

翻译：dozb淺淺淺淺 [英文版](#)

SpiderMonkey

首先你要下载 [SpiderMonkey](#) 的最新版本，下载包是以源代码形式分发的。这就需要你自己编译脚本引擎。对于 Visual C++ 用户可以在 src-目录找到工程文件，编译结果是叫做 'js32.dll' 的动态库。

也可以用 SpiderMonkey 在 Macintosh 和 Unix 平台上，读 `ReadMe.html` 来学习在其他平台如何编译。

第 1 步 - 创建 JavaScript 运行时环境 (runtime)

JavaScript 运行时环境是调用 `JS_NewRuntime` 来初始化。它分配运行时环境的所需内存。你要指定所分配的字节数，超过这个大小碎片收集器将运行。

```
JSRuntime *rt = JS_NewRuntime(1000000L);
if ( rt == NULL )
{
    // Do some error reporting
}
```

上下文指定脚本栈的大小，私有内存的字节数被分配给指定脚本的执行栈。每个脚本关联于自己所拥有的上下文。

但上下文被一个脚本或线程使用时，其他脚本或线程不能使用。可当脚本或线程结束，这个上下文就可以被重用于下一个脚本或线程。

用 `JS_NewContext` 方法创建新的上下文。一个上下文和一个运行时环境关联，你必须指定栈的大小

```
JSContext *cx = JS_NewContext(m_rt, 8192);
if ( cx == NULL )
{
    // Do some error reporting
}
```

第 3 步 - 初始化全局对象 (global object)

在脚本执行前，你必须初始化 general JavaScript 函数和创建用于大多数脚本的 object 类。

全局对象 (global object) 用 JSClass 结构来描述。初始化这个结构如下：

```
JSClass globalClass =
{
    "Global", 0,
    JS_PropertyStub, JS_PropertyStub,
    JS_PropertyStub, JS_PropertyStub,
    JS_EnumerateStub, JS_ResolveStub,
    JS_ConvertStub, JS_FinalizeStub
};
```

现在你可以创建 global object 且初始化它。

```
JSObject *globalObj = JS_NewObject(cx, &globalClass, 0, 0);
JS_InitStandardClasses(cx, globalObj);
```

第 4 步 - 执行脚本

执行脚本的一种途径是用 JS_EvaluateScript 方法。

```
std::string script = "var today = Date(); today.toString();"
jsval rval;
uintN lineno = 0;
JSBool ok = JS_EvaluateScript(cx, globalObj, script.c_str(),
                              script.length(), "script", lineno, &rval);
```

当这个脚本运行没有错误，今天的日期被保存在 rval 中。rval 保存函数最后的执行结果。JS_EvaluateScript 的返回值，运行成功是 JS_TRUE, 发生错误是 JS_FALSE。

从 rval 取回字符串的值如下所示。这里就不演示每个细节。当你需要的时候查看相关 API 的信息。

```
JSString *str = JS_ValueToString(cx, rval);
std::cout << JS_GetStringBytes(str);
```

第 5 步 - 清理脚本引擎

在应用程序结束前，必须清理脚本引擎。

```
JS_DestroyContext(cx);
JS_DestroyRuntime(rt);
```

在 C++ 中定义 JavaScript 类

在例子中使用的类如下：

```
class Customer
{
public:
    int GetAge() { return m_age; }
```

```

void SetAge(int newAge) { m_age = newAge; }
std::string GetName() { return m_name; }
void SetName(std::string newName) { m_name = newName; }

private:
    int m_age;
    std::string m_name;
};

```

第 步 - JavaScript 类 .

创建一个新的 C++ 类，可以源于将在其中使用 JavaScript 的类，或者创建一个新类有一个那个类类型的成员。

在 JavaScript 中用结构 JSClass 来定义类。创建一个这个类型的静态 (static) 成员。声明为 public 成员，因为这个结构要被其他类使用。它可以被用于其他类来确定对象类型。（参考 JS_InstanceOf API）

```

// JSCustomer.h
class JSCustomer
{
public:
    JSCustomer() : m_pCustomer(NULL)
    {
    }

    ~JSCustomer()
    {
        delete m_pCustomer;
        m_pCustomer = NULL;
    }

    static JSClass customerClass;

protected:
    void setCustomer(Customer *customer)
    {
        m_pCustomer = customer;
    }

    Customer* getCustomer()
    {
        return m_pCustomer;
    }

private:
    Customer *m_pCustomer;
};

```

JSClass 结构包含 JavaScript 类的名字，一些标志和 用于引擎回调的函数名。例如一个回调用于当引擎需要从你的类中获取一个属性时。

在 C++ 文件的实现中定义 JSClass 结构，如下所示。

```

// JSCustomer.cpp
JSClass JSCustomer::customerClass =
{
    "Customer", JSCCLASS_HAS_PRIVATE,
    JS_PropertyStub, JS_PropertyStub,
    JSCustomer::JSGetProperty, JSCustomer::JSSetProperty,
    JS_EnumerateStub, JS_ResolveStub,
    JS_ConvertStub, JSCustomer::JSDestructor
};

```

```
};
```

所用的回调是 `JSCustomer::JSGetProperty`, `JSCustomer::JS SetProperty` 和 `JSCustomer::JSDestructor`. `JSGetProperty` 当引擎获取属性时被回调, `JS SetProperty` 当引擎设置属性时被回调, `JSDestructor` 当 JavaScript 对象被销毁时回调。

标志 `JSCCLASS_HAS_PRIVATE` 用于指示引擎开辟内存来绑定数据到 JavaScript 对象. 你可以用 `this` 存储一个指向你的类的指针.

回调是 C++ 类的静态成员函数.

```
static JSBool JSGetProperty(JSContext *cx, JSObject *obj, jsval id, jsval *vp);
static JSBool JS SetProperty(JSContext *cx, JSObject *obj, jsval id, jsval *vp);
static JSBool JSConstructor(JSContext *cx, JSObject *obj, uintN argc,
                           jsval *argv, jsval *rval);
static void JSDestructor(JSContext *cx, JSObject *obj);
```

第 2 步 - 初始化 JavaScript 对象

创建另一个叫 `JSInit` 的静态方法, 见下例. 这个方法将被应用程序调用, 用来创建 JavaScript 运行时环境.

```
static JSObject *JSInit(JSContext *cx, JSObject *obj, JSObject *proto);
```

实现代码如下

```
JSObject *JSCustomer::JSInit(JSContext *cx, JSObject *obj, JSObject *proto)
{
    JSObject *newObj = JS_InitClass(cx, obj, proto, &customerClass,
                                    JSCustomer::JSConstructor, 0,
                                    JSCustomer::customer_properties, JSCustomer::customer_methods,
                                    NULL, NULL);

    return newObj;
}
```

静态方法 `JSConstructor` 当你的对象被脚本实例化的时候被调用. 这个方法非常方便用于绑定你的数据到对象, 通过调用 `JS_SetPrivate` API.

```
JSBool JSCustomer::JSConstructor(JSContext *cx, JSObject *obj, uintN argc,
                                jsval *argv, jsval *rval)
{
    JSCustomer *p = new JSCustomer();

    p->setCustomer(new Customer());
    if ( ! JS_SetPrivate(cx, obj, p) )
        return JS_FALSE;
    *rval = OBJECT_TO_JSVAL(obj);
    return JS_TRUE;
}
```

这个构造器方法可以有多个参数, 能用于初始化你的类. 现在你已经在堆上创建了一个指针, 你也需要一种途径销毁这个指针. 这可以通过静态方法 `JS_Destructor` 来完成.

```
void JSCustomer::JSDestructor(JSContext *cx, JSObject *obj)
{
    JSCustomer *p = JS_GetPrivate(cx, obj);
```

```

    delete p;
    p = NULL;
}

```

第 3 步 - 增加属性

增加一个类型为 `JSPropertySpec` 的静态数组成员。这个数组将包含属性信息。再创建一个属性 ID 号的枚举 (enum)。

```

static JSPropertySpec customer_properties[];
enum
{
    name_prop,
    age_prop
};

```

在实现文件中初始化这个数组，代码如下

```

JSPropertySpec JSCustomer::customer_properties[] =
{
    { "name", name_prop, JSPROP_ENUMERATE },
    { "age", age_prop, JSPROP_ENUMERATE },
    { 0 }
};

```

数组的最后一个元素必须是空 (NULL)。每个元素包含另一个有 3 个元素的数组。第一个元素是名字，将在 JavaScript 脚本中使用。第二个元素是属性的唯一 ID 号，将被传递到回调函数中。第三个元素是一个标志，`JSPROP_ENUMERATE` 表示脚本中当枚举 `Customer` 对象的这个属性时是可见的，就是可以用在脚本中的 `for` 或 `in` 语句中。你可以指定 `JSPROP_READONLY` 属性来说明这个属性是不可以修改的。

现在你能实现获取 (getting) 和设置 (setting) 属性的回调函数。

```

JSBool JSCustomer::JSGetProperty(JSContext *cx, JSObject *obj, jsval id, jsval *vp)
{
    if (JSVAL_IS_INT(id))
    {
        Customer *priv = (Customer *) JS_GetPrivate(cx, obj);
        switch(JSVAL_TO_INT(id))
        {
            case name_prop:
                break;

            case age_prop:
                *vp = INT_TO_JSVAL(priv->getCustomer()->GetAge());
                break;
        }
    }
    return JS_TRUE;
}

```

```

JSBool JSCustomer::JS SetProperty(JSContext *cx, JSObject *obj, jsval id, jsval *vp)
{
    if (JSVAL_IS_INT(id))
    {
        Customer *priv = (Customer *) JS_GetPrivate(cx, obj);
        switch(JSVAL_TO_INT(id))
        {
            case name_prop:
                break;
            case age_prop:
                priv->getCustomer()->SetAge(JSVAL_TO_INT(*vp));
        }
    }
    return JS_TRUE;
}

```

```

        break;
    }
}
return JS_TRUE;
}

```

记得在属性回调中返回 `JS_TRUE` . 当你返回 `JS_FALSE` 将表示在你的对象中没有发现这个属性 .

第 4 步 - 增加方法

创建类型为 `JSFunctionSpec` 的静态成员数组 .

```
static JSFunctionSpec customer_methods[];
```

在实现文件中初始化这个数组 , 代码如下

```
JSFunctionSpec wxJSFrame::wxFrame_methods[] =
{
    { "computeReduction", computeReduction, 1, 0, 0 },
    { 0 }
};

```

数组的最后一个元素必须是空 (`NULL`) . 每个元素包含另一个有 5 个元素的数组 . 第一个元素是脚本中使用的函数名 . 第二个元素是全局或静态成员函数名 . 第三个元素是这个函数的参数个数 . 最后两个元素忽略 .

在类中创建一个静态方法

```
static JSBool computeReduction(JSContext *cx, JSObject *obj, uintN argc,
                              jsval *argv, jsval *rval);
```

当函数成功就返回 `JS_TRUE` . 否则返回 `JS_FALSE` . 你的 JavaScript 方法实际返回值被放到了 `rval` 参数中 .

实现这个方法的例子

```
JSBool JSCustomer::computeReduction(JSContext *cx, JSObject *obj, uintN argc,
                                     jsval *argv, jsval *rval)
{
    JSCustomer *p = JS_GetPrivate(cx, obj);
    if ( p->getCustomer()->GetAge() < 25 )
        *rval = INT_TO_JSVAL(10);
    else
        *rval = INT_TO_JSVAL(5);
    return JS_TRUE;
}

```

一个例子

下面的脚本使用上面创建的对象

```
var c = new Customer();
c.name = "Franky";
c.age = 32;
var reduction = c.computeReduction();

```

不要忘记当创建上下文的时候初始化 JavaScript 对象：

```
JSObject *obj = JSCustomer::JSInit(cx, global);
```

代码

```
//main.cpp 演示如何执行javascript
#define XP_PC

#include <string>
#include <iostream>
#include <fstream>
#include <jsapi.h>

#include "JSCustomer.h"

JSClass globalClass =
{
    "Global", 0,
    JS_PropertyStub, JS_PropertyStub, JS_PropertyStub, JS_PropertyStub,
    JS_EnumerateStub, JS_ResolveStub, JS_ConvertStub, JS_FinalizeStub
};

void main(int argc, char *argv[])
{
    if ( argc < 2 )
    {
        std::cout << "JSExec usage" << std::endl
        << "-----" << std::endl
        << "JSExec <fileName>" << std::endl;
    }

    std::string script;
    std::string buffer;
    std::ifstream istr(argv[1]);
    if ( istr.is_open() )
    {
        do
        {
            std::getline(istr, buffer);
            script += buffer;
        } while (!istr.fail());
    }
    else
    {
        std::cout << "JSExec error" << std::endl
        << "-----" << std::endl
        << "Can't open scriptfile " << argv[1] << std::endl;
        exit(0);
    }

    JSRuntime *rt = JS_Init(1000000L);

    if ( rt )
    {
        JSContext *cx = JS_NewContext(rt, 8192);
        if ( cx )
        {
            JSObject *globalObj = JS_NewObject(cx, &globalClass, 0, 0);
            if ( globalObj )
            {
                JS_InitStandardClasses(cx, globalObj);

                // Init JSCustomer
                JSCustomer::JSInit(cx, globalObj);

                // Execute the script
                jsval rval;
```

```

uintN lineno = 0;

JSString *str;
JSBool ok = JS_EvaluateScript(cx, globalObj, script.c_str(), script.length(), argv[1], lineno, &rval)
if ( ok == JS_TRUE )
{
    str = JS_ValueToString(cx, rval);
    char *s = JS_GetStringBytes(str);
    std::cout << "JSExec result" << std::endl
    << "-----" << std::endl
    << s << std::endl;
}
else
{
    std::cout << "JSExec error" << std::endl
    << "-----" << std::endl
    << "Error in JavaScript file " << argv[1] << std::endl;
}
}
else
{
    std::cout << "Unable to create the global object";
}
JS_DestroyContext(cx);
}
else
{
    std::cout << "Unable to create a context";
}
JS_Finish(rt);
}
else
{
    std::cout << "Unable to initialize the JavaScript Engine";
}
}

//JSCustomer.h 演示Customer JavaScript类的定义
/**
 * JSCustomer.h - Example for my tutorial : Scripting C++ with JavaScript
 * (c) 2002 - Franky Braem
 * http://www.braem17.yucom.be
 */

#ifndef _JSCustomer_H
#define _JSCustomer_H

#include "Customer.h"

class JSCustomer
{
public:
    /**
     * Constructor
     */
    JSCustomer() : m_pCustomer(NULL)
    {
    }

    /**
     * Destructor
     */
    virtual ~JSCustomer()
    {
        delete m_pCustomer;
        m_pCustomer = NULL;
    }

    /**
     * JSGetProperty - Callback for retrieving properties
     */

```



```

static JSBool JSGetProperty(JSContext *cx, JSObject *obj, jsval id, jsval *vp);

/**
 * JS SetProperty - Callback for setting properties
 */
static JSBool JSSetProperty(JSContext *cx, JSObject *obj, jsval id, jsval *vp);

/**
 * JS Constructor - Callback for when a wxCustomer object is created
 */
static JSBool JSConstructor(JSContext *cx, JSObject *obj, uintN argc, jsval *argv, jsval *rval);

/**
 * JS Destructor - Callback for when a wxCustomer object is destroyed
 */
static void JSDestructor(JSContext *cx, JSObject *obj);

/**
 * JS Init - Create a prototype for wxCustomer
 */
static JSObject* JSInit(JSContext *cx, JSObject *obj, JSObject *proto = NULL);

static JSBool computeReduction(JSContext *cx, JSObject *obj, uintN argc, jsval *argv, jsval *rval);

static JSClass Customer_class;

void setCustomer(Customer *customer)
{
    m_pCustomer = customer;
}

Customer* getCustomer()
{
    return m_pCustomer;
}

protected:

private:

Customer *m_pCustomer;

static JSPropertySpec Customer_properties[];
static JSFunctionSpec Customer_methods[];

enum
{
    name_prop,
    age_prop
};

#endif // _JSCustomer_H

//JSCustomer.cpp 演示JSCustomer类的实现
/**
 * JSCustomer.cpp - Example for my tutorial : Scripting C++ with JavaScript
 * (c) 2002 - Franky Braem
 * http://www.braem17.yucom.be
 */

#include <string>

#define XP_PC
#include <jsapi.h>

//#include "Customer.h"
#include "JSCustomer.h"

JSPropertySpec JSCustomer::Customer_properties[] =
{

```

```

{ "name", name_prop, JSPROP_ENUMERATE },
{ "age", age_prop, JSPROP_ENUMERATE },
{ 0 }
};

JSFunctionSpec JSCustomer::Customer_methods[] =
{
{ "computeReduction", computeReduction, 1, 0, 0 },
{ 0, 0, 0, 0, 0 }
};

JSClass JSCustomer::Customer_class =
{
"Customer", JSCCLASS_HAS_PRIVATE, JS_PropertyStub, JS_PropertyStub,
JSCustomer::JSGetProperty, JSCustomer::JSSetProperty,
JS_EnumerateStub, JS_ResolveStub, JS_ConvertStub, JSCustomer::JSDestructor
};

JSBool JSCustomer::JSGetProperty(JSContext *cx, JSObject *obj, jsval id,
jsval *vp)
{
if (JSVAL_IS_INT(id))
{
JSCustomer *p = (JSCustomer *) JS_GetPrivate(cx, obj);
Customer *customer = p->getCustomer();
switch (JSVAL_TO_INT(id))
{
case name_prop:
{
std::string name = customer->GetName();
JSString *str = JS_NewStringCopyN(cx, name.c_str(), name.length());
*vp = STRING_TO_JSVAL(str);
break;
}
case age_prop:
*vp = INT_TO_JSVAL(customer->GetAge());
break;
}
}
return JS_TRUE;
}

JSBool JSCustomer::JSSetProperty(JSContext *cx, JSObject *obj, jsval id,
jsval *vp)
{
if (JSVAL_IS_INT(id))
{
JSCustomer *p = (JSCustomer *) JS_GetPrivate(cx, obj);
Customer *customer = p->getCustomer();

switch (JSVAL_TO_INT(id))
{
case name_prop:
{
JSString *str = JS_ValueToString(cx, *vp);
std::string name = JS_GetStringBytes(str);
customer->SetName(name);
break;
}
case age_prop:
customer->SetAge(JSVAL_TO_INT(*vp));
break;
}
}
return JS_TRUE;
}

JSBool JSCustomer::JSConstructor(JSContext *cx, JSObject *obj, uintN argc,
jsval *argv, jsval *rval)
{
JSCustomer *priv = new JSCustomer();

```

```

priv->setCustomer(new Customer());
JS_SetPrivate(cx, obj, (void *) priv);

return JS_TRUE;
}

void JSCustomer::JSDestructor(JSContext *cx, JSObject *obj)
{
JSCustomer *priv = (JSCustomer*) JS_GetPrivate(cx, obj);
delete priv;
priv = NULL;
}

JSObject *JSCustomer::JSInit(JSContext *cx, JSObject *obj, JSObject *proto)
{
JSObject *newProtoObj = JS_InitClass(cx, obj, proto, &Customer_class,
JSCustomer::JSConstructor, 0,
NULL, JSCustomer::Customer_methods,
NULL, NULL);
JS_DefineProperties(cx, newProtoObj, JSCustomer::Customer_properties);
return newProtoObj;
}

JSBool JSCustomer::computeReduction(JSContext *cx, JSObject *obj, uintN argc,
jsval *argv, jsval *rval)
{
JSCustomer *p = (JSCustomer*) JS_GetPrivate(cx, obj);
if ( p->getCustomer()->GetAge() < 25 )
*rval = INT_TO_JSVAL(10);
else
*rval = INT_TO_JSVAL(5);
return JS_TRUE;
}

```

```

//Customer.h 演示Customer C++类的定义
#ifndef _Customer_H
#define _Customer_H

```

```

class Customer
{
public:

int GetAge()
{
return m_age;
}

void SetAge(int newAge)
{
m_age = newAge;
}

std::string GetName()
{
return m_name;
}

void SetName(std::string newName)
{
m_name = newName;
}

private:

int m_age;
std::string m_name;

};

#endif

```

```
//example.js 演示JavaScript的例子
var c = new Customer();
c.name = "Franky";
c.age = 32;
c.computeReduction();
```

【作者：[dozb](#)】 【访问统计：2058】 【2005年 06月 14日 星期二 18:03】 【[注册](#)】 【[打印](#)】

搜索

Google

站内搜索

搜索

Trackback

你可以使用这个链接引用该篇文章 <http://publishblog.blogchina.com/blog/tb.b?diaryID=1919675>

回复



- 评论人：[faw](#) 賤2007-09-04 14:59:07

验证码：

賤1242

评论内容：

提交

重置