Dingo

Dindo Document

DingoLab

May,2016

Dingo Dindo Document

李约瀚
qinka@live.com
14130140331

May 26th 2016

Version 0.0.3.0
西安, Xi‘an
DingoLab

前言

这个文档是 Dingo 后端 Dindo 的文档，包括后端的大体需求说明，宏观设计说明、详细设计说明、数据库设计与实现、软件源码说明、软件测试说明、软件部署说明件与软件使用说明。

后端 Dindo 使用 Haskell [1]，与 Yesod 框架 [2] 编写的。同时整个后端代码中 Haskell 的部分是使用 Haskell 与 LaTeX 混排的文学编程。所以文档中有一部分为程序代码（及其说明）。

Dindo 的名称由来是在笔者（也是主要维护者）在数学建模的校赛是，使用 Lingo 是受到 Lingo 与 Lindo 的关系而起的名字。

这个后端依次将介绍需求、设计、数据库设计、软件部署、软件使用与维护、Dindo 代码及其说明等内容，以上是正文部分。附录中将会有部分术语表、维护的文档、索引、参考文档等。

[1] Haskell 是一门纯函数式的编程语言。

[2] Yesod 是一个使用 Haskell 作为主要语言，的 RESTful API 的 WEB 应用框架。

目录

1 大体需求说明

2 Dindo 架构设计概论

Dindo 是 Dingo 的核心部分之一，负责客户端与后端的交互，同时负责客户端与数据库的、客户端之间的间接交互。此部分将有负载均衡的大致方法、弹性计算的解决方案、后端 API 与服务程序分割的内容。同时还将说明后端业务流程。

Dindo 是基于 Docker 容器上，采用微服务架构的一个后端。所有的组件将运行与 Docker 容器之中，且方便运行与公有云搭建的 Docker 中，同时价格相对比较便宜。按照灵雀云的收费标准 [1]，按照北京一区（AWS）来计算。当不使用弹性计算中的策略，即仅当容器的大小与数量时确定不变时。负载均衡负载的采用一个 M 级别的容器，运行 5 个 L 级别的容器作为数据库，运行 20 个的 M 级别的容器为处理业务的核心部分。数据库每个容器配置 100G 的挂载点用于存放数据，并计划每天下载数据量有 10G。按上述配置需要[3]

$$((20+1)*0.329+5*0.658)*24*30+10*30*0.93+0.75*100*5=7997.28$$

每个月大致需要不到 8000 元的成本[4]。

Dindo 开发过程依赖敏捷开发，并采用以持续集成为主的测试方式测试，同时采用持续交付的方式交付运营者。由于采用微服务架构、持续交付与 Docker 可以使得后端的版本升级处于“无痛”状态。微服务架构也能使的后端的业务逻辑分布在不同的程序（组件），也可使得后端分布上线。

3 均衡负载设计

均衡负载采用 Nginx 作物负载均衡的软件，

[3] 一个月按 30 天计算。

[4] 当采用弹性计算时，这个成本将继续下降

4 弹性计算设计

5 微服务架构设计

6 业务流程说明

业务流程部分包括后端对事件驱动型的业务处理过程，每个 API 中业务处理过程等。这部分的主要内容将在 Dindo 源码及其结束的部分说明。

7 数据库设计

8 Dindo 部署说明

此部分主要说明 Dindo 的部署问题，包括测试、原型与最后实际运行是的部署。测试与原型的部署有两种方式，一种是直接运行，另一种是基于 Docker [5] 。而最后运营是的部署，目前计划直接部公有云之上，利用 CaaS 服务。

8.1 测试部署方式

测试的部署一般适用于调试与检测。调试一方面是指后端开发时测试验证，另一方面则是指前端开发时测试使用。检测是如安全性测试等方面的检测。而通常运营部署通常不需要调试磨合，直接部署到 CaaS 提供商即可。

8.1.1 原生运行

原生运行首先要构建[6] 然后部署，最后运行。如果已获得构建好的二进制文件，请直接跳过下面构建的过程。

Windows 下的构建 首先需要安装 Haskell Platform 7.10.3 x64，然后克隆 GitHub/DingoLab/DingoBackend 仓库到本地，然后安装 stack，安装方式可参考 Stack Install & Upgrade 。安装完之后跳转到仓库的目录：

```
$ cd DingoBackend
```

[5] 基于的是 Ubuntu（Linux）原声的 Docker，暂不讨论 Mac OS X 与 Windows 下原生的 Docker。

[6] Dindo 是不直接发行二进制文件的，发行的只有 Docker 镜像。

然后执行构建：

```
$ stack build
```

然后在.stack_work 文件夹中某个文件夹下面的 bin 文件夹中可以找到编译好的二进制文件 [7]。

Linux 下的构建 首先安装 GHC[8]。安装的方式通常通过

Max OS 下的构建 部署的方式分为两部分：后端组件与数据库。由于处于测试的目的，并不需要使用均衡负载与法务发现的部分。所以直接载入配置文件就可以启动。对于数据库，要求是实用 PostgreSQL 数据库，并使用 dindo-database 模块中的 SQL 文件初始化数据库并使用。

后端模块的启动 无论是在那个系统下，当获得某个模块的二进制文件时。运行这个文件再将配置传入即可。通常在 UNIX Shell [9] 或与之类似的 Shell 环境中[10] 以用户管理模块为例，假设文件 config.yml 为 YAML 格式的配置文件，则输入如下：

```
$ cat config.yml | dindo-um --form=yaml
```

就可以启动用户管理部分的模块。其中 config.yml 文件的内容如下

```
port: 3000
database-config:
  addr: '192.168.1.224'
  port: '5432'
  user: postgres
  name: dingo
  con-limit: 10
  password: abcdefg
```

其中 port 是指该模块侦听的端口，database-config 部分是数据库的配置。由上到下依次是：数据库地址、数据库侦听端口、数据库用户名、数据库名称、数据库连接数限制与用户密码。启动配置还可以是 JSON 格式：

[7] 为何不直接搜索。

[8] 要求 7.10 以上，之前的版本没有测试过，无法保证可以正常编译运行。

[9] 比如 Bash、Zsh 等。

[10] 例如 Windows 下的 PowerShell。

```
{ "port":3000
, "database-config":
  { "addr" : "192.168.1.224"
  , "port" : "5432"
  , "user" : "postgres"
  , "name" : "dingo"
  , "con-limit" : 10
  , "password" : "johnjing"
  }
}
```

同时启动的命令是:

```
$ cat config.json | dindo-um
```

其中默认的文件格式是 JSON，然而推荐使用 YAML 的格式。同时还可以直接执行可执行文件，然后通过标准输入键入，然后输入文件结束符 EOF [11]。

9 Dindo 软件使用与维护说明

10 Dindo 源码及说明

这一部分是关于 Dindo 源代码及其解释说明。

10.1 dindo-database

这一部分的功能是数据库驱动与数据库内容的表示。

10.1.1 src/Dindo/Database.lhs

数据库内容

```
{-# LANGUAGE TemplateHaskell
           , FlexibleInstances
           , TypeFamilies
```

[11] Windows 下按 Ctrl + Z，Linux 与 Mac 按 Ctrl + D

```
4               , MultiParamTypeClasses
5               , GADTs
6               , GeneralizedNewtypeDeriving
7               , QuasiQuotes
8               #-}
```

```
9  module Dindo.Database where
```

```
10      import Prelude hiding (String)
11      import Import
12      import Data.Text
13      import Data.ByteString
14      import Paths_dindo_database
15      import Data.Version
```

```
16      instance FromJSON ByteString where
17        parseJSON (String x) = pure $ encodeUtf8 x
18      instance ToJSON ByteString where
19        toJSON = String . decodeUtf8
```

```
20      share [mkPersist sqlSettings] [persistLowerCase|
21      Account json sql=table_account
22        Id sql=
23        uid Text sql=key_uid sqltype=varchar(64)
24        pash Text sql=key_pash sqltype=varcher(64)
25        tel Int sql=key_tel
26        name Text sql=key_name sqltype=varchar(64)
27        Primary uid
28        deriving Show Eq
29      Usr json sql=table_usr
30        Id sql=
31        uid Text sql=key_uid sqltype=varchar(64)
32        email Text sql=key_email
33        rname Text sql=key_rname sqltype=varchar(64)
```

```
34          prcid Text sql=key_prcid sqltype=varchar(18)
35          addr Text sql=key_addr
36          status Text sql=key_status sqltype=varchar(1)
37          Primary uid
38          Foreign Account fkuid uid
39          deriving Show Eq
40        Addr json sql=table_addr
41          Id sql=
42          aid Text sql=key_aid sqltype=varchar(64)
43          uid Text sql=key_uid sqltype=varchar(64)
44          zip Text sql=key_zip sqltype=varchar(64)
45          addr Text sql=key_addr
46          Primary aid
47          Foreign Account fkaddruid uid
48          deriving Show Eq
49        Apic sql=table_apic
50          Id sql=
51          pid Text sql=key_pic_id sqltype=varchar(64)
52          uid Text sql=key_uid sqltype=varchar(64)
53          bpic ByteString sql=binary_pic
54          typ Int Maybe sql=key_status default=0
55          Primary pid
56          Foreign Account fkuidb uid
57          deriving Show Eq
58        Task json sql=table_task
59          Id sql=
60          tid Text sql=key_tid sqltype=varchar(64)
61          ca Text Maybe sql=key_ca sqltype=varchar(64)
62          cb Text Maybe sql=key_cb sqltype=varchat(64)
63          Primary tid
64          Foreign Account fkca ca
65          Foreign Account fkvcb cb
66          deriving Show Eq
67        Taskinfo json sql=table_task_info
```

```
68          Id sql=
69          tid Text sql=key_tid sqltype=varchar(64)
70          ew Double sql=key_ew
71          ns Double sql=key_ns
72          r Double sql=key_r
73          w Double sql=key_w
74          size [Double] sql=key_size
75          note Text Maybe sql=key-note
76          cost Int sql=key_note
77          des Text Maybe sql=key_des
78          Primary tid
79          Foreign Task fktid tid
80          deriving Show Eq
81        Taskcost json sql=table_task_cost
82          Id sql=
83          tid Text sql=key_tid sqltype=varchar(64)
84          ad [Int] sql=key_ad
85          bd [Int] sql=key_bd
86          Primary tid
87          Foreign Task fktidb tid
88          deriving Show Eq
89        Dd json sql=table_dd
90          Id sql=
91          did Text sql=key_did sqltype=varchar(64)
92          uid Text sql=key_tid sqltype=varchar(64)
93          dd Text sql=key_dd
94          ew Double sql=key_ew
95          ns Double sql=key_ns
96          r Double sql=key_r
97          Primary did
98          Foreign Account fkuidc uid
99        TmpToken json sql=table_tmptoken
100         Id sql=
101         tt Text sql=key_tmptoken sqltype=varchar(150)
```

```
102             time UTCTime sql=key_timeup
103             uid Text sql=key_uid sqltype=varchar(64)
104             Primary tt
105             Foreign Account fkuidd uid
106           |]
```

```
107         dindo_database_version = version
108         dindo_database_version_quasi = stringE $ showVersion version
```

10.1.2 src/Import.lhs

用于本模块的导入内容，不导出

```
1  module Import
2  ( module X
3  , persistFileWithC
4  ) where
```

```
5  import Language.Haskell.TH as X
6  import Data.Aeson as X
7  import Database.Persist as X
8  import Data.Text.Encoding as X
9  import Database.Persist.TH as X
10 import Database.Persist.Quasi as X
11 import Data.Time as X
```

```
12 persistFileWithC :: PersistSettings
13                  -> FilePath
14                  -> Q Exp
15 persistFileWithC s = persistFileWith s.("../dindo-config/"++)
```

10.2 dindo-common

这一部分是 dindo 各个微组件使用的基础公共设施。

10.2.1 src/Dindo/Import.lhs

这个系列的模块是用来导入模块的，以减少代码重复度

```haskell
module Dindo.Import
    ( module X
    ) where
```

```haskell
      import Data.Maybe as X
      import Data.Time as X
      import Dindo.MicroFramework.Register as X
      import Dindo.MicroFramework.Destory as X
      import Dindo.MicroFramework.API as X
      import Data.Conduit as X
```

10.2.2 src/Dindo/Import/Aeson.lhs

导入 Data.Aeson 及相关内容

```haskell
module Dindo.Import.Aeson
    ( module X
    ) where
      import Data.Aeson as X
```

10.2.3 src/Dindo/Import/ByteString.lhs

导入 bytestring 包中相关模块

```haskell
module Dindo.Import.ByteString
    ( module X
    , fromStrictBS
    ) where
```

```haskell
      import Data.ByteString as X
      import Data.ByteString.Lazy
      fromStrictBS = fromStrict
```

10.2.4 src/Dindo/Import/Database.lhs

导入与数据库相关的模块

```
{-# LANGUAGE TypeFamilies #-}
```

```
module Dindo.Import.Database
    ( module X
    , tryRunDB
    ) where
```

```
      import Database.Persist as X
      import Database.Persist.Postgresql as X
      import Dindo.Database as X
      import Control.Exception
      import Yesod
      tryRunDB :: ( Yesod site
                  , YesodPersist site
                  , YesodPersistBackend site ~ SqlBackend
                  )
                => YesodDB site a -> HandlerT site IO (Either SomeException a)
      tryRunDB f = do
        runInnerHandler <- handlerToIO
        liftIO $ try $ runInnerHandler $ runDB f
```

10.2.5 src/Dindo/Import/Digest.lhs

导入与摘要算法有关的内容模块

```
module Dindo.Import.Digest
    ( module X
    ) where
      import Data.Digest.Pure.SHA as X
```

10.2.6 src/Dindo/Import/Rable.lhs

导入返回值有关的内容模块

```
1 module Dindo.Import.Rable
2     ( module X
3     ) where
```

```
4       import Dindo.Common.Rable as X
5       import Text.Hamlet.XML as X
6       import Text.XML as X
```

10.2.7 src/Dindo/Import/Text.lhs

导入 text 包中相关的模块

```
1 module Dindo.Import.Text
2     ( module X
3     ) where
```

```
4       import Data.Text as X
5       import Data.Text.Encoding as X
```

10.2.8 src/Dindo/Import/TH.lhs

导入与 TemplateHaskell 与 QuasiQuote 有关的模块

```
1 module Dindo.Import.TH
2     ( module X
3     ) where
4
5       import Language.Haskell.TH as X
6       import Language.Haskell.TH.Syntax as X
```

10.2.9 src/Dindo/Import/Yaml.lhs

导入与 Yaml 有关模块

```haskell
module Dindo.Import.Yaml
    ( module X
    ) where
      import Data.Yaml as X
```

10.2.10 src/Dindo/Import/Yesod.lhs

导入与 Yesod 有关的模块

```haskell
{-# LANGUAGE QuasiQuotes
           , TemplateHaskell
           , OverloadedStrings
           #-}
```

```haskell
module Dindo.Import.Yesod
    ( module X
    , mkYesodData
    , mkGetSvrInfoAuthor
    , getSvrtimeR
    , mkSvrinfoR
    ) where
```

```haskell
      import Yesod as X hiding (mkYesodData)
      import qualified Yesod (mkYesodData)
      import Dindo.Common.Rable as X
      import Dindo.Common.Auth as X
      import Dindo.Common.Yesod.Launch as X
      import Dindo.Common.Yesod.Config as X
      import Dindo.Import.TH
      import Data.Maybe
      import Data.Time
      import Data.Text
```

```haskell
      mkYesodData a b = Yesod.mkYesodData a b'
```

```
23          where
24            b' = b ++ [parseRoutes|
25              /svrtime SvrtimeR GET
26              /svrinfo SvrinfoR GET
27              |]
28        mkGetSvrInfoAuthor :: Q [Dec]
29        mkGetSvrInfoAuthor = return $
30          [FunD (mkName "isAuthorized") [Clause [VarP (mkName "GettimeR"),WildP] (
                NormalB (AppE (VarE (mkName "return")) (ConE (mkName "Authorized")))) []]
31          ,FunD (mkName "isAuthorized") [Clause [VarP (mkName "GetinfoR"),WildP] (
                NormalB (AppE (VarE (mkName "return")) (ConE (mkName "Authorized")))) []]
32          ]
33        getSvrtimeR :: Yesod site => HandlerT site IO Text
34        getSvrtimeR = do
35          addD' <- lookupGetParam "add"
36          let addD = fromRational $ toRational $ fromMaybe 0 $ fmap (read.unpack) addD'
37          now <- liftIO getCurrentTime
38          return $ pack $ show $ addUTCTime addD now
39        mkSvrinfoR :: Text -> Q [Dec]
40        mkSvrinfoR info = [d|
41          getSvrinfoR :: Yesod site => HandlerT site IO Text
42          getSvrinfoR = return info
43          |]
```

10.2.11 src/Dindo/Common.lhs

提供版本号的部分

```
1  module Dindo.Common
2      ( dindo_common_version
3      , dindo_common_version_quasi
4      ) where
5
6        import Data.Version
```

```
7       import Paths_dindo_common
8       import Language.Haskell.TH
9       import Language.Haskell.TH.Syntax
10
11      dindo_common_version = version
12      dindo_common_version_quasi = stringE $ showVersion version
```

10.2.12 src/Dindo/Common/Auth.lhs

提供身份确认的函数的部分

```
1  {-# LANGUAGE TypeFamilies
2           , OverloadedStrings
3           #-}
```

```
4  module Dindo.Common.Auth
5      ( runPash
6      , tokenAuth
7      , pskAuth
8      , noAuth
9      , fromEntity
10     , pickF
11     , pickU
12     , getUid
13     ) where
```

```
14      import Yesod
15      import Database.Persist
16      import Database.Persist.Sql
17      import Dindo.Database
18      import Data.Time
19      import Data.Text.Encoding
20      import Data.Maybe
21      import qualified Data.ByteString as B
22      import qualified Data.ByteString.Lazy as B hiding (concat,ByteString)
```

```
23  import Data.Text (unpack,pack,Text)
24  import Data.Digest.Pure.SHA
```

```
25  pickU [] = []
26  pickU ((y,Just x):oth) = (y =. x):pickU oth
27  pickU ((_,Nothing):oth) = pickU oth
28  pickF [] = []
29  pickF ((y,Just x):oth) = (y ==. x):pickF oth
30  pickF ((_,Nothing):oth) = pickF oth
31  getUid :: ( Yesod site
32            , YesodPersist site
33            , YesodPersistBackend site ~ SqlBackend
34            )
35         => HandlerT site IO Text
36  getUid = do
37    tt' <- lookupHeader "TMP-TOKEN"
38    let Just tt = fmap decodeUtf8 tt'
39    rt':_ <- liftHandlerT $ runDB $ selectList [TmpTokenTt ==. tt] []
40    let rt = fromEntity rt'
41    return $ tmpTokenTt rt
```

用于用户验证的 runPash 0 -> uid 1 -> name 2 -> tel

```
42  runPash :: Int -> B.ByteString -> Text -> Text
43  runPash i time pash = pack $ showDigest $ sha512 $ B.fromStrict $ B.concat [pre,
        encodeUtf8 pash,time]
44    where
45      pre = case i of
46        0 -> "uid"
47        1 -> "nnnn"
48        2 -> "+86"
49  runPash _ _ x = id x
50  noAuth :: Yesod site => HandlerT site IO AuthResult
51  noAuth = return Authorized
52
```

```
53      tokenAuth :: ( Yesod site
54                   , YesodPersist site
55                   , YesodPersistBackend site ~ SqlBackend
56                   )
57                => HandlerT site IO AuthResult
58      tokenAuth = do
59        token' <- lookupHeader "TMP-TOKEN"
60        case token' of
61          Nothing -> return $ Unauthorized "Who are you!"
62          Just token -> do
63            rt' <- liftHandlerT $ runDB $ selectList [TmpTokenTt ==. decodeUtf8 token][
                    Desc TmpTokenTime]
64            case rt' of
65              rt:_ -> do
66                now <- liftIO getCurrentTime
67                let time = tmpTokenTime.fromEntity $ rt
68                if diffUTCTime now time >= 0
69                  then return $ Unauthorized "Who are you!"
70                  else return Authorized
71              _ -> return $ Unauthorized "Who are you!"
72
73      pskAuth :: ( Yesod site
74                 , YesodPersist site
75                 , YesodPersistBackend site ~ SqlBackend
76                 )
77              => HandlerT site IO AuthResult
78      pskAuth = checkTime $ \time -> do
79        pash  <- getPash
80        uid'  <- lookupPostParam "uid"
81        name' <- lookupPostParam "name"
82        tel'' <- lookupPostParam "tel"
83        let tel' = fmap (read.unpack) tel'' :: Maybe Int
84        case (uid', name', tel') of
85          (Nothing,Nothing,Nothing) -> return $ Unauthorized "Who are you!"
```

```
86            (Just uid,name,tel) -> do
87              rt <- liftHandlerT $ runDB $ selectList (
88                [AccountUid ==. uid] ++ pickF [(AccountName,name)]++pickF [(AccountTel,
                      tel)]) []
89              checkPash pash rt (runPash 0 time)
90            (Nothing,Just name,tel) -> do
91              rt <- liftHandlerT $ runDB $ selectList (
92                [AccountName ==. name] ++ pickF [(AccountTel,tel)]) []
93              checkPash pash rt (runPash 1 time)
94            (Nothing,Nothing,Just tel) -> do
95              rt <- liftHandlerT $ runDB $ selectList
96                [AccountTel ==. tel] []
97              checkPash pash rt (runPash 2 time)
98            _ -> return $ Unauthorized "Who␣are␣you!"
99          where
100           getPash = do
101             pash' <- lookupPostParam "pash"
102             return $ fromMaybe "" pash'
103           checkPash pash rt f = do
104             case rt of
105               item:_ -> do
106                 let usrPash = f.accountPash.fromEntity $ item
107                 if usrPash == pash
108                   then return Authorized
109                   else return $ Unauthorized "Who␣are␣you!"
110               _ -> return $ Unauthorized "Who␣are␣you!"
111           checkTime f = do
112             time' <- liftHandlerT $ lookupHeader "TIME-STAMP"
113             now <- liftIO getCurrentTime
114             case time' of
115               Just time -> do
116                 let t = read.unpack.decodeUtf8 $ time
117                 let diff = diffUTCTime now t
118                 if diff <= 12 && diff >= (-12)
```

```
119                     then f time
120                     else return $ Unauthorized "I bought a watch last year!"
121                 _ -> return $ Unauthorized "I bought a watch last year!"
122
123         fromEntity :: Entity a -> a
124         fromEntity (Entity _ x) = x
```

10.2.13 src/Dindo/Common/Rable.lhs

提供数据返回的部分部分
返回的类型的通用类型类

```
1  {-# LANGUAGE OverloadedStrings
2              , TemplateHaskell
3              , QuasiQuotes
4              #-}
```

```
5  module Dindo.Common.Rable
6      ( RtType(..)
7      , RtWhere(..)
8      , Varable (..)
9      , defToContent
10     , defToContentXml
11     , defToContentYaml
12     , defToContentJson
13     , Rable (..)
14     , defReturnR
15     , RtStatus (..)
16     , statusHead
17     ) where
```

```
18         import Data.Aeson as A
19         import Data.Yaml as Y
20         import Text.XML as X
21         import Text.Hamlet.XML
```

```haskell
import Data.ByteString.Internal as BI
import Data.ByteString.Lazy as BL (fromStrict, toStrict)
import Data.Text as T
import Data.Text.Encoding
import GHC.Exts(fromList)
import Control.Monad
import Yesod.Core hiding(toContent)
```

JSON,Yaml,XML

```haskell
data RtType = RtJson | RtYaml | RtXml | RtText
  deriving (Eq,Show)
data RtWhere = RtBody | RtOther Text
  deriving (Eq,Show)
```

```haskell
class Show a => Varable a where
  toValue :: a -> Value
  toNodes :: a -> [Node]
  toContents :: RtType -> a -> BI.ByteString
  toContents = defToContent
defToContent :: Varable a => RtType -> a -> BI.ByteString
defToContent RtJson = defToContentJson
defToContent RtYaml = defToContentYaml
defToContent RtXml = defToContentXml
defToContentJson :: Varable a => a -> BI.ByteString
defToContentJson = toStrict. A.encode . toValue
defToContentYaml :: Varable a => a -> BI.ByteString
defToContentYaml = Y.encode . toValue
defToContentXml :: Varable a => a -> BI.ByteString
defToContentXml x = toStrict $ renderLBS def $ Document p root []
  where
    root = Element "data" (fromList []) $ toNodes x
    p = Prologue [] Nothing []
```

```haskell
class Varable a => Rable a where
```

```
52          toWhere :: a -> RtWhere
53          toStatus  :: a -> RtStatus
54          returnR  :: MonadHandler m => a -> m TypedContent
55          returnR = defReturnR
56        defReturnR :: ( MonadHandler m
57                      , Rable a
58                      )
59                   => a -> m TypedContent
60        defReturnR x = do
61          addHeader "STATUS" $ status x
62          if toWhere x == RtBody
63            then addHeader "CONTEXT-WHERE" "BODY"
64            else addHeader "CONTEXT-WHERE" $ (\(RtOther a)-> a) $ toWhere x
65          addContent
66          where
67            status = statusHead.toStatus
68            addContent = case toWhere x of
69              RtBody -> selectRep $ do
70                provideRepType "application/json" $ return $ decodeUtf8 $ toContents RtJson
                      x
71                provideRepType "application/yaml" $ return $ decodeUtf8 $ toContents RtYaml
                       x
72                provideRepType "application/xml"  $ return $ decodeUtf8 $ toContents RtXml
                      x
73              RtOther y -> do
74                addHeader y $ pack $ show x
75                selectRep $  provideRep $ return ("" :: Text)
```

```
76        data RtStatus = RtSucc | RtFail
77        statusHead :: RtStatus -> Text
78        statusHead RtSucc = "SUCCESS"
79        statusHead RtFail = "FAILED"
```

将 Yesod 中的 ErrorResponse 实现 Varable 与 Rable

```
80      instance Varable ErrorResponse where
81        toValue NotFound = A.String "NotFound"
82        toValue ( InternalError  x) = object [" internal-error" .= x]
83        toValue (PermissionDenied x) = object ["permission-denied" .= x]
84        toValue ( InvalidArgs  x) = object [" invalid -args" .= x]
85        toValue NotAuthenticated = A.String "NotAuthenticated"
86        toValue (BadMethod x) = object ["bad-method" .= show x]
87        toNodes NotFound = [xml|NotFound|]
88        toNodes ( InternalError  x) = [xml|<InternalError>#{x}|]
89        toNodes (PermissionDenied x) = [xml|<PermissionDenied>:#{x}|]
90        toNodes (InvalidArgs  x) = [xml|<InvalidArgs>#{x'}|]
91          where
92            x' = T.unlines x
93        toNodes NotAuthenticated = [xml|NotAuthenticated|]
94        toNodes (BadMethod x) = [xml|<BadMethod>#{pack $ show x}|]
95
96      instance Rable ErrorResponse where
97        toWhere _ = RtOther "CONTEXT"
98        toStatus _ = RtFail
```

10.2.14 src/Dindo/Common/Yesod/Config.lhs

提供模块配置的部分

```
1  {-# LANGUAGE RecordWildCards
2             , OverloadedStrings
3             #-}
```

```
4  module Dindo.Common.Yesod.Config
5      ( SvrConfig (..)
6      , DbConfig(..)
7      , dbConfig2Str
8      ) where
```

```
9        import Data.Yaml
```

```haskell
import Data.ByteString as B
import Data.ByteString.Lazy
import Data.String
```

模块配置与数据库链接配置。

svrPost 后端侦听端口

svrDb 后端的数据库配置（由下面的项组成）

dbAddr 数据库的地址（ip / 域名，不包含端口）

dbPort 数据库侦听的端口

dbUser 链接数据库的用户名

dbName 链接的数据库

dbPsk 链接的密码

ConThd 连接数限制

```haskell
data SvrConfig = SvrConfig
  { svrPort :: Int
  , svrDb :: DbConfig
  }
data DbConfig = DbConfig
  { dbAddr :: String
  , dbPort :: String
  , dbUser :: String
  , dbName :: String
  , dbPsk :: String
  , dbConThd :: Int
  }
```

将模块配置与数据库连接设置实现 ToJSON 与 FromJSON 类型类，以供数据转换为 JSON 与 YAML。

```haskell
instance ToJSON SvrConfig where
  toJSON SvrConfig{..} = object
```

```
27            [ "port" .= svrPort
28            , "datebase-bconfig" .= svrDb
29            ]
30        instance ToJSON DbConfig where
31          toJSON DbConfig{..} = object
32            [ "addr" .= dbAddr
33            , "port" .= dbPort
34            , "user" .= dbUser
35            , "name" .= dbName
36            , "con-limit" .= dbConThd
37            , "password" .= dbPsk
38            ]
39        instance FromJSON SvrConfig where
40          parseJSON (Object v) = SvrConfig
41            <$> v .: "port"
42            <*> v .: "database-config"
43        instance FromJSON DbConfig where
44          parseJSON (Object v) =DbConfig
45            <$> v .: "addr"
46            <*> v .: "port"
47            <*> v .: "user"
48            <*> v .: "name"
49            <*> v .: "password"
50            <*> v .: "con-limit"
```

将数据库配置转化成链接字符串。

```
51        dbConfig2Str :: DbConfig -> (B.ByteString,Int)
52        dbConfig2Str DbConfig{..} = (str,dbConThd)
53          where
54            str = toStrict $
55              fromString $     "host=\'" ++ dbAddr
56                          ++ "\' port=\'" ++ dbPort
57                          ++ "\' user=\'" ++ dbUser
58                          ++ "\' password=\'" ++ dbPsk
```

```
59                          ++ "\'␣dbname=\'" ++ dbName
60                          ++ "\'"
```

JSON 与 Yaml 例程。

```
1  { "port":3000
2  , "database-config":
3    { "addr":"127.0.0.1"
4    , "port":"5432"
5    , "user":"postgres"
6    , "name":"postgres"
7    , "password":"postgres"
8    , "con-limit":10
9    }
10 }
```

```
1  port: 3000
2  database-config:
3    addr: '127.0.0.1'
4    port: '5432'
5    user: postgres
6    name: postgres
7    password: postgres
```

这个需要在运行时传入。假设配置文件在 config.yml 中, 启动 UsrManage 模块。

```
# cat config.yml | dindo-um
```

10.2.15 src/Dindo/Common/Yesod/Launch.lhs

提供了启动的相关部分

```
1  module Dindo.Common.Yesod.Launch
2      ( Dindoble(..)
3      ) where
```

```haskell
import Dindo.MicroFramework.Register
import Yesod
import Dindo.Common.Yesod.Config
import Database.Persist.Postgresql
import Control.Monad.Logger
```

Dingo 后端的服务的“标准”

```haskell
class Registrable a => Dindoble a where
  fromPool :: ConnectionPool -> SvrConfig -> a
  warpDindo :: SvrConfig -> (Int -> a -> IO()) -> IO ()
  warpDindo x warpF =
    runStdoutLoggingT $ withPostgresqlPool connStr cT $
      \pool -> liftIO $ do
        let site = fromPool pool x
        register site
        warpF port site
    where
      (connStr,cT) = dbConfig2Str.svrDb $ x
      port = svrPort x
```

微服务架构这一部分，就大部分内容犹豫某些原因为实现，是有能使之运行的空壳。

10.2.16 src/Dindo/MicroFramework/API.lhs

提供了微服务架构中的 API 注册的部分

```haskell
module Dindo.MicroFramework.API
    ( APIble(..)
    , regAPI
    ) where
```

```haskell
import Yesod.Core
```

注册的 API 的类型类

apis 所公开注册的 API，（API 名称，相关 Route 信息）

```
6       class ( RenderRoute a
7             ) => APIble a where
8         apis :: a -> [(String,String)]
```

```
9       regAPI :: APIble a => a -> IO Bool
10      regAPI x = do
11        -- 注册 API
12        -- 实际上应该是 数据生成+http 请求，此处仅输出内容
13        putStrLn "API␣内容"
14        print $ apis x
15        return True
```

10.2.17 src/Dindo/MicroFramework/Destory.lhs

提供了微服务架构中销毁的部分

```
1  module Dindo.MicroFramework.Destory
2      ( Destorible (..)
3      , regDestory
4      ) where
```

```
5       import Yesod.Core
```

服务实例销毁的类型类

destoryAPI 销毁的 API

destoryHead 所需的 Head 中特定 “签名的内容”

```
6       class ( Yesod a
7             ) => Destorible a where
8         destoryAPI :: a -> String
9         destoryHead :: a -> String
```

```
10      regDestory :: Destorible a => a -> IO Bool
11      regDestory x = do
```

```
12          -- 注册 销毁接口
13          -- 实际上应该是 http 请求，此处仅输出内容
14          putStrLn "销毁接口␣注册"
15          print $ destoryAPI x
16          print $ destoryHead x
17          return True
```

10.2.18 src/Dindo/MicroFramework/Register.lhs

提供了微服务架构中服务实例注册的部分

```
1  module Dindo.MicroFramework.Register
2      ( Registrable (..)
3      , Heartbeatable (..)
4      , register
5      ) where
```

```
6          import Yesod.Core
7          import Control.Concurrent
8
9          import Dindo.MicroFramework.API
10         import Dindo.MicroFramework.Destory
```

可注册的服务的类型类。

regSvrAddr 注册目标的地址 ip 或域名

regSvrPost 访问端口

regAddr 注册的服务的地址

regPort 注册的端口

```
11         class ( Yesod a
12                 , APIble a
13                 , Destorible a
14                 , Heartbeatable a
```

```
15              ) => Registrable a where
16            regAddr :: a -> String
17            regAddr = defRegAddr
18            regPort :: a -> Int
19            regPort = defRegPort
20            regSvrAddr :: a -> String
21            regSvrPort :: a -> Int
22          defRegPort _ = 3000
23          defRegAddr _ = "localhost"
```

状态获取的类型类

```
24        class ( Yesod a
25              , RenderRoute a
26              ) => Heartbeatable a where
27            heartbeat :: a -> IO ()
```

注册服务实例的函数

False 注册失败

True 注册成功

```
28          register :: Registrable a => a -> IO Bool
29          register x = do
30            -- 注册 服务
31            -- 实际上应该是 http 请求，此处仅输出内容
32            putStrLn "注册服务的端口"
33            print $ regSvrPort x
34            putStrLn "注册服务的地址"
35            print $ regSvrAddr x
36            putStrLn "被注册的实例的地址"
37            print $ regPort x
38            putStrLn "被注册的实例的端口"
39            print $ regPort x
40            regAPI' $ regDestory' $ do
41              forkIO $ heartbeat x
```

```
42            return True
43          where
44            regAPI' a = do
45              ra <- regAPI x
46              if ra then a else return False
47            regDestory' a = do
48              rd <- regDestory x
49              if rd then a else return True
```

10.3 dindo-launch

这一部分是 dindo 的服务的启动部分。

10.3.1 src/Main.lhs

启动器的主体

```
1 {-# LANGUAGE TemplateHaskell
2              , DeriveDataTypeable
3              #-}
```

```
4 module Main
5     ( main
6     ) where
```

```
7       import Dindo.Std
8       import System.Console.CmdArgs
9       import Dindo.Import.Aeson as A
10      import Dindo.Import.Yaml as Y
11      import Dindo.Import.Yesod
12      import Data.Maybe
13      import qualified Dindo.Import.ByteString as B
14      import qualified Dindo.Import.Text as T
15      import Dindo.Common.Yesod.Launch
16      import Dindo.Common.Yesod.Config
```

```
17  import Paths_dindo_launch
18  import Data.Version
19  import Dindo.Common(dindo_common_version_quasi)
20  import Dindo.Import.Database(dindo_database_version_quasi)
```

```
21  data Launch = Launch {form ::String}
22    deriving (Show,Data,Typeable)
23  launch = Launch{form="json" &= typ "YAML|JSON" &= help "格式"}
24    &= summary ( "dindo-common-"
25                 ++ $(dindo_common_version_quasi)
26                 ++ "; dindo-database-"
27                 ++ $(dindo_database_version_quasi)
28                 ++ "; " ++ $(dindo_module_name) ++ "-"
29                 ++ $(dindo_module_version)
30                 ++ "; dindo-launch-"
31                 ++ showVersion version)
```

```
32  main :: IO ()
33  main = do
34    cfg' <- cmdArgs launch >>= cfg
35    warpDindo cfg' itemWarp
36    where
37      itemWarp :: Int -> $(std) -> IO()
38      itemWarp = warp
39  cfg :: Launch -> IO SvrConfig
40  cfg l = getContents >>= (return.fromMaybe (error "Invailed config json").decode'.
        T.encodeUtf8.T.pack)
41    where
42      decode' = case l of
43        Launch "json" -> A.decode.B.fromStrictBS
44        Launch "yaml" -> Y.decode
45        _ -> error "error form"
```

10.4 dindo-usrmanage

这一部分是 dindo 的用户管理了部分。

10.4.1 src/Dindo/Std.lhs

与 Dindo 启动器对接的部分

```
{-# LANGUAGE TemplateHaskell #-}
```

```
module Dindo.Std
    ( module X
    , std
    , dindo_module_name
    , dindo_module_version
    ) where

      import Dindo.UM as X -- need change
      import Dindo.Import.TH
      import Dindo.Import.TH
      dindo_module_name = stringE "dindo-usrmanage"
      dindo_module_version = dindo_usrmanage_version_quasi
      std = [t|UM|]
```

10.4.2 src/Dindo/UM.lhs

用户管理部分的导出的部分

```
{-# LANGUAGE TemplateHaskell
             , OverloadedStrings
             #-}
```

```
module Dindo.UM
    ( module X
    , dindo_usrmanage_version
    , dindo_usrmanage_version_quasi
```

```
8       ) where
9         import Dindo.UM.Foundation as X
10        import Dindo.UM.Handler as X
11        import Dindo.Import.Yesod
12        import Dindo.Import.TH
13        import Data.Version
14        import Paths_dindo_usrmanage
15
16        dindo_usrmanage_version = version
17        dindo_usrmanage_version_quasi = stringE $ showVersion version
18        mkYesodDispatch "UM" resourcesUM
```

10.4.3 src/Dindo/UM/Data.lhs

定义返回数据的部分

```
1   {-# LANGUAGE OverloadedStrings
2             , QuasiQuotes
3             , RecordWildCards
4             #-}
```

```
5   module Dindo.UM.Data
6       ( RtRegist (..)
7       , RtIdy (..)
8       , RtIdfed (..)
9       , RtCommonSucc(..)
10      , RtUImg(..)
11      , RtUInfo (..)
12      , RtChPsk(..)
13      , RtEaddr(..)
14      , RtGEadd(..)
15      ) where
```

```
16        import Dindo.Import.Rable
17        import Dindo.Import.Aeson as A
```

```
18  import Dindo.Import.Yaml as Y
19  import Dindo.Import.Text as T
20  import Dindo.Import.ByteString as B
21  import Dindo.Import.Yesod
22  import Dindo.Import.Database
```

用户注册返回数据

```
23  data RtRegist = RtRegist
24      { uid :: Text
25      }
26    | RtRegistFail
27      { regReason :: Text
28      }
29    deriving (Eq)
30  instance Show RtRegist where
31    show (RtRegist x) = T.unpack x
32    show (RtRegistFail x) = T.unpack x
33  instance Varable RtRegist where
34    toValue (RtRegist x) = object ["uid" .= x]
35    toValue (RtRegistFail x) = object ["error" .= x]
36    toNodes (RtRegist x) = [xml|<uid>#{x}|]
37    toNodes (RtRegistFail x) =[xml|<error>#{x}|]
38  instance Rable RtRegist where
39    toWhere (RtRegist _) = RtBody
40    toWhere (RtRegistFail _) = RtBody
41    toStatus (RtRegist _) = RtSucc
42    toStatus (RtRegistFail _) = RtFail
```

用户认证信息的返回数据

```
43  data RtIdy = RtIdy
44    | RtIdyFail
45      { idyReason :: Text
46      }
47    deriving (Eq)
```

```
48  instance Show RtIdy where
49    show (RtIdyFail x) = T.unpack x
50  instance Varable RtIdy where
51    toValue RtIdy = Null
52    toValue (RtIdyFail x) = object ["error" .= x]
53    toNodes RtIdy  = [xml|null |]
54    toNodes (RtIdyFail x) = [xml|<error>#{x}|]
55  instance Rable RtIdy where
56    toWhere (RtIdyFail _) = RtBody
57    toWhere RtIdy = RtBody
58    toStatus RtIdy = RtSucc
59    toStatus (RtIdyFail _) = RtFail
```

用户查询认证状态信息

```
60  data RtIdfed = RtIdfedPass | RtIdfedNo
61    deriving (Eq,Show)
62  instance Varable RtIdfed where
63    toValue RtIdfedPass = object ["status" .= ("pass"::Text)]
64    toValue RtIdfedNo   = object ["status" .= ("no"::Text)]
65    toNodes RtIdfedPass = [xml|<status>pass|]
66    toNodes RtIdfedNo   = [xml|<status>no|]
67  instance Rable RtIdfed where
68    toWhere RtIdfedPass = RtBody
69    toWhere RtIdfedNo = RtBody
70    toStatus RtIdfedPass = RtSucc
71    toStatus RtIdfedNo = RtSucc
```

通用成功标志

```
72  data RtCommonSucc = RtCommonSucc
73    deriving (Eq,Show)
74  instance Varable RtCommonSucc where
75    toValue RtCommonSucc = Null
76    toNodes RtCommonSucc = [xml|null|]
77  instance Rable RtCommonSucc where
```

```
78          toWhere RtCommonSucc = RtBody
79          toStatus  RtCommonSucc = RtSucc
```

用户信息查询返回结果

```
80      data RtUInfo = RtUInfo
81          { rtuiUid  :: Text
82          , rtuiName :: Text
83          , rtuiTel  :: Text
84          , rtuiEmail  :: Text
85          }
86        | RtUInfoNSU
87      instance Show RtUInfo where
88        show RtUInfoNSU = "no such a user"
89      instance Varable RtUInfo where
90        toValue RtUInfo{..} = object
91          [ "uid" .= rtuiUid
92          , "name" .= rtuiName
93          , "tel" .= rtuiTel
94          , "email" .= rtuiEmail
95          ]
96        toNodes RtUInfo{..} = [xml|
97        <uid> #{rtuiUid}
98        <name> #{rtuiName}
99        <tel> #{rtuiTel}
100       <email> #{rtuiEmail}
101       |]
102     instance Rable RtUInfo where
103       toWhere RtUInfo{..} = RtBody
104       toWhere RtUInfoNSU = RtOther "CONTEXT"
105       toStatus  RtUInfo{..} = RtSucc
106       toStatus  RtUInfoNSU = RtFail
```

获取用户头像返回内容

```
107     data RtUImg = RtUImg ByteString
```

```
108                 | RtUImgFail
109           deriving (Eq)
110       instance Show RtUImg
111       instance Varable RtUImg
112       instance Rable RtUImg where
113         returnR (RtUImg img) =
114           selectRep $ provideRepType "image/png" $ return img
115         returnR RtUImgFail = do
116           addHeader "CONTEXT-WHERE" "CONTEXT"
117           addHeader "CONTEXT" "Failed on get image"
118           selectRep $ provideRep $ return ("" :: Text)
```

更改密码的返回值

```
120       data RtChPsk = RtChPsk
121                    | RtChPskFail Text
122         deriving (Eq)
123       instance Show RtChPsk where
124         show (RtChPskFail x) = T.unpack x
125       instance Varable RtChPsk where
126         toValue RtChPsk = Null
127         toValue (RtChPskFail x) = object ["error" .= x]
128         toNodes RtChPsk = [xml|null|]
129         toNodes (RtChPskFail x) = [xml|<error>#{x}|]
130       instance Rable RtChPsk where
131         toWhere RtChPsk = RtBody
132         toWhere (RtChPskFail _) = RtBody
133         toStatus RtChPsk = RtSucc
134         toStatus (RtChPskFail _) = RtFail
```

收货地址的增删的返回值

```
135       data RtEaddr = RtEaddrAdd Text
136                    | RtEaddrChn
137                    | RtEaddrDel
138                    | RtEaddrFail Text
```

```
139         deriving (Eq,Show)
140       instance Varable RtEaddr where
141         toValue (RtEaddrAdd x) = object ["aid" .= x]
142         toValue RtEaddrChn = Null
143         toValue RtEaddrDel = Null
144         toValue (RtEaddrFail x) = object ["error" .= x]
145         toNodes (RtEaddrAdd x) = [xml|<aid>#{x}|]
146         toNodes RtEaddrChn = [xml|null|]
147         toNodes RtEaddrDel = [xml|null|]
148         toNodes (RtEaddrFail x) = [xml|<error>#{x}|]
149       instance Rable RtEaddr where
150         toWhere (RtEaddrAdd _) = RtBody
151         toWhere RtEaddrChn = RtBody
152         toWhere RtEaddrDel = RtBody
153         toWhere (RtEaddrFail _) = RtBody
154         toStatus (RtEaddrAdd _) = RtSucc
155         toStatus RtEaddrChn = RtSucc
156         toStatus RtEaddrDel = RtSucc
157         toStatus (RtEaddrFail _) = RtFail
```

获取地址

```
158       data RtGEadd = RtGEadd [Addr]
159                    | RtGEaddFail Text
160         deriving (Eq,Show)
161       instance Varable RtGEadd where
162         toValue (RtGEadd x) = toJSON x
163         toValue (RtGEaddFail x) = object ["error" .= x]
164         toNodes (RtGEadd xs) = [xml|
165           $forall x <- xs
166             <aid>#{addrAid x}
167             <addr>#{addrAddr x}
168             <zip>#{addrZip x}
169           |]
170         toNodes (RtGEaddFail x) = [xml|<error>#{x}|]
```

```
171         instance Rable RtGEadd where
172           toWhere (RtGEadd _ ) = RtBody
173           toWhere (RtGEaddFail _) = RtBody
174           toStatus  (RtGEadd _) = RtSucc
175           toStatus  (RtGEaddFail _) =RtFail
```

10.4.4 src/Dindo/UM/Foundation.lhs

基础的部分

```
1  {-# LANGUAGE OverloadedStrings
2               , TemplateHaskell
3               , TypeFamilies
4               , QuasiQuotes
5               #-}
```

```
6  module Dindo.UM.Foundation
7      ( module Dindo.UM.Foundation
8      , getSvrtimeR
9      ) where
```

```
10         import Dindo.Common
11         import Dindo.Import
12         import Dindo.Import.Yesod
13         import Dindo.Import.Database
14         import Paths_dindo_usrmanage
15         import Dindo.Import.Text as T
16         import Data.Version
```

```
17         data UM = UM
18           { connPool :: ConnectionPool
19           , config   :: SvrConfig
20           }
21         mkYesodData "UM" [parseRoutes|
22           / regist  RegistR POST
```

```
23          /identify IdentifyR POST
24          /identified Identified POST
25          /login LoginR POST
26          /logout LogoutR POST
27          /usrinfo UsrinfoR POST
28          /usrhimg UsrhimgR POST
29          /usrinfochange UsrinfochangeR POST
30          /changpash ChangpashR POST
31          /upeaddr UpeaddrR POST
32          /geteaddr GeteaddR POST
33        |]
```

实现 Yesod 类型类

```
34      instance Yesod UM where
35        errorHandler = returnR
36        isAuthorized SvrinfoR _ = return Authorized
37        isAuthorized SvrtimeR _ = return Authorized
38        isAuthorized RegistR _ = noAuth
39        isAuthorized LoginR _ = pskAuth
40        isAuthorized _ _ = tokenAuth
41      instance YesodPersist UM where
42        type YesodPersistBackend UM = SqlBackend
43        runDB a = getYesod >>= (runSqlPool a.connPool)
44      mkSvrinfoR $ pack $ "dindo-um-" ++ showVersion version ++ "; dindo-common-"
            ++ $(dindo_common_version_quasi)
```

微服务架构

```
45      instance APIble UM where
46        apis _ = []
47      instance Destorible UM where
48        destoryHead _ = ""
49        destoryAPI _ = ""
50      instance Heartbeatable UM where
51        heartbeat _ = return ()
```

```
52      instance Registrable UM where
53        regAddr _ = ""
54        regPort = svrPort . config
55        regSvrPort _ = 80
56        regSvrAddr _ = ""
57      instance Dindoble UM where
58        fromPool = UM
```

10.4.5 src/Dindo/UM/Handler.lhs

处理函数的部分

```
1  {-# LANGUAGE OverloadedStrings
2             , FlexibleContexts
3             , TypeFamilies
4             #-}
```

```
5  module Dindo.UM.Handler
6      ( postRegistR
7      , postUsrinfoR
8      , postLogoutR
9      , postLoginR
10     , postIdentified
11     , postIdentifyR
12     , postUsrinfochangeR
13     , postChangpashR
14     , postUsrhimgR
15     , postUpeaddrR
16     , postGeteaddR
17     ) where
```

```
18      import Dindo.Import
19      import Dindo.Import.Yesod
20      import Dindo.Import.Database
21      import Dindo.UM.Foundation
```

```
22  import Dindo.UM.Data
23  import Dindo.Import.Digest
24  import Dindo.Import.ByteString as B hiding(unpack,pack,splitAt,take,map,null)
25  import Dindo.Import.Text as T hiding(splitAt,take,map,null)
26  import Dindo.Common.Auth(fromEntity,pickU,pickF)
27  import Control.Exception(try,SomeException)
28  import Control.Monad
```

注册的 API

```
29  postRegistR :: Handler TypedContent
30  postRegistR =
31    getParam insertAItem
32    where
33      try' :: IO a -> IO (Either SomeException a)
34      try' = try
35      getParam f = do
36        name' <- lookupPostParam "name"
37        pash' <- lookupPostParam "pash"
38        tel'  <- lookupPostParam "tel"
39        case (name',pash', tel') of
40          (Just name,Just pash,Just tel) -> do
41            x <- liftIO getCurrentTime
42            let (time,p) = splitAt 10 $ show x
43            let to = showDigest $ sha1 $ fromStrictBS $ encodeUtf8 $ T.concat [pash,
                  name]
44            let uid = 'U':time ++ to
45            f (pack uid,name,pash,read (unpack tel))
46          _ -> returnR $ RtRegistFail "param:␣less␣and␣less"
47      insertAItem (uid,name,pash,tel) = do
48        runInnerHandler <- handlerToIO
49        rt <- liftIO $ try' $ runInnerHandler $ runDB $ insert $ Account uid pash tel
              name
50        case rt of
51          Left e -> returnR $ RtRegistFail $ pack $ show e
```

```
52              Right _ -> returnR $ RtRegist uid
```

用户认证的 API

```
53      postIdentifyR :: Handler TypedContent
54      postIdentifyR =
55        checkParam $ addItem $ checkPic addPic
56        where
57          checkParam f = do
58            email' <- lookupPostParam "email"
59            rname' <- lookupPostParam "rname"
60            prcid' <- lookupPostParam "prcid"
61            addr' <- lookupPostParam "addr"
62            case (email', rname', prcid', addr') of
63              (Just email, Just rname, Just prcid, Just addr) ->
64                f (email, rname, prcid, addr)
65              _ -> returnR $ RtIdyFail "param: less and less"
66          checkPic f = do
67            pic' <- lookupFile "pic"
68            case pic' of
69              Just pic -> do
70                rt <- sourceToList $ fileSource pic
71                let bpic = B.concat rt
72                f bpic
73              _ -> returnR $ RtIdyFail "param: picture needed"
74          addItem f (email, rname, prcid, addr) = do
75            uid <- getUid
76            liftHandlerT $ runDB $ insert $ Usr uid email rname prcid addr "N"
77            f
78          addPic pic = do
79            uid <- getUid
80            now <- liftIO getCurrentTime
81            let str = show now
82            let (time, p) = splitAt 10 $ str
```

```
83              let to = showDigest $ sha1 $ fromStrictBS $ encodeUtf8 $ T.concat [uid, pack
                    str]
84              let pid = pack $ 'A':time ++ to
85              liftHandlerT $ runDB $ insert $ Apic pid uid pic $ Just 0
86              returnR $ RtIdy
```

认证状态查询

```
87      postIdentified :: Handler TypedContent
88      postIdentified = do
89        uid <- getUid
90        rt <- liftHandlerT $ runDB $ selectList [UsrUid ==. uid] []
91        case rt of
92          (Entity _ item):_ -> if usrStatus item == "P"
93            then returnR RtIdfedPass
94            else returnR RtIdfedNo
```

用户登录

```
95      postLoginR :: Handler TypedContent
96      postLoginR = do
97        uid' <- lookupPostParam "uid"
98        name' <- lookupPostParam "name"
99        tel' <- lookupPostParam "tel"
100       case (uid', name', tel') of
101         (uid,name,tel) -> do
102           pash <- getPash
103           rt' <- liftHandlerT $ runDB $ selectList (pickF
104             [ (AccountUid, uid)
105             , (AccountName, name)
106             ] ++ pickF
107             [ (AccountTel,fmap (read.unpack) tel)
108             ]) []
109           case rt' of
110             (Entity _ item):_ -> do
111               let uid = accountUid item
```

```
112                 now <- liftIO getCurrentTime
113                 let lim = addUTCTime 3600 now
114                 let time = show lim
115                 let to = showDigest $ sha512 $ fromStrictBS $ encodeUtf8 $ T.concat [uid,
                        pash,pack time]
116                 let tt = pack $ take 22 time ++ to
117                 liftHandlerT $ runDB $ insert $ TmpToken tt lim uid
118                 returnR RtCommonSucc
119           where
120             getPash = do
121               pash' <- lookupPostParam "pash"
122               return $ fromMaybe "" pash'
```

用户登出

```
123       postLogoutR :: Handler TypedContent
124       postLogoutR = do
125         Just token <- lookupHeader "TMP-TOKEN"
126         Just uid <- lookupHeader "USR-ID"
127         liftHandlerT $ runDB $ deleteWhere [TmpTokenTt ==. decodeUtf8 token,
                TmpTokenUid ==. (read.unpack.decodeUtf8) uid]
128         returnR $ RtCommonSucc
```

查询用户信息

```
129       postUsrinfoR :: Handler TypedContent
130       postUsrinfoR = do
131         tuid <- getUid
132         uid' <- lookupPostParam "uid"
133         let uid = fromMaybe tuid uid'
134         rt' <- liftHandlerT $ runDB $ selectList [UsrUid ==. uid] []
135         case rt' of
136           Entity _ rt:_ -> do
137             let email = usrEmail rt
138             Entity _ item:_ <- liftHandlerT $ runDB $ selectList [AccountUid ==. uid] []
```

```
139             returnR $ RtUInfo uid (accountName item) (pack $ show $ accountTel item)
                    email
140           _ -> returnR RtUInfoNSU
```

获得用户头像

```
141     postUsrhimgR :: Handler TypedContent
142     postUsrhimgR = do
143       tuid <- getUid
144       uid' <- lookupPostParam "uid"
145       let uid = fromMaybe tuid uid'
146       rt' <- liftHandlerT $ runDB $ selectList [ApicUid ==. uid] []
147       case rt' of
148         Entity _ rt:_ -> returnR $ RtUImg $ apicBpic rt
149         _ -> returnR $ RtUImgFail
```

用户信息变更

```
150     postUsrinfochangeR :: Handler TypedContent
151     postUsrinfochangeR = check update
152       where
153         updatePic uid pic' = case pic' of
154           Nothing -> return ()
155           Just pic -> do
156             rt <- sourceToList $ fileSource pic
157             let bpic = B.concat rt
158             updateWhere [ApicUid ==. uid,ApicTyp ==. Just 0] [ApicBpic =. bpic]
159         update (a,b,pic) = do
160           uid <- getUid
161           when (not $ null a) $
162             liftHandlerT $ runDB $ updateWhere [AccountUid ==. uid] a
163           when (not $ null b) $
164             liftHandlerT $ runDB $ updateWhere [UsrUid ==. uid] b
165           liftHandlerT $ runDB $ updatePic uid pic
166           returnR $ RtCommonSucc
167         check f = do
```

```
168             name <- liftHandlerT $ lookupPostParam "name"
169             tel  <- liftHandlerT $ lookupPostParam "tel"
170             email <- liftHandlerT $ lookupPostParam "email"
171             rname <- liftHandlerT $ lookupPostParam "rname"
172             prcid <- liftHandlerT $ lookupPostParam "prcid"
173             addr <- liftHandlerT $ lookupPostParam "addr"
174             pic <- liftHandlerT $ lookupFile "pic"
175             let a = pickU [(AccountName,name)]
176             let a' = pickU [(AccountTel,fmap (read.T.unpack) tel)]
177             let b = pickU [(UsrEmail,email),(UsrRname,rname),(UsrPrcid,prcid),(UsrAddr,
                    addr)]
178             f (a++a',b,pic)
```

修改密码

```
179     postChangpashR :: Handler TypedContent
180     postChangpashR = check changePash
181       where
182         changePash pash = do
183           uid <- getUid
184           liftHandlerT $ runDB $ updateWhere [AccountUid ==. uid] [AccountPash =.
                  pash]
185           returnR $ RtChPsk
186         check f = do
187           pash' <- lookupPostParam "pash"
188           case pash' of
189             Nothing -> do
190               returnR $ RtChPskFail "param:␣less␣and␣less"
191             Just x -> f x
```

收获地址

```
192     postUpeaddrR :: Handler TypedContent
193     postUpeaddrR = spl
194       where
195         changeItem aid a = do
```

```
196             liftHandlerT $ runDB $ updateWhere [AddrAid ==. aid] a
197             returnR $ RtEaddrChn
198           checkChn f = do
199             addr <- liftHandlerT $ lookupPostParam "addr"
200             zipcode <- liftHandlerT $ lookupPostParam "zip"
201             aid' <- liftHandlerT $ lookupPostParam "aid"
202             case aid' of
203               Just aid -> f aid $ pickU [(AddrAddr,addr),(AddrZip,zipcode)]
204               Nothing -> returnR $ RtEaddrFail "param:change:␣less␣and␣less"
205           delItem aid = do
206             liftHandlerT $ runDB $ deleteWhere [AddrAid ==. aid]
207             returnR $ RtEaddrDel
208           checkDel f = do
209             aid' <- liftHandlerT $ lookupPostParam "aid"
210             case aid' of
211               Just aid -> f aid
212               Nothing -> returnR $ RtEaddrFail "param:del:␣less␣and␣less"
213           addItem (addr,zipcode) = do
214             uid <- getUid
215             now <- liftIO getCurrentTime
216             let aid' = showDigest $ sha256 $ fromStrictBS $ encodeUtf8 addr
217             let aid = pack $ "A"++show now++aid'
218             liftHandlerT $ runDB $ insert $ Addr aid uid zipcode addr
219             returnR $ RtEaddrAdd aid
220           checkAdd f = do
221             addr' <- liftHandlerT $ lookupPostParam "addr"
222             zip' <- liftHandlerT $ lookupPostParam "zip"
223             case (addr', zip') of
224               (Just addr, Just zipcode) -> f (addr,zipcode)
225               _ -> returnR $ RtEaddrFail "param:add:␣less␣and␣less"
226           spl = do
227             opt <- liftHandlerT $ lookupHeader "OPT"
228             case opt of
229               Just "ADD" -> checkAdd addItem
```

```
230                 Just "DEL" -> checkChn changeItem
231                 Just "CHANGE" -> checkDel delItem
232                 _ -> returnR $ RtEaddrFail "header:opt:␣less␣and␣less"
```

获取收货地址

```
233       postGeteaddR :: Handler TypedContent
234       postGeteaddR = spl
235         where
236           getByUid uid = do
237             rt <- liftHandlerT $ runDB $ selectList [AddrUid ==. uid] []
238             returnR $ RtGEadd $ map fromEntity rt
239           getByAid aid = do
240             uid <- getUid
241             rt <- liftHandlerT $ runDB $ selectList [AddrAid ==. aid,AddrUid ==. uid] []
242             returnR $ RtGEadd $ map fromEntity rt
243           spl = do
244             uid' <- liftHandlerT $ lookupPostParam "uid"
245             aid' <- liftHandlerT $ lookupPostParam "aid"
246             case (uid', aid') of
247               (Just uid, _) -> getByUid uid
248               (Nothing, Just aid) -> getByAid aid
249               _ -> returnR $ RtGEaddFail "param:␣less␣and␣less"
```

10.5 dindo-tools

dindo 的辅助工具
dindo-pash 测试用的辅助工具

10.5.1 src/pash/Main.lhs

主函数部分
产生密钥的工具

```
1  {-# LANGUAGE TemplateHaskell
2              , DeriveDataTypeable
```

```
3           #-}
```

```
4  module Main
5      ( main
6      ) where
```

```
7          import System.Environment
8          import Dindo.Import
9          import Dindo.Common.Auth
10         import Dindo.Import.Digest
11         import qualified Dindo.Import.Text as T
12         import qualified Dindo.Import.ByteString as B
13         import Dindo.Common(dindo_common_version_quasi)
14         import Data.Version
15         import System.Console.CmdArgs
16         import Paths_dindo_tools
```

```
17         main :: IO ()
18         main = do
19           Pash key t at <- cmdArgs pash
20           now' <- getCurrentTime
21           let now = addUTCTime (fromIntegral at) now
22           pash <- getPash t key now
23           a' <- getContents
24           let a = concat.lines $ a'
25           case t of
26             100 -> putStr $ a ++ "␣-d␣\"pash="++pash++"\""
27             _ -> putStr $ a ++ "␣-d␣\"pash="++pash++"\"␣-H␣\"TIME-STAMP:"++
                       show now++"\""
28           return ()
29           where
30             getPash typ key now = case typ of
31               100 -> return $ showDigest $ sha256 $ B.fromStrictBS $ T.encodeUtf8 $ T.pack
                         key
```

```
32            x -> do
33              let k = T.pack $ showDigest $ sha256 $ B.fromStrictBS $ T.encodeUtf8 $ T.
                    pack key
34              let time = T.encodeUtf8.T.pack.show $ now
35              return $ T.unpack $ runPash x time k
```

dindo-pash 使用说明 一共有两个参数：一个是密码，另一个是散列方式，也就是认证方式。

100 注册时

0 使用 uid 登录时

1 使用 name 登录时

2 使用 tel 登录时

有一个 flag 开关是关于时间矫正的，矫正单位是秒。

```
36      data Pash = Pash {pKey :: String,pType :: Int,aTime :: Int}
37        deriving (Show,Data,Typeable)
38      pash = Pash
39        { pKey = def &= args &= typ "PASSWORD" &= help "密码"
40        , pType = def &= args &= typ "Identify␣type" &= help "认证方式"
41        , aTime = 0 &= typ "UTCDiffTime" &= help "时间矫正"
42        } &= summary ( "dindo-common:-"
43                       ++ $(dindo_common_version_quasi)
44                       ++ ";␣dindo-tools-"
45                       ++ showVersion version
46                       )
```

11 Dindo 公共组件

这部分是关于 Dindo 的公共组件的。由于 Dingo 后端采用的微服务架构[12]，不同的微服务之间，会有包括服务发现[13]、数据库 [14]、授权认证等是共用的。所以为了减少代码的重复使用，则独立出这一部分。

[12]后面随时可能会称之为微架构。

[13]目前的版本并没有开发实际的服务发现的内容，直接使用 Nginx 进行做均衡负载等。

[14]这一部分单独出来的。

12 Dindo 数据库

13 Dindo Launcher

14 Dindo 微服务组件——用户管理

15 DIndo 测试说明

15.1 如何测试

A 术语解释

CaaS Container as a Server，是指将容器（Docker）提供作为一种服务。是云计算中的概念，与 PaaS、SaaS 等概念对等。

B Docker 中 Weave 的配置

Weave 是能将 Docker 中每个物理主机中的连接起来一个工具，也就是能使的 Docker 容器跨主机互联。下面是配置（安装）Weave 的 Shell 脚本:

Listing 1: Weave 安装

```
1  #!/bin/sh
2  wget -O /usr/local/bin/weave \
3  https://github.com/zettio/weave/releases/download/latest_release/weave
4  chmod a+x /usr/local/bin/weave
5  dao pull weaveworks/weave:1.5.1
6  dao pull weaveworks/plugin:1.5.1
7  dao pull weaveworks/weaveexec:1.5.1
8  apt-get update
9  apt-get install bridge-utils
10 dao pull weaveworks/weavedb:latest
11 weave launch 192.168.1.181
```

运行容器需要使用

```
# weave run <ip> <repo>
```

C 后端附带工具使用说明

C.1 dindo-pash

dindo-pash 是用于测试期间生成密码的工具，具体使用请参照 **??** 部分。

D 发行（发布）的二进制文件镜像与包的命名规则

这一部分的内容是关于发布或发行的二进制文件包或者 Docker 镜像的命名规则。(构建类型 _ 构建编号)-([commit hash] | [tag name])-(操作系统体系 _ 发行版本)-(编译系统体系 _ 版本)-(cpu 架构体系)-[llvm_ 版本]-[threaded]-[其他特性]-(模块) 例如某二进制包的文件名：single-7a8c900-win32_windows_10_rs1_14342-x86_64-GHC_8.0.1-llvm_3.8-threaded-all_in_one.tar.xz

参考文献

[1] 灵雀云收费标准 2016 年 5 月，Alauda-Price