Próximo:Outro hash, Anterior:Acesso de hash, Acima:Tabelas de hash [Conteúdo][Índice]

8.3 Definindo Comparações de Hash

Você pode definir novos métodos de pesquisa de chave por meio de `define-hash-table-test`. Para usar esse recurso, você precisa entender como as tabelas de hash funcionam e o que significa um *código de hash* .

Você pode pensar em uma tabela de hash conceitualmente como uma grande matriz de muitos slots, cada um capaz de manter uma associação. Para procurar uma chave, `gethash`primeiro calcula um número inteiro, o código hash, da chave. Ele pode reduzir este módulo inteiro do comprimento do array, para produzir um índice no array. Em seguida, ele procura nesse slot e, se necessário, em outros slots próximos, para ver se encontrou a chave que está sendo procurada.

Assim, para definir um novo método de pesquisa de chave, você precisa especificar uma função para calcular o código hash de uma chave e uma função para comparar duas chaves diretamente. As duas funções devem ser consistentes entre si: ou seja, os códigos de hash de duas chaves devem ser os mesmos se as chaves forem comparadas como iguais. Além disso, como as duas funções podem ser chamadas a qualquer momento (como pelo coletor de lixo), as funções devem estar livres de efeitos colaterais e devem retornar rapidamente, e seu comportamento deve depender apenas das propriedades das chaves que não mudam .

Função: define-hash-table-test *name test-fn hash-fn*

Esta função define um novo teste de tabela de hash, chamado *name* .

Depois de definir o *nome* dessa maneira, você pode usá-lo como o argumento de *teste*`make-hash-table` em . Ao fazer isso, a tabela de hash usará *test-fn* para comparar valores de chave e *hash-fn* para calcular um código de hash a partir de um valor de chave.

A função *test-fn* deve aceitar dois argumentos, duas chaves e retornar non- `nil`se forem considerados iguais.

A função *hash-fn* deve aceitar um argumento, uma chave, e retornar um inteiro que seja o código hash dessa chave. Para bons resultados, a função deve usar todo o intervalo de números fixos para códigos hash, incluindo números fixos negativos.

As funções especificadas são armazenadas na lista de propriedades *name* sob a propriedade `hash-table-test`; a forma do valor da propriedade é . (*`test-fn hash-fn`*)

Função: *obj* igual a sxhash

Esta função retorna um código hash para o objeto Lisp *obj* . Este é um inteiro que reflete o conteúdo de *obj* e os outros objetos Lisp para os quais ele aponta.

Se dois objetos *obj1* e *obj2* são `equal`, então e são o mesmo inteiro. (`sxhash-equal` *`obj1`*) (`sxhash-equal` *`obj2`*)

Se os dois objetos não forem `equal`, os valores retornados por `sxhash-equal`geralmente são diferentes, mas nem sempre; de vez em quando, por sorte, você encontrará dois objetos de aparência distinta que dão o mesmo resultado de `sxhash-equal`.

Nota do Common Lisp: No Common Lisp, uma função semelhante é chamada `sxhash`. O Emacs fornece esse nome como um alias de compatibilidade para `sxhash-equal`.

Função: sxhash-eq *obj*

Esta função retorna um código hash para o objeto Lisp *obj* . Seu resultado reflete a identidade de *obj* , mas não seu conteúdo.

Se dois objetos *obj1* e *obj2* são `eq`, então e são o mesmo inteiro. `(sxhash-eq obj1)(sxhash-eq obj2)`

Função: sxhash-eql *obj*

Esta função retorna um código hash para o objeto Lisp *obj* adequado para `eql`comparação. Ou seja, reflete a identidade do *obj* exceto no caso em que o objeto é um bignum ou um número float, caso em que um código hash é gerado para o valor.

Se dois objetos *obj1* e *obj2* são `eql`, então e são o mesmo inteiro. `(sxhash-eql obj1)(sxhash-eql obj2)`

Este exemplo cria uma tabela de hash cujas chaves são strings que são comparadas sem distinção entre maiúsculas e minúsculas.

```
(defun case-fold-string= (ab)
  (eq t (compare strings a nil nil b nil nil t)))
(defun case-fold-string-hash (a)
  (sxhash-igual (a maiúscula)))

(define-hash-table-test 'case-fold
  'case-fold-string='case-fold-string-hash)

(make-hash-table :test 'case-fold)
```

Aqui está como você pode definir um teste de tabela de hash equivalente ao valor de teste predefinido `equal`. As chaves podem ser qualquer objeto Lisp, e objetos de aparência igual são considerados a mesma chave.

```
(define-hash-table-test 'contents-hash 'equal 'sxhash-equal)

(make-hash-table :test 'contents-hash)
```

Programas Lisp *não* devem depender de códigos hash preservados entre sessões do Emacs, pois a implementação das funções hash usa alguns detalhes do armazenamento de objetos que podem mudar entre sessões e entre diferentes arquiteturas.

Próximo:Outro hash, Anterior:Acesso de hash, Acima:Tabelas de hash [Conteúdo][Índice]