ShadowBuilder Service

The **ShadowBuilder service** (see the PropertyShadowBuilder API) is used to build a special, delegating kind of service implementation that, essentially, allows a property of another service to be exposed as its own service.

For example, the tapestry-core module provides a Request property as a shadow of the RequestGlobals service's request property:

Related Articles

- ShadowBuilder Service
- PipelineBuilder Service
- StrategyBuilder Service
- ChainBuilder Service

```
public Request build()
{
  return shadowBuilder.build(requestGlobals, "request", Request.class);
}
```

(shadowBuilder and requestGlobals are injected into the module class instance)

This can be thought of as similar to:

```
public Request build()
{
  return requestGlobals.getRequest();
}
```

However there is a *critical* difference between the two: a shadow property is *re-evaluated on each method invocation*. In the former case, the Request service will always obtain the current value of the request property from the per-thread RequestGlobals service. The second example is more than likely broken, since it will expose whatever value is in the request property of the RequestGlobals *at the time the Request service implementation is realized*.

Notice that in this example, the Request service is a normal singleton. This service can be freely injected into any service throughout the framework or application. Invoking methods on this service will always delegate to the current thread's request. Callers don't have to be aware of this internal delegation; it just happens.

Non-Reflective

When the shadow is created, reflection is used to translate the property name to a method name. This information is used to build a new class (at runtime) that is instantiated as the service implementation.

A typical method is implemented as (approximately):

```
private final RequestGlobals source;

public String getParameter(String name)
{
  return source.getRequest().getParameter(name);
}
```

That is, the shadow implementation holds onto the target object (in the above example, the RequestGlobals service) and invokes a method on it directly, not using reflection, no differently than you would if you wrote the code yourself.