Infrastructure and Web Applications for Application Programming Interface (API) of Bio-database, Ocean Data Bank (ODB) 海洋資料庫生物資料庫應用程式介面之基礎建構與網際網路應用

Chiyu Weng 翁其羽

2022-03-15

Table of Contents

# 1 Introduction

## 1.1 An infrastructure for information services of ODB’s bio-database

How can we design an infrastructure that help people easier, more safer, more convenient for adding ideas to use the bio-database of Ocean Data Bank (ODB)? ODB is dedicated to help academia to curate and use the databases whose data is mainly collected through marine research vessels which are supported by Ministry of Science and Technology (MOST), Taiwan. The data of ODB is in privacy under the restriction of the data release policies of MOST, and ODB compiles raw data after reviewing someone’s application. This restriction limits open usage of the information services we deliver. To bridge ODB’s information services between open usage and databases is a prerequisite, whereas a well-defined application programming interface (API) is often the answer ([Box 1.1](#box1-1)). In addition, ODB’s information services are usually not for implementing new theory or algorithm on scientific researches, but for helping academia to use data, check data patterns, and create data modeling in their researches. These considerations form the basis for how to construct the infrastructure of information services to use the bio-database of ODB:

Box1.1

✓ API is a set of interfacing specifications for machine-to-machine communications. Here we focus on web API, i.e., data transport upon HTTP request/response structure. Most popular web API protocol is REST (Representational State Transfer), and increasingly used, GraphQL API. Commonly used data formats are JSON and XML.

✓ API can provide a secure way to access the bio-database (internal APIs, Fig. ) with authority management. On the other hand, API can also provide open-access methods for public information compiled from the bio-database (public APIs).

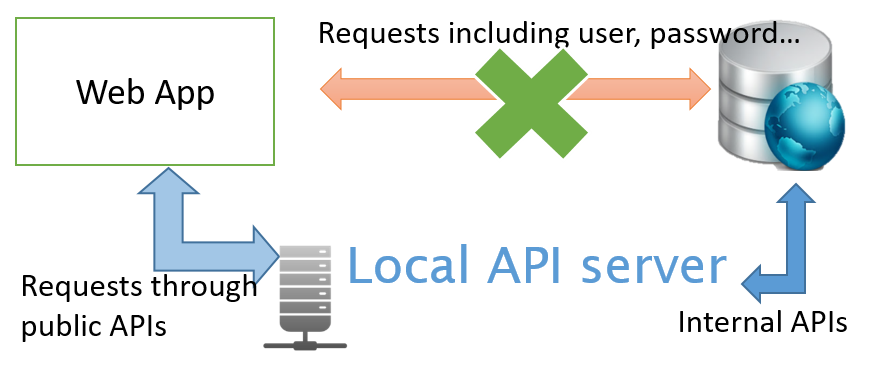


Fig. . Simplified schematic for web API of the bio-database

1. Don’t reinvent the wheel: ODB’s information services should be focused on problem solving. “Don’t reinvent the wheel” here means use existing, proven, and open source software (OSS) to get solutions. There are many OSS packages in data manipulation, data visualization, database-engine wrapper, Geographic Information System (GIS) processing, and statistics which are the major fields of data science that used for ecological applications. To be able to add these OSS packages upon the infrastructure in a more flexible way facilitate us to bring ideas into practice.
2. Faster development and deployment: It’s practical to design the infrastructure of ODB’s information services under the premise that most of the researchers are not programmers and no need to have background for database when using the data. But the researchers usually have to demonstrate their ideas, to check data patterns, and to modeling their data more efficiently. That’s why we need to have an infrastructure that is suitable for faster development and deployment.
3. Scalability, reproducibility, and consistency: To be easier to combine user data with ODB’s data so that the researchers can check data patterns in different scales. It’s also necessary to make the results be reproducible and consistent.

We present an infrastructure which may fulfill these requirements for constructing ODB’s information services upon (Fig. 1.2). We use R ()

# 2 **Re-structuring bio-database**

## 2.1 *Mutating a new bio-database on PostgreSQL*

The bio-database of ODB had been constructed since 2009 and completely set-up on Microsoft SQL Server since 2015. The data curation is basically done by a Microsoft C# program. These protocols of data curation was documented in 圖輯. For developing an open cross-platform framework for web APIs of bio-database, first, we mutated a bio-database on PostgreSQL, i.e., exported from SQL Server, and then re-imported to a new database by using PostgreSQL.

Box2.1

Why not just use original Microsoft SQL Server as backend database for developing APIs of bio-database?

✓ SQL Server is a commercial software with annual license fees.

✓ SQL Server is suited for the Microsoft Server based framework, but not for an open cross-platform framework.

Why choose PostgreSQL?

✓ PostgreSQL is a powerful open-source object-relational database systems.

✓ PostgreSQL can be well integrated in Geographic Information System (GIS) applications by its PostGIS extension. - It’s an important reason why we choose PostgreSQL. Most of APIs in ODB are used to develop GIS related applications.

As a backend database for API in production, we do not need some columns of variables originally in SQL Server (Fig. 2a) that used to remark some quality control (QC) issues. So when mutating the bio-database on PostgreSQL, we also did some simplifications for database schema (Fig. 2b). Roughly speaking, we left the “remark\_xxx” or “flag\_xxx” columns in SQL Server only, and in PostgreSQL, re-arranged the tables to make frequently-used variables more easily be accessed in table “cast\_site” (site information) and “taxa\_data” (abundance data of taxonomic groups). Another important modification is to construct classification of taxonomic groups in table ’taxon\_group” that can help us in analytical statistics among taxonomic groups when using bio-database data. The major parts of schema for bio-database on PostgreSQL include: taxon\_group:

1. cast\_site:
2. c00\_cast\_tbl:
3. t00\_taxa\_tbl:
4. taxa\_data:

# NOW on Ubuntu, this driver not support old SQL Server 2008-R2. It only works on Windows  
# On Ubuntu, the driver setting is in /etc/odbcinst.ini, e.g. driver={ODBC Driver 18 for SQL Server}  
library(RODBC) #R interface of ODBC driver to connect SQL Server  
library(sqldf) #runing SQL statements on R data frames  
library(data.table) #data.table is superb fast in R data manipulation  
library(magrittr) #pipe function  
ms\_conn <-   
 paste0('driver={SQL Server};server=', sqlServerHost,  
 ';database=', sqlServerDB,  
 ';uid=', sqlServerUser,  
 ';pwd=', sqlServerPass) %>%  
 odbcDriverConnect()  
  
# Query all taxonomic abundance in SQL server  
taxa\_data <- sqlQuery(ms\_conn, 'select \* from dbo.Taxa\_record') %>% setDT()  
  
# Query site  
cast\_site <- sqlQuery(ms\_conn, 'select \* from dbo.Cast') %>% setDT()  
  
close(ms\_conn) # Close database connection

Table  Taxaonomic abundance in bio-database

| taxarec\_id | cast\_id | taxonomic\_name | taxon\_count | original\_unit |
| --- | --- | --- | --- | --- |
| T0000000001 | C00000013 | Acartia bifilosa | 18.9960 | per m3 |
| T0000000002 | C00000014 | Acartia negligens | 31.9352 | per m3 |
| T0000000003 | C00000015 | Acartia negligens | 14.4690 | per m3 |
| T0000000004 | C00000001 | Acartia negligens | 48.5928 | per m3 |
| T0000000005 | C00000002 | Acartia negligens | 39.0825 | per m3 |
| T0000000006 | C00000003 | Acartia negligens | 47.9880 | per m3 |

Table  Casting sites in bio-database

| cast\_id | station\_id | date | depth\_lower\_bound | mesh\_size | gear\_type |
| --- | --- | --- | --- | --- | --- |
| C00000001 | S000001 | 1999-08-16 | 163 | 150 | NORPAC Net |
| C00000002 | S000002 | 1999-08-16 | 150 | 150 | NORPAC Net |
| C00000003 | S000003 | 1999-08-16 | 124 | 150 | NORPAC Net |
| C00000004 | S000004 | 1999-08-16 | 132 | 150 | NORPAC Net |
| C00000005 | S000005 | 1999-08-18 | 75 | 150 | NORPAC Net |
| C00000006 | S000006 | 1999-08-18 | 68 | 150 | NORPAC Net |

library(RPostgres) #R (using Rcpp) Interface to PostgreSQL  
library(odbapi) #Internal API package, which is explained in   
#e.g., A dataset with species Creseis conica, Abudefduf saxatilis, Labidocera gallensis, Metridia macrura, and M. asymmetrica is ready to be imported.  
test\_sp <- c("Creseis conica", "Abudefduf saxatilis",   
 "Labidocera gallensis", "Metridia macrura", "Metridia asymmetrica")  
  
pconn <- dbConnect(drv = RPostgres::Postgres(),  
 host = odbHost, port = odbPort, dbname = odbBioDB,  
 user = odbUser, password = odbPass)  
#Got taxonomy of bio-database on PostgreSQL  
taxonomy <- dbReadTable(pconn, name="taxon\_group") %>% setDT()  
#getSciName first retrieve query from existed taxonomy in "taxon\_group" table.  
#For those scientific names not in bio-database, retrieve them by web API of WORMS, GBIF, etc. Internally its magic is done through R package "taxize".  
new\_taxa <- odbapi::getSciName(dbuser = odbUser, dbhost = odbHost, dbname = odbBioDB,  
 taxon = test\_sp[!test\_sp %chin% taxonomy$show\_name],  
#Only the latter three species are new to “taxon\_group” table. This check can be ignored.  
 source = c("worms", "gbif")) #see Table 2.3

## [1] "connection to odbio database stage.."  
## [1] "fetch odbio taxon\_group.."  
## [1] "3 taxonomic names need to be queryed in taxize db. Please wait and check the interactive prompt..."  
## == 3 queries ===============  
## Error : (400) Bad Request - Labidocera gallensis  
## [1] "Warning: source not connect: worms"  
## == 3 queries ===============  
## v Found: Labidocera gallensis  
## v Found: Metridia macrura  
## v Found: Metridia asymmetrica  
## == Results =================  
##   
## \* Total: 3   
## \* Found: 3   
## \* Not Found: 0

Table  Update table of taxonomy (parts) in bio-database

| query | rank | taxon | class | order | family | genus | species |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Labidocera gallensis | species | Labidocera gallensis | Hexanauplia | Calanoida | Pontellidae | Labidocera | Labidocera gallensis |
| Metridia macrura | species | Metridia macrura | Hexanauplia | Calanoida | Metridinidae | Metridia | Metridia macrura |
| Metridia asymmetrica | species | Metridia asymmetrica | Hexanauplia | Calanoida | Metridinidae | Metridia | Metridia asymmetrica |

## 2.2 *Collaboration with larval fish database*

fconn <- dbConnect(drv = RPostgres::Postgres(),  
 host = larvaHost, port = odbPort, dbname = larvaDB,  
 user = odbUser, password = odbPass)  
# Similar schema to get the same tables from two different data source on PostgreSQL  
# Advantage: to design an integrated API to access them is more easier  
bio\_site <- dbReadTable(pconn, name="cast\_site") %>% setDT()  
larva\_site <- dbReadTable(fconn, name="cast\_site") %>% setDT()  
  
library(sf) # R package that handles geometries in GIS applications   
sites <- list(bio\_site[, source:="ODB"], larva\_site[, source:="CHIU"]) %>%   
 rbindlist(use.names = TRUE, fill = TRUE) %>%   
 .[, season := odbapi::datex\_season(date)] %>% # Internal API: convert date to season  
 .[, season := factor(season, levels = c(0,1,2,3),  
 labels=c("Spring","Summer","Autumn","Winter"))] %>%  
 st\_as\_sf(coords = c("longitude", "latitude"), crs = 4326)  
bbox <- st\_bbox(sites) # Bounding box, used in plotting Fig. 2.4  
dbDisconnect(pconn) # Close database connection  
dbDisconnect(fconn)

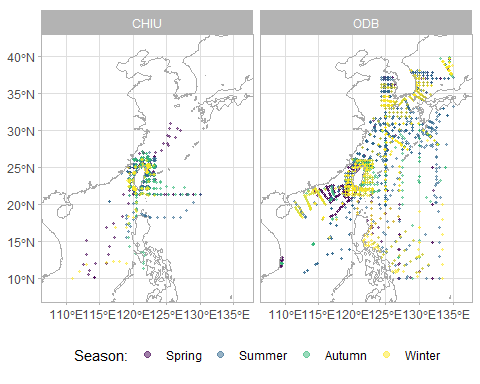


Fig. . Distribution of sampling sites in ODB bio-database

## 2.3 *Taxonomic composition of bio-database*

ODB bio-database collected a total of 137,819 records of abundance data, mostly zooplankton and larval fish, at 6700 sampling sites in 475 cruises. Fig. 2.5 shows the taxonomic composition in bio-database with the number of records and proportions. The largest proportion larval fish data, mainly comes from Dr. Chiu’s research, sums up to a 61,165 records with 673 species, 500 genera, and 190 families (Table 2.4).

## 2.4 *Eco-environment database in ODB*

This is a reference to table .

Table  Dataset demo

| name | data source | description | URL | variables | x resolution | y resolution | records | longitude | latitude | years |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| nasa\_nobm\_v2017 | NASA/Goddard Space Flight Center | NASA Ocean Biogeochemical Model (NOBM) | https://gmao.gsfc.nasa.gov/reanalysis/MERRA-NOBM | longitude, latitude, date, season, mixlayer, seaice, chl, nitrate, iron, chlorophyte, diatom, coccolith, cyanobacteria | 1.25 | 0.667 | 658800 | 88.75 - 150.00 | 0.00 - 40.00 | 1998 - 2015 |
| nasa\_neo | NASA/Goddard Space Flight Center | NASA Earth Observations (NEO) | https://neo.gsfc.nasa.gov | longitude, latitude, date, season, sst, chl | 0.10 | 0.100 | 30927976 | 90.10 - 150.00 | 0.00 - 40.00 | 2002 - 2018 |
| ncep\_godas | NOAA/National Weather Service | NCEP Global Ocean Data Assimilation System (GODAS) | https://www.cpc.ncep.noaa.gov/products/GODAS | longitude, latitude, date, season, depth, salinity | 1.00 | 0.333 | 99118656 | 89.50 - 150.50 | -0.17 - 40.17 | 1980 - 2016 |
| noaa\_woa\_v2013 | NOAA/National Centers for Environmental Information | WORLD OCEAN ATLAS 2013 version 2 (WOA13) | https://www.nodc.noaa.gov/OC5/woa13 | longitude, latitude, month, season, depth, nitrate, phosphate, silicate | 1.00 | 1.000 | 116018 | 89.00 - 140.00 | 9.00 - 40.00 | - 2012 |
| svp\_pacific\_v2017\_025d | NOAA/Atlantic Oceanographic and Meteorological Laboratory (AOML) | Global Lagrangian Drifter Data | https://www.aoml.noaa.gov/phod/gdp | longitude, latitude, season, u, v | 0.25 | 0.250 | 58564 | 107.00 - 137.00 | 2.00 - 32.00 | - 2012 |
| ctd\_odb | ODB, IONTU | Marine hydrographic data from Conductivity-Temperature-Depth Profiler (CTD) | https://www.odb.ntu.edu.tw | longitude, latitude, date, season, cruise, depth, temperature, salinity, fluorescence, transmission, oxygen | NA | NA | 15076533 | 105.10 - 156.17 | 1.98 - 32.75 | 1985 - 2017 |
| depth\_odb\_v2011\_025d | ODB, IONTU | Bathymetric data in 0.5-degree compiled by ODB | https://www.odb.ntu.edu.tw | longitude, latitude, sea\_depth | 0.25 | 0.250 | 16093 | 105.00 - 135.00 | 2.00 - 35.00 | - 2011 |
| depth\_odb\_v2011\_ea1000m | ODB, IONTU | Bathymetric data about 1000m-gridded compiled by ODB | https://www.odb.ntu.edu.tw | longitude, latitude, sea\_depth | 0.01 | 0.010 | 9906301 | 105.00 - 135.00 | 2.00 - 35.00 | - 2011 |

odbapi::getEnv\_glossary()

| var | unit | name | label |
| --- | --- | --- | --- |
| temperature | (unit: deg C) | Sea temperature | Temperature |
| salinity | (unit: PSU) | Salinity | Salinity |
| fluorescence | (unit: uM) | Fluorescence | Fluorescence |
| transmission | (unit: %) | Transmission | Transmission |
| oxygen | (unit: ug/l) | Dissolved oxygen | Oxygen |
| mixlayer | (unit: m) | Mixed-layer depth | Mixlayer |
| seaice | (unit: %) | Sea ice cover | Seaice |
| chl | (unit: mg/m3) | Chlorophyll | Chla |
| nitrate | (unit: mmole/l) | Nitrate | Nitrate |
| iron | (unit: nano more/l) | Iron | Iron |
| chlorophyte | (unit: mg/m3) | Chlorophytes | Chlorophyte |
| diatom | (unit: mg/m3) | Diatom | Diatom |
| coccolith | (unit: mg/m3) | Coccolithophores | Coccolithophore |
| cyanobacteria | (unit: mg/m3) | Cyanobacteria | Cyanobacteria |
| sst | (unit: deg C) | SST | SST |
| sea\_depth | (unit: m) | Sea depth | Depth |
| phosphate | (unit: mmole/l) | Phosphate | Phosphate |
| silicate | (unit: mmole/l) | Silicate | Silicate |

# 3 **OpenCPU framework for internal web APIs of bio-database**

## 3.1 *OpenCPU server configuration*

library(data.table)  
library(magrittr)  
library(odbapi)  
  
args = list(dbhost = larvaHost, dbuser = odbUser, dbname = larvaDB, varname = "metadata\_raw",  
 taxon ='fish', taxon\_lvl = 'species', value\_unit = 'perm3', appends='all')  
fd0 <- do.call(getBio, args)

## [1] "connection to PostgreSQL database stage.."  
## [1] "SQL Query cast\_id from cast\_tbl.."  
## [1] "fetch new taxa list.."  
## [1] "SQL Query cast sites.."  
## [1] "SQL Query taxa\_data by filtering taxon, and other criteria.."

# curl https://bio.odb.ntu.edu.tw/ocpu/library/odbapi/R/getBio/json -H "Content-Type: application/json" -d "{\"dbuser\":\"odbguest\",\"dbhost\":\"Your-Host\", \"dbname\":\"Your-DB\",\"value\_unit\":\"perm3\"}" -o output.json  
library(jsonlite)  
library(httr)  
# Note: The issue (github.com/opencpu/opencpu/issues/328) indicates that:   
# The default digits argument in jsonlite::toJSON() set to 4. Add JSON encoding parameters as http parameters to the ../json URL. Try appending ?digits=10 or use\_signif=TRUE  
res <- POST(  
 url = "https://bio.odb.ntu.edu.tw/ocpu/library/odbapi/R/getBio/json?use\_signif=TRUE",  
 body = jsonlite::toJSON(args), encode = "json",  
 add\_headers("Content-Type" = "application/json")  
 )   
httr::stop\_for\_status(res)  
fd0t <- jsonlite::fromJSON(content(res, as="text")) %>% setDT()

diffobj::diffPrint(fd0t, fd0)   
# The difference between the results from API call/R-package is in the number of digits, which can be solved by ?digits=10.  
## < fd0t ## Results from API   
## > fd0 ## Results from R-package  
## rank taxarec\_id cast\_id taxon\_count mlength date time depth   
## < 1: species 33377 1029 0.0011680 14.4 1993-05-21 2200 17   
## > 1: species 33377 1029 0.0011675131 14.4 1993-05-21 2200 17  
## < 2: species 33887 1036 0.0009689 16.6 1993-05-22 0400 35   
## > 2: species 33887 1036 0.0009688737 16.6 1993-05-22 0400 3

library(curl)  
print(res$headers$`x-ocpu-session`)

## [1] "x0facf10b17bfb4"

## [1] "x0ef9155ba8e8a5" # Current session ID to keep the state of HTTP REST API  
rez <- paste0('https://bio.odb.ntu.edu.tw/ocpu/tmp/',  
 res$headers$`x-ocpu-session`,'/R/.val/json?use\_signif=TRUE') %>%   
 curl\_fetch\_memory() # Get the results from a storage of this session generated by OpenCPU  
fd0z <- rawToChar(rez$content) %>% jsonlite::fromJSON() %>% setDT()   
all.equal(fd0t, fd0z) # [1] TRUE # Definitely, it’s just from the same storage of this session

## [1] TRUE

## 3.2 *Internal API package: odbapi*

library(Rcpp) # implement R functions in C++  
library(RcppArmadillo) # include a C++ library <Armadillo> for linear algebra that can speedup vector computing  
cppsrc <- '  
Rcpp::List grid\_combineC (SEXP x, SEXP y, int grd\_sel) {  
 NumericVector xi(x);  
 NumericVector yi(y);  
 IntegerVector sgnx = sign(xi);  
 IntegerVector sgny = sign(yi);  
 NumericVector xs = abs(xi);  
 NumericVector ys = abs(yi);  
 vec xt = floor( xs \* 100.00 + 0.5 ) / 100.00;  
 vec yt = floor( ys \* 100.00 + 0.5 ) / 100.00;  
 vec xl = floor(xt);  
 vec yl = floor(yt);  
 colvec lon(xl.begin(), xl.size(), false);   
 colvec lat(yl.begin(), yl.size(), false);  
 if (grd\_sel==1) {  
 int n = lon.size();  
 for(int j = 0; j < n; j++) {  
 lon[j] = (xt[j]-lon[j])>=0.5? lon[j]+0.5: lon[j];  
 lat[j] = (yt[j]-lat[j])>=0.5? lat[j]+0.5: lat[j];  
 } }  
 return Rcpp::List::create(Named("lon") = lon % as<vec>(sgnx), Named("lat") = lat % as<vec>(sgny));  
}'  
sourceCpp("src/grid\_combine.cpp") # A simplified C++ source code for odbapi::grdxy\_comb. Only test 0.5-degree grid here.   
grid\_combineR <- function (x, y, grd\_sel=1) {  
 xt <- round(x,2);  
 yt <- round(y,2)  
 xl <- as.integer(xt)  
 yl <- as.integer(yt)  
 if (grd\_sel == 1) { #Simplified case: only test 0.5-degree grid here.  
 return(list(lon = ifelse((xt-xl)>=0.5, xl+0.5, xl),  
 lat = ifelse((yt-yl)>=0.5, yl+0.5, yl)))   
 }  
 return(list(lon = x,lat = y))  
}  
# Generate a testing data.table with 1 million rows  
testdt <- do.call("rbind", replicate(  
 30, fd0[,.(show\_name, longitude, latitude)], simplify = FALSE)  
) # Benchmark: The 1st and 2nd term using Rcpp are just the same, comparing with 3rd term using R, get 3x improvement in speed.

res <- microbenchmark::microbenchmark(  
 "grd\_odbapi" = copy(testdt) %>% .[,c("lon","lat") := grdxy\_comb(longitude,latitude,1)],  
 "grd\_cpp" = copy(testdt) %>% .[,c("lon","lat") := grid\_combineC(longitude,latitude,1)],  
 "grd\_R" = copy(testdt) %>% .[,c("lon","lat") := grid\_combineR(longitude,latitude,1)],  
 times = 50  
) %>% print()

## Unit: milliseconds  
## expr min lq mean median uq max neval  
## grd\_odbapi 47.7582 58.5363 77.16663 61.34715 69.6767 178.5297 50  
## grd\_cpp 48.4187 60.4455 81.75065 62.43385 70.3393 179.2449 50  
## grd\_R 145.2124 155.3240 194.55008 159.81860 260.5772 277.3182 50

# Unit: milliseconds  
# expr min lq mean median uq max neval  
# grd\_odbapi 46.5205 54.8443 73.98525 56.3518 63.7002 208.5724 50  
# grd\_cpp 46.6298 54.1312 70.14181 57.0878 64.0498 157.3354 50  
# grd\_R 136.2115 146.1351 173.52936 150.7337 181.3523 276.3082 50

odbapi::getBio(   
 dbuser, dbhost, dbname,  
 tblname = NA, varname = NA, taxon = NA, taxon\_lvl = "All",   
 season\_sel = NA, grd\_sel = NA, grd\_additive = 0L,   
 lng\_rng = NA, lat\_rng = NA, date\_rng = NA, depth\_rng = NA, mesh\_rng = NA,   
 lifestages = NA, value\_unit = NA, site = NA, appends = NA, parse\_simple = TRUE  
)

# 4

#{r, child=c('larvalfish-regional-distribution.Rmd')} #

# 5 **Public web APIs of bio-database: Open-API project**

## 5.1 *Open-API package: exapi*

exapi::geo2map(  
 geo = NULL, geopoly = NULL, file = NULL, format = "csv", type = "POLYGON",  
 crs = 4326, site = NULL, stfile = NA\_character\_, en\_intersect = FALSE,   
 env\_layer = NA\_character\_, envsrc = NA\_character\_, by\_season = TRUE, grd\_env = 1L,   
 env\_fill\_palette = "inferno", env\_label = NA\_character\_, force\_krig\_off = FALSE,   
 en\_current = FALSE, bubble\_datasrc = NULL, bubble\_rng = c(0.5, 7.5), bubble\_offset = 0,   
 val\_layer = NA\_character\_, val\_group = NA\_character\_, val\_label = val\_layer, val\_trans = "exp",  
 en\_poly\_border = TRUE, enlarge\_poly\_bbox = TRUE, scale\_res = "medium",   
 legend\_pos = "topleft", color = NA\_character\_, debug\_mode = FALSE)

# 6 **A web application for the APIs of bio-database: Bio-Query**

## 6.1 *Bio-Query implemented on a R-Shiny web framework*

# in server.R  
library(shiny)  
library(promises)  
library(odbapi)  
library(R.cache)  
  
setCacheRootPath(path="./.Rcache")   
  
shinyServer(function(input, output, session) {  
 ggmapFx <- function(scale="medium") {  
 ggplot() + geom\_sf(data = ne\_coastline(scale = scale, returnclass = "sf"))  
 }  
  
 memFx <- function (FUN, akey=NULL, use\_akey\_only=FALSE, flatten\_key=FALSE, ...) {  
 if (is.null(akey)) {  
 key <- list(...)  
 } else {  
 if (use\_akey\_only) {  
 key <- as.list(akey)  
 } else {  
 key <- list(akey,...)  
 }  
 }  
 if (flatten\_key) key <- purrr::flatten(key)  
 fn <- as.character(substitute(FUN))  
 data <- R.cache::loadCache(key)  
 if (!is.null(data)) {  
 print(paste0("Loaded ", fn, " cached data"))  
 return(data)  
 }  
 print(paste0("Generating ", fn, " data from key: ", key))  
 data <- FUN(...)  
 R.cache::saveCache(data, key=key, comment=paste0(fn,"()"))  
 return(data)  
 }  
  
 gbasemap <-   
 memFx(FUN=ggmapFx, akey=list(funx="ggmapFx"), use\_akey\_only=TRUE, scale="medium") + coord\_sf()  
   
 xst <- future(getBio(dbuser=def\_user,dbhost=def\_host,dbname=def\_biodb,  
 tblname='cast\_site', appends = c('date','cite\_ref')))  
 xsite <- memFx(FUN=future::value, akey=list(timestamp=Lastest\_Modified\_TimeStamp, funx="site\_odb")  
  
 bioData <- reactiveValues(site = xsite, abundance = data.table(), mode="site")  
 bioAction <- reactiveValues(renew = FALSE, content = ggplot())  
   
}  
   
 bio\_future <- reactive({  
 argx <- bio\_paramx() # A reactive() value changed by UI actions observed in oberserveEvent({})   
 future(do.call(getBio, argx))  
 })  
   
 lfish\_future <- reactive({  
 argx <- lfish\_paramx()  
 future(do.call(getBio, argx))  
 })  
   
 bdt <- bio\_future() %...>% (function(dt) {  
 if (error\_handling(dt)) return(data.table())  
 return(dt)  
 })  
   
 fdt <- lfish\_future() %...>% (function(dt) {  
 if (error\_handling(dt)) return(data.table())  
 return(dt)  
 })  
  
 observeEvent(input$specific\_user\_event, {  
 req(input$specific\_user\_event)  
   
 promise\_all(bdt, fdt) %...>% {  
 dt <- rbindlist(l=., fill=TRUE, use.names = TRUE)  
 bioData$taxadt<- xdt  
 bioData$mode <- "abundance"  
 }  
 })  
   
 site\_Plot <- reactive({  
 dt <- bioData$site  
 future({  
 gbasemap + geom\_point(data=dt, aes(x=longitude, y=latitude, ...)) + ...  
 })   
 })  
   
 data\_Plot <- reactive({  
 dt <- bioData$abundance  
 future({  
 gbasemap + geom\_point(data=dt, aes(x=longitude, y=latitude, ...)) + ...  
 })   
 })  
   
 output$Canvas <- renderPlot({  
 if (bioData$mode == "abundance") {  
 g1 <- data\_Plot()  
 } else {  
 g1 <- site\_Plot()  
 }  
 g1 %...>% (function(gx) {  
 bioAction$content <- gx  
 bioAction$renew <- TRUE  
 bioAction$content  
 })  
 })  
})