Dingo

Dindo Document

DingoLab

June,2016

Dingo Dindo Document

李约瀚
qinka@live.com
14130140331

June 2nd 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/Dingo-Lab/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

```haskell
module Dindo.Database where
```

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

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

```
import Import
import Data.Text
import Data.ByteString
import Paths_dindo_database
import Data.Version
```

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

```
share [mkPersist sqlSettings] [persistLowerCase|
Account json sql=table_account
  Id sql=
  uid Text sql=key_uid sqltype=varchar(64)
  pash Text sql=key_pash sqltype=varcher(64)
  tel Int sql=key_tel
  name Text sql=key_name sqltype=varchar(64)
  Primary uid
  deriving Show Eq
Usr json sql=table_usr
  Id sql=
  uid Text sql=key_uid sqltype=varchar(64)
  email Text sql=key_email
  rname Text sql=key_rname sqltype=varchar(64)
  prcid Text sql=key_prcid sqltype=varchar(18)
  addr Text sql=key_addr
  status Text sql=key_status sqltype=varchar(1)
  Primary uid
  Foreign Account fkuid uid
  deriving Show Eq
Addr json sql=table_addr
  Id sql=
  aid Text sql=key_aid sqltype=varchar(64)
```

```
    uid  Text  sql=key_uid sqltype=varchar(64)
    zip  Text  sql=key_zip sqltype=varchar(64)
    addr  Text  sql=key_addr
    Primary aid
    Foreign  Account fkaddruid  uid
    deriving  Show  Eq
  Apic  sql=table_apic
    Id  sql=
    pid  Text  sql=key_pic_id sqltype=varchar(64)
    uid  Text  sql=key_uid sqltype=varchar(64)
    bpic  ByteString  sql=binary_pic
    typ  Int  Maybe sql=key_status default=0
    Primary pid
    Foreign  Account fkuidb  uid
    deriving  Show  Eq
  Task json  sql=table_task
    Id  sql=
    tid  Text  sql=key_tid sqltype=varchar(64)
    ca  Text  Maybe sql=key_ca sqltype=varchar(64)
    cb  Text  Maybe sql=key_cb sqltype=varchat(64)
    Primary  tid
    Foreign  Account fkca  ca
    Foreign  Account fkvcb cb
    deriving  Show  Eq
  Taskinfo  json  sql=table_task_info
    Id  sql=
    tid  Text  sql=key_tid sqltype=varchar(64)
    ew  Double sql=key_ew
    ns  Double sql=key_ns
    r  Double sql=key_r
    wei  Double sql=key_wei
    size  [Double] sql=key_size
    note  Text  Maybe sql=key_note
    cost  Int  sql=key_cost
    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
    deriving Show Eq
  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
    deriving Show Eq
   |]
```

```
  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
      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

```
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

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

```
      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

```
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

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

```haskell
  import Dindo.Common.Rable as X
  import Text.Hamlet.XML as X
  import Text.XML as X
```

10.2.7 src/Dindo/Import/Text.lhs

导入 text 包中相关的模块

```haskell
module Dindo.Import.Text
    ( module X
    , showT
    , readT
    ) where

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

```haskell
  showT :: Show a => a -> Text
  showT = pack.show
```

```haskell
  readT :: Read a => Text -> a
  readT = read.unpack
```

10.2.8 src/Dindo/Import/TH.lhs

导入与 TemplateHaskell 与 QuasiQuote 有关的模块

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

  import Language.Haskell.TH as X
  import Language.Haskell.TH.Syntax as X
```

10.2.9 src/Dindo/Import/Yaml.lhs

导入与 Yaml 有关模块

```
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

```
module Dindo.Import.Yesod
    ( module X
    , mkYesodData
    , mkShomeR
    ) where
```

```
      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
      import qualified Data.Text.Encoding as TE
      import Data.Aeson
      import Data.ByteString.Lazy as BL hiding(unpack)
```

```
      mkYesodData a b = Yesod.mkYesodData a b'
        where
          b' = b ++ [parseRoutes|/ ShomeR GET|]
      homeR :: Yesod site
            => Text
            -> HandlerT site IO Text
      homeR info = do
        addD' <- lookupGetParam "add"
        let addD = fromRational $ toRational $ fromMaybe 0 $ fmap (read.unpack) addD'
        now <- fmap (show.addUTCTime addD) $ liftIO getCurrentTime
        return $ TE.decodeUtf8 $ toStrict $ encode $ object
          [ "server-time" .= now
          , "server-info" .= info
          ]
      mkShomeR :: Text -> Q [Dec]
      mkShomeR info = [d|
        getShomeR :: Yesod site => HandlerT site IO Text
        getShomeR = homeR info
         |]
```

10.2.11 src/Dindo/Common.lhs

提供版本号的部分

```
module Dindo.Common
    ( dindo_common_version
    , dindo_common_version_quasi
    ) where

      import Data.Version
      import Paths_dindo_common
      import Language.Haskell.TH
      import Language.Haskell.TH.Syntax

      dindo_common_version = version
      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

```
module Dindo.Common.Auth
      ( runPash
      , tokenAuth
      , pskAuth
      , noAuth
      , fromEntity
      , pickF
      , pickU
      , getUid
      ) where
```

```
        import Yesod
        import Database.Persist
        import Database.Persist.Sql
        import Dindo.Database
        import Data.Time
        import Data.Text.Encoding
        import Data.Maybe
        import qualified Data.ByteString as B
        import qualified Data.ByteString.Lazy as B hiding (concat,ByteString)
        import Data.Text (unpack,pack,Text)
        import Data.Digest.Pure.SHA
```

```
        pickU [] = []
        pickU ((y,Just x):oth) = (y =. x):pickU oth
        pickU ((_,Nothing):oth) = pickU oth
        pickF [] = []
        pickF ((y,Just x):oth) = (y ==. x):pickF oth
        pickF ((_,Nothing):oth) = pickF oth
        getUid :: ( Yesod site
                  , YesodPersist site
```

```
                    , YesodPersistBackend site ~ SqlBackend
                    )
                => HandlerT site IO Text
getUid = do
  tt' <- lookupHeader "TMP-TOKEN"
  let Just tt = fmap decodeUtf8 tt'
  rt':_ <- liftHandlerT $ runDB $ selectList [TmpTokenTt ==. tt] []
  let rt = fromEntity rt'
  return $ tmpTokenTt rt
```

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

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

tokenAuth :: ( Yesod site
             , YesodPersist site
             , YesodPersistBackend site ~ SqlBackend
             )
          => HandlerT site IO AuthResult
tokenAuth = do
  token' <- lookupHeader "TMP-TOKEN"
  case token' of
    Nothing -> return $ Unauthorized "Who␣are␣you!"
    Just token -> do
```

```haskell
          rt' <- liftHandlerT $ runDB $ selectList [TmpTokenTt ==. decodeUtf8 token][
              Desc TmpTokenTime]
          case rt' of
            rt:_ -> do
              now <- liftIO getCurrentTime
              let time = tmpTokenTime.fromEntity $ rt
              if diffUTCTime now time >= 0
                then return $ Unauthorized "Who are you!"
                else return Authorized
            _ -> return $ Unauthorized "Who are you!"

  pskAuth :: ( Yesod site
             , YesodPersist site
             , YesodPersistBackend site ~ SqlBackend
             )
          => HandlerT site IO AuthResult
  pskAuth = checkTime $ \time -> do
    pash  <- getPash
    uid'  <- lookupPostParam "uid"
    name' <- lookupPostParam "name"
    tel''  <- lookupPostParam "tel"
    let tel' = fmap (read.unpack) tel'' :: Maybe Int
    case (uid',name',tel') of
      (Nothing,Nothing,Nothing) -> return $ Unauthorized "Who are you!"
      (Just uid,name,tel) -> do
        rt <- liftHandlerT $ runDB $ selectList (
          [AccountUid ==. uid] ++ pickF [(AccountName,name)]++pickF [(AccountTel,
              tel)]) []
        checkPash pash rt (runPash 0 time)
      (Nothing,Just name,tel) -> do
        rt <- liftHandlerT $ runDB $ selectList (
          [AccountName ==. name] ++ pickF [(AccountTel,tel)]) []
        checkPash pash rt (runPash 1 time)
      (Nothing,Nothing,Just tel) -> do
```

```
              rt <- liftHandlerT $ runDB $ selectList
                [AccountTel ==. tel] []
              checkPash pash rt (runPash 2 time)
            _ -> return $ Unauthorized "Who␣are␣you!"
        where
          getPash = do
            pash' <- lookupPostParam "pash"
            return $ fromMaybe "" pash'
          checkPash pash rt f = do
            case rt of
              item:_ -> do
                let usrPash = f.accountPash.fromEntity $ item
                if usrPash == pash
                  then return Authorized
                  else return $ Unauthorized "Who␣are␣you!"
              _ -> return $ Unauthorized "Who␣are␣you!"
          checkTime f = do
            time' <- liftHandlerT $ lookupHeader "TIME-STAMP"
            now <- liftIO getCurrentTime
            case time' of
              Just time -> do
                let t = read.unpack.decodeUtf8 $ time
                let diff = diffUTCTime now t
                if diff <= 12 && diff >= (-12)
                  then f time
                  else return $ Unauthorized "I␣bought␣a␣watch␣last␣year!"
              _ -> return $ Unauthorized "I␣bought␣a␣watch␣last␣year!"

      fromEntity :: Entity a -> a
      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

```
module Dindo.Common.Rable
    ( RtType(..)
    , RtWhere(..)
    , Varable (..)
    , defToContent
    , defToContentXml
    , defToContentYaml
    , defToContentJson
    , Rable (..)
    , defReturnR
    , RtStatus (..)
    , statusHead
    , RtCommon(..)
    ) 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 -> Bl.ByteString
  toContents = defToContent
defToContent :: Varable a => RtType -> a -> Bl.ByteString
defToContent RtJson = defToContentJson
defToContent RtYaml = defToContentYaml
defToContent RtXml = defToContentXml
defToContentJson :: Varable a => a -> Bl.ByteString
defToContentJson = toStrict. A.encode . toValue
defToContentYaml :: Varable a => a -> Bl.ByteString
defToContentYaml = Y.encode . toValue
defToContentXml :: Varable a => a -> Bl.ByteString
defToContentXml x = toStrict $ renderLBS def $ Document p root []
  where
    root = Element "data" (fromList []) $ toNodes x
    p = Prologue [] Nothing []
```

```
class Varable a => Rable a where
  toWhere :: a -> RtWhere
  toStatus :: a -> RtStatus
  returnR :: MonadHandler m => a -> m TypedContent
  returnR = defReturnR
defReturnR :: ( MonadHandler m
              , Rable a
              )
           => a -> m TypedContent
defReturnR x = do
  addHeader "Status" $ status x
  if toWhere x == RtBody
    then addHeader "Context-Where" "Body"
    else addHeader "Context-Where" $ (\(RtOther a)-> a) $ toWhere x
```

```haskell
      addContent
      where
        status  = statusHead.toStatus
        addContent = case toWhere x of
          RtBody -> selectRep $ do
            provideRepType "application/json" $ return $ decodeUtf8 $ toContents RtJson
                x
            provideRepType "application/yaml" $ return $ decodeUtf8 $ toContents RtYaml
                x
            provideRepType "application/xml"  $ return $ decodeUtf8 $ toContents RtXml
                x
          RtOther y -> do
            addHeader y $ pack $ show x
            selectRep $  provideRep $ return ("" :: Text)
```

```haskell
    data RtStatus = RtSucc | RtFail
    statusHead :: RtStatus -> Text
    statusHead RtSucc = "Success"
    statusHead RtFail  = "Failed"
```

将 Yesod 中的 ErrorResponse 实现 Varable 与 Rable

```haskell
    instance Varable ErrorResponse where
      toValue NotFound = A.String "NotFound"
      toValue ( InternalError  x) = object [" internal -error" .= x]
      toValue (PermissionDenied x) = object ["permission-denied" .= x]
      toValue ( InvalidArgs  x) = object [" invalid -args" .= x]
      toValue NotAuthenticated = A.String "NotAuthenticated"
      toValue (BadMethod x) = object ["bad-method" .= show x]
      toNodes NotFound = [xml|NotFound|]
      toNodes ( InternalError  x) = [xml|<InternalError>#{x}|]
      toNodes (PermissionDenied x) = [xml|<PermissionDenied>:#{x}|]
      toNodes (InvalidArgs  x) = [xml|<InvalidArgs>#{x'}|]
        where
          x' = T.unlines x
    toNodes NotAuthenticated = [xml|NotAuthenticated|]
    toNodes (BadMethod x) = [xml|<BadMethod>#{pack $ show x}|]

  instance Rable ErrorResponse where
    toWhere _ = RtBody
    toStatus _ = RtFail
```

通用成功与失败标志

```
  data RtCommon = RtCommonSucc
                | RtCommonSuccT Text
                | RtCommonFail Text
    deriving (Eq,Show)
  instance Varable RtCommon where
    toValue RtCommonSucc = Null
    toValue (RtCommonSuccT t) = object ["tmp-token" .= t]
    toValue (RtCommonFail x) = String x
    toNodes RtCommonSucc = [xml|null|]
    toNodes (RtCommonSuccT x) = [xml|<tmp-token>#{x}|]
    toNodes (RtCommonFail x) = [xml|<error>#{x}|]
  instance Rable RtCommon where
    toWhere RtCommonSucc = RtBody
    toWhere (RtCommonFail _) = RtBody
    toWhere (RtCommonSuccT _) = RtBody
    toStatus RtCommonSucc = RtSucc
    toStatus (RtCommonSuccT _) = RtSucc
    toStatus (RtCommonFail _) = RtFail
```

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

提供模块配置的部分

```
module Dindo.Common.Yesod.Config
    ( SvrConfig (..)
    , DbConfig(..)
    , ScError (..)
```

```
  , scError
  , dbConfig2Str
  ) where
```

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

```
  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。

```
instance ToJSON SvrConfig where
  toJSON SvrConfig{..} = object
    [ "port" .= svrPort
    , "datebase-bconfig" .= svrDb
    ]
instance ToJSON DbConfig where
  toJSON DbConfig{..} = object
    [ "addr" .= dbAddr
    , "port" .= dbPort
    , "user" .= dbUser
    , "name" .= dbName
    , "con-limit" .= dbConThd
    , "password" .= dbPsk
    ]
instance FromJSON SvrConfig where
  parseJSON (Object v) = SvrConfig
    <$> v .: "port"
    <*> v .: "database-config"
  parseJSON _ = throw $ ScError "Invailed"
instance FromJSON DbConfig where
  parseJSON (Object v) =DbConfig
    <$> v .: "addr"
    <*> v .: "port"
    <*> v .: "user"
    <*> v .: "name"
    <*> v .: "password"
    <*> v .: "con-limit"
  parseJSON _ = throw $ ScError "Invailed"
```

将数据库配置转化成链接字符串。

```
dbConfig2Str :: DbConfig -> (B.ByteString,Int)
dbConfig2Str DbConfig{..} = (str,dbConThd)
  where
    str = toStrict $
      fromString $    "host=\'" ++ dbAddr
                   ++ "\' port=\'" ++ dbPort
                   ++ "\' user=\'" ++ dbUser
                   ++ "\' password=\'" ++ dbPsk
                   ++ "\' dbname=\'" ++ dbName
                   ++ "\'"
```

设置读写异常

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

JSON 与 Yaml 例程。

```
{ "port":3000
, "database-config":
  { "addr":"127.0.0.1"
  , "port":"5432"
  , "user":"postgres"
  , "name":"postgres"
  , "password":"postgres"
  , "con-limit":10
  }
}
```

```
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

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

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

```
      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）

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

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

10.2.17 src/Dindo/MicroFramework/Destory.lhs

提供了微服务架构中销毁的部分

```
module Dindo.MicroFramework.Destory
    ( Destorible (..)
```

```haskell
    , regDestory
    ) where
```

```haskell
    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”

```haskell
    class ( Yesod a
          ) => Destorible a where
      destoryAPI :: a -> String
      destoryHead :: a -> String
```

```haskell
    regDestory :: Destorible a => a -> IO Bool
    regDestory x = do
      -- 注册 销毁接口
      -- 实际上应该是 http 请求，此处仅输出内容
      putStrLn "销毁接口␣注册"
      print $ destoryAPI x
      print $ destoryHead x
      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

```haskell
module Dindo.MicroFramework.Register
    ( Registrable (..)
    , Heartbeatable (..)
    , register
    ) 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

```
register :: Registrable a => a -> IO Bool
register x = do
  -- 注册 服务
  -- 实际上应该是 http 请求，此处仅输出内容
  putStrLn "注册服务的端口"
  print $ regSvrPort x
  putStrLn "注册服务的地址"
  print $ regSvrAddr x
  putStrLn "被注册的实例的地址"
  print $ regPort x
  putStrLn "被注册的实例的端口"
  print $ regPort x
  regAPI' $ regDestory' $ do
    forkIO $ heartbeat x
    return True
  where
    regAPI' a = do
      ra <- regAPI x
      if ra then a else return False
    regDestory' a = do
      rd <- regDestory x
      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
      import System.Exit
      import Control.Concurrent
      import System.Signal
```

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
        tid  <− myThreadId
         installHandler  sigINT $ \ sig  −> do
          if  sig  == sigINT
            then do
              putStrLn "going␣to␣turn␣down"
               killThread  tid
              exitSuccess
            else  putStrLn $ "catch" ++ show sig
        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 ll = form l in
            case map toLower ll of
              "auto" -> decA
              "json" -> evaluate.decJ
              "yaml" -> evaluate.decY
              _ -> 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

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

      import Dindo.UM as X -- need change
      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

```
module Dindo.UM
    ( module X
    , dindo_usrmanage_version
    , dindo_usrmanage_version_quasi
    ) where
      import Dindo.UM.Foundation as X
      import Dindo.UM.Handler as X
      import Dindo.Import.Yesod
      import Dindo.Import.TH
      import Data.Version
      import Paths_dindo_usrmanage

      dindo_usrmanage_version = version
      dindo_usrmanage_version_quasi = stringE $ showVersion version
      mkYesodDispatch "UM" resourcesUM
```

10.4.3 src/Dindo/UM/Data.lhs

定义返回数据的部分

```
module Dindo.UM.Data
    ( RtRegist (..)
    , RtIdy (..)
    , RtIdfed (..)
    , RtUImg(..)
    , RtUInfo (..)
    , RtChPsk(..)
```

```
  , RtEaddr(..)
  , RtGEadd(..)
  ) where

  import Dindo.Import.Rable
  import Dindo.Import.Aeson as A
  import Dindo.Import.Yaml as Y
  import Dindo.Import.Text as T
  import Dindo.Import.ByteString as B
  import Dindo.Import.Yesod
  import Dindo.Import.Database
```

用户注册返回数据

```
  data RtRegist = RtRegist
      { uid  ::  Text
      }
    | RtRegistFail
      { regReason :: Text
      }
    deriving (Eq)
  instance Show RtRegist where
    show (RtRegist x) = T.unpack x
    show (RtRegistFail x) = T.unpack x
  instance Varable RtRegist where
    toValue (RtRegist x) = object ["uid" .= x]
    toValue ( RtRegistFail x) = object ["error" .= x]
    toNodes (RtRegist x) = [xml|<uid>#{x}|]
    toNodes ( RtRegistFail x) =[xml|<error>#{x}|]
  instance Rable RtRegist where
    toWhere (RtRegist _) = RtBody
    toWhere (RtRegistFail _) = RtBody
    toStatus (RtRegist _) = RtSucc
    toStatus ( RtRegistFail _) = RtFail
```

用户认证信息的返回数据

```
data RtIdy = RtIdy
  | RtIdyFail
    { idyReason :: Text
    }
  deriving (Eq)
instance Show RtIdy where
  show (RtIdyFail x) = T.unpack x
instance Varable RtIdy where
  toValue RtIdy = Null
  toValue ( RtIdyFail x) = object ["error" .= x]
  toNodes RtIdy  = [xml|null |]
  toNodes (RtIdyFail x) = [xml|<error>#{x}|]
instance Rable RtIdy where
  toWhere (RtIdyFail _) = RtBody
  toWhere RtIdy = RtBody
  toStatus RtIdy = RtSucc
  toStatus ( RtIdyFail _) = RtFail
```

用户查询认证状态信息

```
data RtIdfed = RtIdfedPass | RtIdfedNo
  deriving (Eq,Show)
instance Varable RtIdfed where
  toValue RtIdfedPass = object ["status" .= ("pass" :: Text)]
  toValue RtIdfedNo   = object ["status" .= ("no" :: Text)]
  toNodes RtIdfedPass = [xml|<status>pass|]
  toNodes RtIdfedNo   = [xml|<status>no|]
instance Rable RtIdfed where
  toWhere RtIdfedPass = RtBody
  toWhere RtIdfedNo = RtBody
  toStatus RtIdfedPass = RtSucc
  toStatus RtIdfedNo = RtSucc
```

用户信息查询返回结果

```
data RtUInfo = RtUInfo
    { rtuiUid  :: Text
    , rtuiName :: Text
    , rtuiTel  :: Text
    , rtuiEmail :: Text
    }
  | RtUInfoNSU
instance Show RtUInfo where
  show RtUInfoNSU = "no␣such␣a␣user"
instance Varable RtUInfo where
  toValue RtUInfo{..} = object
    [ "uid" .= rtuiUid
    , "name" .= rtuiName
    , "tel" .= rtuiTel
    , "email" .= rtuiEmail
    ]
  toNodes RtUInfo{..} = [xml|
  <uid> #{rtuiUid}
  <name> #{rtuiName}
  <tel> #{rtuiTel}
  <email> #{rtuiEmail}
  |]
instance Rable RtUInfo where
  toWhere RtUInfo{..} = RtBody
  toWhere RtUInfoNSU = RtOther "CONTEXT"
  toStatus RtUInfo{..} = RtSucc
  toStatus RtUInfoNSU = RtFail
```

获取用户头像返回内容

```
data RtUImg = RtUImg ByteString
            | RtUImgFail
  deriving (Eq)
instance Show RtUImg
instance Varable RtUImg
instance Rable RtUImg where
  returnR (RtUImg img) =
    selectRep $ provideRepType "image/png" $ return img
  returnR RtUImgFail = do
    addHeader "CONTEXT-WHERE" "CONTEXT"
    addHeader "CONTEXT" "Failed␣on␣get␣image"
    selectRep $ provideRep $ return ("" :: Text)
```

更改密码的返回值

```
data RtChPsk = RtChPsk
             | RtChPskFail Text
  deriving (Eq)
instance Show RtChPsk where
  show (RtChPskFail x) = T.unpack x
instance Varable RtChPsk where
  toValue RtChPsk = Null
  toValue (RtChPskFail x) = object ["error" .= x]
  toNodes RtChPsk = [xml|null|]
  toNodes (RtChPskFail x) = [xml|<error>#{x}|]
instance Rable RtChPsk where
  toWhere RtChPsk = RtBody
  toWhere (RtChPskFail _) = RtBody
  toStatus RtChPsk = RtSucc
  toStatus (RtChPskFail _) = RtFail
```

收货地址的增删的返回值

```
data RtEaddr = RtEaddrAdd Text
             | RtEaddrChn
             | RtEaddrDel
             | RtEaddrFail Text
  deriving (Eq,Show)
instance Varable RtEaddr where
  toValue (RtEaddrAdd x) = object ["aid" .= x]
```

```
    toValue RtEaddrChn = Null
    toValue RtEaddrDel = Null
    toValue (RtEaddrFail x) = object ["error" .= x]
    toNodes (RtEaddrAdd x) = [xml|<aid>#{x}|]
    toNodes RtEaddrChn = [xml|null|]
    toNodes RtEaddrDel = [xml|null|]
    toNodes (RtEaddrFail x) = [xml|<error>#{x}|]
  instance Rable RtEaddr where
    toWhere (RtEaddrAdd _) = RtBody
    toWhere RtEaddrChn = RtBody
    toWhere RtEaddrDel = RtBody
    toWhere (RtEaddrFail _) = RtBody
    toStatus (RtEaddrAdd _) = RtSucc
    toStatus RtEaddrChn = RtSucc
    toStatus RtEaddrDel = RtSucc
    toStatus (RtEaddrFail _) = RtFail
```

获取地址

```
  data RtGEadd = RtGEadd [Addr]
               | RtGEaddFail Text
    deriving (Eq,Show)
  instance Varable RtGEadd where
    toValue (RtGEadd x) = toJSON x
    toValue (RtGEaddFail x) = object ["error" .= x]
    toNodes (RtGEadd xs) = [xml|
       $forall x <- xs
         <aid>#{addrAid x}
         <addr>#{addrAddr x}
         <zip>#{addrZip x}
       |]
    toNodes (RtGEaddFail x) = [xml|<error>#{x}|]
  instance Rable RtGEadd where
    toWhere (RtGEadd _ ) = RtBody
    toWhere (RtGEaddFail _) = RtBody
    toStatus (RtGEadd _) = RtSucc
    toStatus (RtGEaddFail _) =RtFail
```

10.4.4 src/Dindo/UM/Foundation.lhs

基础的部分

```
module Dindo.UM.Foundation where

    import Dindo.Common
    import Dindo.Import
    import Dindo.Import.Yesod
    import Dindo.Import.Database
    import Paths_dindo_usrmanage
    import Dindo.Import.Text as T
    import Data.Version
```

定义基本类型路由表

```
    data UM = UM
      { connPool :: ConnectionPool
      , config   :: SvrConfig
      }
    mkYesodData "UM" [parseRoutes|
      /regist  RegistR POST
      /identify  IdentifyR  POST
      /identified   Identified  POST
      /login  LoginR POST
      /logout  LogoutR POST
      /usrinfo  UsrinfoR  POST
      /usrhimg UsrhimgR POST
      /usrinfochange  UsrinfochangeR POST
      /changpash ChangpashR POST
      /upeaddr UpeaddrR POST
      /geteaddr GeteaddR POST
     |]
```

实现 Yesod 类型类

```
instance Yesod UM where
  errorHandler  = returnR
  isAuthorized  ShomeR _ = return Authorized
  isAuthorized  RegistR _ = noAuth
  isAuthorized  LoginR _ = pskAuth
  isAuthorized  _ _ = tokenAuth
instance YesodPersist  UM where
  type YesodPersistBackend UM = SqlBackend
  runDB a = getYesod >>= (runSqlPool a.connPool)
mkShomeR $ pack $ "dindo-um-" ++ showVersion version ++ ";␣dindo-common-"
    ++ $(dindo_common_version_quasi)
```

微服务架构

```
instance APIble UM where
  apis _ = []
instance  Destorible  UM where
  destoryHead _ = ""
  destoryAPI _ = ""
instance Heartbeatable  UM where
  heartbeat  _ = return ()
instance  Registrable  UM where
  regAddr _ = ""
  regPort  = svrPort  .  config
  regSvrPort  _ = 80
  regSvrAddr _ = ""
instance Dindoble UM where
  fromPool = UM
```

10.4.5 src/Dindo/UM/Handler.lhs

处理函数的部分

```haskell
module Dindo.UM.Handler
      ( postRegistR
      , postUsrinfoR
      , postLogoutR
      , postLoginR
      ,  postIdentified
      ,  postIdentifyR
      , postUsrinfochangeR
      , postChangpashR
      , postUsrhimgR
      , postUpeaddrR
      , postGeteaddR
      ) where
```

```haskell
         import Dindo.Import
         import Dindo.Import.Rable
         import Dindo.Import.Yesod
         import Dindo.Import.Database
         import Dindo.UM.Foundation
         import Dindo.UM.Data
         import Dindo.Import.Digest
         import Dindo.Import.ByteString as B hiding(unpack,pack,splitAt,take,map,null)
         import Dindo.Import.Text as T hiding(splitAt,take,map,null)
         import Dindo.Common.Auth(fromEntity,pickU,pickF)
         import Control.Exception(try,SomeException)
         import Control.Monad
```

注册的 API

```haskell
         postRegistR :: Handler TypedContent
         postRegistR =
            getParam insertAItem
            where
               getParam f = do
                  name' <- lookupPostParam "name"
```

```
        pash'  <- lookupPostParam "pash"
        tel '  <- lookupPostParam "tel"
        case (name',pash', tel ') of
          (Just name,Just pash,Just tel) -> do
            x <- liftIO getCurrentTime
            let (time,p) = splitAt 10 $ show x
            let to = showDigest $ sha1 $ fromStrictBS $ encodeUtf8 $ T.concat [pash,
                name]
            let uid = 'U':time ++ to
            f (pack uid,name,pash,read (unpack tel))
          _ -> returnR $ RtRegistFail "param:␣less␣and␣less"
      insertAItem (uid,name,pash,tel) = do
        rt <- liftHandlerT $ tryRunDB $
          insert $ Account uid pash tel name
        returnR $ case rt of
          Left e -> RtRegistFail $ pack $ show e
          Right _ -> RtRegist uid
```

用户认证的 API

```
  postIdentifyR :: Handler TypedContent
  postIdentifyR =
    checkParam $ addItem $ checkPic addPic
    where
      checkParam f = do
        email'  <- lookupPostParam "email"
        rname' <- lookupPostParam "rname"
        prcid '  <- lookupPostParam "prcid"
        addr'  <- lookupPostParam "addr"
        case (email ', rname', prcid ', addr') of
          (Just email, Just rname,Just prcid , Just addr) ->
            f (email,rname,prcid,addr)
          _ -> returnR $ RtIdyFail "param:␣less␣and␣less"
      checkPic f ins = do
        pic' <- lookupFile "pic"
```

```
            case pic' of
              Just pic -> do
                rt <- sourceToList $ fileSource pic
                let bpic = B.concat rt
                f (bpic, ins)
              _ -> returnR $ RtIdyFail "param:␣picture␣needed"
        addItem f (email,rname,prcid,addr) =
          f $ \uid -> Usr uid email rname prcid addr "N"
        addPic (pic,usr) = do
          uid <- getUid
          now <- liftIO getCurrentTime
          let str = show now
          let (time,p) = splitAt 10 $ str
          let to = showDigest $ sha1 $ fromStrictBS $ encodeUtf8 $ T.concat [uid, pack
              str]
          let pid = pack $ 'A':time ++ to
          rt <- liftHandlerT $ tryRunDB $ do
            insert $ usr uid
            insert $ Apic pid uid pic $ Just 0
          returnR $ case rt of
            Left e -> RtIdyFail $ pack $ show e
            Right _ -> RtIdy
```

认证状态查询

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

用户登录

```
postLoginR :: Handler TypedContent
postLoginR = do
  uid'  <- lookupPostParam "uid"
  name' <- lookupPostParam "name"
  tel'  <- lookupPostParam "tel"
  case (uid', name', tel') of
    (uid,name,tel) -> do
      pash <- getPash
      rt' <- liftHandlerT $ runDB $ selectList (pickF
        [ (AccountUid, uid)
        , (AccountName, name)
        ] ++ pickF
        [ (AccountTel,fmap (read.unpack) tel)
        ]) []
      case rt' of
        (Entity _ item):_ -> do
          let uid  = accountUid item
          now <- liftIO getCurrentTime
          let lim  = addUTCTime 3600 now
          let time = show lim
          let to  = showDigest $ sha512 $ fromStrictBS $ encodeUtf8 $ T.concat [uid,
              pash,pack time]
          let tt  = pack $ take 22 time ++ to
          liftHandlerT $ runDB $ insert $ TmpToken tt lim uid
          returnR $ RtCommonSuccT tt
  where
    getPash = do
      pash' <- lookupPostParam "pash"
      return $ fromMaybe "" pash'
```

用户登出

```
postLogoutR :: Handler TypedContent
postLogoutR = do
```

```
      Just token <- lookupHeader "TMP-TOKEN"
      Just uid <- lookupHeader "USR-ID"
      rt <- liftHandlerT $ tryRunDB $ deleteWhere [TmpTokenTt ==. decodeUtf8 token,
          TmpTokenUid ==. decodeUtf8 uid]
      returnR $ case rt of
        Left e -> RtCommonFail $ pack $ show e
        Right _ -> RtCommonSucc
```

查询用户信息

```
postUsrinfoR :: Handler TypedContent
postUsrinfoR = do
  tuid <- getUid
  uid' <- lookupPostParam "uid"
  let uid = fromMaybe tuid uid'
  rt' <- liftHandlerT $ runDB $ selectList [UsrUid ==. uid] []
  case rt' of
    Entity _ rt:_ -> do
      let email = usrEmail rt
      Entity _ item:_ <- liftHandlerT $ runDB $ selectList [AccountUid ==. uid] []
      returnR $ RtUInfo uid (accountName item) (pack $ show $ accountTel item)
          email
    _ -> returnR RtUInfoNSU
```

获得用户头像

```
postUsrhimgR :: Handler TypedContent
postUsrhimgR = do
  tuid <- getUid
  uid' <- lookupPostParam "uid"
  let uid = fromMaybe tuid uid'
  rt' <- liftHandlerT $ runDB $ selectList [ApicUid ==. uid] []
  case rt' of
    Entity _ rt:_ -> returnR $ RtUImg $ apicBpic rt
    _ -> returnR $ RtUImgFail
```

用户信息变更

```haskell
postUsrinfochangeR :: Handler TypedContent
postUsrinfochangeR = check update
  where
    updatePic uid pic' = case pic' of
      Nothing -> return ()
      Just pic -> do
        rt <- sourceToList $ fileSource pic
        let bpic = B.concat rt
        updateWhere [ApicUid ==. uid,ApicTyp ==. Just 0] [ApicBpic =. bpic]
    update (a,b,pic) = do
      uid <- getUid
      rt <- liftHandlerT $ tryRunDB $ do
        when (not $ null a) $
          updateWhere [AccountUid ==. uid] a
        when (not $ null b) $
          updateWhere [UsrUid ==. uid] b
        updatePic uid pic
      returnR $ case rt of
        Left e -> RtCommonFail $ pack $ show e
        Right _ -> RtCommonSucc
    check f = do
      name <- liftHandlerT $ lookupPostParam "name"
      tel   <- liftHandlerT $ lookupPostParam "tel"
      email <- liftHandlerT $ lookupPostParam "email"
      rname <- liftHandlerT $ lookupPostParam "rname"
      prcid <- liftHandlerT $ lookupPostParam "prcid"
      addr <- liftHandlerT $ lookupPostParam "addr"
      pic <- liftHandlerT $ lookupFile "pic"
      let a = pickU [(AccountName,name)]
      let a' = pickU [(AccountTel,fmap (read.T.unpack) tel)]
      let b = pickU [(UsrEmail,email),(UsrRname,rname),(UsrPrcid,prcid),(UsrAddr,
          addr)]
      f (a++a',b,pic)
```

修改密码

```
postChangpashR :: Handler TypedContent
postChangpashR = check changePash
  where
    changePash pash = do
      uid <− getUid
      rt <− liftHandlerT $ tryRunDB $ updateWhere [AccountUid ==. uid] [
          AccountPash =. pash]
      returnR $ case rt of
        Left e −> RtChPskFail $ pack $ show e
        Right _ −> RtChPsk
    check f = do
      pash' <− lookupPostParam "pash"
      case pash' of
        Nothing −> do
          returnR $ RtChPskFail "param:␣less␣and␣less"
        Just x −> f x
```

收获地址

```
postUpeaddrR :: Handler TypedContent
postUpeaddrR = spl
  where
    changeItem aid a = do
      rt <− liftHandlerT $ tryRunDB $ updateWhere [AddrAid ==. aid] a
      returnR $ case rt of
        Left e −> RtEaddrFail $ pack $ show e
        Right _ −> RtEaddrChn
    checkChn f = do
      addr <− liftHandlerT $ lookupPostParam "addr"
      zipcode <− liftHandlerT $ lookupPostParam "zip"
      aid' <− liftHandlerT $ lookupPostParam "aid"
      case aid' of
```

```
          Just aid  -> f aid $ pickU [(AddrAddr,addr),(AddrZip,zipcode)]
          Nothing -> returnR $ RtEaddrFail "param:change:␣less␣and␣less"
      delItem aid = do
        rt <- liftHandlerT $ tryRunDB $ deleteWhere [AddrAid ==. aid]
        returnR $ case rt of
          Left e -> RtEaddrFail $ pack $ show e
          Right _ -> RtEaddrDel
      checkDel f = do
        aid' <- liftHandlerT $ lookupPostParam "aid"
        case aid' of
          Just aid  -> f aid
          Nothing -> returnR $ RtEaddrFail "param:del:␣less␣and␣less"
      addItem (addr,zipcode) = do
        uid <- getUid
        now <- liftIO getCurrentTime
        let aid' = showDigest $ sha256 $ fromStrictBS $ encodeUtf8 addr
        let aid = pack $ "A"++show now++aid'
        rt <- liftHandlerT $ tryRunDB $ insert $ Addr aid uid zipcode addr
        returnR $ case rt of
          Left e -> RtEaddrFail $ pack $ show e
          Right _ -> RtEaddrAdd aid
      checkAdd f = do
        addr' <- liftHandlerT $ lookupPostParam "addr"
        zip' <- liftHandlerT $ lookupPostParam "zip"
        case (addr', zip') of
          (Just addr,Just zipcode) -> f (addr,zipcode)
          _ -> returnR $ RtEaddrFail "param:add:␣less␣and␣less"
      spl = do
        opt <- liftHandlerT $ lookupHeader "OPT"
        case opt of
          Just "ADD" -> checkAdd addItem
          Just "DEL" -> checkChn changeItem
          Just "CHANGE" -> checkDel delItem
          _ -> returnR $ RtEaddrFail "header:opt:␣less␣and␣less"
```

获取收货地址

```
postGeteaddR :: Handler TypedContent
postGeteaddR = spl
  where
    getByUid uid = do
      rt <- liftHandlerT $ runDB $ selectList [AddrUid ==. uid] []
      returnR $ RtGEadd $ map fromEntity rt
    getByAid aid = do
      uid <- getUid
      rt <- liftHandlerT $ runDB $ selectList [AddrAid ==. aid,AddrUid ==. uid] []
      returnR $ RtGEadd $ map fromEntity rt
    spl = do
      uid' <- liftHandlerT $ lookupPostParam "uid"
      aid' <- liftHandlerT $ lookupPostParam "aid"
      case (uid', aid') of
        (Just uid, _) -> getByUid uid
        (Nothing, Just aid) -> getByAid aid
        _ -> 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

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

```
import qualified GHC.IO.Encoding as E
import System.IO
import System.Environment
import Dindo.Import
import Dindo.Common.Auth
import Dindo.Import.Digest
import qualified Dindo.Import.Text as T
import qualified Dindo.Import.ByteString as B
import Dindo.Common(dindo_common_version_quasi)
import Data.Version
import System.Console.CmdArgs
import Paths_dindo_tools
```

```
main :: IO ()
main = do
#ifndef WithoutUTF8
  E.setLocaleEncoding E.utf8
  hSetEncoding stdout utf8
#endif
  Pash key t at <- cmdArgs pash
  now' <- getCurrentTime
  let now = addUTCTime (fromIntegral at) now'
  pash <- getPash t key now
  a' <- getContents
  let a = concat.lines $ a'
  case t of
    100 -> putStr $ a ++ " -d \"pash="++pash++"\""
    _ -> putStr $ a ++ " -d \"pash="++pash++"\" -H \"TIME-STAMP:"++
        show now++"\""
  return ()
  where
    getPash typ key now = case typ of
      100 -> return $ showDigest $ sha256 $ B.fromStrictBS $ T.encodeUtf8 $ T.pack
```

```
              key
            x -> do
              let k = T.pack $ showDigest $ sha256 $ B.fromStrictBS $ T.encodeUtf8 $ T.
                  pack key
              let time = T.encodeUtf8.T.pack.show $ now
              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。

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

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

[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。

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

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

```
#!/bin/sh
wget -O /usr/local/bin/weave \
https://github.com/zettio/weave/releases/download/latest_release/weave
chmod a+x /usr/local/bin/weave
dao pull weaveworks/weave:1.5.1
dao pull weaveworks/plugin:1.5.1
dao pull weaveworks/weaveexec:1.5.1
apt-get update
apt-get install bridge-utils
dao pull weaveworks/weavedb:latest
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