토치 유틸리티 함수들

2015년 10월 22일에 한국어로 옮겨짐

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

목차

이 함수들은 모든 토치 패키지에서 클래스들을 생성하고 다루는 데 사용됩니다. 가장 흥미로운 함수는 아마도 `torch.class()`일 것입니다. 이 함수는 사용자가 새 클래스들을 쉽게 생성할 수 있게 합니다. 토치7(*Torch7*) 객체의 클래스 종류를 확인하는 `torch.typename()`도 흥미로울 수도 있습니다.

더 (지식이나 기술이) 앞선 사용자들을 위한 다른 함수들도 있습니다.

[metatable] torch.class(name, [parentName])

`name`이라 불리는 새 토치(`Torch`) 클래스 하나를 만듭니다. 만약 `parentName`이 제공되면, 그 클래스는 `parentName` 메소드들을 상속합니다. 한 클래스는 한 특정 메타테이블을 가진 한 테이블입니다.

만약 `name`이 `package.className` 형태이면, 그 `className` 클래스는 그 특정된 `package`에 추가됩니다. 그 경우, `package`는 유효한(그리고 이미 로드된) 패키지여야 합니다. 만약 `name`이 어떤 .도 포함하지 않으면, 그 클래스는 전역(global) 환경에서 정의됩니다.

만약 `module`이 제공된 테이블이면, 그 클래스는 이 테이블의 `className` 키에서 정의될 것입니다.

하나[또는 두] (메타)테이블들이 리턴됩니다. 이 테이블들은 그 클래스[그리고 만약 있으면 그 부모 클래스]에 의해 제공된 모든 메소드들을 포함합니다. `torch.class()`의 호출 뒤, 당신은 반드시 그 메타테이블을 가득 채워야 합니다.

그 클래스 정의가 끝난 뒤, `name()`을 호출함으로써 한 새로운 클래스 `name`이 구성됩니다. 만약 `lua__init()`이 존재하면, `name()`은 우선 `lua__init()` 메소드를 호출할 것입니다. `name()`의 모든 인자들 또한 전달됩니다.

```
-- 이름 붙이기 편하기 위해
do
   --- 한 클래스 "Foo"를 만듭니다.
```

```
   local Foo = torch.class('Foo')

   --- 초기화 메소드
   function Foo:__init()
      self.contents = 'this is some text'
   end

   --- 한 메소드
   function Foo:print()
      print(self.contents)
   end

   --- 또다른 메소드
   function Foo:bip()
      print('bip')
   end

end

--- 이제 Foo의 한 인스턴스를 만듭니다.
foo = Foo()

--- 그것을 써봅니다.
foo:print()

--- 한 클래스 torch.Bar를 만듭니다.
--- 그것은 Foo를 상속합니다.
do
   local Bar, parent = torch.class('torch.Bar', 'Foo')

   --- 초기화 메소드
   function Bar:__init(stuff)
      --- 우리 자신에 있는 부모 초기화 메소드를 호출합니다.
      parent.__init(self)

      --- 약간의 일을 합니다.
      self.stuff = stuff
   end

   --- 한 새 메소드
   function Bar:boing()
      print('boing!')
   end

   --- 부모의 메소드를 오버라이드 합니다.
   function Bar:print()
      print(self.contents)
      print(self.stuff)
   end
end

--- 한 새 인스턴스를 만들고 그것을 사용합니다.
bar = torch.Bar('ha ha!')
bar:print() -- 메소드 오버라이드
```

```
bar:boing() -- 자식 메소드
bar:bip()   -- 부모의 메소드
```

(지식이나 기술이) 앞선 사용자들을 위해, `torch.class()`는 사실 한 특정 생성자를 가진 `torch.newmetatable()`를 호출한다는 점을 언급하고자 합니다. 그 생성자는 한 루아 테이블을 만들고 그것에 대한 올바른 메타테이블을 설정합니다. 그리고 만약 그 메타테이블에 `lua__init()`이 존재하면 `lua__init()`을 호출합니다. `lua__init()`은 또한 이 클래스의 빈 객체 하나를 생성하는 것이 가능한 한 factory 필드(field) `lua__factory`를 설정합니다.

[string] torch.type(object)

`object`가 메타테이블을 가지는지 확인합니다. 만약 `object`가 메타테이블을 가지고 `object`가 토치(`Torch`) 클래스이면, 그 클래스의 이름을 담고있는 문자열 하나를 리턴합니다. 그렇지 않으면, 이 함수는 그 객체의 루아 `type(object)`를 리턴합니다. `torch.typename()`와 달리, 모든 출력은 문자열입니다.

```
> torch.type(torch.Tensor())
torch.DoubleTensor
> torch.type({})
table
> torch.type(7)
number
```

[string] torch.typename(object)

`object`가 메타테이블을 가지는지 확인합니다. 만약 `object`가 메타테이블을 가지고 `object`가 토치(`Torch`) 클래스이면, 그 클래스의 이름을 담고있는 문자열 하나를 리턴합니다. 다른 경우에 대해서는 `nil`을 리턴합니다.

```
> torch.typename(torch.Tensor())
torch.DoubleTensor
> torch.typename({})

> torch.typename(7)
```

한 토치 클래스는 `torch.class()` 또는 `torch.newmetatable()`로 만들어집니다.

[userdata] torch.typename2id(string)

`string`으로 특정된 한 토치의 클래스 이름이 인자로 입력되면, (그 클래스의 내부 구조를 가리키는 한 `lightuserdata`로 정의된) 상응하는 한 고유한 id를 리턴합니다. 만약 `torch.id()`와 함께 사용되면, 이것은 한 객체에 있는 클래스를 빨리 확인하는 데 유용할 수 있습니다. 이것은 문자열 비교 또한 하지 않습니다.

만약 한 토치 객체가 `string`으로 특정되지 않으면, `nil`을 리턴합니다.

[userdata] torch.id(object)

인자로 입력된 토치7(*Torch7*) 객체의 `class`에 상응하는 고유한 id를 리턴합니다. 그 id는 그 클래스의 내부 구조를 가리키는 한 `lightuserdata`로 정의됩니다.

만약 `object`가 토치 객체가 아니면, `nil`을 리턴합니다.

이것은 `torch.pointer()`에 의해 리턴되는 `object` id와는 다릅니다.

[boolean] isTypeOf(object, typeSpec)

인자로 입력된 `object`가 `typeSpec`으로 특정된 타입의 인스턴스인지 확인합니다. `typeSpec`은 (`string.find` 패턴을 포함하는) 한 문자열 또는 한 토치 클래스를 위한 생성자 객체일 수 있습니다. 이 함수는 클래스 계층을 넘나듭니다. 만약 b가 A의 한 서브클래스인 B의 인스턴스이면, `torch.isTypeOf(b, B)`와 `torch.isTypeOf(b, A)`는 모두 `true`를 리턴할 것입니다.

[table] torch.newmetatable(name, parentName, constructor)

인자로 입력된 문자열 `name`을 가진 한 토치 타입의 새로운 메타테이블 하나를 등록합니다. 그 새 메타테이블이 리턴됩니다.

만약 그 문자열 `parentName`이 `nil`이 아니고 유효한 (`torch.newmetatable()`로 이전에 생성된) 토치 타입이면, 그 상응하는 메타테이블을 그 리턴된 새 메타테이블로의 한 메타테이블로 설정합니다.

만약 인자로 입력된 `constructor` 함수가 `nil`이 아니면, 변수 `name`에 그 인자로 입력된 생성자를 할당합니다. 그 `name`은 `package.className` 형태일 수도 있습니다. 이 경우, 그 `className`은 그 특정된 `package`의 지역(local)일 것입니다. 그 경우, `package`는 반드시 유효하고 이미 로드된 패키지여야 합니다.

[function] torch.factory(name)

토치 클래스 `name`의 factory 함수를 리턴합니다. 만약 그 클래스 이름이 유효하지 않거나 그 클래스가 factory를 가지지 않으면, `nil`을 리턴합니다.

한 토치 클래스는 `torch.class()` 또는 `torch.newmetatable()`로 만들어집니다.

한 factory 함수는 그것의 상응하는 클래스의 한 새로운 (빈) 객체를 리턴할 수 있습니다. 이것은 객체 직렬화에 도움이됩니다.

[table] torch.getmetatable(string)

한 `string`이 인자로 주어지면, `string`에 의해 특정된 그 토치 클래스에 상응하는 한 메타테이블을 리턴합니다. 만약 그 클래스가 존재하지 않으면, `nil`을 리턴합니다.

한 토치 클래스는 `torch.class()` 또는 `torch.newmetatable()`로 만들어집니다.

예:

```
> for k, v in pairs(torch.getmetatable('torch.CharStorage')) do print(k, v) end

__index__       function: 0x1a4ba80
__typename      torch.CharStorage
write           function: 0x1a49cc0
__tostring__    function: 0x1a586e0
__newindex__    function: 0x1a4ba40
string          function: 0x1a4d860
__version       1
read            function: 0x1a4d840
copy            function: 0x1a49c80
__len__         function: 0x1a37440
fill            function: 0x1a375c0
resize          function: 0x1a37580
__index         table: 0x1a4a080
size            function: 0x1a4ba20
```

[boolean] torch.isequal(object1, object2)

만약 인자로 주어진 두 객체가 *루아* 테이블(또는 *Torch7* 객체)이고 그 두 객체가 메모리에서 같은 주소를 가지면, `true`를 리턴합니다. 그 밖의 경우, `false`를 리턴합니다.

한 토치 클래스는 `torch.class()` 또는 `torch.newmetatable()`로 만들어집니다.

[string] torch.getdefaulttensortype()

현재 *Torch7*이 사용중인 기본 텐서 타입을 나타내는 문자열 하나를 리턴합니다.

[table] torch.getenv(function or userdata)

주어진 `function` 또는 `userdata`의 루아 `table` 환경을 리턴합니다. 환경에 대해 더 알고 싶으시면, `lua_setfenv()`와 `lua_getfenv()` 문서를 읽으십시오.

[number] torch.version(object)

주어진 객체의 필드 `lua__version`를 리턴합니다. 이것은 시간에 따른 한 클래스에서의 변이들을 다루는 데 도움이 될 수도 있습니다.

[number] torch.pointer(object)

주어진 `object`의 한 고유한 id(포인터)를 리턴합니다. 그 `object`는 하나의 *Torch7* 객체, 테이블, 스레드, 또는 함수일 수 있습니다.

이것은 `torch.id()`에 의해 리턴되는 `class` id와는 다릅니다.

torch.setdefaulttensortype([typename])

이 시점에서 할당된 모든 텐서들의 기본 텐서 타입을 설정합니다. 유효한 타입들은 다음과 같습니다.

- `torch.ByteTensor`
- `torch.CharTensor`
- `torch.ShortTensor`

- torch.IntTensor
- torch.FloatTensor
- torch.DoubleTensor

torch.setenv(function or userdata, table)

주어진 function 또는 userdata의 루아 환경으로 table을 지정(assign)합니다. 환경에 대해 더 알기 위해, lua_setfenv()와 lua_getfenv()의 문서를 읽으십시오.

[object] torch.setmetatable(table, classname)

classname으로 명명된 토치 객체의 메타테이블로 주어진 table의 메타테이블을 설정합니다. 이 함수는 반드시 주의 깊게 사용되어야 합니다.

[table] torch.getconstructortable(string)

버그 있음. string으로 특정된 토치 클래스의 생성자 테이블을 리턴합니다.

[table] torch.totable(object)

텐서 하나 또는 스토리지 하나를 루아 테이블 하나로 바꿉니다. 또한 메소드로도 사용할 수 있습니다. tensor:totable()과 storage:totable()이 그것입니다. 다차원 텐서들은 소스 텐서와 모양을 맞추어 중첩된(nested) 테이블들로 구성된 집합 하나로 변환됩니다.

```
> print(torch.totable(torch.Tensor({1, 2, 3})))
{
  1 : 1
  2 : 2
  3 : 3
}
```

목차

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