erlang学习(3)

李小红 lixiaohong@gmail.com
黄冬 huangdong@gmail.com

你将会学到

- 支持多个连接的Table Server
- 进程的创建
- Socket、进程与消息的深入处理
- 使用erlang doc

处理多个连接

simple_server的问题和改进

- 只接受了一个连接
- 改进
  - 支持多个连接
- 先不用关注
  - Table的delete
  - 并发连接

```
-module(simple_server).¬
-export([start_nano_server/0,nano_client_eval/1]).¬
¬
start_nano_server() ->¬
    {ok, Listen} = gen_tcp:listen(5555, [binary, {packet, 2},¬
                                         {reuseaddr, true},¬
                                         {active, true}]),¬
    {ok, Socket} = gen_tcp:accept(Listen),¬
    Table = ets:new(?MODULE,[set]),¬
    gen_tcp:close(Listen),¬
    loop(Socket,Table),¬
    ets:delete(Table).¬
```

开工干活

简单的改动

```
start_nano_server() ->
    {ok, Listen} = gen_tcp:listen(5555, [binary, {packet, 2},
                                         {reuseaddr, true},
                                         {active, true}]),
    Table = ets:new(?MODULE,[set]),
    seq_accept(Listen,Table).

seq_accept(Listen,Table) ->
    {ok, Socket} = gen_tcp:accept(Listen),
    loop(Socket,Table),
    seq_accept(Listen,Table).
```

处理多个并发连接

seq_server的问题和改进

- 同时只能有一个连接顺序连上来
- 改进
  - 支持并发连接

```
seq_accept(Listen,Table) ->
    {ok, Socket} = gen_tcp:accept(Listen),
    loop(Socket,Table),
    seq_accept(Listen,Table).
```

创建进程

```
spawn(Module, Name, Args) -> pid()
  Module = Name = atom()
  Args = [Arg1,...,ArgN]
    ArgI = term()
```

模块

参考

ERLANG

Glossary | Modules | Index

Release highlights
Potential incompatibilities
Expand All | Contract All

- Erlang/OTP
- Erlang Programming
  - Getting Started
  - Erlang Reference Manual
  - Programming Examples
  - Efficiency Guide
  - Interoperability Tutorial
- Working with OTP
- Basic Applications
- Database Applications
- Operation & Maintenance Applications
- Object Request Broker & IDL Applications
- Interface and Communication Applications
- Tool Applications

Erlang/OTP R12B

Welcome to Erlang/OTP, a complete development environment for concurrent programming.

Some hints that may get you started faster

- The complete Erlang language is described in the Erlang Reference Manual. An Erlang tutorial can be found in Getting Started With Erlang.

  In addition to the documentation here Erlang is described in the book "Programming Erlang", ISBN 978-1-934356-00-5, which we highly recommend as a start.

- Erlang/OTP is divided into a number of OTP applications. An application normally contains Erlang modules. Some OTP applications, such as the C interface *erl_interface*, are written in other languages and have no Erlang modules.

  Note that functions that are not imported or prefixed with a module name belong to the module erlang (in the *Kernel* application).

- On a Unix system you can view the manual pages from the command line using

  ```
  % erl -man <module>
  ```

- You can of course use any editor you like to write Erlang programs, but if you use Emacs there exists editing support such as indentation, syntax highlighting, electric commands, module name verification, comment support including paragraph filling, skeletons, tags support and more. See the Tools application for details.

  There is also an Erlang plugin (Erllde) for Eclipse if you prefer a more graphical environment. Erllde is under development and should at the time of writing be quite stable and useful.

- When developing with Erlang/OTP you usually test your programs from the interactive shell (see Getting Started With Erlang) where you can call individual functions. There is also a number of tools available, such as the graphical Debugger, the process manager Pman and table viewer TV.

  Also note that there are some shell features like history list (control-p and control-n), in line editing (emacs key bindings) and module and function name completion (tab) if the module is loaded.

- OpenSource users can ask questions and share experiences on the Erlang questions mailing list.

把loop给spawn了

修改代码

```
seq_accept(Listen,Table) ->
    {ok, Socket} = gen_tcp:accept(Listen),
    spawn(fun() -> loop(Socket,Table) end),
    seq_accept(Listen,Table).
```

出问题了

- 客户端执行后，Server/Client都没有任何反映
- 为什么?

进程与消息

- loop在一个新的进程里，它收不到Socket回来的消息
- 消息还是放在了spawn之前的进程消息队列中

介绍一个新朋友

Socket消息转移

controlling_process(Socket, Pid) -> ok | {error, eperm}

Types:

Socket = socket()
Pid = pid()

将消息转到新进程

```
seq_accept(Listen,Table) ->
    {ok, Socket} = gen_tcp:accept(Listen),
    Pid = spawn(fun() -> loop(Socket,Table) end),
    gen_tcp:controlling_process(Socket,Pid),
    seq_accept(Listen,Table).
```

出错了

```
HD@~/work/xbaytable/erlangtut/ch3$erl
Erlang (BEAM) emulator version 5.6 [source] [smp:2] [async-threads:0] [kernel-
poll:false]

Eshell V5.6  (abort with ^G)
1>  para_server1:start_nano_server().
Server received binary = <<131,104,3,100,0,3,115,101,116,107,0,2,72,68,107,0,2,
                           72,68>>
Server (unpacked) {set,"HD","HD"}

=ERROR REPORT==== 28-Dec-2007::17:54:16 ===
Error in process <0.33.0> with exit value: {badarg,[{ets,insert,[15,{"HD","HD"}]},
{para_server1,handle_cmd,2},{para_server1,loop,2}]}
```

ets不能跨进程

还有新朋友

erlang包中的字典

get(Key) -> Val | undefined

Types:

Key = Val = term()

Returns the value Valassociated with Key in the process dictionary, or undefined if Key does not exist.

```
> put(key1, merry),
put(key2, lambs),
put({any, [valid, term]}, {are, playing}),
get({any, [valid, term]}).
{are,playing}
```

改造

```
    seq_accept(Listen).

seq_accept(Listen) ->
    {ok, Socket} = gen_tcp:accept(Listen),
    Pid = spawn(fun() -> loop(Socket) end),
    gen_tcp:controlling_process(Socket,Pid),
    seq_accept(Listen).

loop(Socket) ->
    receive
        {tcp, Socket, Bin} ->
            io:format("Server received binary = ~p~n" ,[Bin]),
            Str = binary_to_term(Bin),
            io:format("Server (unpacked) ~p~n" ,[Str]),
            Reply = handle_cmd(Str),
            io:format("Server replying = ~p~n" ,[Reply]),
            gen_tcp:send(Socket, term_to_binary(Reply)),
            loop(Socket);
        {tcp_closed, Socket} ->
            io:format("Server socket closed~n" )
    end.
```

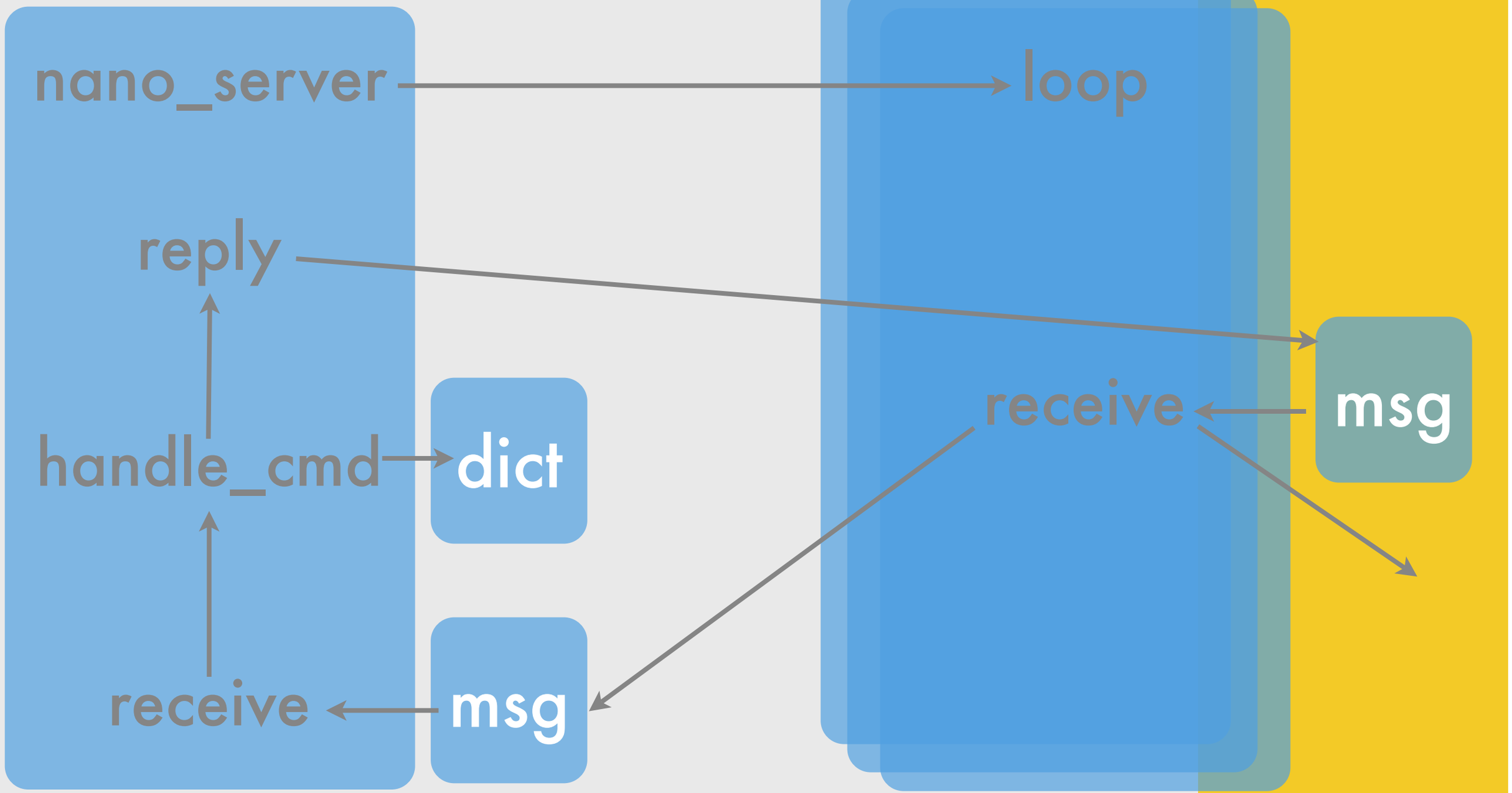
```
handle_cmd({set,Name,Value}) ->
    erlang:put(Name,Value),
    ok;

handle_cmd({get,Name}) ->
    case erlang:get(Name) of
        undefined ->
            {error,notfound};
        Value ->
            {ok,Value}
    end;

handle_cmd({delete,Name}) ->
    erlang:erase(Name),
    ok;

handle_cmd(Cmd) ->
    {error, Cmd}.
```

出错，为什么?

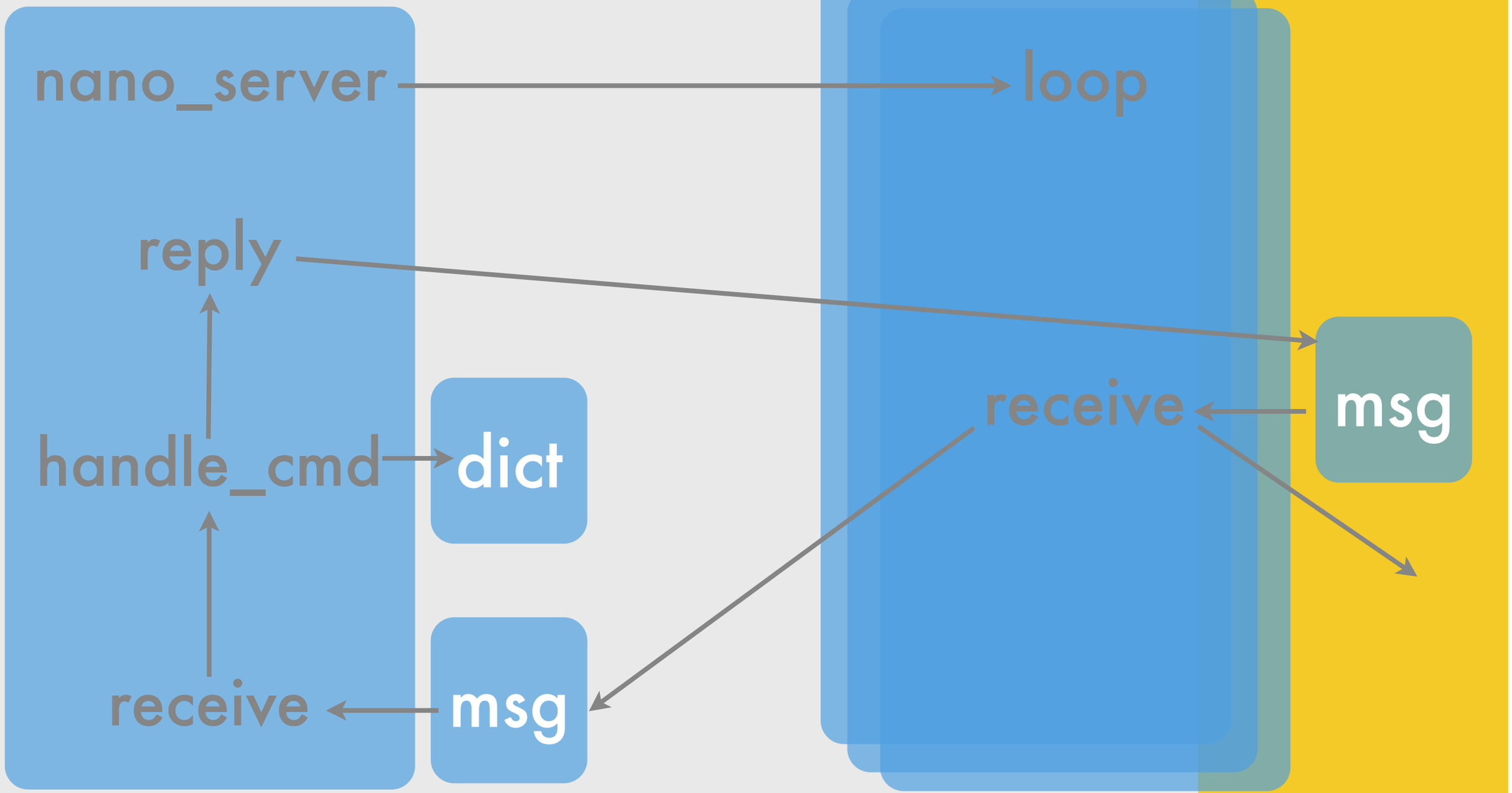
```
HD@~/work/xbaytable/erlangtut/ch3$erl
Erlang (BEAM) emulator version 5.6 [source] [smp:2] [async-threads:0] [kernel-
poll:false]

Eshell V5.6  (abort with ^G)
1> para_server2:nano_client_eval({set,"HD","HD"}).
Client received binary = <<131,100,0,2,111,107>>
Client result = ok
ok
2> para_server2:nano_client_eval({get,"HD"}).
Client received binary = <<131,104,2,100,0,5,101,114,114,111,114,100,0,8,110,
                           111,116,102,111,117,110,100>>
Client result = {error,notfound}
ok
3>
```

消息、进程大汇总

处理者与传输者


nano_server
loop
reply
handle_cmd
dict
receive
msg
receive
msg
receive
msg

处理者与传输者


nano_server
loop
reply
handle_cmd
dict
receive
msg
receive
msg
receive
msg

巨大的工程

- 发送消息
  - Pid ! msg
- receive 别忘记递归
- 区别对待各个来源

看代码-loop

```
loop(Socket) ->
    receive
        {tcp, Socket, Bin} ->
            io:format("Server received binary = ~p~n" ,[Bin]),
            Str = binary_to_term(Bin),
            io:format("Server (unpacked) ~p~n" ,[Str]),
            Selfid = self(),
            ?MODULE ! {Selfid,Str},
            receive
                {Selfid,Reply} ->
                    io:format("Server replying = ~p~n" ,[Reply]),
                    gen_tcp:send(Socket, term_to_binary(Reply))
            end,
            loop(Socket);
    {tcp_closed, Socket} ->
        io:format("Server socket closed~n" )
end.
```

nano-server的处理

```
start_nano_server() ->
    {ok, Listen} = gen_tcp:listen(5555, [binary, {packet, 2},
                                         {reuseaddr, true},
                                         {active, true}]),
    register(?MODULE,self()),
    spawn(fun() -> seq_accept(Listen) end),
    process_loop().

process_loop() ->
    receive
        {Pid,Str} ->
            Reply = handle_cmd(Str),
            Pid ! {Pid,Reply},
            process_loop()
    end.
```

我们已经留下了三个脚印