Announcement
Rob Tweed · Sep 28, 2022

glsdb: JavaScript Objects that are actually IRIS Objects

I'd like to announce the release of something really rather interesting - revolutionary in fact. That may sound like hyperbole, but I don't think you'll have seen anything quite like this, or even thought it possible!

We've pushed out a new JavaScript/Node.js module named *glsdb* which you can read all about here in detail:

https://github.com/robtweed/glsdb

However, for the purposes of this announcement here, I just want to focus on one part of *glsdb*: its APIs that abstract IRIS (or Cache) Classes as equivalent JavaScript Objects.

Let's take a quick look at what I mean by that, because I mean JavaScript Objects that are actually IRIS Objects residing in the database!

Suppose I've installed the SAMPLES data from the Intersystems repository, eg as described here:

https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls...

So, to access this SAMPLES data using *glsdb*, I first have to open a connection to IRIS (*glsdb* uses our *mg-dbx* interface for making that connection), for example:

```
const {glsDB} = await import('glsdb');
let glsdb = new glsDB('iris');
let options = {
  connection: 'network',
  host: '172.17.0.2',
  tcp_port: 7042,
  username: "_SYSTEM",
  password: "xxxxxx",
  namespace: "SAMPLES"
}

glsdb.open(options);
```

Now that I've done that, I can do the following to access an Employee record in the SAMPLES database:

```
let Employee = glsdb.irisClass('Sample.Employee');
let employee = Employee(101);
```

So far, so unsurprising, perhaps.

But here's where the mind-boggling fun begins. We've created a JavaScript object named *employee* which appears to represent the Employee instance with an OID of 101, but it's actually a rather special JavaScript Object: it's a JavaScript Proxy Object that is directly mapped to the physical IRIS Class instance in the database. I have direct access to all the IRIS Class's properties and methods, and when I manipulate the Proxy Object, I'm actually manipulating the IRIS Class instance in the database! So I can now do things like this:

```
let salary = employee.Salary;
console.log('employee.Salary = ' + salary);
let company = employee.Company;
let name = company.Name;
console.log('Company: ' + name);
```

The important thing to realise here is that my *employee* object isn't an in-memory JavaScript copy of the IRIS Class instance - it is the ***actual*** IRIS Class instance, **in the database**!

And as you can see above, I can even chain through the relationship between the Employee and the Company as you would in ObjectScript. In fact I can even do this in one go:

```
console.log(employee.Company.Name);
```

and chain directly through the properties, directly in the physical IRIS Class in the database!

I can save changes to the object too - for that I use a special method named *save()* , eg: *employee.save()*

To create a new instance of an employee, I'd just not specify an OID, eg:

```
let newEmployee = Employee();
```

glsdb provides a shortcut way of defining and saving a new record all in one go, using the *set()* method, eg:

```
let ok = newEmployee._set({
    Title: 'Manager'
    Salary: 60000
})
```

You can even inspect the methods and properties available for the Class:

```
console.log(employee._properties);
console.log(employee._methods);
```

So, that's a quick look at *glsdb*: I hope it gives you a taste of how crazy it is!

You're probably wondering how is it even possible to do this kind of thing.

Well, in summary, it's all down to a relatively new feature of JavaScript: Proxy Objects. Proxy Objects are special objects that act, as their name implies, as a proxy to another real object. But their real power is in the fact that you can set traps for access or change to the Proxy Object's properties, and (and this is the key bit) you can apply custom logic to those traps. In the case of *glsdb*, that custom logic uses the *mg-dbx* APIs behind the scenes to perform the equivalent access or change that's happening to the Proxy Object to a corresponding IRIS class.

Anyway, that's a quick glimpse at the crazy world made possible with *glsdb!* What I've put out is an early release so I'm sure there are aspects of the IRIS proxying that will need tweaking. if you want to try it out and see what it can do, please let me know if you find anything that doesn't behave properly.

Oh and by the way, check out the other APIs that *glsdb* provides - they're equally interesting and mind-boggling!

And finally, *glsdb* is Open Source software, so please play around with the code and post PRs on the Github repository if you'd like to help out with its onward development.

#JavaScript #Node.js #Object Data Model #Caché #InterSystems IRIS