如何将C/C++应用转化为JS

——Emscripten: llvm javascript编译器

Emscripten

- Emscripten 是 Mozilla 的 Alon Zakai 开发的一个独特 LLVM 后端，可以将 LLVM 中间码编译成 JavaScript，Emscripten 并非通常的 LLVM 后端，本身使用 JavaScript 写成。
- 它可以将任何通过 LLVM 前端(比如 C/C++ Clang)生成的 LLVMIR 中间码编译成 JavaScript。 LLVM可以将代码编译成比特码，然后Emscripten将其翻译成JavaScript。
- 很多大型的项目已经可以使用 Emscripten 转换为 JavaScript 了，比如 Python, Ruby, Lua 和 Doom。
- LLVM可以将代码编译成比特码，然后Emscripten将其翻译成JavaScript。

LLVM

- LLVM 是 Low Level Virtual Machine 的简称，这个库提供了与编译器相关的支持，可以作为多种语言编译器的后台来使用。能够进行程序语言的编译期优化、链接优化、在线编译优化、代码生成，是一个模块化和可重复使用的编译器和工具技术的集合。
- 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成了Apple的官方支持的编译器。Apple已经将它用在OpenCL的流水线优化，Xcode已经能使用llvm-gcc编译代码。

准备工作

- Install Java
  - sudo apt-get install openjdk-7-jdk openjdk-7-jre-headless
- Install node.js

```
cd /tmp
 wget http://nodejs.org/dist/v0.10.12/node-v0.10.12.tar.gz
tar -xzvf node-v0.10.12.tar.gz
cd node-v0.10.12
./configure
make
sudo make install
```

准备工作

- Install LLVM 3.2

```
sudo mkdir /llvm-3.2
sudo chmod 777 /llvm-3.2
cd /llvm-3.2
svn co http://llvm.org/svn/llvm-project/llvm/tags/RELEASE_32/final/ llvm
cd llvm/tools/
svn co http://llvm.org/svn/llvm-project/cfe/tags/RELEASE_32/final clang
cd ../..
cd llvm/projects
svn co http://llvm.org/svn/llvm-project/compiler-rt/tags/RELEASE_32/final compiler-rt
cd ../..
mkdir build
cd build
../llvm/configure --prefix=/usr/local/llvm-3.2 --enable-cxx11 --enable-optimized
CXX=/usr/bin/g++-4.7 CC=/usr/bin/gcc-4.7
make -j 8
sudo make install
```

准备工作

- Download emscripten

```
sudo mkdir /emscripten
sudo chmod 777 /emscripten
git clone git://github.com/kripken/emscripten.git /emscripten
```

- Install Python 2.7.3
  - sudo ln -s /usr/bin/python /usr/bin/python2
- Install gcc4.7/g++4.7
- Add /emscripten to $PATH
  - export PATH=$PATH:/emscripten

使用步骤

- 测试环境
  - clang tests/hello_world.cpp ./a.out
  - node tests/hello_world.js
- 设置Emscripten
  - ./emcc
  - 编辑~/.emscripten，设置LLVM_ROOT和NODE_JS
- 运行**Emscripten**
  - ./emcc tests/hello_world.cpp
  - node a.out.js

使用技巧

- 生成HTML
  - ./emcc tests/hello_world_sdl.cpp -o hello.html
- 使用文件
  - ./emcc tests/hello_world_file.cpp -o hello.html --preload-file tests/hello_world_file.txt
- 优化
  - ./emcc -O1 tests/hello_world.cpp
- 测试
  - python tests/runner.py test_hello_world

需要注意的几点

- 尽量减少使用structure，因为这将会占用更多的内存
- 尽量用if/else代替switch，用for代替while
- 不要使用64bit的Int类型
- 不要使用多线程
- 不能直接操作硬盘上的文件系统
- 编译的时候尽量使用静态编译，这样的话依赖的库也会编译成llvm代码，这样可以一块转换成js代码

		Search First Time	JS vs Native(ms)		
pinyin	candidates	compare times	Emscripten Opted (1,000 times)	Native (1,000 times)	IME Opt Nightly / Native
a	7	36	86	8	10.75
b	674	16086	843	380	2.163157895
c	948	24828	1278	620	1.982258065
d	736	17614	1038	440	2.065909091
e	85	1095	84	20	7.2
f	473	10497	587	200	2.66
g	613	14289	887	320	2.34375
h	814	21237	1097	520	1.957692308
z	1431	37846	1945	1120	1.725892857
ba	43	5147	61	20	9.25
da	31	5440	155	20	7.1
fa	23	3089	36	20	6.65
ga	11	4109	85	0	
ha	4	6237	31	0	

总结

- 代码越简单，转化后差别越大。
- 部分文件转换后不能直接使用，需要进行修改。
- 转换后的代码经过压缩和原生代码编译出来的二进制代码大小差别不大。
- 转换后执行效率降低，但基本能达到原生应用的一半，而且随着js引擎的优化和硬件加速比如asm.js,webgl，和本地执行代码差距会越来越小。

Demos

- BananaBread - C++ 3D游戏引擎移植后使用 JavaScript + WebGL + HTML。
- 一些2D游戏，包括使用SDL。
- OpenGL ES 2.0渲染的WebGL。
- 文本朗读。
- PNG，XML， SQLite，PDF，OpenJPEG， zlib， LZMA移植到JS。
- Python, Ruby, Lua等的语言的转换。

参考

- https://github.com/kripken/emscripten
- https://github.com/kripken/emscripten/wiki
- https://earthserver.com/Setting_up_emscripten_development_environment_on_Linux