12_Workspace_organization_uf

September 22, 2019

1 12. WORKSPACE ORGANIZATION

The problem of global variables - In q, there is no lexical scope - In q there are local variables (within a function) and global variables that exist in the entire workspace. - Mutable shared state is the root of all evil: - q is single-threaded by default - multi-threading is q is implemented so that it does not support mutating global variables - However, the existence of global variables can lead to name clashes in q.

1.1 12.1. Namespaces

- A partial solution to name clashes is namespaces:
  - Namespaces is a hierarchical structure of names separated by a specific separating character
    - E.g.: forward/back/in file systems
    - Java package hierarchy
    - Web address hierarchy
  - In q, the dot (.) is the separator, which separates the containers from their contents:
    - In q, the containers are called *contexts*
    - E.g.: .jab.x, where
      - .jab is the context, which has the symbolic name ‘.jab and x is the variable name
      - and the fully qualified name of the variable is ˙.jab.x
      - .jab refers to the context called jab in the root context
      - varName would refer to a global variable in the root context
    - From within a function you cannot differentiate between a local and a global variable with the same name:
      - .jab, in this case holds all q entities inside the .jab context in a dictionary form, which can be overwritten

1.2 12.2. Contexts

- q contexts are dictionaries the keys of which are variables in the given context
- you can have nested contexts in the form of .ctx1.ctx2.ctx3.varName
- distinction between a context and a variable name is made based on if it is a properly constructed context dictionary or not?
- reserved namespaces by Kx Systems:
  - .kx (for their own use)

- .h Markup output of http
- .j Json serialization/ deserialization
- .o
- .q (buil-in functions written not in C)
- .u
- .z System variables, callbacks
- .Q Utilities functions:
  - general
  - environment
  - InterProcess Communication
  - datatype
  - database
  - partitioned database

[5]:
```
.h.xd xmlTable:([id:til 4] col1:`a`b`c`d; col2:(1;2;3;4))
```

[5]:
```
"<R>"
"<r><id>0</id><col1>a</col1><col2>1</col2></r>"
"<r><id>1</id><col1>b</col1><col2>2</col2></r>"
"<r><id>2</id><col1>c</col1><col2>3</col2></r>"
"<r><id>3</id><col1>d</col1><col2>4</col2></r>"
"</R>"
```

1.3 12.3. Creating contexts

- When we create the first variable in a namespace, the context is automatically created, in this case, foo: .foo.a

[7]:
```
key ` / returns all contexts in the root context
```

[7]:
```
`q`Q`h`j`o`qpk`P`p`b64
```

[12]:
```
key `.q / returns all variables in the `.q context
```

[12]:
```
``neg`not`null`string`reciprocal`floor`ceiling`signum`mod`xbar`xlog`and`or`ea..
```

[18]:
```
value `.q.xbar
```

[18]:
```
k){x*y div x:$[16h=abs[@x];"j"$x;x]}
```

1.4 12.4. Context as dictionary