

# Erlang实战

- 构建IP查询服务

litaocheng@gmail.com ECUG IV  
2009.11.07



Concurrent Distributed FP Language

# 目的

---

通过一个具体的开发实例，比较全面的描述erlang项目开发过程。

包含的内容：

- 项目目录结构
- OTP
- makefile
- 代码风格
- edoc
- type/spec, dialyzer
- eunit
- common testserver
- release



# 开发任务

---

## IP在线查询服务

基于 HTTP 的 IP 查询服务, 用户调用 HTTP GET 查询某个 IP, 返回此 IP 所属国家, 地区等相关的地理信息.

本项目采用 Erlang 开发, 基于其他开源项目.

项目地址:

<http://code.google.com/p/erlips/>

获取代码:

svn checkout <http://erlips.googlecode.com/svn/trunk/> erlips-read-only



# 开发任务-IP在线查询服务

---

```
GET geoip?ip=123.8.36.135
```

```
{  
  "cuntroy": "China",  
  "region": "22",  
  "city": "Beijing",  
  "long": 116.39,  
  "lat": 39.93  
}
```

返回数据为json字符串 (<http://json.org>)



# 设计实现

---

- 开发迅速

Erlang 开发

mochiweb 为web server

IP数据库为 maxmind GeoLiteCity binary db

egeoip - GeoLiteCity Erlang client library

- 性能出众

采用合适的查询算法

- 易于扩展

逻辑扩展 Erlang Hot code load/swap

分布式支持, 满足性能要求



# 做好准备？

---

- 一点WEB开发经验（HTTP，js 总得听过吧）
- 一点基本网络知识（IP不是“挨批”）
- 一点Erlang知识（至少要在 google 上搜索过Erlang）

最重要的一点：

对Erlang充满了兴趣！  
勤学习，勤思考，勤动手！



# 安装mochiweb

---

1. 从 google code checkout mochiweb
2. 将 mochiweb 放置再 \$ERL\_LIB 下 (\$ERL\_LIB 下所有的 app 均自动添加到 code path list 中)
3. 编译 mochiweb

通过以上步骤，我们项目中将可以使用 mochiweb.  
为了方便提供脚本：*erlips/scripts/install\_mochiweb.sh*

```
litaop@litaopc # ./install_mochiweb.sh
check if Erlang/OTP exists... ok
get the erlang lib path... ok
ERL_TOP is /usr/local/lib/erlang/lib
checkout the mochiweb codes from the google code...
ok
(cd src;make all)
```



# 项目目录结构

---

- `src`  
Erlang 源代码 (`*.erl`, `*.hrl`)
- `ebin`  
编译生成的 Erlang 目标码 (`*.beam`), `.app` 文件
- `priv`  
项目专有的一些文件, 如 `c` 代码, 专有数据, `code:priv_dir/1`  
获取应用的`priv`目录
- `include`  
包含头文件 (`.hrl`)
- `test`  
测试脚本

## Note

遵循OTP的目录结构规范, 让他人更容易理解你的项目, 让 Erlang 相关工具更好的理解你的项目





# erlips directory

---

```
litao@litaopc: ~  
litao@litaopc:~$ tree -d erlips  
erlips  
|-- doc  
|-- ebin  
|-- include  
|-- priv  
|-- scripts  
|-- src  
`-- test  
  
7 directories  
litao@litaopc:~$
```



ERLANG

# 安装geoip & egeoip

---

## geoip

从 maxmind 获取 geoip ip database, 执行脚本:

```
# erlips/scripts/get_geoip.sh
```

完成后在 priv 目录下将会产生一个名为:  
GeoIPCity.dat.gz 的文件

## egeoip

从 google code 获取 egeoip (<http://code.google.com/p/egeoip/>)

将名为 egoip.erl 的源文件, 添加到 erlips/src 目录中



# OTP是什么

---

- 在Erlang(FP, 并发, message-based)的世界, OO的设计模式不再合适
- Erlang自己的“设计模式”
- 依照OTP设计出更加规范清晰的 application
- 通过对项目的提炼, 升华出几种 behaviour:
  - `gen_server` (使用频率极高! 80%)
  - `gen_fsm`
  - `gen_event`
  - `supervisor`
  - `application`后4种都可以通过 `gen_server`来实现



# OTP behaviour gen\_server

---

- `gen_server` 用来维护某个状态(数据) 或 提供某项功能(接口)
- 很多逻辑都可以抽象成 `client/server` 的结构, `gen_server` 广泛使用
- `gen_server` 提供通用的 `server` 框架, 用户模块定义相关 `callback` 函数
- `gen_server` 支持 `call(sync)`, `cast(asyn)`, `multi_call`, `abcast`
- `gen_server` 运行在一个独立的 `process` 中, 复杂的操作会阻塞其他 `call/cast` 调用
- `gen_server` 别搞成死锁: `gen_server:handle_call/3` 中调用 `gen_server:call/2`
- 可以使用 `sys` module 对 `gen_server` 进行诊断, 获取各种信息



# OTP behaviour gen\_fsm

- 一个有限状态机代码框架
- 处理 `client lifetime`, 协议交互, `server` 状态等场景下使用
- 在一个独立的 `erlang process` 中执行, 耗时的操作会阻塞其他调用
- 对于不同的 `StateName`, 需要 `export` 相关的 `Mod`:  
`StateName/2, 3` 函数
- `send_event`, `send_all_state_event` 为异步调用,  
`sync_send_event`, `sync_send_all_event` 为同步调用
- 同 `gen_server` 一样, 支持 `start/3, 4` 以独立方式启动  
`gen_fsm`, 或 `start_link/3, 4` 以属于 `supervisor tree` 方式启动



# OTP behaviour gen\_event

---

- 事件处理框架（包括管理器和处理者）
- 在需要对某个事件进行多种处理时，推荐采用 `gen_event`，如 Erlang 中的 `error_logger` 便是采用 `gen_event`，我们可以方便的添加自己的事件处理模块，进行log处理
- `start/0,1`, `start_link/0,1` 启动一个事件管理器
- 通过 `add_handler/3`, `delete_handler/3`, `swap_handler/3` 对事件管理其中的处理者进行操作
- `notify/2` 为异步调用，`sync_notify`为同步调用，总是返回ok，`call`为同步调用，返回对应结果



# OTP behaviour supervisor

管理 `gen_server`, `gen_fsm`, `gen_event`, `supervisor` 或其他 `process`.  
包含多种策略:

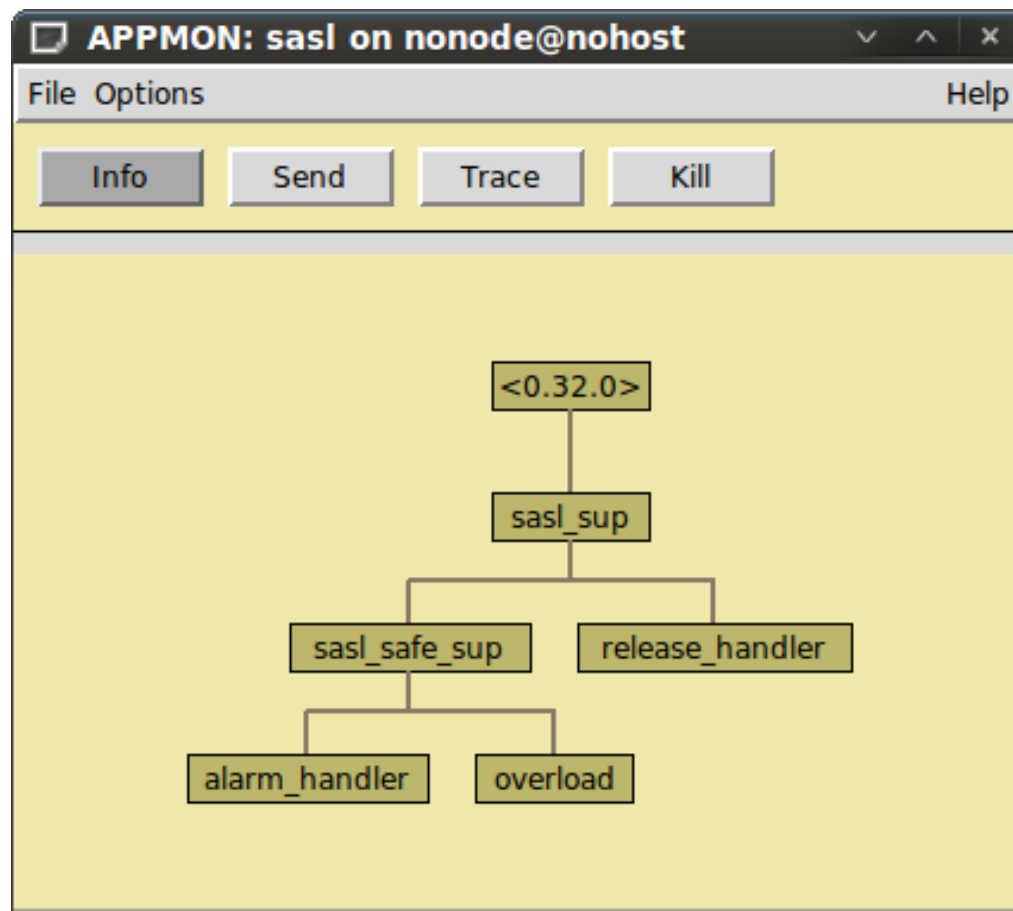
- `one_for_one` - 当某个 `child process` 结束并需要重启时, 仅仅重启这个 `child`
- `one_for_all` - 当某个 `child process` 结束并需要重启时, 所有其他 `child process` 均先停止, 随后全部重启
- `rest_for_one` - 当某个 `child process` 结束并需要重启时, 所有此节点的“后继”节点均先停止 (根据启动顺序判定后继节点), 随后后继节点及终止节点全部重新启动
- `simple_one_for_one` - 所有的 `child` 均是动态添加的实例, 其运行相同的代码. `terminate_child/2`, `delete_child/2` and `restart_child/2` 均无效

supervisor 基于 `gen_server` 实现



# appmon 工具查看supervisor tree

```
litao@litaopc:~$ erl -boot start_sasl  
1> appmon:start(). % makesure tcl/tk >= tcl8.3
```





# Emakefile

Emakefile 为 Erlang 自带的 Make 工具，定义如下：

```
Modules.  
{Modules,Options}.  
% Options: 参看 compile module
```

erlips Emakefile:

```
{"src/*", [{i, "include"},  
          {outdir, "./ebin"}]}
```

编译(当前路径为Emakefile所在路径):

linux shell: `$ erl -make`

erlang shell: `1> make:all()`.



# GNU Makefile

---

Emakefile不足：

- Emakefile只能用来编译erl代码
- 缺乏依赖支持
- 无法进行更加复杂的自动化工作

使用GNU Makefile

- make
- make clean
- make test
- make edoc

满足日常所需的各种编译，测试任务。

(推荐：GNU autoconf提供一些Erlang相关的Macro，制作更加规范，可移植的生成方法)



# erlips的Makefile

```
SHELL := /bin/bash
.PHONY: all test edoc dialyzer clean
PLT=".dialyzer_plt"

all:
    (cd src;$(MAKE))
test:
    (cd src;$(MAKE) TEST=true)
    (erl -pa ./ebin -eval "eunit:test(\"./ebin\",
[verbose]), init:stop()")
edoc:
    (mkdir -p ./edoc)
    (cd src; $(MAKE) edoc)
plt :
    (./scripts/gen_plt.sh -a sasl)
dialyzer: clean
    (cd src;$(MAKE) DEBUG=true)
    (dialyzer --plt $(PLT) -Werror_handling -
Wrace_conditions -Wunderspecs -r .)
clean:
    (cd src;$(MAKE) clean)
```



# Makefile

```
$ make all
    编译 erl 代码
$ make test
    调用所有 module 的 eunit test
$ make edoc
    根据代码中的 edoc 标记, 生成edoc
$ make plt
    调用 erlips/scripts/gen_plt.sh 生成 plt 文件: .
    dialyzer_plt
$ make dialyzer
    根据type, spec信息对代码进行静态分析
$ make clean
    清理生成的目标码
```



# 代码

---

- erlipsapp.erl  
application, supervisor callback module
- erlips\_httpd.erl  
httpd module, based on mochiweb
- \_ips\_geoip.erl  
handle module for "/ips/geoip?" request
- egeoip.erl  
解析GeoLiteCity binary file, 提供ip查询服务
- erlips\_ctl.erl  
erlips ctl 对应module
- \_demo.erl
- handle "/demo" path



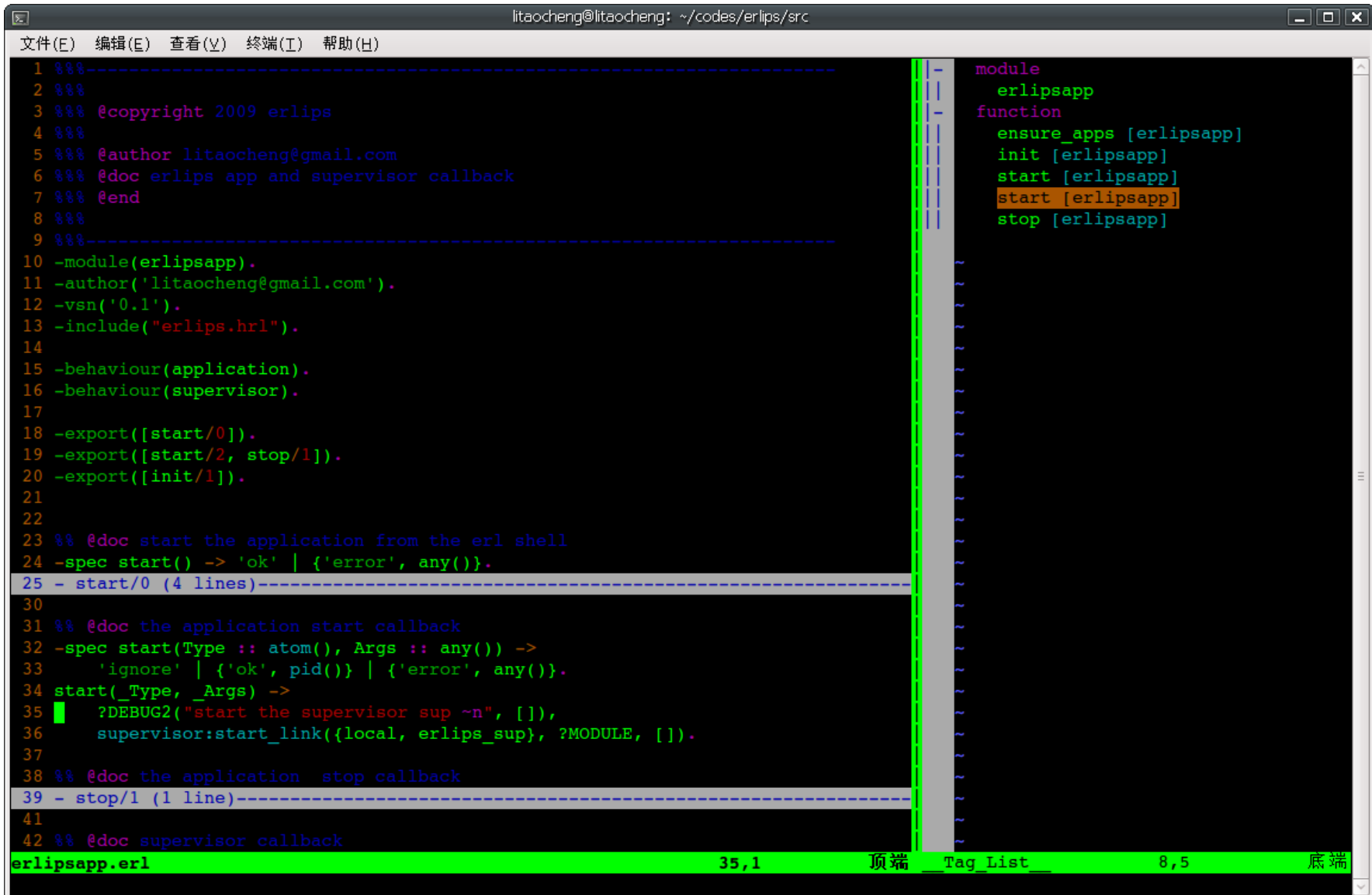
# Erlang Module Template

---

```
%%%-----  
%%%  
%%% @copyright your company 2009  
%%%  
%%% @author litao cheng <litaocheng@gmail.com>  
%%% @version 0.1  
%%% @doc some desc  
%%%  
%%%-----  
-module(mod_demo).  
-export([some_fun/0]).  
  
%% the function  
some_fun() ->  
    ...  
    % some comment  
    some_return.
```



# Erlang Source In Vim



```
litaocheng@litaocheng: ~/codes/erlips/src
文件(E) 编辑(E) 查看(V) 终端(T) 帮助(H)
1 %%%-----
2 %%%
3 %%% @copyright 2009 erlips
4 %%%
5 %%% @author litaocheng@gmail.com
6 %%% @doc erlips app and supervisor callback
7 %%% @end
8 %%%
9 %%%-----
10 -module(erlipsapp).
11 -author('litaocheng@gmail.com').
12 -vsn('0.1').
13 -include("erlips.hrl").
14
15 -behaviour(application).
16 -behaviour(supervisor).
17
18 -export([start/0]).
19 -export([start/2, stop/1]).
20 -export([init/1]).
21
22
23 %% @doc start the application from the erl shell
24 -spec start() -> 'ok' | {'error', any()}.
25 - start/0 (4 lines)-----
30
31 %% @doc the application start callback
32 -spec start(Type :: atom(), Args :: any()) ->
33   'ignore' | {'ok', pid()} | {'error', any()}.
34 start(_Type, _Args) ->
35   ?DEBUG2("start the supervisor sup ~n", []),
36   supervisor:start_link({local, erlips_sup}, ?MODULE, []).
37
38 %% @doc the application stop callback
39 - stop/1 (1 line)-----
41
42 %% @doc supervisor callback
erlipsapp.erl 35,1 顶端 Tag_List 8,5 底端
```



ERLANG

Concurrent Distributed FP Language

# edoc tags

---

@doc : 书写模块或函数的文档

@copyright : 显示版权信息

@doc : 文档描述信息

@version : 版本信息

@spec : 显示函数的spec信息（目前edoc无法使用-spec）

@type : 显示自定义的type信息

@deprecated : 表示对应信息不推荐使用，将被废除

@private : 私有信息

...

## Note

edoc 目前(R13B02-1)对 unicode 支持不好，如果doc中含有中文会产生错误.

参看: <http://www.nabble.com/UTF8-and-EDoc-td25676638.html>





# edoc examples (src/\_ips\_geoip.erl)

```
%%%-----  
%%%  
%%% @copyright 2009 erlips  
%%%  
%%% @author litaocheng@gmail.com  
%%% @doc the module handle the request path:  
%%% "http://host/ips/geoip"  
%%% @end  
%%%  
%%%-----  
  
%% @doc handle the /ips/geoip request  
%% @spec handle(Req :: any(), Method :: atom()) ->  
%%      {pos_integer(), list(), iodata()}  
-spec handle(Req :: any(), Method :: atom()) -> {pos_integer(),  
list(), iodata()}.  
handle(Req, 'GET') ->  
    .....  
    {200, [], <<"ok">>}.
```

# edoc examples (src/\_ips\_geoip.erl)

## Modules

[\\_demo](#)  
[\\_echo](#)  
[\\_ips\\_geoip](#)  
[egeip](#)  
[erlips\\_ctl](#)  
[erlips\\_httpd](#)  
[erlipsapp](#)

[Overview](#)



## Module `_ips_geoip`

[Description](#)  
[Function Index](#)  
[Function Details](#)

the module handle the request path: "http://host/ips/geoip".

Copyright © 2009 erlips

Authors: [litaocheng@gmail.com](mailto:litaocheng@gmail.com).

### Description

the module handle the request path: "http://host/ips/geoip"

### Function Index

<a href="#">handle/2</a>	handle the /ips/geoip request.
--------------------------	--------------------------------

### Function Details

#### handle/2

```
handle(Req::any(), Method::atom()) -> {pos_integer(), list(), iodata()}
```

handle the /ips/geoip request

[Overview](#)



ERLANG

Concurrent Distributed FP Language

# type and spec

---

## 预定义类型:

```
any(), none(), pid(), port(), ref(), [], atom(),  
binary(), float(), fun(), integer(), list(),  
tuple(), boolean(), char(), string() ...
```

## 自定义类型:

```
-type gender() :: 'male' | 'female'.  
-type age() :: 1 .. 150.  
-type http_code() :: pos_integer().  
-type header_list() :: [{binary() |  
string(), binary() | string()}].
```



# type and spec

定义type的目的，是为了定义函数的 Specifications，可以使用 dialyzer 进行静态分析。

目前Erlang OTP 中大部分 lib 的 exported 函数都定义了 spec

```
-spec Module:Function(ArgType1, ..., ArgTypeN) ->  
    ReturnType.
```

如果是在同一个 module 中:

```
-spec Function(ArgType1, ..., ArgTypeN) -> ReturnType.
```

为提供更好的文档:

```
-spec Function(ArgName1 :: Type1, ..., ArgNameN ::  
TypeN) -> RT.
```



# type and spec

---

Erlang 中同一个函数可以有多个函数子句(function clauses), 所以同一函数也可以具有多个 spec (以分号分割)

```
-spec foo(T1, T2) -> T3 ;  
      (T4, T5) -> T6.
```

type spec 参考:

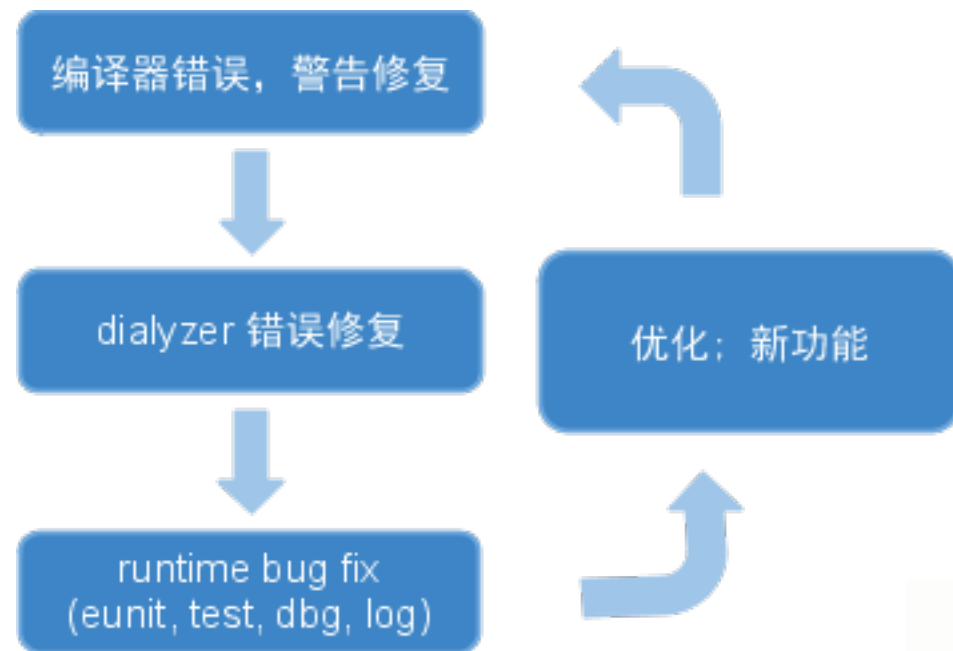
[eep 8] <http://www.erlang.org/eeps/eep-0008.html>



# 如何发觉程序中的问题？

开发过程中的错误，可以通过很多手段进行发掘：

- compile 编译期，主要为语法错误，确保没有任何错误，认真分析每个warning
- dialyzer 静态分析，运行dialyzer根据type/spec信息，分析函数调用的参数，返回值，及时发现错误
- runtime 运行时，通过eunit, common test, log, dbg 等工具在运行时发现程序的错误



# Compile

确保没有任何 编译警告!

```
litaocheng@litaocheng:~/codes/erlips$ make
(cd src;make)
make[1]: 正在进入目录 `/home/litaocheng/codes/erlips/src'
erlc -W -I ../include -o ../ebin _demo.erl
erlc -W -I ../include -o ../ebin _echo.erl
erlc -W -I ../include -o ../ebin egeoip.erl
erlc -W -I ../include -o ../ebin erlipsapp.erl
erlc -W -I ../include -o ../ebin erlips_ctl.erl
erlc -W -I ../include -o ../ebin erlips_httpd.erl
erlc -W -I ../include -o ../ebin _ips_geoip.erl
cp erlips.app ../ebin/erlips.app
make[1]:正在离开目录 `/home/litaocheng/codes/erlips/src'
```

# Dialyzer

## 1, 生成plt

```
$ dialyzer --build_plt --verbose --  
output_plt .dialyzer_plt -r \  
    $ERL_TOP/lib/erts-*/ebin \  
    $ERL_TOP/lib/kernel-*/ebin \  
    $ERL_TOP/lib/stdlib-*/ebin
```

ERL\_TOP 代表 Erlang otp 安装目录 ( `code:root_dir()` )  
erlips 中, 我们使用 Makefile 生成 plt 文件:

```
$ make plt
```

其调用我们书写的一个script  
(`erlips/scripts/gen_plt.sh`), 在 `erlips` 目录下生  
成 `.dialyzer_plt`





# Dialyzer con't

## 2, 运行dialyzer

运行dialyzer很简单: `$ dialyzer --plt .dialyzer_plt -r erlips`

使用 Makefile 运行dialyzer:

```
$ make dialyzer
(dialyzer --plt ".dialyzer_plt" -Werror_handling -
Wrace_conditions -r .)
  Checking whether the PLT .dialyzer_plt is up-to-date...
yes
  Proceeding with analysis...
Unknown functions:
  mochiweb_headers:to_list/1
  mochiweb_http:start/1
  mochiweb_util:path_split/1
done in 0m3.88s
done (passed successfully)
```



# Dialyzer con't

## 3, 制造一个 显而易见的错误

下面我们故意设置一个错误 (*src/\_ips\_geoip.erl*):

```
27 + Ip = proplists:get_value("ip", <<>>),
```

随后进行 dialyzer 分析:

```
$ make dialyzer
Proceeding with analysis...
...
_ips_geoip.erl:27: The call proplists:get_value("ip",
<<>>) will never return since the success typing is
(any(),[any()]) -> any() and the contract is (Key::term(),
List::[term()]) -> term()
...
done (warnings were emitted)
make: *** [dialyzer] 错误 2
```



# Dialyzer con't

---

dialyzer 提示错误!

因为 `proplists:get_value("ip", <<>>)` 的第二个参数与 `proplists:get/2` 的 `spec` 声明不一致!

修改 `src/_ips_geoip.erl` Line 27为:

```
27 Ip = proplists:get_value("ip", []),
```

或者将本行删除, 则dialyzer分析成功.

## Note

dialyzer 仅仅分析出了一小部分bug, 需要通过更多的单元测试, 集成测试修正bug



# EUnit

---

module 中 include eunit 头文件  
`-include_lib("eunit/include/eunit.hrl").`  
自动导出 test/0 函数

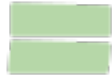
Macros:

- `assert(BoolExpr)`
- `assertNot(BoolExpr)`
- `assertMatch(GuardedPattern, Expr)`
- `assertEqual(Expect, Expr)`
- `assertException(ClassPattern, TermPattern, Expr)`
- `assertError(TermPattern, Expr)`
- `assertExit(TermPattern, Expr)`
- `assertThrow(TermPattern, Expr)`
- `assertCmd(CommandString)`
- `assertCmd(CommandString)`

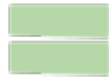


# EUnit

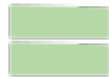
```
basic1_test() ->  
  ?assert(1 + 1 == 2).
```



```
basic2_test_() ->  
  fun () -> ?assert(1 + 1 == 2) end.
```



```
basic3_test_() ->  
  ?_test(?assert(1 + 1 == 2)).
```



```
basic4_test_() ->  
  ?_assert(1 + 1 == 2).
```

- `xxx_test()` 为 test 函数
- `xxx_test_()` 为 test 生成函数
- 模块 `m` 的 unit test 可以放置在一个独立的 `m_tests` 模块中
- `eunit:test(Dir, Opts)` 调用目录 `Dir` 下所有的 module 的 `test/0` 函数
- `eunit:test(Mod, Opts)` 调用某个 `Mod` 的 `test/0` 函数



# EUnit

erlips 模块比较少, 因此 eunit 测试相关的代码也比较少.

在 (erlips/src/\_ips\_geoip.erl) 中, unit test 用来测试 f2s/1 函数, 其将一个 float 转化为只有 2 个小数位的字符串.

```
-ifdef(EUNIT).  
f2s_test_() ->  
[  
    ?_assertEqual("2.00", f2s(2)),  
    ?_assertEqual(["2.01"], f2s(2.01)),  
    ?_assertEqual(["0.00"], f2s(0.00)),  
    ?_assertEqual(["2.01"], f2s(2.0102)),  
    ?_assertError(function_clause, f2s('2.00'))  
].  
-endif.
```

# EUnit

```
$ make test
...
(erl -pa ./ebin -eval "eunit:test(\"./ebin\", [verbose]), init:stop())
1> ===== EUnit =====
directory "./ebin"
  module 'erlips_httpd'
  module 'erlips_ctl'
  module '_echo'
  module '_ips_geoip'
    _ips_geoip:61: f2s_test_...ok
    _ips_geoip:62: f2s_test_...ok
    _ips_geoip:63: f2s_test_...ok
    _ips_geoip:64: f2s_test_...[0.001 s] ok
    _ips_geoip:65: f2s_test_...ok
  [done in 0.014 s]
  module 'erlipsapp'
  module '_demo'
  module 'egeoip'
    egeoip:653: ip2long_test_...ok
    egeoip:662: lookup_test_...ok
  [done in 0.112 s]
[done in 0.161 s]
=====
All 7 tests passed.
```

# common test

---

## Install

```
$ cd $ERL_ROOT/lib/common_test-1.4.5/bin  
$ ./install local  
$ sudo ln -s $ERL_ROOT/lib/common_test-1.4.5  
/priv/bin/run_test /usr/bin/run_test
```

编写 test SUITE: *geoip\_SUITE.erl*

执行 common test 时, 会自动编译 test SUITE, 执行相关的 test case, 最终生成基于 html 测试报告





# common test

---

## Run

```
$ make comm_test
Common Test v1.4.5 starting (cwd is /home/litaocheng/codes/erlips)
Eshell V5.7.3 (abort with ^G)
(ct@litaocheng)1>
Common Test: Running make in test directories...
Recompile: geoip_SUITE
./geoip_SUITE.erl:69: Warning: variable 'Config' is unused

CWD set to: "/home/litaocheng/codes/erlips/test/log/ct_run.ct@litaocheng.2009-10-12_16.24.48"

TEST INFO: 1 test(s), 1 case(s) in 1 suite(s)
Testing codes.erlips: Starting test, 1 test cases
=INFO REPORT==== 12-Oct-2009::16:24:48 ===
  application: inets
  exited: stopped
  type: temporary
Testing codes.erlips: TEST COMPLETE, 1 ok, 0 failed of 1 test cases

Updating /home/litaocheng/codes/erlips/test/log/index.html... done
Updating /home/litaocheng/codes/erlips/test/log/all_runs.html... done
```



# common test

Test Results - Vimperator

Google  搜索 书签 翻译 >> litaoc...

Test Results

[All Test Runs in this directory](#)

Name	Test Run Started	Successful	Failed	Skipped (User/Auto)	Missing Suites	Node	CT Log	Old Runs
<a href="#">codes.erlips</a>	Mon Oct 12 2009 16:24:48	1	0	0 (0/0)	0	ct@litaocheng	<a href="#">CT Log</a>	none
<b>Total</b>		<b>1</b>	<b>0</b>	0 (0/0)	<b>0</b>			

Copyright © 2009 [Open Telecom Platform](#)  
Updated: Mon Oct 12 2009 16:24:48

file:///home/litaocheng/codes/erlips/test/log/index.html [+]

[1/1] All

ERLANG

Concurrent Distributed FP Language

# common test

Test "codes.erlips" results - Vimperator

Google  搜索 ☆ >> litaoc...

Test "codes.erlips" results

**Results from test "codes.erlips"**

Test started at 2009-10-12 16:24:48

Host:  
Run by litaocheng on litaocheng  
Used Erlang 5.7.3 in /usr/local/lib/erlang.

[Full textual log](#)  
[Coverage log](#)

Suite contains 1 test cases.

Num	Module	Case	Log	Time	Result	Comment
	geoup_SUITE	<a href="#">init_per_suite</a>	< >	0.034s	Ok	
1	geoup_SUITE	<a href="#">test_egeoup</a>	< >	0.029s	Ok	
	geoup_SUITE	<a href="#">end_per_suite</a>	< >	0.001s	Ok	
	<b>TOTAL</b>			0.148s	<b>Ok</b>	1 Ok, 0 Failed of 1

file:///home/litaocheng/codes/erlips/test/log/ct\_run.ct@litaocheng.2009-10- [1/1] Top  
-- PASS THROUGH --



# release

使用 reltool 生成 target system.配置文件如下:

```
{sys,  
  [{lib_dirs,["/home/litaocheng/erlang"]},  
   {boot_rel, "erlips"},  
   {rel, "erlips", "0.1", [kernel, stdlib, sasl, mochiweb, erlips]},  
   {relocatable, true},  
   {profile, embedded},  
   {app_file, keep},  
   {debug_info, strip},  
   {mod_cond, all},  
   {incl_cond, derived},  
   {incl_app_filters, ["^include", "^priv", "^ebin", "^src"]},  
   {excl_app_filters, []},  
   {incl_archive_filters, []},  
   {excl_archive_filters, [".*"]},  
   {app, kernel, [{incl_cond, include}]},  
   {app, stdlib, [{incl_cond, include}]},  
   {app, sasl, [{incl_cond, include}]},  
   {app, erlips, [{incl_cond, include}, {incl_app_filters, [".*"]},  
                  {excl_app_filters, ["^log", "^var", "^release"]}]},  
   {app, mochiweb, [{incl_cond, include}]},  
   {app, runtime_tools, [{incl_cond, include}]}  
  ]  
}.
```



ERLANG

# release

---

使用 reltool 生成 target sytem 步骤:

1. 定义 reltool 用来生成 target system 的各种配置参数 Config
  2. reltool:start\_server(Config) 启动 reltool
  3. reltool:get\_target\_spec/1 获取根据 Config 产生的用来创建 target system 的一系列"动作" (Spec)
  4. reltool:eval\_target\_spec/3 根据 Spec 生成 target system
  5. 书写辅助的脚本, 用来生成最终的安装包 (可以为tar包, 或者zip等)
- 编写脚本 `erlips/release/gen_release` 包含以上步骤, 用来生成 target sytem.

```
# ./gen_release erlips.config
```



# release

---

## 运行

最终生成的 tar 包，包含了 erlips 运行所需的环境（包括 erl, epmd, erts 等相关的系统文件，以及依赖的各种 application, 如 kernel, stdlib, sasl 等）。

将此 tar 包分发到没有安装 Erlang 的系统中，解压。

执行 `erlipsctl` 控制 `erlips`:

```
# ./erlipsctl live # 交互方式启动
# ./erlipsctl start # 以后台方式启动
# ./erlipsctl status # 查看当前系统状态
# ./erlipsctl debug # 创建与后台启动的erlips的交互shell
# ./erlipsctl stop # 停止erlips
```



# 运行



# Erlang 更多

---

trace, trace\_pattern, dbg  
port, prot driver, c node  
appmon, pman, etop, percept  
crashviewer, et  
mnesia, fragment, global overlay  
match spec, abstract code  
cover, fprof  
application, release, appup, release handling, reltool  
erlang VM  
...





# 必读

---

Erlang FAQ

<http://www.erlang.org/faq/faq.html>

OTP Design Principles

[http://www.erlang.org/doc/design\\_principles/part\\_frame.html](http://www.erlang.org/doc/design_principles/part_frame.html)

Efficiency Guide:

[http://www.erlang.org/doc/efficiency\\_guide/part\\_frame.html](http://www.erlang.org/doc/efficiency_guide/part_frame.html)



---

Thanks For Everyone!  
Any Questions, Please  
litaocheng@gmail.com  
<http://erlangdisplay.javaeye.com>

