Real Code

Scoped Values

Sunit

Lets Find Out!?

Will using

Scoped Values

instead of

Thread-Local

variables

make multi threaded

coding?

- ✓ Clear
- ✓ Simple
- ✓ Robust
- ✓ Performant

Sunit

Use Case

- Large and complex java multi threaded applications need to share data between components

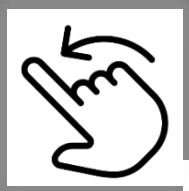
- To have variables having multiple incarnations, one per thread

Use Case

- A better way to share data between components than wiring it into a cascade of untrusted method invocations

- Immutable shared data between parent-child hierarchy

- Bounded lifetime of the shared data



Sunit

ThreadLocal

Thread-local variables

Each thread that accesses it has its own, independently initialized copy of the variable

Typically private static fields in classes

Each thread, as long as alive, holds an implicit reference to its copy of a thread-local variable

ThreadLocal Issues ☹

Unconstrained mutability

Every thread-local variable is mutable

Unbounded lifetime

Developers often forget to remove the value

Expensive inheritance

Child thread has to allocate storage for every inherited thread-local variable as they are mutable

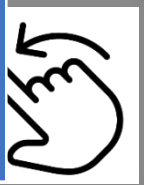
ScopedValue

Allows for safely and efficiently sharing data for a bounded period of execution

Without passing the data as method arguments

A value that is set once and is then available for reading for a bounded period of execution by a thread

Like a thread-local variable, a scoped value has multiple incarnations, one per thread



Sunit

Create

To create a variable use this method

```
Creates a scoped value that is initially unbound for all threads.

Returns: a new ScopedValue

public static <T> ScopedValue<T> newInstance()
    return new ScopedValue<T>();
```

no arg constructor is private

Sunit

Initialize

To initialize the variable use this method

```
Params: key – the ScopedValue key
        value – the value, can be null
Returns: a new Carrier with a single mapping
public static <T> Carrier where(ScopedValue<T>
    return Carrier.of(key, value);
```

Can be chained like

ScopedValue.where(k1, v1).where(k2, v2).run(() -> ...);

Check

To check if a value has been set in the scope of the thread

```
Returns { true} if this scoped value is bound in the current thread.
Returns: { true} if this scoped value is bound in the current thread
public boolean isBound() {
```

if not set, the logic should handle the absence of value.

Sunit

Use Value

To get value that has been set in the scope of the thread

```
Returns the value of the scoped value if bound in the current thread.
Returns: the value of the scoped value if bound in the current thread
Throws: NoSuchElementException – if the scoped value is not bound
@{ ... }
public T get() {
```

Always check for binding to the thread by isBound()

Sunit

Enough of talk and articles....

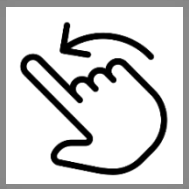
Lets

Refer GitHub Link

See some code in action

Refer GitHub Link

Scoped Values



Sunit

Create

Scoped Value

```
final static ScopedValue<URL> SITE_URL =
        ScopedValue.newInstance();
```

```
final static ThreadLocal<HttpSecurity> SECURE =
        ThreadLocal.withInitial
                (() → HttpSecurity.https);
```

Thread Local

Note: it has withInitial()

Sunit

Initialize

Sets the value

```
case SCOPED_VALUE:
    for (URL url : myFavSitesURLList) {
        ScopedValue.where(SITE_URL, url)
                .run(Utils::fetchURL);
    }
```

Pass the runnable method here

Sunit

Check

To check if a value has been set in the scope of the thread

```
blic static boolean isSiteContentSecure() {
  System.out.println(String.format("For thread
                    " was " +
                    "fetched by %s protocol", Thr
          SITE_URL.isBound() ? SITE_URL.get()
                    : "Unknown", SECURE.get()));
```

if not set, the logic should handle the absence of value.

Use Value

To get value that has been set in the scope of the thread

```
returnStr =  String.format( The site %s
        "due to " +
        "exception %s, details: %s",
 SITE_URL.isBound() ? SITE_URL.get() :
```

Always check for binding to the thread by isBound()

Sunit

A Future represents the result of an asynchronous computation. Methods are provided to check if the computation is com

Takeaway

We discussed Thread Local, its disadvantages.

We discussed the ways to use Scoped Values

Happy Coding !!!

Sunit

Follow

Sunit Ghosh

to get #interesting
and latest #titbits
on #java, #AiMl, #cloud
technologies