone https://github.com/
➥ mozilla-b2g/gaia gaia
```

Após o download, entre na pasta `src` recém-criada, edite o arquivo `.mozconfig` e o modifique com o conteúdo a seguir:

```
mk_add_options MOZ_OBJDIR=../build
mk_add_options MOZ_MAKE_FLAGS="
➥ -j9 -s"

ac_add_options
➥ --enable-application=b2g
ac_add_options
➥ --disable-libjpeg-turbo

# This option is required if you
➥ want to be able to run Gaia's
➥ tests
ac_add_options --enable-tests

# turn on mozTelephony/mozSms
➥ interfaces
# Only turn this line on if you
➥ actually have a dev phone
# you want to forward to. If you
➥ get crashes at startup,
# make sure this line is
➥ commented.
# ac_add_options -enable-b2g-ril
```

Em seguida, inciaremos a compilação com o tradicional comando `make`, conforme o exemplo abaixo. Durante esta operação, sugiro ao leitor uma pausa para um café...

```
$ make -f client.mk build
```

Ao término da compilação, basta entrar na pasta `GAIA`, compilá-la com

Listagem 1: Requisitos para o Ubuntu

```
01 sudo apt-get install mercurial g++ make autoconf2.13 yasm libgtk2.0-dev libglib2.0-dev libdbus-1-dev
➥ libdbus-glib-1-dev libasound2-dev libcurl4-openssl-dev libiw-dev libxt-dev mesa-common-dev
```

Listagem 2: Requisitos para o Debian

```
02 sudo aptitude install mercurial libasound2-dev libcurl4-openssl-dev libnotify-dev libxt-dev libiw-dev
➥ libidl-dev mesa-common-dev autoconf2.13 yasm libgtk2.0-dev libdbus-1-dev libdbus-glib-1-dev
```

Listagem 3: Requisitos para o RedHat Enterprise/CentOS/Fedora

```
01 sudo yum groupinstall 'Development Tools' 'Development Libraries' 'GNOME Software Development'
02 sudo yum install mercurial autoconf213 glibc-static libstdc++-static yasm wireless-tools-devel
➥ mesa-libGL-devel alsa-lib-devel libXt-devel
```

Listagem 4: Requisitos para o openSUSE/SUSE

```
01 sudo zypper install make cvs mercurial zip gcc-c++ gtk2-devel xorg-x11-libXt-devel libidl-devel
➥ freetype2-devel fontconfig-devel pkg-config dbus-1-glib-devel mesa-devel libcurl-devel libnotify-devel
➥ alsa-devel autoconf213 libiw-devel yasm
```

Listagem 5: Requisitos para o Arch Linux

```
01 sudo pacman -Syu base-devel zip unzip freetype2 fontconfig pkg-config gtk2 dbus-glib iw libidl2 python2
➥ mercurial alsa-lib curl libnotify libxt mesa autoconf2.13 yasm wireless_tools
```

o comando `make` e logo em seguida executar o binário `b2g`, conforme as instruções:

```
$ cd ../gaia/
$ make
$ ../build/dist/bin/b2g
➥ -profile profile
```

Dicas para o B2G desktop client

Caso ocorra algum problema de renderização, basta inserir o conteúdo abaixo no arquivo: `[PATH]Gaia/profile/prefs.js`

```
user_pref("layers.acceleration.
➥ disabled", true);
```

A resolução do emulador pode ser definida com parâmetros `--screen`, e deve ser informada a resolução no formato `<largura>x<altura>@<dpi>` exemplo:

```
$ ../build/dist/bin/b2g -profile
➥ profile --screen=320x240@160
```

Se tudo estiver funcionando corretamente, teremos a seguinte tela do B2G desktop client. Com este módulo é possível testar a loja de aplicativos e recursos de software desenvolvidos (**figura 1**).

Instalação do B2G emulator a partir do código-fonte

A principal diferença entre o B2G emulator e o desktop client é a capacidade de interagir com o dispositivos virtual. O emulador pode ser compilado para x86 ou ARM. Claro, se optarmos pela primeira opção, o desempenho será superior em computadores pessoais. O emulador permite conexão e acesso ao Shell, assim como configuração de ambientes e tudo que um desenvolvedor precisa para criar aplicativos nesta plataforma.

Se uma pasta não foi criada conforme as instruções anteriores, seja pelo motivo da não instalação do B2G desktop client, seja por outro motivo qualquer, crie então uma pasta e efetue o download do código-fonte utilizando o comando a seguir da ferramenta Git; logo após, execute o comando `./config.sh emulator-x86` para efetuar a compilação para a plataforma x86; para finalizar, execute o comando `./build.sh` (o usuário deve responder a algumas perguntas como nome e email).

```
$ mkdir firefoxOS
$ cd firefoxOS/
$ git clone https://github.com/
➥ mozilla-b2g/B2G.git
$ cd B2G
$ ./config.sh emulator-x86
$ ./build.sh
```

Depois de algumas horas de download e compilação, e se tudo

Figura 1 Na tela do B2G desktop client é possível testar a loja de aplicativos e recursos desenvolvidos.

Figura 2 Imagem de instalação do B2G emulator a partir do código-fonte funcionando corretamente.

funcionou corretamente, basta executar o comando `./run-emulator.sh` para visualizar algo como a **figura 2**. Se algum erro ocorreu, sem sombra de dúvida foi por conta de alguma dependência do pacote de desenvolvimento que não foi atendida. Ressalto que o ambiente utilizado foi a plataforma openSUSE 12.1 64 bits.

A memória do emulador pode ser alterada com o parâmetro `-memory`, assim como a visualização – por padrão, o emulador utiliza a resolução 320x480, porém, se desejarmos modificá-la, basta editar o script `run-emulator.sh` e adicionar o parâmetro `--skin` seguido do layout desejado (conforme relação abaixo).

- HVGA (320x480)
- QVGA (240x320)
- WQVGA (240x400)
- WQVGA432 (240x432)
- WSVGA (1024x600)
- WVGA800 (480x800)
- WVGA854 (480x854)
- WXGA720 (1280x720)
- WXGA800 (1280x800)

Insira na variável `PATH` a localização do binário `adb` (*Android Debug Bridge*) pois, conforme mencionado no início do texto, parte do FirefoxOS foi extraída do Android (camada de kernel). Em seguida, teste a comunicação com o emulador utilizando o aplicativo *adb*, conforme o exemplo:

```
$  adb start-server
* daemon not running. starting it
➥ now on port 5037 *
* daemon started successfully *
$ adb devices
List of devices attached
emulator-5554   device
$ PATH=$PATH:[SUA PASTA]/b2g/B2G/
➥ out/host/linux-x86/bin
```

Se por algum motivo a conectividade do dispositivo virtual não estiver funcionando (sem acesso à Internet), basta executar o comando no Shell:

```
$ adb shell setprop net.dns1
➥ 10.0.2.3
```

Figura 3 Aviso de instalação do aplicativo MozillaBall.

Figura 4 Para remover o MozillaBall, pressione o ícone por alguns segundos.

Desenvolva aplicativos para FirefoxOS

Agora que o seu ambiente está instalado e configurado com sucesso, o próximo passo é entender como desenvolver um aplicativo para esta plataforma de software. Mas, antes, vamos conhecer um pouco a ferramenta *Marionette*.

Baseado no protocolo JSON, o Marionette é um módulo que trabalha como um servidor no dispositivo virtual. Seu principal objetivo é obter acesso à chamada WebAPI, tornando possível acessar o estado de diversos recursos do sistema operacional. Em outras palavras, podemos testar o framework do sistema FirefoxOS apenas digitando as chamadas desejadas. Para acessar este recurso, utilize o comando conforme o exemplo abaixo:

```
$ adb forward tcp:2828 tcp:2828
```

Agora devemos instalar o Marionette client a partir do código-fonte, e em seguida entrar na pasta `marionette_client` para configurar a localização do Python e depois ativar o Marionette client na pasta `marionette_venv`, conforme o exemplo:

```
$ git clone https://github.com/
➥ mozilla/marionette_client/
cd marionette_client/marionette/
sh venv_test.sh
➥ /usr/bin/python2.7
cd ../marionette_venv
$. bin/activate
(marionette_venv)$
```

Agora executaremos o interpretador Python e invocaremos uma seção Marionette interativa:

```
(marionette_venv)$ python
Python 2.7.2 (default, Aug 19
```

```
➥ 2011, 20:41:43) [GCC] on linux2
Type "help", "copyright",
➥ "credits" or "license" for
➥ more information.
>>> from marionette import
➥ Marionette
>>> marionette =
➥ Marionette('localhost', 2828)
>>> marionette.start_session()
u'5-b2g'
```

Em seguida podemos executar diversos comandos para testar o framework de telefonia, SMS, bateria, GPS e outros.

```
>>> marionette.execute_script
➥ ("return navigator.mozTelephony;")
{u'oncallschanged': None,
➥ u'calls': [], u'muted': False,
➥ u'onincoming': None,
➥ u'speakerEnabled': False,
➥ u'active': None}

>>> marionette.execute_script
➥ ("return navigator.battery;")
{u'onlevelchange': None, u'level':
➥ 0.5, u'dischargingTime': None,
➥ u'onchargingchange': None,
➥ u'ondischargingtimechange':
➥ None, u'onchargingtimechange':
➥ None, u'chargingTime': None
```

```
>>> marionette.execute_script
➥ ("return navigator.battery.
➥ level;")
0.5

>>> marionette.execute_
➥ script("return navigator.
➥ geolocation;")
{}

>>> marionette.execute_
➥ script("return navigator.mozSms;")
{u'onreceived': None,
➥ u'ondelivered': None, u'onsent':
➥ None}
```

Para finalizar a seção do Marionete, utilize a função `quit()` seguida do comando `deactivate`.

```
>>> quit()
$ deactivate
```

Como executar um HelloWord

Para desenvolver um programa para o FirefoxOS, basta obter sólidos conhecimentos sobre HTML5 e CSS JavaScript (ou seja, Internet). Então não pretendo escrever como utilizar o HTML5 neste texto, partiremos direto para a ação. Pegaremos como exemplo o aplicativo *MozillaBall*, que tem como finalidade exibir uma bola "ricocheteando" na tela [2]. Primeiramente, faça o download do aplicativo e separe o projeto em uma pasta visível ao seu Apache:

```
# cp -r /home/cabelo/mozilla-
➥ openwebapps-0c6856f/examples/
➥ mozillaball /srv/www/htdocs/
```

Agora criaremos o arquivo `install.html`, que será responsável pela instalação do aplicativo em nosso device virtual. O segredo desta operação está no arquivo com estrutura JSON com o nome `manifest.webapp` que apresenta as informações necessárias para o FirefoxOS efetuar a instalação com sucesso.

```
<html>
    <head>
      <title></title>
      <meta content="">
      <style></style>
    </head>
<script>
var manifest_url = "http://
➥ MINHA-MAQUINA/mozillaball/
➥ manifest.webapp";
</script>
    <body>
    <a onclick="navigator.mozApps.
➥ install(manifest_url); return
➥ false;" href="#">Install</a>
    </body>
</html>
```

Agora entre no emulador, execute o Firefox e teremos a nossa primeira página de instalação de aplicativos. Se tudo estiver funcionando corretamente, clique no link *Install* e receberá uma mensagem de aviso referente à instalação do aplicativo MozillaBall (**figura 3**). Então clique no botão *YES* e o ícone do aplicativo aparecerá na tela.

Agora basta clicar no ícone e o programa entrará em execução. Pronto! Para removê-lo, basta pressionar o ícone por alguns segundos e um "X" aparecerá sobre o aplicativo (**figura 4**). ■

O autor

Alessandro de Oliveira Faria (Cabelo) é sócio-proprietário da NETi TECNOLOGIA, fundada em Junho de 1996 (*http://www.netitec.com.br*) e especializada em desenvolvimento de software e soluções biométricas. Consultor Biométrico na tecnologia de reconhecimento facial, atuando na área de tecnologia desde 1986. Leva o Linux a sério desde 1998 com desenvolvimento de soluções open-source, é membro colaborador da comunidade Viva O Linux e mantenedor da biblioteca open-source de vídeo captura, entre outros projetos.

Mais informações

[1] Potencial do B2G http://reviews.cnet.com/8301-13970_7-57385616-78/telefonica-mozillaphone-is-ten-times-cheaper-than-an-iphone/

[2] Aplicativo Mozilla Ball: https://github.com/mozilla/openwebapps/tree/develop/examples/mozillaball

Gostou do artigo?

Queremos ouvir sua opinião. Fale conosco em cartas@linuxmagazine.com.br

Este artigo no nosso site: http://lnm.com.br/article/7783