[SDCFE]新人指南-feutils

作者：liyubei

- Fserver
- Fslide
- Fcooder
- Fsprite

新人指南-feutils

一组工具集来减少重复的工作，提高工作效率，降低学习成本。

安装

首先系统上确保你的系统上已经安装了 svn 相关的工具，以及 python 和 java 这两个编程环境。 环境的安装步骤请参考这里(env.text)

我们假设这次的工作目录是 WORK_DIR ，对于Windows平台用户，我们创建目录 c:\work ，对于NIX 平台用户，我们创建目录 $HOME\work 。

第一步：切换到 WORK_DIR ，然后执行命令：

```
svn co http://fe.baidu.com/repos/doc/sdcfe sdcfe
```

第二步：设置一下系统变量，对于Windows平台用户，我们设置的是 %PATH% ，把这两个目录添加进去：

```
WORK_DIR\tools\bin;WORK_DIR\tools\ant\bin
```

对于NIX平台用户，我们设置的是 $PATH ，把这两个目录添加进去：

```
WORK_DIR/tools/bin:WORK_DIR/tools/ant/bin
```


系统属性
常规
计算机名
硬件
高级
系统还原
自动更新
远程
环境变量
编辑系统变量
变量名(N):
Path
变量值(V):
am Files\MySQL\MySQL Server 5.1\bin
确定
取消
系统变量(S)
变量
值
NUMBER_OF_PR...
1
OS
Windows_NT
Path
C:\WINDOWS\system32;C:\WINDOWS;...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;....
PROCESSOR_AR...
x86
新建(W)
编辑(I)
删除(L)
确定
取消

使用

打开一个终端，执行命令 Fhelp ，如果能看到类似下面的输出，说明环境已经配置OK了

```
$ Fhelp
Available commands:
  Fapp
  Fhelp
  Fcodereview
  Ffixjsstyle
  Fgjslint
  Fws
  Fjs
  Fformat
  Fdoc
  Flint
  Fcompressor
  Foptipng
  Foutline
```

提醒

Fformat,Fdoc,Flint,Fcompressor,Foptipng,Foutline这几个在Windows下面还无法使用，后续会升级。

ant

关于 apache ant 的基本概念，可以google一下，此外 sdcfe/src/main/build.xml 里面提供了一些项目 中很常用的用法，可以了解一下。

Fws

启动一个简单的Web Server，默认的根目录是执行命令的目录，默认的端口是8000，如果想切换端口，添加 参数即可，例如：

```
Fws 8080 # 启动Web Server, 端口是8080
```

Fapp

创建一个简单的app，可以通过添加 -h 参数查看使用方式。这里需要注意的一个地方是，执行这个命令的时候， 需要保证当前目录存在 base.js 。这个文件在 DAN , CLB , COUP 项目里面都是存在的。对于

sdcfe这个repos， 我们切换到 sdcfe/src/main 目录即可使用这个命令。

一般的用法是:

```
Fapp -n news
```

就会在当前目录下面生成一个 news 目录，里面有如下几个文件:

```
app.html  app.js  config.js  init.js  mockup.js  tpl.html
```

可以根据名字了解它们的角色。如果想查看效果的话，需要额外的一个步骤，切换到 base.js 所在的目录，执行命令

```
make deps
```

然后在当前目录执行 Fws ，再用浏览器打开 http://localhost:8000/news/app.html 即可看到效果。

如果是第一次执行这个命令，会提示你输入 用户名 和 邮箱 ，这两个信息是用来生成代码的时候自动添加内容使用的。

Fcodereview

请参考这篇文章 (codereview.text)

Fgjslint & Ffixjsstyle

就是 google closure linter 里面 gjslint 和 fixjsstyle，只是名字不同而已。

如何更新

如果工具集更新了，我们只需要切换到 WORK_DIR，执行命令:

```
svn update
```

或者

```
git svn fetch
git svn rebase
```

ChangeLog

1. 添加了 Fzip，Fformat 的支持
2. Linux平台支持了 Fsvn add，Fmake
3. Windows平台下面添加了Finstall和Funinstall的命令
4. 升级了Fhelp

Fzip

支持压缩js,css,json，未来还会支持压缩html

```
Usage: Fzip.py [options] input

Options:
  -h, --help            show this help message and exit
  -e ENGINE, --engine=ENGINE
                        compressor engine
  -c CHARSET, --charset=CHARSET
                        input charset
  -o OUTPUT, --output=OUTPUT
                        output file
```

Fformat

跟 Fzip 的作用相反，把压缩之后的文件格式化，方便查看代码，现在支持js,css,json，未来还会支持html

```
Usage: Fformat.py [options] input

Options:
  -h, --help            show this help message and exit
  -t TYPE, --type=TYPE  input type
  -o OUTPUT, --output=OUTPUT
                        output file
```

Finstall

仅限于Windows平台，执行之后，在右键菜单里面添加了 Process with Fformat 和 Process with Fzip 两个命令

Funinstall

仅限于Windows平台，执行之后，把右键菜单里面的两个命令去掉

Fupload

开始支持通过cms的API，直接把静态文件上传到http://img.baidu.com的服务器上，使用方法如下：

```
Fupload ~/a.png
```

第一次使用的时候，会提示你输入邮箱的用户名和密码，来进行认证，之后就不需要

了。如果以后修改了 用户名和密码，导致认证失败，无法上传，可以通过命名 Fupload -c 重新设置即可。

上传成功之后，会返回线上的地址，例如 http://img.baidu.com/adm/a.png ，默认只是上传到 http://img.baidu.com/adm 目录下面。

Ftangram

平时使用tangram开发应用的时候，一般为了方便，都是把整个tangram引入到项目中来，例如，经常看到如下的代码：

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <script type="text/javascript" src="http://the/path/to
    <script type="text/javascript" src="app.js"></script>
  </head>
  <body>
    foobar
  </body>
</html>
```

而我们 app.js 的内容可能是下面这样子的:

```
function Ad() {
}
Ad.prototype.render = function() {
  ...
  baidu.g("id").innerHTML = html;
```

```
    ...
    baidu.on("id", "click", function(){});
    ...
}
```

当我们要发布的时候，可以直接用线上的tangram，如果你考虑到tangram还是比较大，可能会手工通过tangram的**codesearch工具(http://tangram.baidu.com/codesearch/codesearch.html)** 合并一份儿体积比较小的代码出来，但是手工的方式很不方便，而且容易出错，万一漏掉某个函数就麻烦了。因此想写这个工具来解决 这个问题，我们开始的时候，直接引用最全的tangram，发布的之后，通过工具 Ftangram ，分析代码中使用到

的 tangram 函数的地方，然后 自动把这些函数的实现获取下来。例如：

```
Ftangram app.js -o app.compiled.js
```

大家就能看到， app.compiled.js 里面已经有了 baidu.g 和 baidu.event.on 的实现了，其它都没有了，之后我们再使用 Fzip 这个工具，压缩 一下就能发布了，o(∩∩)o...哈哈.

Ftangram 有很多参数，可以通过 --help 查看。

```
Usage: Ftangram.py [options] file1 [file2 [file3] ... ]
```

```
Options:
  -h, --help            show this help message and exit
  -o OUTPUT, --output=OUTPUT
                        output file
  -f FUNCTIONS, --functions=FUNCTIONS                    #
  -l, --keep-first-file-position                         #
  -w OUTPUT_WRAPPER, --output_wrapper=OUTPUT_WRAPPER     #
                        default is %output%
  -c CHARSET, --charset=CHARSET                          #
```

Fserver

基于er的项目的一个本地调试工具。

一般来说，er的项目本地调试的时候，都是采用nginx，然后配置proxy_pass到后端去获取数据。如果想在ie下面 调试的时候，因为ie下面的**@import限制的问题**

(http://social.msdn.microsoft.com/Forums/en-US/iewebdevelopment/thread/ad1b6e88-bbfa-4cc4-9e95-3889b82a7c1d/)，还需要把对 css的请求proxy_pass到本地的一个static_server，这个server会返回合并之后的css文件，这样子一般来说就需要 有两套server（nginx和static_server）.

Fserver就是为了解决这个问题而准备的，可以理解为 `Fserver = nginx + static_server` ，但是具备更好的扩展性和跨平台性，因为是 用node.js开发的，大家都熟悉这个语言.

如何使用

1. 要使用之前，首先需要更新 feutils，也就是切换到 WORK_DIR，然后执行 svn update

2. 然后我们创建一个最简单的 server.js，示例如下：

```
var er = require('er-server');

var server = new er.ErServer();
server.start();
```

3. 我们把这个文件保存在 c:\work\server.js，打开一个终端，

切换到 c:\work 这个目录，然后执行命令：

```
Fserver server.js
```

4. 正常情况下，如果没有报错，说明服务已经启动了，我们此时可以访问 **http://localhost:8080/server.js (http://localhost:8080/server.js)** 查看我们这个文件了.

5. OK，到现在为止，一个最简单的静态文件服务已经完成了，仅仅3行代码而已。

6. 现在我们要实现 proxy_pass 的功能了，因为一个er的应用，除了需要请求静态文件之外，我们还需要跟后端交互来获取数据， 而这里的实现逻辑是如果发现本地没有这个文件，那么就考虑去后端服务请求，此时我们仅仅需要一个配置文件而已：

```
{
    "localhost:8090" : "jn.e.shifen.com"
}
```

把这部分内容放到 c:\work\online.config.json 这个文件里面，然后重启服务，添加一个 --config 参数，如下：

```
Fserver server.js --config online.config.json
```

7. 此时我们再次访问**http://localhost:8080/login_cfg/read (http://localhost:8080/login_cfg/read)** 这个地址，因为本地找不到 c:\work\login_cfg\read 这个文件，那么 就会去请求 http://jn.e.shifen.com/login_cfg/read 这个地址，然后把结果返回回来，符合预期.

8. OK，到现在为止，一个具备proxy_pass的静态文件服务器也完成了，仅仅3行代码和一个配置文件而已.

9. 现在我们要考虑完成自定义handler的功能，也就是当访问某个url的时候，我们希望返回自定义的内容，不是文件的真正内容。例如 当我访问 http://localhost:8080/server.js (http://localhost:8080/server.js)，需要返回 HELLO Fserver，那么我们可以很简单的这么写：

```
var er = require("er-server");

var server = new er.ErServer();
server.addHandler("/server.js", function(){
  return "HELLO Fserver";
});
server.start();
```

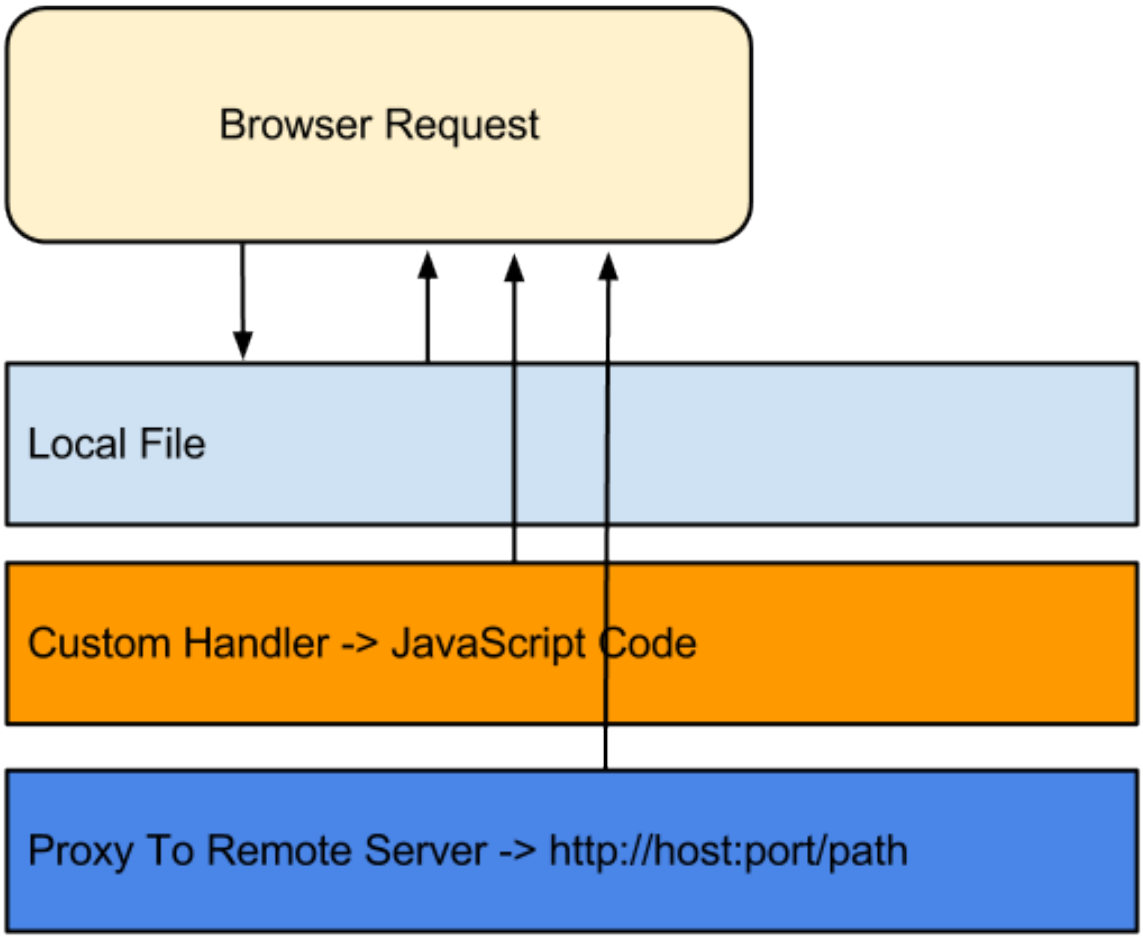
10. 重启服务，然后再次访问 http://localhost:8080/server.js (http://localhost:8080/server.js)，就会看到浏览器显示的结果已经变成 HELLO Fserver 了.

11. 如果想返回另外一个文件的内容怎么办呢？我们可以参考NodeJS的API (http://nodejs.org/docs/latest/api/fs.html)，最简单的方式是 这么写：

```
server.addHandler("/no_such_url.js", function(){
  var fs = require("fs");
  return fs.readFileSync("online.config.json", "utf
});
```

12. exit(0)，更多请参考**锦囊里面的server.js (jn.server.js)**

整体架构


Browser Request
Local File
Custom Handler -> JavaScript Code
Proxy To Remote Server -> http://host:port/path

注意事项

1. 文件的编码都必须是utf8，其它类型的编码nodejs暂时还不太容易处理.
2. addHandler的第一个参数是url，最好要有后缀名，Fserver会自动添加Response Header中的Content-Type.
3. 恺华同学测试，发现 Win7 64bit 下面因为node.js的问题，暂时无法正常使用，需要等待node.js升级支持 Win7 64bit 才可以.

Fslide

对 landside 的一个包装，只需要更新feutils即可使用，无须安装，推荐使用 Markdown 的语法写PPT，例如：

```
# 这是一个PPT

------

## 这是标题1
1. xxx
2. yyy
3. zzz

-------

## 这是标题2

    !javascript
    var a = 10;
    var b = 20;
    var c = /30/gi;
```

执行命令

```
Fslide -i demo.markdown
```

之后的效果可以从这里查看
(presentation.html)

Fcooder

仅仅是Linux平台下，为了规避
http://cooder.baidu.com
(http://cooder.baidu.com) 的缺陷而写一个脚本，一般不推荐使用.

Fsprite

FE经常的一个需求是希望能自动合并css中的背景图，从而优化页面的资源请求，加快页面的加载速度，但是人肉合并这些背景图 是很耗时的工作，而且容易出错，为了减少这些重复性的工作，找到了 **http://yostudios.github.com/Spritemapper/ (http://yostudios.github.com/Spritemapper/)** 这个工具，感觉 不错，能满足70%的需求，所以包装了一下，放到feutils里面，命名为 `Fsprite` 。这里介绍一下简单的用法：

Fsprite现在只支持处理png图片，因此写css的请注意。

```
/** assets/css/sina.css */
.icon {
  display: inline-block;
  margin-right: 10px;
}
.icon.a {
  width: 16px;
  height: 16px;
  background: url('../img/sina_16x16.png') no-repeat;
}
.icon.b {
  width: 24px;
  height: 24px;
  background: url('../img/sina_24x24.png') no-repeat;
}
...
...
.icon.l {
  width: 48px;
  height: 48px;
  background: url('../img/sina_logo_48x48.png') no-repeat;
}
```

```
.icon.m {
  width: 64px;
  height: 64px;
  background: url('../img/sina_logo_64x64.png') no-repeat;
}
```

执行命令

```
Fsprite assets/css/sina.css
```

生成的CSS文件是：

```
.icon {
  display: inline-block;
  margin-right: 10px;
}
.icon.a {
```

```
  width: 16px;
  height: 16px;
  background: url('../img.png') no-repeat -17px 0;
}
.icon.b {
  width: 24px;
  height: 24px;
  background: url('../img.png') no-repeat -65px -17px;
}
...
...
.icon.l {
  width: 48px;
  height: 48px;
  background: url('../img.png') no-repeat 0 -82px;
}
.icon.m {
  width: 64px;
  height: 64px;
  background: url('../img.png') no-repeat 0 -17px;
}
```

生成的图片如下：