介绍 CouchDB 的 couch_ref_counter 模块

1. 是一个 gen_server，内部保存这样一个 record 作为 state.

```
-record(srv,
    {
    referrers=dict:new(), %% 记录格式: {Pid, {MonitorRef, Count}}
    child_procs=[]        %% 子进程列表
    }).
```

<1> child_procs 于 gen_server 进程相关连，当 gen_server 进程启动的时候，会 link/1 到所有的 child processes; 当 gen_server 进程停止(terminate)的时候，会停止所有的子进程后，然后自己退出.

<2> refferers 是一个 dictionary()，内部存储的是 Key/Value 对，也就是监控进程的引用信息.

```
Key = Pid
Value = {MonitorRef, Count}
```

dict 中的所有的 Key(也就是 Pid)，都会被这个 gen_server 进程监控 erlang:monitor/2

a. 当 count 大于等于 1 的时候，gen_server 进程监控这个 Pid，我们可以通过 add/1，add/2，drop/1，drop/2 APIs 来修改 Pid 对应的 count 的值;

b. 当 count 小于 1 的时候，gen_server 进程将调用 erlang:demonitor(MonitorRef, [flush])取消对这个 Pid 的监控，并同时把这个 Pid 对应的记录从 dict 中删除.

c. 当 gen_server 进程监控的进程退出的时候，gen_server 进程会收到'DOWN'消息，会把这个退出进程的记录从 dict 中删除(保持 dict 中的记录于 monitor 的进程信息同步).

d. 当这个 gen_server 没有监控任何进程的时候，也就是 process_info(self(), monitors)返回[]的时候，gen_server 也会退出. 也就是说 gen_server 进程运行的条件是至少有一个 Pid 被这个 gen_server 进程所监控.

2. gen_server 的启动:

我们可以通过 couch_ref_counter:start(ChildProcs)来启动这个 gen_server，gen_server 启动的时候会做两件事情:

a. 连接(link)到所有的 child processes

b. 构造一个新的 dict，并插入{Pid, {MonitorRef, 1}}这样一条新记录
   (其中 Pid 就是 count_ref_count:start/1 调用进程的 Pid).

启动之后，gen_server 进程开始运行，并且监控一个进程，也就是它的调用进程.

3. 小技巧:

<1> 如何获得当前进程监控 erlang:monitor/2 的进程的信息?

```
Erlang:process_info(self(), monitors)
```

<2> 如何同步的停止一个进程?

```
shutdown_sync(Pid) when not is_pid(Pid)->
    ok;
shutdown_sync(Pid) ->
    MRef = erlang:monitor(process, Pid),
    try
        catch unlink(Pid),
        catch exit(Pid, shutdown),
        receive
        {'DOWN', MRef, _, _, _} ->
            ok
        end
    after
        erlang:demonitor(MRef, [flush])
    end.
```