介绍 CouchDB 的配置文件管理模块

1．用来管理配置文件，可以对配置文件进行读写操作，涉及两个模块：
couch_config_writer 和 couch_config

2．配置文件本质上是一个 Key/Value 对，其中 Key 是 Section 加上一个可选的 Option 组成．
当插入数据的时候，简单说如果 Key = Section+Option 存在，则用新的 Value 替换老的；
如果 Section+Option 不存在，则插入新数据(虽然 Option 是可选的，但我们在使用的时候，
约定为 Option 指定一个非空的值)．

3．使用 couch_config_writer 来写数据到配置文件
该模块之提供了一个 API，用来写入配置信息到配置文件中．

```
@spec save_to_file(
             Config::{{Section::string(), Option::string()}, Value::string()},
             File::filename()) -> ok
```

注意：
这个 API 在使用的时候必须保证 File 文件是存在的，否则会抛出下面 Exception：

```
exception error: no match of right hand side value {error,enoent}
```

例子：

```
couch_config_writer:save_to_file({{"sec1", "key1"},"val1"}, "file1").
couch_config_writer:save_to_file({{"sec1", ""},"val"}, "file1"). %% 不建议这样使用
couch_config_writer:save_to_file({{"sec1", "key2"},"val2"}, "file1").
couch_config_writer:save_to_file({{"sec2", ""},"abc"}, "file1").  %% 不建议这样使用
couch_config_writer:save_to_file({{"sec2", "xxxxkey"},"xxxxval"}, "file1").
```

产生的 file1 文件内容如下：

```
$file1

[sec1]
key1 = val1
 = val                %% 在解析的时候会被忽略
key2 = val2

[sec2]
 = abc                %% 在解析的时候会被忽略
xxxxkey = xxxxval
```

4．使用 couch_config 来解析配置文件
我们可以使用下面这个 API 来单独解析配置文件:
```
@spec parse_ini_file(IniFile :: string()) -> {ok, [tuple()]}.
```

例如(使用这个 API 解析 3 中生成的 file1 文件):
```
couch_config:parse_ini_file("file1").
{ok,[{{"sec2","xxxxkey"},"xxxxval"},
     {{"sec1","key2"},"val2"},
     {{"sec1","key1"},"val1"}]}.
```

注意:
解析结果中 Option=""的数据被解析器忽略，这也就是我们为什么建议在使用 couch_config_writer 写数据到配置文件的时候 Option 不为空，因为 Option 为空的数据会被解析器忽略掉.
实质上，Val=""的数据也会被解析器忽略，所以我们在使用解析模块的时候，因该保证 Section，Option，Value 三部分都非空.

5．couch_config 支持同时解析多个配置文件，它连续的解析这些配置文件中的 key/value 对，然后把它们统一存储在一张 ets 表中，以{Sec，Option}作为 Key 存储，如果指定了多个配置文件，则后面配置文件的信息会"覆盖"前面配置文件的信息(ets 表的写入机制而已:)，在这种情况下最后一个配置文件会存储 set，delete 对应的修改，也就是 set，delete 的修改配置信息的操作会同步到配置文件中.

启动 & 停止
```
couch_config:start_link(IniFiles :: [string()]) -> {ok, pid()}.
couch_config:stop().
```

6．对于配置文件的操作，我们可以通过 get，set，delete API 来进行操作.

7．set，delete 回调函数 & 回调进程
这个模块提供了两个 API 可以用来指定 set，delete 操作时候的 callback function，也就是对配置文件进行修改的时候的回调函数，一个回调函数对应一个回调进程，当执行 set，delete 操作的时候，如果回调模块对应的进程仍然存活，则会调用:
Fun(Sec，Key，Val，Persist)来执行一次回调.
注册的回调函数参数可以从 1-4，如果参数为 1，则 key，Val，Persist 参数在回调的时候会被忽略，以此类推.

回调进程实质上是通过 erlang:monitor(process，Pid)来被 couch_config 这 gen_server 进程监控的，如果回调进程死亡，则 couch_config 进程会收到{'DOWN'，_，_，DownPid，_}消息，从而把回调进程对应的回调函数从回调函数列表中删除.

我们可以注册多个回调函数，在 set，delete 的时候这些回调函数都会被调用.

Persist 是为回调函数引入的一个概念，默认为 true；这个数据不存储，只是在回调函数调用的时候作为参数传递.

```
@spec register(Fun :: fun()) -> ok.  %% 默认是调用该函数的进程是回调模块对应的进程
@spec register(Fun :: fun(), Pid :: pid()) -> ok.
```

例如:

```
couch_config:start_link(["file1"]).
{ok,<0.95.0>}
couch_config:register(fun(A,B,C,D) -> io:format("Section:~p, Key:~p, Val:~p,
Persist:~p~n", [A,B,C,D]) end). %% 注册一个四个参数的回调函数
ok
couch_config:set("sec1", "newkey1", "newval1").
Section:"sec1", Key:"newkey1", Val:"newval1", Persist:true  %% 触发回调函数
ok
couch_config:set("sec1", "newkey1", "newval1", "mypersist").
Section:"sec1", Key:"newkey1", Val:"newval1", Persist:"mypersist" %% 触发回调函数
ok
couch_config:delete("sec1", "key111").  %% 在删除数据的时候 Value 都是是'deleted'
Section:"sec1", Key:"key111", Val:deleted, Persist:true
ok
couch_config:delete("sec1", "key111", "ddasdf").
Section:"sec1", Key:"key111", Val:deleted, Persist:"ddasdf"
ok
```