Dingo

Dindo Document

DingoLab

May,2016

Dingo Dindo Document

李约瀚
qinka@live.com
14130140331

May 29th 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

```
module Dindo.Database where
```

```
    import Prelude hiding (String)
```

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

```
3   import Import
4   import Data.Text
5   import Data.ByteString
6   import Paths_dindo_database
7   import Data.Version
```

```
8   instance FromJSON ByteString where
9     parseJSON (String x) = pure $ encodeUtf8 x
10  instance ToJSON ByteString where
11    toJSON = String . decodeUtf8
```

```
12  share [mkPersist sqlSettings] [persistLowerCase|
13  Account json sql=table_account
14    Id sql=
15    uid Text sql=key_uid sqltype=varchar(64)
16    pash Text sql=key_pash sqltype=varcher(64)
17    tel Int sql=key_tel
18    name Text sql=key_name sqltype=varchar(64)
19    Primary uid
20    deriving Show Eq
21  Usr json sql=table_usr
22    Id sql=
23    uid Text sql=key_uid sqltype=varchar(64)
24    email Text sql=key_email
25    rname Text sql=key_rname sqltype=varchar(64)
26    prcid Text sql=key_prcid sqltype=varchar(18)
27    addr Text sql=key_addr
28    status Text sql=key_status sqltype=varchar(1)
29    Primary uid
30    Foreign Account fkuid uid
31    deriving Show Eq
32  Addr json sql=table_addr
33    Id sql=
34    aid Text sql=key_aid sqltype=varchar(64)
```

```
35          uid Text sql=key_uid sqltype=varchar(64)
36          zip Text sql=key_zip sqltype=varchar(64)
37          addr Text sql=key_addr
38          Primary aid
39          Foreign Account fkaddruid uid
40          deriving Show Eq
41        Apic sql=table_apic
42          Id sql=
43          pid Text sql=key_pic_id sqltype=varchar(64)
44          uid Text sql=key_uid sqltype=varchar(64)
45          bpic ByteString sql=binary_pic
46          typ Int Maybe sql=key_status default=0
47          Primary pid
48          Foreign Account fkuidb uid
49          deriving Show Eq
50        Task json sql=table_task
51          Id sql=
52          tid Text sql=key_tid sqltype=varchar(64)
53          ca Text Maybe sql=key_ca sqltype=varchar(64)
54          cb Text Maybe sql=key_cb sqltype=varchat(64)
55          Primary tid
56          Foreign Account fkca ca
57          Foreign Account fkvcb cb
58          deriving Show Eq
59        Taskinfo json sql=table_task_info
60          Id sql=
61          tid Text sql=key_tid sqltype=varchar(64)
62          ew Double sql=key_ew
63          ns Double sql=key_ns
64          r Double sql=key_r
65          w Double sql=key_w
66          size [Double] sql=key_size
67          note Text Maybe sql=key-note
68          cost Int sql=key_note
```

```
      des Text Maybe sql=key_des
      Primary tid
      Foreign Task fktid tid
      deriving Show Eq
    Taskcost json sql=table_task_cost
      Id sql=
      tid Text sql=key_tid sqltype=varchar(64)
      ad [Int] sql=key_ad
      bd [Int] sql=key_bd
      Primary tid
      Foreign Task fktidb tid
      deriving Show Eq
    Dd json sql=table_dd
      Id sql=
      did Text sql=key_did sqltype=varchar(64)
      uid Text sql=key_tid sqltype=varchar(64)
      dd Text sql=key_dd
      ew Double sql=key_ew
      ns Double sql=key_ns
      r Double sql=key_r
      Primary did
      Foreign Account fkuidc uid
    TmpToken json sql=table_tmptoken
      Id sql=
      tt Text sql=key_tmptoken sqltype=varchar(150)
      time UTCTime sql=key_timeup
      uid Text sql=key_uid sqltype=varchar(64)
      Primary tt
      Foreign Account fkuidd uid
    |]
```

```
    dindo_database_version = version
    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

```
module Import
( module X
, persistFileWithC
) where
```

```
import Language.Haskell.TH as X
import Data.Aeson as X
import Database.Persist as X
import Data.Text.Encoding as X
import Database.Persist.TH as X
import Database.Persist.Quasi as X
import Data.Time as X
```

```
persistFileWithC :: PersistSettings
                 -> FilePath
                 -> Q Exp
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

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

```
      import Data.Maybe as X
      import Data.Time as X
```

```
6         import Dindo.MicroFramework.Register as X
7         import Dindo.MicroFramework.Destory as X
8         import Dindo.MicroFramework.API as X
9         import Data.Conduit as X
```

10.2.2 src/Dindo/Import/Aeson.lhs

导入 Data.Aeson 及相关内容

```
1 module Dindo.Import.Aeson
2     ( module X
3     ) where
4         import Data.Aeson as X
```

10.2.3 src/Dindo/Import/ByteString.lhs

导入 bytestring 包中相关模块

```
1 module Dindo.Import.ByteString
2     ( module X
3     , fromStrictBS
4     ) where
```

```
5         import Data.ByteString as X
6         import Data.ByteString.Lazy
7         fromStrictBS = fromStrict
```

10.2.4 src/Dindo/Import/Database.lhs

导入与数据库相关的模块

```
1 module Dindo.Import.Database
2     ( module X
3     , tryRunDB
4     ) where
```

```
5       import Database.Persist as X
6       import Database.Persist.Postgresql as X
7       import Dindo.Database as X
8       import Control.Exception
9       import Yesod
```

```
10      tryRunDB :: ( Yesod site
11                  , YesodPersist site
12                  , YesodPersistBackend site ~ SqlBackend
13                  )
14                => YesodDB site a -> HandlerT site IO (Either SomeException a)
15      tryRunDB f = do
16        runInnerHandler <- handlerToIO
17        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

```
1  module Dindo.Import.Digest
2      ( module X
3      ) where
4        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

```
1 module Dindo.Import.Yaml
2     ( module X
3     ) where
4       import Data.Yaml as X
```

10.2.10 src/Dindo/Import/Yesod.lhs

导入与 Yesod 有关的模块

```
1 module Dindo.Import.Yesod
2     ( module X
```

```
3       , mkYesodData
4       , mkGetSvrInfoAuthor
5       , getSvrtimeR
6       , mkSvrinfoR
7       ) where
```

```
8       import Yesod as X hiding (mkYesodData)
9       import qualified Yesod (mkYesodData)
10      import Dindo.Common.Rable as X
11      import Dindo.Common.Auth as X
12      import Dindo.Common.Yesod.Launch as X
13      import Dindo.Common.Yesod.Config as X
14      import Dindo.Import.TH
15      import Data.Maybe
16      import Data.Time
17      import Data.Text
```

```
18      mkYesodData a b = Yesod.mkYesodData a b'
19        where
20          b' = b ++ [parseRoutes|
21            /svrtime SvrtimeR GET
22            /svrinfo SvrinfoR GET
23            |]
24      mkGetSvrInfoAuthor :: Q [Dec]
25      mkGetSvrInfoAuthor = return $
26        [FunD (mkName "isAuthorized") [Clause [VarP (mkName "GettimeR"),WildP] (
              NormalB (AppE (VarE (mkName "return")) (ConE (mkName "Authorized")))) []]
27        ,FunD (mkName "isAuthorized") [Clause [VarP (mkName "GetinfoR"),WildP] (
              NormalB (AppE (VarE (mkName "return")) (ConE (mkName "Authorized")))) []]
28        ]
29      getSvrtimeR :: Yesod site => HandlerT site IO Text
30      getSvrtimeR = do
31        addD' <- lookupGetParam "add"
32        let addD = fromRational $ toRational $ fromMaybe 0 $ fmap (read.unpack) addD'
```

```
33          now <- liftIO getCurrentTime
34          return $ pack $ show $ addUTCTime addD now
35        mkSvrinfoR :: Text -> Q [Dec]
36        mkSvrinfoR info = [d|
37          getSvrinfoR :: Yesod site => HandlerT site IO Text
38          getSvrinfoR = return info
39          |]
```

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  module Dindo.Common.Auth
2      ( runPash
3      , tokenAuth
4      , pskAuth
5      , noAuth
6      , fromEntity
```

```
7       , pickF
8       , pickU
9       , getUid
10      ) where
```

```
11      import Yesod
12      import Database.Persist
13      import Database.Persist.Sql
14      import Dindo.Database
15      import Data.Time
16      import Data.Text.Encoding
17      import Data.Maybe
18      import qualified Data.ByteString as B
19      import qualified Data.ByteString.Lazy as B hiding (concat,ByteString)
20      import Data.Text (unpack,pack,Text)
21      import Data.Digest.Pure.SHA
```

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

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

```
39  runPash :: Int -> B.ByteString -> Text -> Text
40  runPash i time pash = pack $ showDigest $ sha512 $ B.fromStrict $ B.concat [pre,
        encodeUtf8 pash,time]
41    where
42      pre = case i of
43        0 -> "uid"
44        1 -> "nnnn"
45        2 -> "+86"
46  runPash _ _ x = id x
47  noAuth :: Yesod site => HandlerT site IO AuthResult
48  noAuth = return Authorized
49
50  tokenAuth :: ( Yesod site
51               , YesodPersist site
52               , YesodPersistBackend site ~ SqlBackend
53               )
54            => HandlerT site IO AuthResult
55  tokenAuth = do
56    token' <- lookupHeader "TMP-TOKEN"
57    case token' of
58      Nothing -> return $ Unauthorized "Who are you!"
59      Just token -> do
60        rt' <- liftHandlerT $ runDB $ selectList [TmpTokenTt ==. decodeUtf8 token][
              Desc TmpTokenTime]
61        case rt' of
62          rt:_ -> do
63            now <- liftIO getCurrentTime
64            let time = tmpTokenTime.fromEntity $ rt
65            if diffUTCTime now time >= 0
66              then return $ Unauthorized "Who are you!"
67              else return Authorized
68          _ -> return $ Unauthorized "Who are you!"
69
```

```
70      pskAuth :: ( Yesod site
71                 , YesodPersist site
72                 , YesodPersistBackend site ~ SqlBackend
73                 )
74              => HandlerT site IO AuthResult
75      pskAuth = checkTime $ \time -> do
76        pash  <- getPash
77        uid'  <- lookupPostParam "uid"
78        name' <- lookupPostParam "name"
79        tel'' <- lookupPostParam "tel"
80        let tel' = fmap (read.unpack) tel'' :: Maybe Int
81        case (uid', name', tel') of
82          (Nothing,Nothing,Nothing) -> return $ Unauthorized "Who␣are␣you!"
83          (Just uid,name,tel) -> do
84            rt <- liftHandlerT $ runDB $ selectList (
85              [AccountUid ==. uid] ++ pickF [(AccountName,name)]++pickF [(AccountTel,
                    tel)]) []
86            checkPash pash rt (runPash 0 time)
87          (Nothing,Just name,tel) -> do
88            rt <- liftHandlerT $ runDB $ selectList (
89              [AccountName ==. name] ++ pickF [(AccountTel,tel)]) []
90            checkPash pash rt (runPash 1 time)
91          (Nothing,Nothing,Just tel) -> do
92            rt <- liftHandlerT $ runDB $ selectList
93              [AccountTel ==. tel] []
94            checkPash pash rt (runPash 2 time)
95          _ -> return $ Unauthorized "Who␣are␣you!"
96        where
97          getPash = do
98            pash' <- lookupPostParam "pash"
99            return $ fromMaybe "" pash'
100         checkPash pash rt f = do
101           case rt of
102             item:_ -> do
```

```
103                 let usrPash = f.accountPash.fromEntity $ item
104                 if usrPash == pash
105                   then return Authorized
106                   else return $ Unauthorized "Who␣are␣you!"
107               _ -> return $ Unauthorized "Who␣are␣you!"
108           checkTime f = do
109             time' <- liftHandlerT $ lookupHeader "TIME-STAMP"
110             now <- liftIO getCurrentTime
111             case time' of
112               Just time -> do
113                 let t = read.unpack.decodeUtf8 $ time
114                 let diff = diffUTCTime now t
115                 if diff <= 12 && diff >= (-12)
116                   then f time
117                   else return $ Unauthorized "I␣bought␣a␣watch␣last␣year!"
118               _ -> return $ Unauthorized "I␣bought␣a␣watch␣last␣year!"
119
120       fromEntity :: Entity a -> a
121       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  module Dindo.Common.Rable
2      ( RtType(..)
3      , RtWhere(..)
4      , Varable (..)
5      , defToContent
6      , defToContentXml
7      , defToContentYaml
8      , defToContentJson
9      , Rable (..)
```

```
    , defReturnR
    , RtStatus (..)
    , statusHead
    ) where
```

```
    import Data.Aeson as A
    import Data.Yaml as Y
    import Text.XML as X
    import Text.Hamlet.XML
    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

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

```
    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
```

```
40  defToContentYaml :: Varable a => a -> BI.ByteString
41  defToContentYaml = Y.encode . toValue
42  defToContentXml :: Varable a => a -> BI.ByteString
43  defToContentXml x = toStrict $ renderLBS def $ Document p root []
44    where
45      root = Element "data" (fromList []) $ toNodes x
46      p = Prologue [] Nothing []
```

```
47  class Varable a => Rable a where
48    toWhere :: a -> RtWhere
49    toStatus :: a -> RtStatus
50    returnR :: MonadHandler m => a -> m TypedContent
51    returnR = defReturnR
52  defReturnR :: ( MonadHandler m
53                , Rable a
54                )
55              => a -> m TypedContent
56  defReturnR x = do
57    addHeader "Status" $ status x
58    if toWhere x == RtBody
59      then addHeader "Context-Where" "Body"
60      else addHeader "Context-Where" $ (\(RtOther a)-> a) $ toWhere x
61    addContent
62    where
63      status = statusHead.toStatus
64      addContent = case toWhere x of
65        RtBody -> selectRep $ do
66          provideRepType "application/json" $ return $ decodeUtf8 $ toContents RtJson
                x
67          provideRepType "application/yaml" $ return $ decodeUtf8 $ toContents RtYaml
                x
68          provideRepType "application/xml" $ return $ decodeUtf8 $ toContents RtXml
                x
69        RtOther y -> do
```

```
70                addHeader y $ pack $ show x
71                selectRep $  provideRep $ return ("" :: Text)
```

```
72        data RtStatus = RtSucc | RtFail
73        statusHead :: RtStatus -> Text
74        statusHead RtSucc = "Success"
75        statusHead RtFail = "Failed"
```

将 Yesod 中的 ErrorResponse 实现 Varable 与 Rable

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

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

提供模块配置的部分

```
1   module Dindo.Common.Yesod.Config
2       ( SvrConfig (..)
```

```
3       , DbConfig(..)
4       , ScError (..)
5       , scError
6       , dbConfig2Str
7       ) where
```

```
8       import Data.Yaml
9       import Data.ByteString as B
10      import Data.ByteString.Lazy
11      import Data.String
12      import Control.Exception
```

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

```
13      data SvrConfig = SvrConfig
14        { svrPort :: Int
15        , svrDb :: DbConfig
16        }
17      data DbConfig = DbConfig
18        { dbAddr :: String
19        , dbPort :: String
20        , dbUser :: String
21        , dbName :: String
```

```
22          , dbPsk  :: String
23          , dbConThd :: Int
24          }
```

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。

```
25      instance ToJSON SvrConfig where
26        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        parseJSON _ = throw $ ScError "Invailed"
44      instance FromJSON DbConfig where
45        parseJSON (Object v) =DbConfig
46          <$> v .: "addr"
47          <*> v .: "port"
48          <*> v .: "user"
49          <*> v .: "name"
50          <*> v .: "password"
51          <*> v .: "con-limit"
52        parseJSON _ = throw $ ScError "Invailed"
```

将数据库配置转化成链接字符串。

```
53  dbConfig2Str :: DbConfig -> (B.ByteString,Int)
54  dbConfig2Str DbConfig{..} = (str,dbConThd)
55    where
56      str = toStrict $
57        fromString $    "host=\'" ++ dbAddr
58                   ++ "\' port=\'" ++ dbPort
59                   ++ "\' user=\'" ++ dbUser
60                   ++ "\' password=\'" ++ dbPsk
61                   ++ "\' dbname=\'" ++ dbName
62                   ++ "\'"
```

设置读写异常

```
63  data ScError = ScError String
64    deriving (Eq)
65  scError = throw.ScError
66  instance Show ScError where
67    show (ScError e) = "parse server config file FAILED:\n\t" ++ e
68  instance Exception ScError where
69    displayException e = "parse server config file FAILED:\n\t"
```

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
```

```
database-config:
  addr: '127.0.0.1'
  port: '5432'
  user: postgres
  name: postgres
  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

```
module Dindo.Common.Yesod.Launch
    ( Dindoble(..)
    ) where
```

```
      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”

```
      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
```

```
19              (connStr,cT) = dbConfig2Str.svrDb $ x
20              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

```
1  module Dindo.MicroFramework.API
2      ( APIble(..)
3      , regAPI
4      ) where
```

```
5      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
```

```haskell
import Yesod.Core
import Control.Concurrent

import Dindo.MicroFramework.API
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

```haskell
class ( Yesod a
      , APIble a
      , Destorible a
      , Heartbeatable a
      ) => Registrable a where
  regAddr :: a -> String
  regAddr = defRegAddr
  regPort :: a -> Int
  regPort = defRegPort
  regSvrAddr :: a -> String
  regSvrPort :: a -> Int
defRegPort _ = 3000
defRegAddr _ = "localhost"
```

状态获取的类型类

```haskell
class ( Yesod a
      , RenderRoute a
      ) => Heartbeatable a where
  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

```haskell
module Main
    ( main
    ) where
```

```haskell
    import qualified GHC.IO.Encoding as E
    import System.IO
    import Dindo.Std
    import System.Console.CmdArgs
    import Dindo.Import.Aeson as A
    import Dindo.Import.Yaml as Y
    import Dindo.Import.Yesod
    import Data.Maybe
    import qualified Dindo.Import.ByteString as B
    import qualified Dindo.Import.Text as T
    import Dindo.Common.Yesod.Launch
    import Dindo.Common.Yesod.Config
    import Paths_dindo_launch
    import Data.Version
    import Dindo.Common(dindo_common_version_quasi)
    import Dindo.Import.Database(dindo_database_version_quasi)
    import Control.Exception(try,SomeException,ErrorCall (..),throw,evaluate)
    import Data.Char
```

启动方式是通过标准输入流输入，输入的格式是 JSON 或者是 YAML， “–form=” 这个选项是控制输入或输出的是的，是 JSON 或者是 YAML。

```haskell
    data Launch = Launch {form ::String}
      deriving (Show,Data,Typeable)
    launch = Launch{form="auto" &= typ "AUTO|YAML|JSON" &= help "格式"}
      &= summary ( "dindo-common-"
                   ++ $(dindo_common_version_quasi)
                   ++ "; dindo-database-"
                   ++ $(dindo_database_version_quasi)
                   ++ "; " ++ $(dindo_module_name) ++ "-"
                   ++ $(dindo_module_version)
```

```
                    ++ ";␣dindo-launch-"
                    ++ showVersion version)
```

```
      main :: IO ()
      main = do
#ifndef WithoutUTF8
        E.setLocaleEncoding E.utf8
        hSetEncoding stdout utf8
#endif
        cfg' <- cmdArgs launch >>= cfg
        warpDindo cfg' itemWarp
        where
          itemWarp :: Int -> $(std) -> IO()
          itemWarp = warp
      cfg :: Launch -> IO SvrConfig
      cfg l = getContents >>= (decode'.T.encodeUtf8.T.pack)
        where
          tryList :: [a -> SvrConfig] -> [ScError] -> a -> IO SvrConfig
          tryList [] es a = scError.concatWith "\n\t".map getError $ es
          tryList (x:xs) es a = do
            rt <- try.evaluate $ x a :: IO (Either ScError SvrConfig)
            case rt of
              Left e -> tryList xs (e:es) a
              Right sc -> return sc
          getError (ScError a) = a
          concatWith a xs = foldr sig "all␣failed" xs
            where
              sig x os = x ++ a ++ os
          decJ = fromMaybe (throw $ ScError "Invailed␣JSON").A.decode.B.fromStrictBS
          decY = fromMaybe (throw $ ScError "Invailed␣YAML").Y.decode
          decA = tryList [decY,decJ] []
          decode' = let Launch ll = l in
            case map toLower ll of
              "auto" -> decA
```

```
64                  "json" -> evaluate.decJ
65                  "yaml" -> evaluate.decY
66                  _ -> 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

```
1  module Dindo.Std
2      ( module X
3      , std
4      , dindo_module_name
5      , dindo_module_version
6      ) where
7
8        import Dindo.UM as X -- need change
9        import Dindo.Import.TH
10       import Dindo.Import.TH
11       dindo_module_name = stringE "dindo-usrmanage"
12       dindo_module_version = dindo_usrmanage_version_quasi
13       std = [t|UM|]
```

10.4.2 src/Dindo/UM.lhs

用户管理部分的导出的部分

```
1  module Dindo.UM
2      ( module X
3      , dindo_usrmanage_version
4      , dindo_usrmanage_version_quasi
5      ) where
6        import Dindo.UM.Foundation as X
```

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

10.4.3 src/Dindo/UM/Data.lhs

定义返回数据的部分

```
1  module Dindo.UM.Data
2      ( RtRegist (..)
3      , RtIdy (..)
4      , RtIdfed (..)
5      , RtCommon(..)
6      , RtUImg(..)
7      , RtUInfo (..)
8      , RtChPsk(..)
9      , RtEaddr(..)
10     , RtGEadd(..)
11     ) where
```

```
12        import Dindo.Import.Rable
13        import Dindo.Import.Aeson as A
14        import Dindo.Import.Yaml as Y
15        import Dindo.Import.Text as T
16        import Dindo.Import.ByteString as B
17        import Dindo.Import.Yesod
18        import Dindo.Import.Database
```

用户注册返回数据

```
19  data RtRegist = RtRegist
20      { uid :: Text
21      }
22    | RtRegistFail
23      { regReason :: Text
24      }
25    deriving (Eq)
26  instance Show RtRegist where
27    show (RtRegist x) = T.unpack x
28    show (RtRegistFail x) = T.unpack x
29  instance Varable RtRegist where
30    toValue (RtRegist x) = object ["uid" .= x]
31    toValue (RtRegistFail x) = object ["error" .= x]
32    toNodes (RtRegist x) = [xml|<uid>#{x}|]
33    toNodes (RtRegistFail x) =[xml|<error>#{x}|]
34  instance Rable RtRegist where
35    toWhere (RtRegist _) = RtBody
36    toWhere (RtRegistFail _) = RtBody
37    toStatus (RtRegist _) = RtSucc
38    toStatus (RtRegistFail _) = RtFail
```

用户认证信息的返回数据

```
39  data RtIdy = RtIdy
40    | RtIdyFail
41      { idyReason :: Text
42      }
43    deriving (Eq)
44  instance Show RtIdy where
45    show (RtIdyFail x) = T.unpack x
46  instance Varable RtIdy where
47    toValue RtIdy = Null
48    toValue (RtIdyFail x) = object ["error" .= x]
49    toNodes RtIdy = [xml|null|]
```

```
50          toNodes (RtIdyFail x) = [xml|<error>#{x}|]
51        instance Rable RtIdy where
52          toWhere (RtIdyFail _) = RtBody
53          toWhere RtIdy = RtBody
54          toStatus RtIdy = RtSucc
55          toStatus (RtIdyFail _) = RtFail
```

用户查询认证状态信息

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

通用成功与失败标志

```
68        data RtCommon = RtCommonSucc
69                      | RtCommonSuccT Text
70                      | RtCommonFail Text
71          deriving (Eq,Show)
72        instance Varable RtCommon where
73          toValue RtCommonSucc = Null
74          toValue (RtCommonSuccT t) = object ["tmp-token" .= t]
75          toValue (RtCommonFail x) = String x
76          toNodes RtCommonSucc = [xml|null|]
77          toNodes (RtCommonSuccT x) = [xml|<tmp-token>#{x}|]
78          toNodes (RtCommonFail x) = [xml|<error>#{x}|]
79        instance Rable RtCommon where
```

```
80          toWhere RtCommonSucc = RtBody
81          toWhere (RtCommonFail _) = RtBody
82          toWhere (RtCommonSuccT _) = RtBody
83          toStatus RtCommonSucc = RtSucc
84          toStatus (RtCommonSuccT _) = RtSucc
85          toStatus (RtCommonFail _) = RtFail
```

用户信息查询返回结果

```
86      data RtUInfo = RtUInfo
87          { rtuiUid :: Text
88          , rtuiName :: Text
89          , rtuiTel :: Text
90          , rtuiEmail :: Text
91          }
92        | RtUInfoNSU
93      instance Show RtUInfo where
94        show RtUInfoNSU = "no such a user"
95      instance Varable RtUInfo where
96        toValue RtUInfo{..} = object
97          [ "uid" .= rtuiUid
98          , "name" .= rtuiName
99          , "tel" .= rtuiTel
100         , "email" .= rtuiEmail
101         ]
102       toNodes RtUInfo{..} = [xml|
103       <uid> #{rtuiUid}
104       <name> #{rtuiName}
105       <tel> #{rtuiTel}
106       <email> #{rtuiEmail}
107       |]
108     instance Rable RtUInfo where
109       toWhere RtUInfo{..} = RtBody
110       toWhere RtUInfoNSU = RtOther "CONTEXT"
111       toStatus RtUInfo{..} = RtSucc
```

```
112         toStatus RtUInfoNSU = RtFail
```

获取用户头像返回内容

```
113     data RtUImg = RtUImg ByteString
114                 | RtUImgFail
115       deriving (Eq)
116     instance Show RtUImg
117     instance Varable RtUImg
118     instance Rable RtUImg where
119       returnR (RtUImg img) =
120         selectRep $ provideRepType "image/png" $ return img
121       returnR RtUImgFail = do
122         addHeader "CONTEXT-WHERE" "CONTEXT"
123         addHeader "CONTEXT" "Failed on get image"
124         selectRep $ provideRep $ return ("" :: Text)
```

更改密码的返回值

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

收货地址的增删的返回值

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

获取地址

```
164 data RtGEadd = RtGEadd [Addr]
165                | RtGEaddFail Text
166   deriving (Eq,Show)
167 instance Varable RtGEadd where
168   toValue (RtGEadd x) = toJSON x
169   toValue (RtGEaddFail x) = object ["error" .= x]
170   toNodes (RtGEadd xs) = [xml|
171     $forall x <- xs
```

```
172                 <aid>#{addrAid x}
173                 <addr>#{addrAddr x}
174                 <zip>#{addrZip x}
175               |]
176           toNodes (RtGEaddFail x) = [xml|<error>#{x}|]
177         instance Rable RtGEadd where
178           toWhere (RtGEadd _ ) = RtBody
179           toWhere (RtGEaddFail _) = RtBody
180           toStatus  (RtGEadd _) = RtSucc
181           toStatus  (RtGEaddFail _) =RtFail
```

10.4.4 src/Dindo/UM/Foundation.lhs

基础的部分

```
1  module Dindo.UM.Foundation
2      ( module Dindo.UM.Foundation
3      , getSvrtimeR
4      ) where
```

```
5          import Dindo.Common
6          import Dindo.Import
7          import Dindo.Import.Yesod
8          import Dindo.Import.Database
9          import Paths_dindo_usrmanage
10         import Dindo.Import.Text as T
11         import Data.Version
```

```
12         data UM = UM
13           { connPool :: ConnectionPool
14           , config   :: SvrConfig
15           }
16         mkYesodData "UM" [parseRoutes|
17           / regist  RegistR POST
18           / identify  IdentifyR  POST
```

```
19          / identified   Identified  POST
20          /login  LoginR POST
21          /logout  LogoutR POST
22          /usrinfo  UsrinfoR  POST
23          /usrhimg UsrhimgR POST
24          /usrinfochange  UsrinfochangeR POST
25          /changpash ChangpashR POST
26          /upeaddr UpeaddrR POST
27          /geteaddr  GeteaddR POST
28        |]
```

实现 Yesod 类型类

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

微服务架构

```
40      instance APIble UM where
41        apis  _ = []
42      instance  Destorible  UM where
43        destoryHead _ = ""
44        destoryAPI _ = ""
45      instance Heartbeatable  UM where
46        heartbeat  _ = return ()
47      instance  Registrable  UM where
```

```
48            regAddr _ = ""
49            regPort = svrPort . config
50            regSvrPort _ = 80
51            regSvrAddr _ = ""
52          instance Dindoble UM where
53            fromPool = UM
```

10.4.5 src/Dindo/UM/Handler.lhs

处理函数的部分

```
1  module Dindo.UM.Handler
2      ( postRegistR
3      , postUsrinfoR
4      , postLogoutR
5      , postLoginR
6      , postIdentified
7      , postIdentifyR
8      , postUsrinfochangeR
9      , postChangpashR
10     , postUsrhimgR
11     , postUpeaddrR
12     , postGeteaddR
13     ) where
```

```
14         import Dindo.Import
15         import Dindo.Import.Yesod
16         import Dindo.Import.Database
17         import Dindo.UM.Foundation
18         import Dindo.UM.Data
19         import Dindo.Import.Digest
20         import Dindo.Import.ByteString as B hiding(unpack,pack,splitAt,take,map,null)
21         import Dindo.Import.Text as T hiding(splitAt,take,map,null)
22         import Dindo.Common.Auth(fromEntity,pickU,pickF)
23         import Control.Exception(try,SomeException)
```

```
24  import Control.Monad
```

注册的 API

```
25  postRegistR :: Handler TypedContent
26  postRegistR =
27    getParam insertAItem
28    where
29      getParam f = do
30        name' <- lookupPostParam "name"
31        pash' <- lookupPostParam "pash"
32        tel'  <- lookupPostParam "tel"
33        case (name',pash', tel') of
34          (Just name,Just pash,Just tel) -> do
35            x <- liftIO getCurrentTime
36            let (time,p) = splitAt 10 $ show x
37            let to = showDigest $ sha1 $ fromStrictBS $ encodeUtf8 $ T.concat [pash,
                  name]
38            let uid = 'U':time ++ to
39            f (pack uid,name,pash,read (unpack tel))
40          _ -> returnR $ RtRegistFail "param:␣less␣and␣less"
41      insertAItem (uid,name,pash,tel) = do
42        rt <- liftHandlerT $ tryRunDB $
43          insert $ Account uid pash tel name
44        returnR $ case rt of
45          Left e -> RtRegistFail $ pack $ show e
46          Right _ -> RtRegist uid
```

用户认证的 API

```
47  postIdentifyR :: Handler TypedContent
48  postIdentifyR =
49    checkParam $ addItem $ checkPic addPic
50    where
51      checkParam f = do
52        email' <- lookupPostParam "email"
```

```
53              rname' <- lookupPostParam "rname"
54              prcid' <- lookupPostParam "prcid"
55              addr' <- lookupPostParam "addr"
56              case (email', rname', prcid', addr') of
57                (Just email, Just rname, Just prcid, Just addr) ->
58                  f (email, rname, prcid, addr)
59                _ -> returnR $ RtIdyFail "param: less and less"
60            checkPic f ins = do
61              pic' <- lookupFile "pic"
62              case pic' of
63                Just pic -> do
64                  rt <- sourceToList $ fileSource pic
65                  let bpic = B.concat rt
66                  f (bpic, ins)
67                _ -> returnR $ RtIdyFail "param: picture needed"
68            addItem f (email, rname, prcid, addr) =
69              f $ \uid -> Usr uid email rname prcid addr "N"
70            addPic (pic, usr) = do
71              uid <- getUid
72              now <- liftIO getCurrentTime
73              let str = show now
74              let (time, p) = splitAt 10 $ str
75              let to = showDigest $ sha1 $ fromStrictBS $ encodeUtf8 $ T.concat [uid, pack
                    str]
76              let pid = pack $ 'A':time ++ to
77              rt <- liftHandlerT $ tryRunDB $ do
78                insert $ usr uid
79                insert $ Apic pid uid pic $ Just 0
80              returnR $ case rt of
81                Left e -> RtIdyFail $ pack $ show e
82                Right _ -> RtIdy
```

认证状态查询

```
83          postIdentified :: Handler TypedContent
```

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

用户登录

```
92        postLoginR :: Handler TypedContent
93        postLoginR = do
94          uid' <- lookupPostParam "uid"
95          name' <- lookupPostParam "name"
96          tel' <- lookupPostParam "tel"
97          case (uid', name', tel') of
98            (uid,name,tel) -> do
99              pash <- getPash
100             rt' <- liftHandlerT $ runDB $ selectList (pickF
101               [ (AccountUid, uid)
102               , (AccountName, name)
103               ] ++ pickF
104               [ (AccountTel,fmap (read.unpack) tel)
105               ]) []
106             case rt' of
107               (Entity _ item):_ -> do
108                 let uid = accountUid item
109                 now <- liftIO getCurrentTime
110                 let lim = addUTCTime 3600 now
111                 let time = show lim
112                 let to = showDigest $ sha512 $ fromStrictBS $ encodeUtf8 $ T.concat [uid,
                        pash,pack time]
113                 let tt = pack $ take 22 time ++ to
114                 liftHandlerT $ runDB $ insert $ TmpToken tt lim uid
```

```
115                 returnR $ RtCommonSuccT tt
116           where
117             getPash = do
118               pash' <- lookupPostParam "pash"
119               return $ fromMaybe "" pash'
```

用户登出

```
120       postLogoutR :: Handler TypedContent
121       postLogoutR = do
122         Just token <- lookupHeader "TMP-TOKEN"
123         Just uid <- lookupHeader "USR-ID"
124         rt <- liftHandlerT $ tryRunDB $ deleteWhere [TmpTokenTt ==. decodeUtf8 token,
                TmpTokenUid ==. (read.unpack.decodeUtf8) uid]
125         returnR $ case rt of
126           Left e -> RtCommonFail $ pack $ show e
127           Right _ -> RtCommonSucc
```

查询用户信息

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

获得用户头像

```
140       postUsrhimgR :: Handler TypedContent
```

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

用户信息变更

```
149     postUsrinfochangeR :: Handler TypedContent
150     postUsrinfochangeR = check update
151       where
152         updatePic uid pic' = case pic' of
153           Nothing -> return ()
154           Just pic -> do
155             rt <- sourceToList $ fileSource pic
156             let bpic = B.concat rt
157             updateWhere [ApicUid ==. uid,ApicTyp ==. Just 0] [ApicBpic =. bpic]
158         update (a,b,pic) = do
159           uid <- getUid
160           rt <- liftHandlerT $ tryRunDB $ do
161             when (not $ null a) $
162               updateWhere [AccountUid ==. uid] a
163             when (not $ null b) $
164               updateWhere [UsrUid ==. uid] b
165             updatePic uid pic
166           returnR $ case rt of
167             Left e -> RtCommonFail $ pack $ show e
168             Right _ -> RtCommonSucc
169         check f = do
170           name <- liftHandlerT $ lookupPostParam "name"
171           tel  <- liftHandlerT $ lookupPostParam "tel"
172           email <- liftHandlerT $ lookupPostParam "email"
```

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

修改密码

```
181     postChangpashR :: Handler TypedContent
182     postChangpashR = check changePash
183       where
184         changePash pash = do
185           uid <- getUid
186           rt <- liftHandlerT $ tryRunDB $ updateWhere [AccountUid ==. uid] [
                  AccountPash =. pash]
187           returnR $ case rt of
188             Left e -> RtChPskFail $ pack $ show e
189             Right _ -> RtChPsk
190         check f = do
191           pash' <- lookupPostParam "pash"
192           case pash' of
193             Nothing -> do
194               returnR $ RtChPskFail "param:␣less␣and␣less"
195             Just x -> f x
```

收获地址

```
196     postUpeaddrR :: Handler TypedContent
197     postUpeaddrR = spl
198       where
199         changeItem aid a = do
200           rt <- liftHandlerT $ tryRunDB $ updateWhere [AddrAid ==. aid] a
```

```
201             returnR $ case rt of
202               Left e -> RtEaddrFail $ pack $ show e
203               Right _ -> RtEaddrChn
204           checkChn f = do
205             addr <- liftHandlerT $ lookupPostParam "addr"
206             zipcode <- liftHandlerT $ lookupPostParam "zip"
207             aid' <- liftHandlerT $ lookupPostParam "aid"
208             case aid' of
209               Just aid -> f aid $ pickU [(AddrAddr,addr),(AddrZip,zipcode)]
210               Nothing -> returnR $ RtEaddrFail "param:change:␣less␣and␣less"
211           delItem aid = do
212             rt <- liftHandlerT $ tryRunDB $ deleteWhere [AddrAid ==. aid]
213             returnR $ case rt of
214               Left e -> RtEaddrFail $ pack $ show e
215               Right _ -> RtEaddrDel
216           checkDel f = do
217             aid' <- liftHandlerT $ lookupPostParam "aid"
218             case aid' of
219               Just aid -> f aid
220               Nothing -> returnR $ RtEaddrFail "param:del:␣less␣and␣less"
221           addItem (addr,zipcode) = do
222             uid <- getUid
223             now <- liftIO getCurrentTime
224             let aid' = showDigest $ sha256 $ fromStrictBS $ encodeUtf8 addr
225             let aid = pack $ "A"++show now++aid'
226             rt <- liftHandlerT $ tryRunDB $ insert $ Addr aid uid zipcode addr
227             returnR $ case rt of
228               Left e -> RtEaddrFail $ pack $ show e
229               Right _ -> RtEaddrAdd aid
230           checkAdd f = do
231             addr' <- liftHandlerT $ lookupPostParam "addr"
232             zip' <- liftHandlerT $ lookupPostParam "zip"
233             case (addr', zip') of
234               (Just addr, Just zipcode) -> f (addr,zipcode)
```

```
235                 _ -> returnR $ RtEaddrFail "param:add:␣less␣and␣less"
236             spl = do
237               opt <- liftHandlerT $ lookupHeader "OPT"
238               case opt of
239                 Just "ADD" -> checkAdd addItem
240                 Just "DEL" -> checkChn changeItem
241                 Just "CHANGE" -> checkDel delItem
242                 _ -> returnR $ RtEaddrFail "header:opt:␣less␣and␣less"
```

获取收货地址

```
243       postGeteaddR :: Handler TypedContent
244       postGeteaddR = spl
245         where
246           getByUid uid = do
247             rt <- liftHandlerT $ runDB $ selectList [AddrUid ==. uid] []
248             returnR $ RtGEadd $ map fromEntity rt
249           getByAid aid = do
250             uid <- getUid
251             rt <- liftHandlerT $ runDB $ selectList [AddrAid ==. aid,AddrUid ==. uid] []
252             returnR $ RtGEadd $ map fromEntity rt
253           spl = do
254             uid' <- liftHandlerT $ lookupPostParam "uid"
255             aid' <- liftHandlerT $ lookupPostParam "aid"
256             case (uid', aid') of
257               (Just uid, _) -> getByUid uid
258               (Nothing, Just aid) -> getByAid aid
259               _ -> 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  module Main
2      ( main
3      ) where
```

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

```
16         main :: IO ()
17         main = do
18 #ifndef WithoutUTF8
19             E.setLocaleEncoding E.utf8
20             hSetEncoding stdout utf8
21 #endif
22             Pash key t at <- cmdArgs pash
23             now' <- getCurrentTime
24             let now = addUTCTime (fromIntegral at) now'
25             pash <- getPash t key now
26             a' <- getContents
27             let a = concat.lines $ a'
28             case t of
```

```
29            100 -> putStr $ a ++ "␣-d␣\"pash="++pash++"\""
30            _ -> putStr $ a ++ "␣-d␣\"pash="++pash++"\"␣-H␣\"TIME-STAMP:"++
                  show now++"\""
31          return ()
32          where
33            getPash typ key now = case typ of
34              100 -> return $ showDigest $ sha256 $ B.fromStrictBS $ T.encodeUtf8 $ T.pack
                    key
35              x -> do
36                let k = T.pack $ showDigest $ sha256 $ B.fromStrictBS $ T.encodeUtf8 $ T.
                      pack key
37                let time = T.encodeUtf8.T.pack.show $ now
38                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。

```
39      data Pash = Pash {pKey :: String,pType :: Int,aTime :: Int}
40        deriving (Show,Data,Typeable)
41      pash = Pash
42        { pKey = def &= argPos 1 &= typ "PASSWORD"
43        , pType = def &= argPos 2 &= typ "IDENTIFY-TYPE"
44        , aTime = 0 &= typ "UTCDiffTime" &= help "时间矫正"
45        } &= summary ( "dindo-common:-"
46                       ++ $(dindo_common_version_quasi)
47                       ++ ";␣dindo-tools-"
48                       ++ showVersion version
49                       )
```

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 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

[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。

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。dindo-pash 直接输出的是对应着 cURL 的参数名称。同时输入的内容应该是 cURL 对应的其他内容。

```
$ echo 'curl --some-flags url://host' | dindo-pash password
```

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