Erweiterbare Arrays und Key-Value Dictionary

Robuste Software für parallele Computerarchitekturen SS2014

Merlin Göttlinger - 2262143

Technische Hochschule Nürnberg Georg-Simon-Ohm

19. Mai 2014

Gliederung

1 Array

2 Dictionary

Grundlegendes

Typischerweise (C#, Java):

- feste Größe
- 0-basierter Index
- veränderlicher Inhalt

Besonderheiten in Erlang:

- feste oder variable Größe!
- leere Stellen enthalten `undefined` (oder selbst definierten Standardwert)
- (intern) kein wahlfreier Zugrif!

Grundlegendes

Typischerweise (C#, Java):

- feste Größe
- 0-basierter Index
- veränderlicher Inhalt

Besonderheiten in Erlang:

- feste oder variable Größe!
- leere Stellen enthalten `undefined` (oder selbst definierten Standardwert)
- (intern) kein wahlfreier Zugrif!

Achtung

Alle Datenstrukturen in Erlang sind immutable. Funktionen die in anderen Sprachen Änderungen duchführen würden, geben eine geänderte Instanz zurück, statt die bestehende zu ändern.

Benutzung Modul array

Anlegen

```
new()
new(Options::array_opts())
new(Size::integer() >= 0, Options::array_opts())
```

Benutzung Modul array

Anlegen

```
new()
new(Options::array_opts())
new(Size::integer() >= 0, Options::array_opts())
```

Lesen

```
get(I::array_indx(), Array)
```

Benutzung Modul array

Anlegen

```
new()
new(Options::array_opts())
new(Size::integer() >= 0, Options::array_opts())
```

Lesen

```
get(I::array_indx(), Array)
```

Schreiben/Ändern

```
set(I::array_indx(), Value, Array)
```

Benutzung Modul array

Größe

```
size(Array)
```

Benutzung Modul array

Größe

```
size(Array)
```

Variabel → Fest

```
fix(Array)
```

Benutzung Modul array

Größe

```
size(Array)
```

Variabel → Fest

```
fix(Array)
```

Fest → Variabel

```
relax(Array)
```

Benutzung Modul array

Größe

```
size(Array)
```

Variabel → Fest

```
fix(Array)
```

Fest → Variabel

```
relax(Array)
```

Fest?

```
is_fix(Array)
```

Benutzung Modul array

“echte” Größe

```
sparse_size(Array)
```

Benutzung Modul array

“echte” Größe

```
sparse_size(Array)
```

Größe verändern

```
resize(Array)
resize(Size::integer() >= 0, Array)
```

Benutzung Modul array

Anlegen aus Liste

```
from_list(List::[Value::Type])
from_list(List::[Value::Type], Default::term())
```

Benutzung Modul array

Anlegen aus Liste

```
from_list(List::[Value::Type])
from_list(List::[Value::Type], Default::term())
```

Anlegen aus Orddict

```
from_orddict(Orddict::indx_pairs(Value::Type))
from_orddict(Orddict::indx_pairs(Value::Type),
             Default::Type)
```

Benutzung Modul array

Konvertieren in Orddict/Liste

```
to_list(Array)
to_orddict(Array)
```

Benutzung Modul array

Konvertieren in Orddict/Liste

```
to_list(Array)
to_orddict(Array)
```

Sparse Konversion

```
sparse_to_list(Array)
sparse_to_orddict(Array)
```

Benutzung Modul array

Fold Left

```
foldl(Function, InitialAcc::A, Array)->B
```

Benutzung Modul array

Fold Left

```
foldl(Function, InitialAcc::A, Array)->B
```

```
foldl(fun(I, E, A)->A + E end, 0, from_list([1, 2, 3]))
```

Benutzung Modul array

Fold Left

```
foldl(Function, InitialAcc::A, Array)->B
```

```
foldl(fun(I, E, A)->A + E end, 0, from_list([1, 2, 3]))
```

Rechnung: `((0 + 1) + 2) + 3`

Ergebnis: `6`

Benutzung Modul array

Fold Left

```
foldl(Function, InitialAcc::A, Array)->B
```

```
foldl(fun(I, E, A)->A + E end, 0, from_list([1, 2, 3]))
```

Rechnung: `((0 + 1) + 2) + 3`

Ergebnis: `6`

Fold Right

```
foldr(Function, InitialAcc::A, Array)->B
```

Benutzung Modul array

Fold Left

```
foldl(Function, InitialAcc::A, Array)->B
```

```
foldl(fun(I, E, A)->A + E end, 0, from_list([1, 2, 3]))
```

Rechnung: `((0 + 1) + 2) + 3`

Ergebnis: `6`

Fold Right

```
foldr(Function, InitialAcc::A, Array)->B
```

```
foldr(fun(I, E, A)->E + A end, 0, from_list([1, 2, 3]))
```

Benutzung Modul array

Fold Left

```
foldl(Function, InitialAcc::A, Array)->B
```

```
foldl(fun(I, E, A)->A + E end, 0, from_list([1, 2, 3]))
```

Rechnung: `((0 + 1) + 2) + 3`

Ergebnis: `6`

Fold Right

```
foldr(Function, InitialAcc::A, Array)->B
```

```
foldr(fun(I, E, A)->E + A end, 0, from_list([1, 2, 3]))
```

Rechnung: `1 + (2 + (3 + 0))`

Ergebnis: `6`

Benutzung Modul array

Sparse Folds

```
sparse_foldl/3
sparse_foldr/3
```

Benutzung Modul array

Sparse Folds

```
sparse_foldl/3
sparse_foldr/3
```

Sparse Map

```
sparse_map(Function, Array::array(T1))->array(T2)
```

Benutzung Modul array

Sparse Folds

```
sparse_foldl/3
sparse_foldr/3
```

Sparse Map

```
sparse_map(Function, Array::array(T1))->array(T2)
```

```
sparse_map(fun(I, A)->A * A end, from_list([1, 2, 3]))
```

Benutzung Modul array

Sparse Folds

```
sparse_foldl/3
sparse_foldr/3
```

Sparse Map

```
sparse_map(Function, Array::array(T1))->array(T2)
```

```
sparse_map(fun(I, A)->A * A end, from_list([1, 2, 3]))
```

Ergebnis: `from_list([1, 4, 9])`

Benutzung Modul array

Default

```
default(Array)
```

Benutzung Modul array

Default

```
default(Array)
```

Löschen

```
reset(I::array_indx(), Array)
```

Benutzung Modul array

Default

```
default(Array)
```

Löschen

```
reset(I::array_indx(), Array)
```

Array?

```
is_array(X::term())
```

Gliederung

1 Array

2 Dictionary

Grundlegendes

- Ähnlich `Dictionary<Key, Value>` in Java oder C#
- Abbildung von einem Key auf Menge an Values 1:n
- Keine Reihenfolge

dict nutzt `=:=`

orddict nutzt == für Gleichheitsbestimmung der Keys

Grundlegendes

- Ähnlich `Dictionary<Key, Value>` in Java oder C#
- Abbildung von einem Key auf Menge an Values 1:n
- Keine Reihenfolge

dict nutzt `=:=`

orddict nutzt `==` für Gleichheitsbestimmung der Keys

Achtung

Auch hier an Immutability denken!

Benutzung Module `dict` und `orddict`

Anlegen

```
new()
```

Benutzung Module dict und orddict

Anlegen

```
new()
```

Hinzufügen

```
append(Key, Value, Dict)
append_list(Key, ValList, Dict)
```

Benutzung Module `dict` und `orddict`

Anlegen

```
new()
```

Hinzufügen

```
append(Key, Value, Dict)
append_list(Key, ValList, Dict)
```

Löschen

```
erase(Key, Dict)
```

Benutzung Module dict und orddict

Lesen

```
find(Key, Dict) ->{ok, Value}| error
fetch(Key, Dict) ->Value
```

Benutzung Module dict und orddict

Lesen

```
find(Key, Dict) ->{ok, Value}| error
fetch(Key, Dict) ->Value
```

Aktualisieren

```
update(Key, Fun, Dict)
  Fun = fun((Value1) ->Value2)
update(Key, Fun, Initial::Value, Dict)
update_counter(Key, Increment::number(), Dict)
```

Benutzung Module dict und orddict

Lesen

```
find(Key, Dict) ->{ok, Value}| error
fetch(Key, Dict) ->Value
```

Aktualisieren

```
update(Key, Fun, Dict)
  Fun = fun((Value1) ->Value2)
update(Key, Fun, Initial::Value, Dict)
update_counter(Key, Increment::number(), Dict)
```

Aktualisieren/Hinzufügen

```
store(Key, Value, Dict)
```

Benutzung Module `dict` und `orddict`

Anzahl Keys

```
is_empty(Dict)
size(Dict)
```

Benutzung Module `dict` und `orddict`

Anzahl Keys

```
is_empty(Dict)
size(Dict)
```

Keys auslesen

```
fetch_keys(Dict)
```

Benutzung Module `dict` und `orddict`

Anzahl Keys

```
is_empty(Dict)
size(Dict)
```

Keys auslesen

```
fetch_keys(Dict)
```

Existiert der Key?

```
is_key(Key, Dict)
```

Benutzung Module `dict` und `orddict`

Listen

```
to_list(Dict)
from_list(List)
```

Benutzung Module dict und orddict

Listen

```
to_list(Dict)
from_list(List)
```

Merge

```
merge(Fun, Dict1, Dict2)
```

Benutzung Module dict und orddict

Listen

```
to_list(Dict)
from_list(List)
```

Merge

```
merge(Fun, Dict1, Dict2)
```

```
merge(fun(K, A, B) ->lists:append([A , B]) end,
      from_list([{"a", [1]}, {"b", [2]}]),
      from_list([{"a", [3]}, {"c", [4]}]))
```

Benutzung Module dict und orddict

Listen

```
to_list(Dict)
from_list(List)
```

Merge

```
merge(Fun, Dict1, Dict2)
```

```
merge(fun(K, A, B) ->lists:append([A , B]) end,
      from_list([{"a", [1]}, {"b", [2]}]),
      from_list([{"a", [3]}, {"c", [4]}]))
```

Ergebnis:

```
from_list([{"a", [1, 3]}, {"b", [2]}, {"c", [4]}])
```

Benutzung Module dict und orddict

Faltung

```
fold(Fun, Acc0, Dict) ->Acc1
  Fun = fun((Key, Value, AccIn) ->AccOut)
```

Benutzung Module dict und orddict

Faltung

```
fold(Fun, Acc0, Dict) ->Acc1
  Fun = fun((Key, Value, AccIn) ->AccOut)
```

Map

```
map(Fun, Dict1) ->Dict2
  Fun = fun((Key, Value1) ->Value2)
```

Benutzung Module dict und orddict

Faltung

```
fold(Fun, Acc0, Dict) ->Acc1
  Fun = fun((Key, Value, AccIn) ->AccOut)
```

Map

```
map(Fun, Dict1) ->Dict2
  Fun = fun((Key, Value1) ->Value2)
```

Filterung

```
filter(Pred, Dict1) ->Dict2
  Pred = fun((Key, Value) ->boolean())
```

Quellen

http://www.erlang.org/doc/man/array.html
http://www.erlang.org/doc/man/dict.html
http://www.erlang.org/doc/man/orddict.html
https://github.com/erlang/otp/blob/maint/lib/stdlib/src/array.erl
https://github.com/erlang/otp/blob/maint/lib/stdlib/src/dict.erl
https://github.com/erlang/otp/blob/maint/lib/stdlib/src/orddict.erl
http://stackoverflow.com/q/16447921/1876344
http://jkndrkn.livejournal.com/240948.html