스토리지

2015년 10월 22일에 한국어로 옮겨짐
2015년 11월 4일에 마지막으로 갱신됨

원문: https://github.com/torch/torch7/blob/master/doc/storage.md

목차

스토리지(storage)는 기본적으로 `C` 포인터나 배열의 메모리에 접근하기 위한 루아(`Lua`)의 한 방법입니다. 스토리지는 또한 한 파일의 내용을 메모리로 사상(map)할 수 있습니다. 스토리지 하나는 기본적인 c 타입 중 하나인 배열입니다. 토치(`Torch`) 객체들의 배열을 위해, 루아 테이블이 사용됩니다.

모든 기본 c 타입을 위한 스토리지 클래스가 존재합니다. 그 스토리지 클래스는 이름만 보고도 그 타입을 알 수 있습니다. 이를테면 `ByteStorage`, `CharStorage`, `ShortStorage`, `IntStorage`, `LongStorage`, `FloatStorage`, `DoubleStorage`가 그것입니다.

유념하십시오. `ByteStorage`와 `CharStorage`는 모두 두 배열이 바이트(byte) 타입임을 나타냅니다. 그러나 `ByteStorage`는 *unsigned* char 배열임을, `CharStorage`는 *signed* char 배열임을 나타냅니다.

두 `Storage` 타입 사이의 변환은 `copy`를 사용하여 수행될 수도 있습니다.

```
x = torch.IntStorage(10):fill(1)
y = torch.DoubleStorage(10):copy(x)
```

고전적인 스토리지들은 직렬화할 수 있습니다. 한 파일을 사상하는 스토리지들 또한 직렬화할 수 있으나 *한 보통 스토리지*로 저장될 것입니다. 고수준 직렬화 명령어들은 직렬화 절에서 설명됩니다.

torch.setdefaulttensortype 함수로 설정된 당신이 선호하는 스토리지 타입으로 한 별명 `torch.Storage()`가 만들어집니다. 기본적으로, 이것은 `torch.DoubleStorage`을 가리킵니다.

생성자와 접근 메소드

---

torch.TYPEStorage([size [, ptr]])

타입이 `TYPE`인 새 스토리지 하나를 리턴합니다. 유효한 `TYPE`에는 `Byte`, `Char`, `Short`, `Int`, `Long`, `Float`, `Double`이 있습니다. 만약 `size`가 주어지면, 그 스토리지의 크기가 `size`로 바뀝니다. 그렇지 않으면, 빈 스토리지 하나를 만듭니다.

선택적 두 번째 인자인 `ptr`은 숫자 하나입니다. 그 숫자의 값은 (이를테면 `torch.data()` 메소드의 리턴 값으로 나온) 크기가 `size*sizeof(TYPE)`인 메모리 덩어리를 가리키는 포인터입니다. 그 스토리지는 그 메모리 덩어리의 메모리 할당 해제(freeing)를 책임집니다.

예:

```
-- double 타입 요소 10개를 저장할 수 있는 스토리지 하나를 만듭니다.
x = torch.DoubleStorage(10)
```

그 스토리지의 데이터는 *초기화되지 않습니다*.

torch.TYPEStorage(table)

`table`은 숫자로 구성된 루아 배열이라고 가정됩니다. 생성자는 `TYPE`으로 특정된 새 스토리지 하나를 리턴합니다. 그 스토리지의 크기와 요소는 인자로 전달된 테이블과 같게 유지됩니다. 단, 테이블에 있던 요소들은 그 스토리지의 타입으로 타입이 바뀝니다.

예:

```
> = torch.IntStorage({1,2,3,4})
 1
 2
 3
 4
[torch.IntStorage of size 4]
```

torch.TYPEStorage(storage [, offset [, size]])

타입이 TYPE인 새 스토리지 하나를 리턴합니다. 그 스토리지는 첫 번째 인자에 대한 뷰(view)입니다. 첫 번째 인자의 타입은 반드시 TYPE과 같아야 합니다. 선택적 인자로 offset이 입력될 수 있습니다 (기본값은 1입니다). 새 스토리지의 크기를 제한하기 위한 선택적 인자로 size 또한 입력될 수 있습니다 (기본값은 storage:size()-(offset-1)입니다).

예:

```
-- double 타입 요소가 10개 들어갈 수 있는 크기의 스토리지 하나를 만듭니다.
> x = torch.DoubleStorage(10)

-- 오프셋 3에서 시작하고 크기가 5인 이 스토리지에 대한 뷰 하나를 만듭니다.
> y = torch.DoubleStorage(x, 3, 5)

-- y의 요소들의 바꾸면 x의 요소들 또한 바뀝니다.
> x:fill(0)
> y:fill(1)
> print(x)
 0
 0
 1
 1
 1
 1
 1
 0
 0
 0
[torch.DoubleStorage of size 10]
```

torch.TYPEStorage(filename [, shared [, size [, sharedMem]]])

한 새로운 종류의 스토리지를 리턴합니다. 그 스토리지는 인자로 입력된 filename의 내용들을 메모리로 사상(map)합니다. 유효한 TYPE에는 Byte, Char, Short, Int, Long, Float, Double이 있습니다. 만약 선택적 인자인 shared가 true이면, 사상된 메모리는 그 컴퓨터의 모든 프로세스들에 공유됩니다.

shared가 true일 때, 그 파일은 반드시 읽기-쓰기 모드로 접근되어야 합니다. 스토리지의 내용이 바뀌면, 파일의 내용도 함께 바뀝니다. 다만 그 바뀜은 스토리지가 없어진 다음에야 파일에 적용될지도 모릅니다.

shared가 false일 때 (또는 인자로 입력되지 않았을 때), 파일은 반드시 최소한 읽을 수 있어야 합니다. 스토리지의 내용이 바뀌어도, 파일의 내용은 바뀌지 않을 것입니다. 주의하십시오. 일단 스토리지가 만들어진 다음에 생긴 파일에 대한 변화는 스토리지 내용에 불특정한 영향을 미칩니다.

만약 size가 입력되면, 그것은 리턴될 스토리지의 size를 정합니다 (요소 개수). 이 경우, 만약 shared가 false이면, 그 파일은 반드시 최소한 다음 크기(bytes)를 이미 가지고 있어야 합니다.

```
size*(TYPE 크기)
```

만약 shared가 true이면, 그 파일은 필요에 따라 생성될 것이고, 필요한 크기(bytes)로 그 길이가 확장될 것입니다.

만약 size가 특정되지 않으면, 리턴된 스토리지의 size는 (인자로) 제공된 이미 존재하는 빈 파일 하나의

```
(byte 단위 파일 크기)/(TYPE 크기)
```

개 요소들일 것입니다.

만약 sharedMem이 true이면, 그 파일은 shm_open()을 사용하여 공유된 메모리 지역(area)에서 생성(또는 사상)될 것입니다. 리눅스 시스템에서 이것은 RAM 상의 프로세스 사이 통신을 위해 /dev/shm 부분에서 구현됩니다.

예:

```
$ echo "Hello World" > hello.txt
$ lua
Lua 5.1.3  Copyright (C) 1994-2008 Lua.org, PUC-Rio
> require 'torch'
> x = torch.CharStorage('hello.txt')
> = x
  72
 101
 108
 108
 111
  32
  87
 111
 114
```

```
 108
 100
  10
[torch.CharStorage of size 12]

> = x:string()
Hello World

> = x:fill(42):string()
**********
>
$ cat hello.txt
Hello World
$ lua
Lua 5.1.3  Copyright (C) 1994-2008 Lua.org, PUC-Rio
> require 'torch'
> x = torch.CharStorage('hello.txt', true)
> = x:string()
Hello World

> x:fill(42)
>
$ cat hello.txt
**********
```

[number] #self

그 스토리지에 있는 요소의 개수를 리턴합니다. size()와 같습니다.

[number] self[index]

그 스토리지의 `index` 위치에 있는 요소를 리턴하거나 설정합니다. `index`의 유효 범위는 1에서 size()입니다.

예:

```
x = torch.DoubleStorage(10)
print(x[5])
```

[self] copy(storage)

또다른 `storage`를 복사합니다. 두 스토리지의 타입은 다를 수도 있습니다. 그 경우, 인자로 입력된 스토리지의 타입이 바뀝니다 (물론, 이 바뀜은 정밀도 손실이나 반올림을 유발할 수도 있습니다). 이 메소드는 self를 리턴합니다.

```
x = torch.IntStorage(10):fill(1)
y = torch.DoubleStorage(10):copy(x) -- y는 nil이 아닐 것입니다!
```

[self] fill(value)

주어진 값으로 그 스토리지를 채웁니다. 이 메소드는 self를 리턴합니다.

```
x = torch.IntStorage(10):fill(0) -- x는 nil이 아닐 것입니다!
```

[self] resize(size)

그 스토리지의 크기를 인자로 입력된 `size`로 바꿉니다. 새로 추가된 요소들은 초기화되지 않습니다.

이 함수는 self를 리턴합니다. 다음과 같은 것도 허용됩니다.

```
x = torch.DoubleStorage(10):fill(1)
y = torch.DoubleStorage():resize(x:size()):copy(x) -- y는 nil이 아닐 것입니다!
```

[number] size()

그 스토리지에 있는 요소의 개수를 리턴합니다. #와 같습니다.

[self] string(str)

이 함수는 오직 `ByteStorage`와 `CharStorage`에만 사용될 수 있습니다.

이 메소드는 제공된 문자열 `str`의 길이로 그 스토리지의 크기를 바꾸어, `str`의 내용을 그 스토리지로 복사합니다. 그리고 그 스토리지를 리턴합니다. `NULL` 종료 문자는 복사되지 않습니다. 그러나 `str`은 `NULL` 문자를 포함할 수도 있습니다.

```
> x = torch.CharStorage():string("blah blah")
> print(x)
  98
 108
  97
 104
  32
  98
 108
  97
 104
[torch.CharStorage of size 9]
```

[string] string()

이 함수는 오직 `ByteStorage`와 `CharStorage`에만 사용될 수 있습니다.

그 스토리지의 내용이 문자열 하나로 리턴됩니다. 그 문자열에는 `NULL` 문자가 포함될 수도 있습니다.

```
> x = torch.CharStorage():string("blah blah")
> print(x:string())
blah blah
```

참조 횟수를 세는 메소드들

루아는 텐서들의 참조 횟수를 셉니다. 한 객체(C 또는 루아 상태)가 한 텐서를 참조하여 보유할 필요가 있을 때마다, 그에 상응하는 텐서 참조 카운터가 증가됩니다. 그 참조 카운터는 그 텐서가 더 이상 필요 없어질 때 감소됩니다.

이 메소드들은 각별히 조심해서 사용해야 합니다. 일반적으로, 당신이 의도적으로 사용하지 않는 이상, 이 메소드들은 결코 쓰일 일이 없습니다. 참조들은 자동으로 조작되기 때문입니다. 이 메소드들은 스레드 환경에서 유용할 수 있습니다. 이 메소드들은 atomic 연산임을 유념하십시오.

retain()

그 스토리지의 참조 카운터를 증가.

free()

그 스토리지의 참조 카운터를 감소. 만약 카운터가 0이면, 그 스토리지를 free(그 스토리지에 할당된 메모리를 회수)함.

목차

❖ 틀렸거나 보완할 점을 본문에 댓글로 또는 저에게 이메일로 알려 주시면 감사하겠습니다.