arcgis R api

Josiah Parry

Goals:

- be as fast as possible (use Rcpp where it makes sense)
- be lazy
- be familiar (integrate with dplyr)

Be familiar

- R users expect tibbles
- R users expect sf objects

tibbles

> “Tibbles are data.frames that are lazy and surly”

- type-safe
- strict

```
# A tibble: 3 × 2
  name  plays

1 John  guitar
2 Paul  bass
3 Keith guitar
```

sf object

- extension of `data.frame`s with an explicit geometry column
- `sf == "simple features"`
  - uses simple feature standard (OGC)

```
Simple feature collection with 5 features and 1 field
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -69.44558 ymin: 7.883846 xmax: 34.34388 ymax: 87.83796
Geodetic CRS:  WGS 84
  x                   geometry
1 1 POINT (-16.33905 87.83796)
2 2  POINT (34.34388 7.883846)
3 3  POINT (29.67758 29.29599)
4 4 POINT (-57.85205 48.63379)
5 5 POINT (-69.44558 34.10791)
```

be lazy

- delay computation as long as possible
- `{dbplyr}` is a dplyr interface for databases
  - dplyr becomes a front-end and backend agnostic

dbplyr example

```r
car <- tbl(con, "mtcars")

car
```

```
# Source:   table [?? x 11]
# Database: sqlite 3.40.0 [:memory:]
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb

 1  21        6  160    110  3.9   2.62  16.5     0     1     4     4
 2  21        6  160    110  3.9   2.88  17.0     0     1     4     4
 3  22.8      4  108     93  3.85  2.32  18.6     1     1     4     1
 4  21.4      6  258    110  3.08  3.22  19.4     1     0     3     1
 5  18.7      8  360    175  3.15  3.44  17.0     0     0     3     2
 6  18.1      6  225    105  2.76  3.46  20.2     1     0     3     1
 7  14.3      8  360    245  3.21  3.57  15.8     0     0     3     4
 8  24.4      4  147.    62  3.69  3.19  20       1     0     4     2
 9  22.8      4  141.    95  3.92  3.15  22.9     1     0     4     2
10  19.2      6  168.   123  3.92  3.44  18.3     1     0     4     4
# … with more rows
```

behaves like a data frame

```
qry <- car |>
  group_by(cyl) |>
  summarise(avg_mpg = mean(mpg, na.rm = TRUE))
```

behaves like a data frame

```r
qry <- car |>
  group_by(cyl) |>
  summarise(avg_mpg = mean(mpg, na.rm = TRUE))


show_query(qry)
```

```
SELECT `cyl`, AVG(`mpg`) AS `avg_mpg`
FROM `mtcars`
GROUP BY `cyl`
```

behaves like a data frame

```
1 qry <- car |>
2   group_by(cyl) |>
3   summarise(avg_mpg = mean(mpg, na.rm = TRUE))
4
5
6 compute(qry)
```

```
# Source:   table [3 x 2]
# Database: sqlite 3.40.0 [:memory:]
    cyl avg_mpg

1     4    26.7
2     6    19.7
3     8    15.1
```

behaves like a data frame

```
1 qry <- car |>
2   group_by(cyl) |>
3   summarise(avg_mpg = mean(mpg, na.rm = TRUE))
4
5
6 collect(qry)
```

```
# A tibble: 3 × 2
    cyl avg_mpg

1     4    26.7
2     6    19.7
3     8    15.1
```

api R interface prototype

Implemented:

- Authentication (client & code)
- Feature Layer
- Table
- Feature Server
- sf -> Esri geometry conversion
- Feature Layer & table dplyr interface

`FeatureLayer`

- metadata list with nice print method
- behaves similar to a connection
- stores query
  - only executed with `collect()`

FeatureLayer

```
library(arcgis)

# define the feature layer url
furl <- "https://services.arcgis.com/P3ePLMYs2RVChkJx/ArcGIS/rest/services/

# create a feature layer
layer <- feature_layer(furl)
layer
```

```
#> <FeatureLayer <3143 features, 12 fields>>
#> Name: USA Counties - Generalized
#> Geometry Type: esriGeometryPolygon
#> CRS: 4326
#> Capabilities: Query,Extract
```

FeatureLayer

extract fields

```
# A tibble: 12 × 10
   name       type  alias sqlType nulla…¹ edita…² domain defau…³ length

 1 OBJECTID   esri… OBJE… sqlTyp… FALSE   FALSE   NA     NA           NA
 2 NAME       esri… Name  sqlTyp… TRUE    TRUE    NA     NA           50
 3 STATE_NA… esri… Stat… sqlTyp… TRUE    TRUE    NA     NA           20
 4 STATE_FI… esri… Stat… sqlTyp… TRUE    TRUE    NA     NA            2
 5 FIPS       esri… FIPS  sqlTyp… TRUE    TRUE    NA     NA            5
 6 SQMI       esri… Area… sqlTyp… TRUE    TRUE    NA     NA           NA
 7 POPULATI… esri… 2020… sqlTyp… TRUE    TRUE    NA     NA           NA
 8 POP_SQMI   esri… Peop… sqlTyp… TRUE    TRUE    NA     NA           NA
 9 STATE_AB… esri… Stat… sqlTyp… TRUE    TRUE    NA     NA            2
10 COUNTY_F… esri… Coun… sqlTyp… TRUE    TRUE    NA     NA            3
11 Shape__A… esri… Shap… sqlTyp… TRUE    FALSE   NA     NA           NA
12 Shape__L… esri… Shap… sqlTyp… TRUE    FALSE   NA     NA           NA
# … with 1 more variable: description … and abbreviated variable
```

FeatureLayer

- build a query using `tidyselect` helpers

```
layer |>
  select(starts_with("STATE"))
```

```
#> <FeatureLayer <3143 features, 12 fields>>
#> Name: USA Counties - Generalized
#> Geometry Type: esriGeometryPolygon
#> CRS: 4326
#> Capabilities: Query,Extract
#>
#> ── Query
#> outFields: STATE_NAME,STATE_FIPS,STATE_ABBR
```

FeatureLayer

- add `where` clause with `filter()`

```
qry <- layer |>
  select(starts_with("STATE")) |>
  filter(STATE_ABBR == "CA")

qry
```

```
#> <FeatureLayer <3143 features, 12 fields>>
#> Name: USA Counties - Generalized
#> Geometry Type: esriGeometryPolygon
#> CRS: 4326
#> Capabilities: Query,Extract
#>
#> ── Query
#> outFields: STATE_NAME,STATE_FIPS,STATE_ABBR
#> where: STATE_ABBR = 'CA'
```

FeatureLayer

- stop being lazy (bring into memory)

```
1 collect(qry)
```

```
Simple feature collection with 58 features and 3 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -124.3926 ymin: 32.53578 xmax: -114.1252 ymax: 42.00219
Geodetic CRS:  WGS 84
# A tibble: 58 × 4
   STATE_NAME STATE_FIPS STATE_ABBR                                  geometry
 *
 1 California 06         CA         (((-121.4721 37.47772, -121.8587 …
 2 California 06         CA         (((-120.0152 38.43224, -120.05 38…
 3 California 06         CA         (((-120.9838 38.22236, -121.0162 …
 4 California 06         CA         (((-121.6148 39.30518, -121.8968 …
 5 California 06         CA         (((-120.6423 37.82521, -120.9193 …
 6 California 06         CA         (((-122.3475 38.9245, -122.4086 3…
 7 California 06         CA         (((-122.3076 37.89176, -122.3715 …
```

FeatureServer

- a collection of feature layers and tables

```
furl <- "https://services2.arcgis.com/j80Jz20at6Bi0thr/ArcGIS/rest/services

ft_srv <- feature_server(furl)
```

```
#> <FeatureServer <2 Features>>
#> CRS: 3857
#> Capabilities: Query
#> ┌ Features ──────────────────────────────────────────┐
#> │ 24: Adoption_Facilities (esriGeometryPolygon)     │
#> │ 27: Adoption_Org (Table)                          │
#> └──────── List of adoption providers in each state ┘
```

- extract with `get_layer(id)` or `get_all_layers()`
  - returns a `FeatureLayer` or `Table` or named list

FeatureServer

```
#> <FeatureLayer <51 features, 9 fields>>
#> Name: Adoption_Facilities
#> Geometry Type: esriGeometryPolygon
#> CRS: 3857
#> Capabilities: Query
```

Table

- `feature_table()` behaves just like `FeatureLayer`

```
#> <Table <281 features, 8 fields>>
#> Name: Adoption_Org
#> Capabilities: Query
```

JSON conversion

- sf object conversion to Esri JSON
- written in Rcpp
  - fairly fast
- 3 types of objects: sfg, sfc, sf

sf object types:

- sfg: simple feature geometry
- sfc: simple feature column
- sf: simple feature (data.frame + sfc)

sf object mapping:

- sfg -> Geometry Object
- sfc -> FeatureSet with no attributes
- sf -> FeatureSet with attributes

st_as_geometry()

```r
xyz <- st_point(c(0, 1, 3, 4))
st_as_geometry(xyz) |>
  jsonify::pretty_json()
```


```
{
    "hasZ": true,
    "hasM": true,
    "x": 0.0,
    "y": 1.0,
    "z": 3.0,
    "m": 4.0,
    "spatialReference": {
        "wkid": 4326
    }
}
```

st_as_geometry()

```r
lines <-st_multipoint(x = matrix(runif(4, -90, 90), ncol = 2))

st_as_geometry(lines) |>
  jsonify::pretty_json()
```


```
{
    "hasZ": false,
    "hasM": false,
    "points": [
        [
            76.20709268376231,
            40.36714492365718
        ],
        [
            -52.82256934326142,
            2.591481409035623
        ]
    ],
    "spatialReference": {
        "wkid": 4326
    }
```

st_as_geometry()

```r
# polygon
m <- matrix(
  c(0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 2, 2, 1, 2, 3, 1, 3, 2, 0, 0, 0),
  ncol = 3,
  byrow = TRUE
)

poly <- st_polygon(list(m))

st_as_geometry(poly) |>
  jsonify::pretty_json()
```


```
{
    "hasZ": true,
    "hasM": false,
    "rings": [
        [
            [
                0.0,
                0.0,
                0.0
            ],
            [
                0.0,
```

```
            0.0,
            1.0
        ],
```

st_as_featureset()

```
1 st_as_featureset(sfnetworks::roxel[1,]) |>
2   jsonify::pretty_json()
```


```
{
    "geometryType": "esriGeometryPolyline",
    "spatialReference": {
        "wkid": 4326
    },
    "hasZ": false,
    "hasM": false,
    "features": [
        {
            "attributes": {
                "name": "Havixbecker Strasse",
                "type": 5
            },
            "geometry": {
                "paths": [
                    [
```

Need help with:

- querying image API endpoint
- creating new layers from API
  - can’t find endpoint
- updating field definitions
  - can’t understand `addToDefinition` docs