

blackrock / NamedTuples.jl


Unwatch28Unstar12Fork9

<> Code 1 Issues 4 2 Pull requests 2 0 Projects 0 Wiki Pulse Graphs

NamedTuples.jl

70 commits 2 branches 7 releases 7 contributors Apache-2.0

Branch: masterNew pull requestCreate new fileUpload filesFind fileClone or download

 mdcfrancis committed on GitHub Merge pull request #21 from davidanthoff/julia-0-6-only ... Latest commit 683f855 12 days ago

src	Make things julia 0.6 only	14 days ago
test	Replace => syntax with =	7 months ago
.gitignore	Added CI related files	8 months ago
.gitlab-ci.yml	r -> t	8 months ago
.travis.yml	Add nightly build to travis	13 days ago
LICENSE.md	NamedTuples.jl generated files.	2 years ago
README.md	Replace => syntax with =	7 months ago
REQUIRE	Make things julia 0.6 only	14 days ago

README.md

NamedTuples

NamedTuples.jl provides a high performance implementation of named tuples for Julia (cf named tuples in python). Julia tuples are restricted to supporting index based access. This new implementation allows both index and property based access. NamedTuples may be used anywhere that a tuple is currently being used, for example in the definition of a method or as the return value of a method. NamedTuples are implemented using Julia's macro system, ensuring that the run time cost is equivalent to constructing a regular immutable type.

NamedTuples may also be used in cases where a small typed immutable dictionary is desired.

Syntax

```
@NT( a, b )                -> Defines a tuple with a and b as members
@NT( a::Int64, b::Float64 ) -> Defines a tuple with the specific arg types as members
@NT( a = 1, b = "hello" )  -> Defines and constructs a tuple with the specified members and values
@NT( a, b )( 1, "hello" )  -> Is equivalent to the above definition
@NT( a::Int64 )( 2.0 )     -> Calls `convert( Int64, 2.0 )` on construction and sets `a`
@NT( ::Int64, ::Float64 ) -> Defines a named tuple with automatic names
```

NamedTuples may be used anywhere you would use a regular Tuple, this includes method definition and return values.

```
module Test
using NamedTuples

function foo( y )
    a = 1
    x = 3
    return @NT( a = 1, b = "world", c = "hello", d=a/x, y = a/y )
end

function bar( nt::@NT( a::Int64, c::ASCIIString ))
    return repeat( nt.c, nt.a )
end

end
```

```
Test.foo( 1 ) # Returns a NamedTuple of 5 elements
Test.bar( @NT( a= 2, c="hello")) # Returns `hellohello`
```

There is at most one instance of a NamedTuple type with a given set of Members and Types, hence

```
typeof( @NT( a::Int64, b::Float64 )(1, 3.0) ) == typeof( @NT( a = 1, b = 2.0 ))
```

NamedTuple definitions are shared across all modules. The underlying immutable types are constructed at first use.

NamedTuples support iteration and indexing, and behave as immutable associative containers.

```
@NT( a = 1 ).a == 1
@NT( a = 1 )[1] == 1
@NT( a = 1 )[:a] == 1
length( @NT( a = 1 )) == 1
length( @NT( a = 1, b = 2.0 )) == 2
first( @NT( a = 1, b = 2.0 )) == 1
last( @NT( a = 1, b = 2.0 )) == 2.0
for( (k,v) in @NT( a = 1, b = 1 ))
    println( "$k = $v" )
end
slice( @NT( a = 1, b = 2, c = 3 ), [1:2]) # Named tuple (a=1,b=2)
```

NamedTuples additionally support operations to merge, update, add and delete elements. Since NamedTuples are immutable, these operations make a copy of the data and return a new NamedTuple. The current implementation of these operations is functional rather than performance oriented.

```
nt = @NT( a=1, b=2, c=3 )
x = NamedTuples.setindex( nt, :x, 123 )
NamedTuples.delete( x, :a ) # (b=2,c=3,x=123)
merge( nt, @NT( d = "hello", e = "world")) # ( a=1,b=2,c=3,d="hello",e="world")
```

Note the use of `setindex/delete` and not `setindex!/delete!` as these operations do NOT modify in place.

build passing

