Erlang实战

- 构建IP查询服务

litaocheng@gmail.com ECUG IV
2009.10.24

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所属的国家,地区等相关的地理信息. 项目简洁,高效,具有很好的扩展性.
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
  Distributed 满足性能要求
- 易于扩展
  逻辑扩展 Erlang Hot code load/swap

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）
“三点”都准备齐了吧！

最重要的一点：

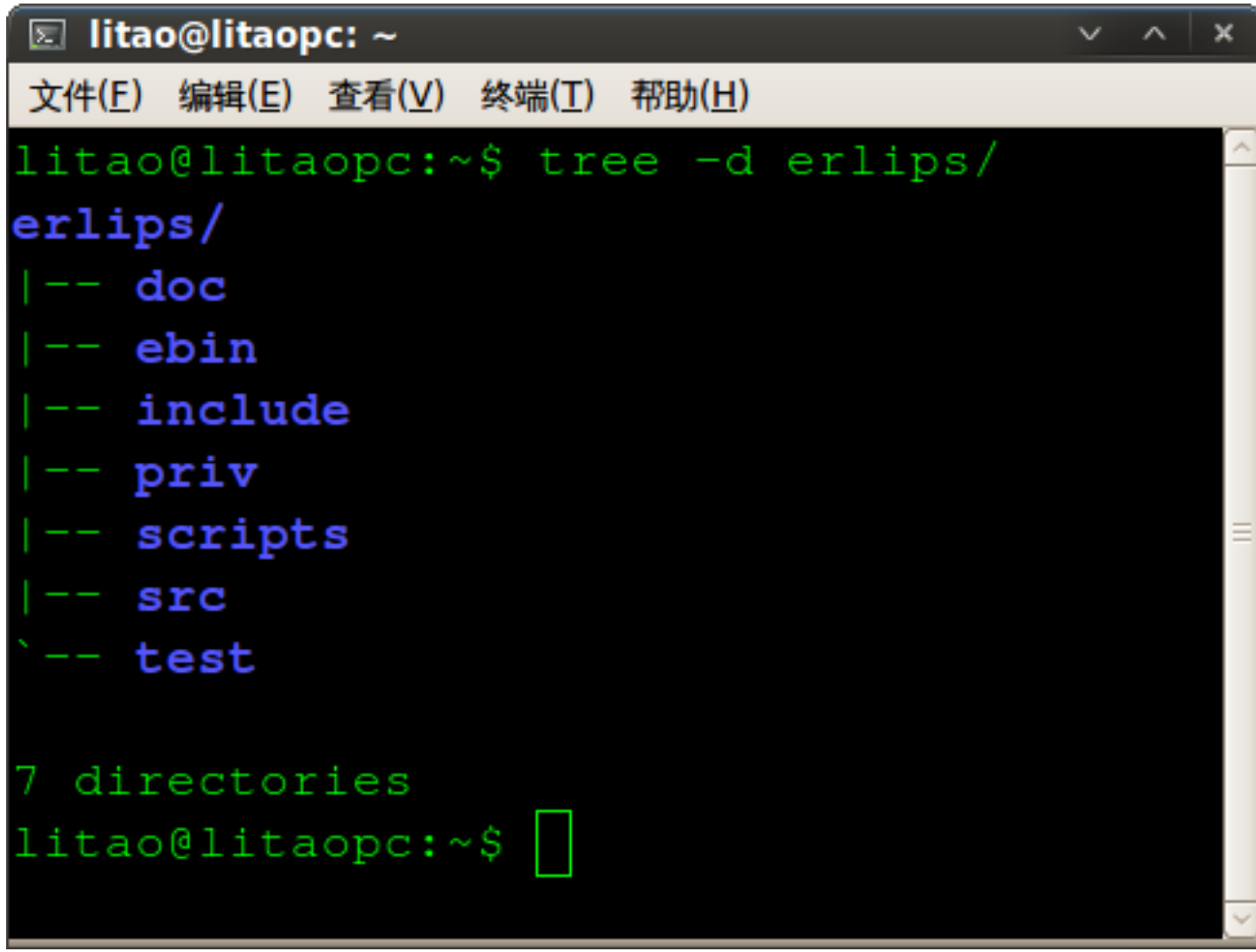
你对Erlang充满了好奇，看看它是如何完成某些工作！

项目目录结构

- srv
  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



ERLANG
Concurrent Distributed FP Language

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 behavious 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

OTP behavious gen_fsm con't

- 一个有限状态机代码框架
- 在处理 client lifetime，协议交互，server 状态等场景下使用
- 在一个独立的 process 中执行，耗时的操作会阻塞其他调用
- 对于“对象”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 behavious gen_event con't

- 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着进行操作
- 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

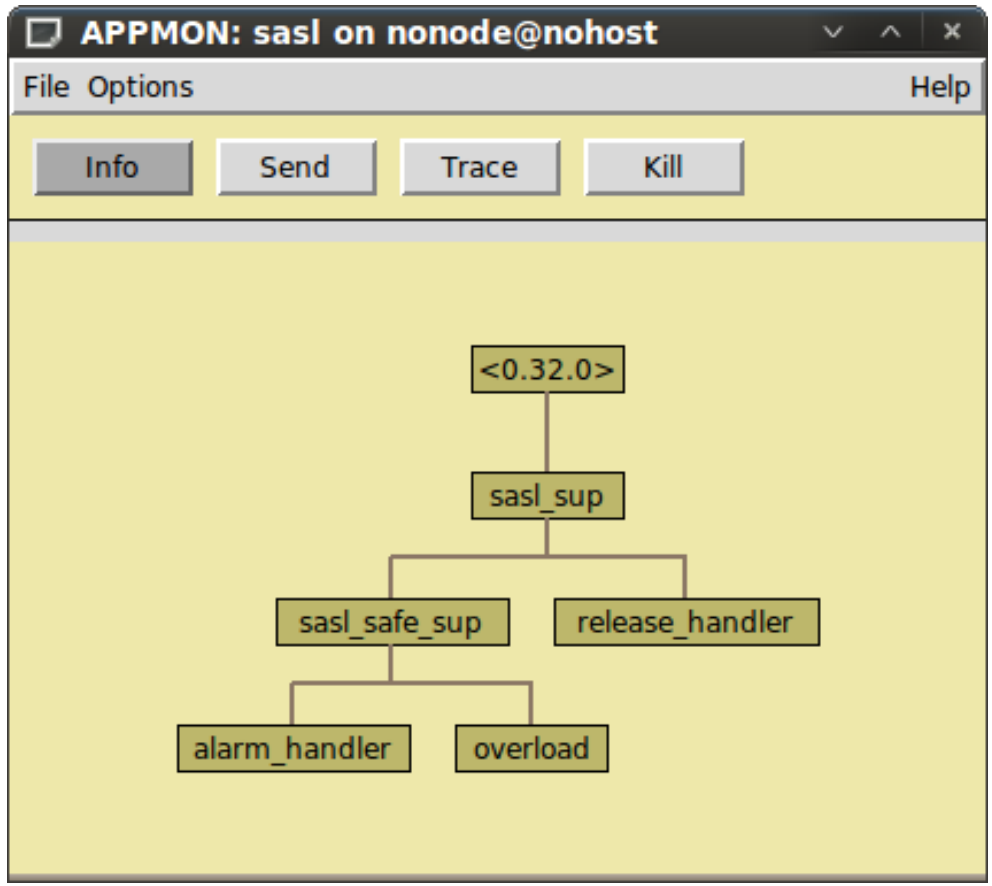
OTP behavious supervisor con't

管理 gen_server, gen_fsm, gen_event, supervisor 或其他process

- one_for_one
- one_for_all
- rest_for_one
- simple_one_for_one

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().*

ERLANG
Concurrent Distributed FP Language

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@doc相关描述，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

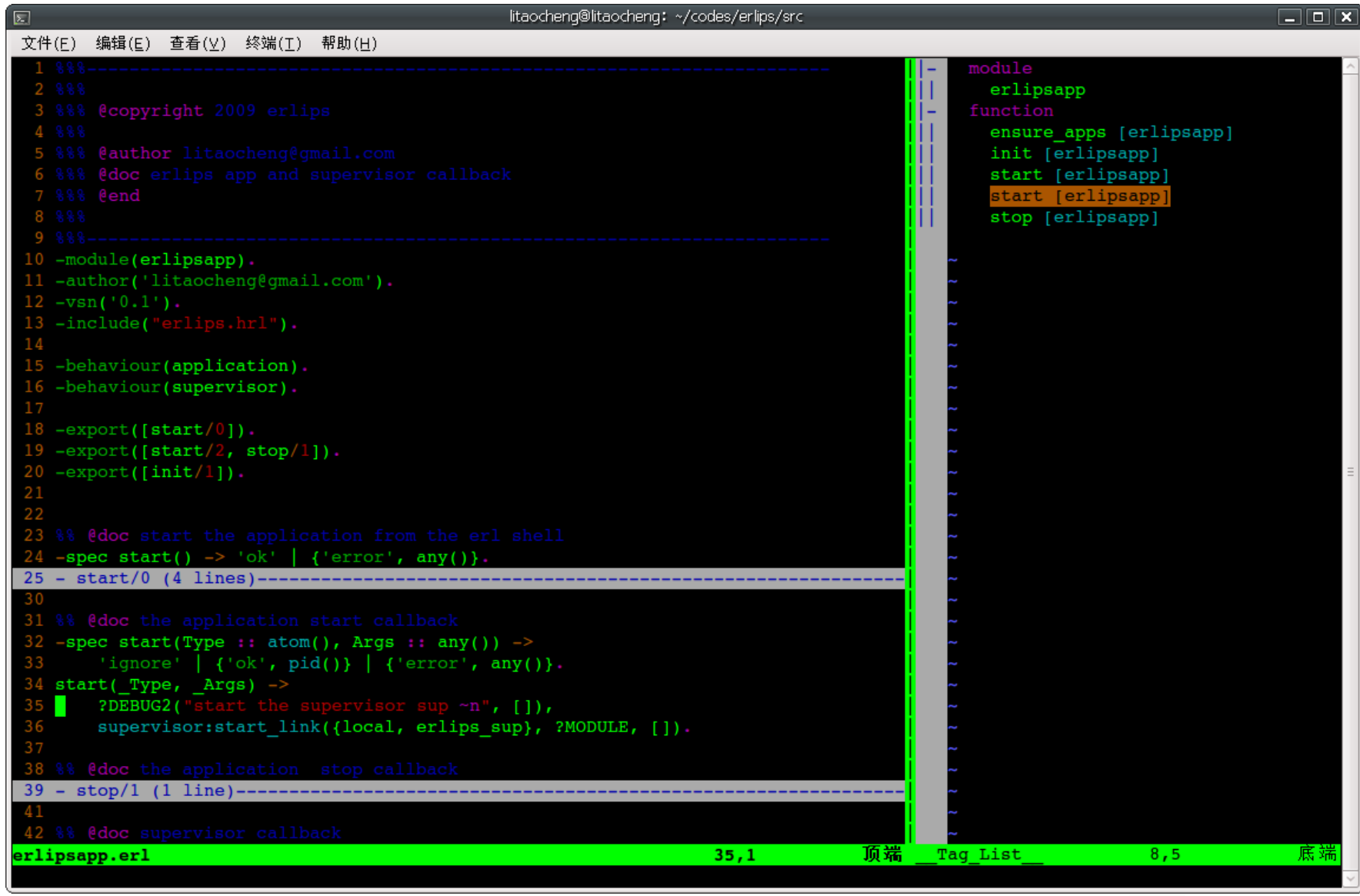
- erlipsapp.erl
  application, supervisor callback module
- erlips_httpd.erl
  httpd module, based on mochiweb
- _ips_geoip.erl
  handle module for "/ips/geoip?“ request
- egeoip.erl
  解析GeoLiteCity binary file，提供ip查询服务
- erlips_ctl.erl
  erlips ctl 对应module
- _demo.erl
- handle "/demo" path

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
    ...
```

Erlang Module In Vim



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 ...
edoc 目前(R13B02-1)对 unicode 支持不好，还有一些bug
如果doc中含有中文会产生错误，许要进行一些修正，参看
(http://www.nabble.com/UTF8-and-EDoc-td25676638.html)

edoc examples (src/_ips_geoip.erl)

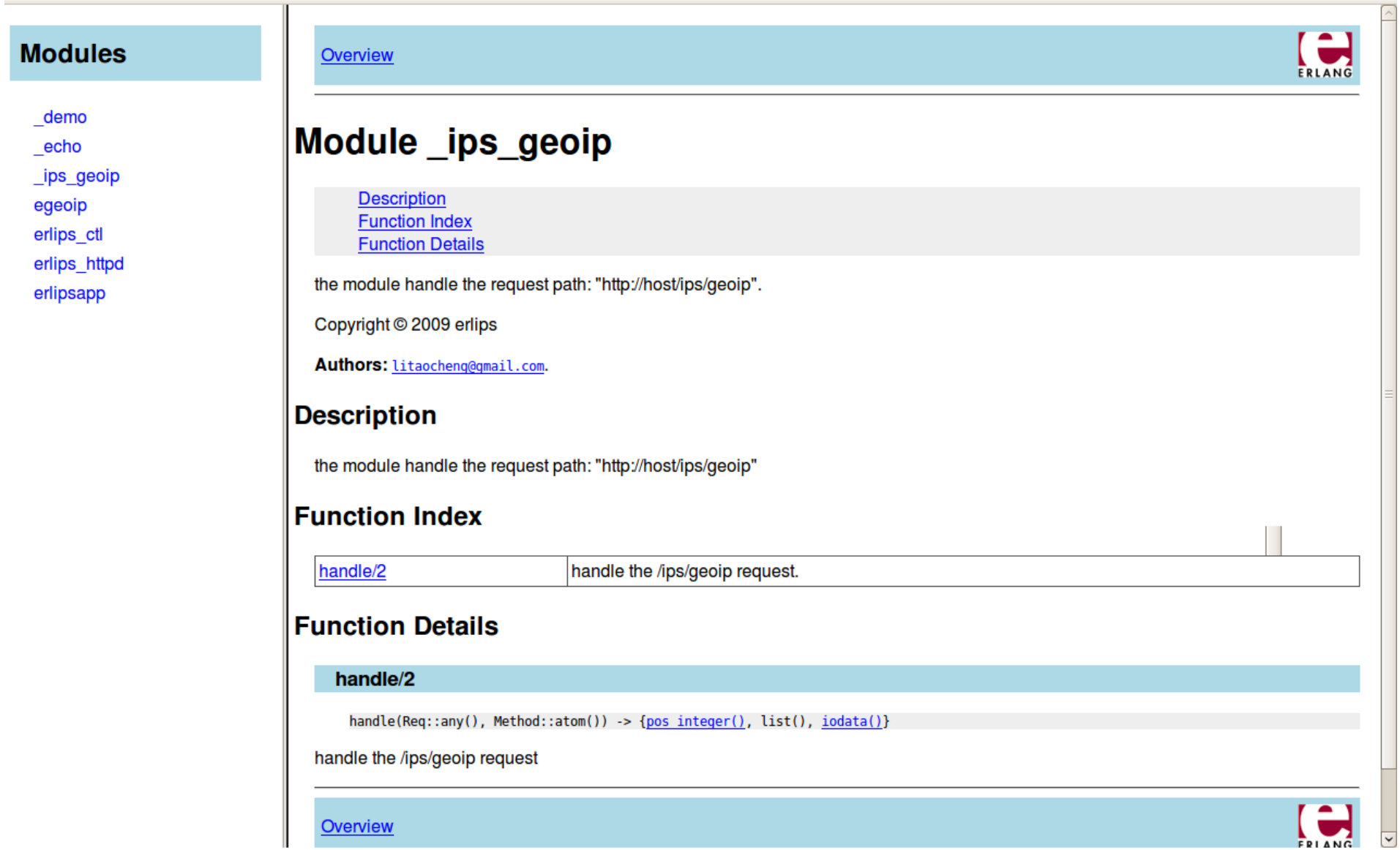
```
%%%----------------------------------------------------------------------
%%%
%%% @copyright 2009 erlips
%%%
%%% @author litaocheng@gmail.com
%%% @doc the module handle the request path:
%%%  "http://host/ips/geoip"
%%% @end
%%%
%%%----------------------------------------------------------------------

%% @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">>}.
```

ERLANG

Concurrent Distributed FP Language

edoc examples (src/_ips_geoip.erl)



Modules

- _demo
- _echo
- _ips_geoip
- egeoip
- 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.

Description

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

Function Index

handle/2	handle the /ips/geoip request.

Function Details

handle/2

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

handle the /ips/geoip request

Overview

type and spec

Type

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

Spec

定义type的目的，是为了定义函数的Specifications（函数参数，返回值类型），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

格式如下:

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

如果是在同一个 module 中，可以:

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

为提供更好的文档，可以:

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

type and spec

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）

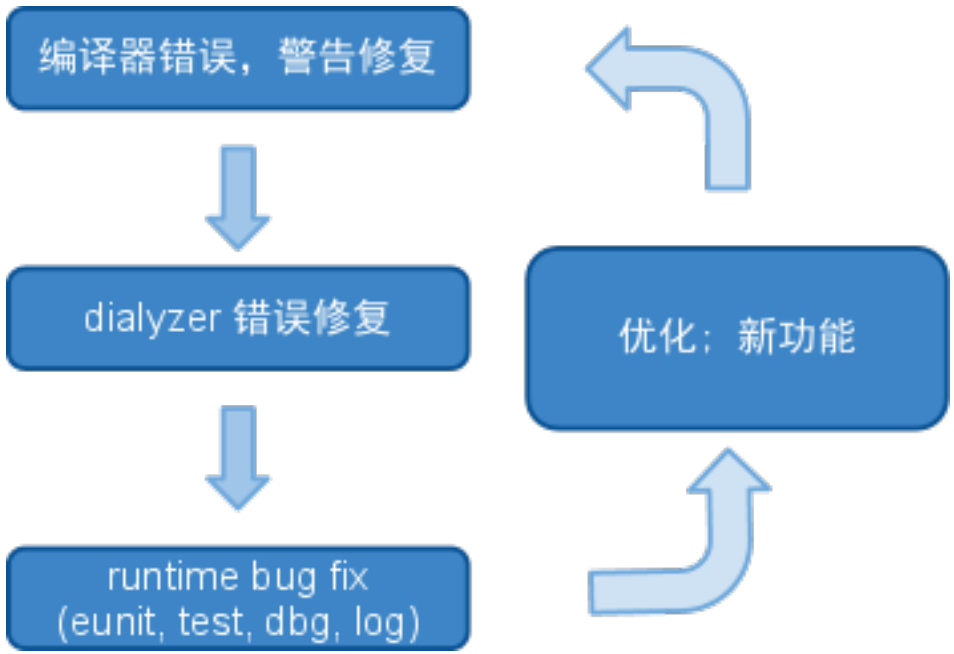
```
-spec foo(T1, T2) -> T3 ;
      (T4, T5) -> T6.
```

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



ERLANG
Concurrent Distributed FP Language

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)
```

Concurrent Distributed FP Language

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()
  ...
```

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

Note: dialyzer 仅仅分析出了一小部分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

Concurrent Distributed FP Language

release

运行

Erlang 更多

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

Concurrent Distributed FP Language