Shared Arrays

`SharedArrays.SharedArray` — Type

```
SharedArray{T}(dims::NTuple; init=false, pids=Int[])
SharedArray{T,N}(...)
```

Construct a `SharedArray` of a bits type `T` and size `dims` across the processes specified by `pids` - all of which have to be on the same host. If `N` is specified by calling `SharedArray{T,N}(dims)`, then `N` must match the length of `dims`.

If `pids` is left unspecified, the shared array will be mapped across all processes on the current host, including the master. But, `localindices` and `indexpids` will only refer to worker processes. This facilitates work distribution code to use workers for actual computation with the master process acting as a driver.

If an `init` function of the type `initfn(S::SharedArray)` is specified, it is called on all the participating workers.

The shared array is valid as long as a reference to the `SharedArray` object exists on the node which created the mapping.

```
SharedArray{T}(filename::AbstractString, dims::NTuple, [offset=0]; mode=nothing,
SharedArray{T,N}(...)
```

Construct a `SharedArray` backed by the file `filename`, with element type `T` (must be a bits type) and size `dims`, across the processes specified by `pids` - all of which have to be on the same host. This file is mmapped into the host memory, with the following consequences:

- The array data must be represented in binary format (e.g., an ASCII format like CSV cannot be supported)
- Any changes you make to the array values (e.g., `A[3] = 0`) will also change the values on disk

If `pids` is left unspecified, the shared array will be mapped across all processes on the current host, including the master. But, `localindices` and `indexpids` will only refer to worker processes. This facilitates work distribution code to use workers for actual computation with the master process acting as a driver.

`mode` must be one of `"r"`, `"r+"`, `"w+"`, or `"a+"`, and defaults to `"r+"` if the file specified by `filename` already exists, or `"w+"` if not. If an `init` function of the type `initfn(S::SharedArray)` is specified, it is called on all the participating workers. You cannot specify an `init` function if the file is not writable.

`offset` allows you to skip the specified number of bytes at the beginning of the file.

`SharedArrays.SharedVector` — Type

```
SharedVector
```

A one-dimensional `SharedArray`.

`SharedArrays.SharedMatrix` — Type

```
SharedMatrix
```

A two-dimensional `SharedArray`.

`Distributed.procs` — Method

```
procs(S::SharedArray)
```

Get the vector of processes mapping the shared array.

`SharedArrays.sdata` — Function

```
sdata(S::SharedArray)
```

Returns the actual `Array` object backing `S`.

`SharedArrays.indexpids` — Function

```
indexpids(S::SharedArray)
```

Returns the current worker's index in the list of workers mapping the `SharedArray` (i.e. in the same list returned by `procs(S)`), or 0 if the `SharedArray` is not mapped locally.

`SharedArrays.localindices` — Function

```
localindices(S::SharedArray)
```

Returns a range describing the "default" indices to be handled by the current process. This range should be interpreted in the sense of linear indexing, i.e., as a sub-range of `1:length(S)`. In multi-process contexts, returns an empty range in the parent process (or any process for which `indexpids` returns 0).

It's worth emphasizing that `localindices` exists purely as a convenience, and you can partition work on the array among workers any way you wish. For a `SharedArray`, all indices should be equally fast for each worker process.

« Serialization Sockets »

Powered by Documenter.jl and the Julia Programming Language.