ChainBuilder Service

The **ChainBuilder Service** is a built-in service used to implement of one of the most useful of the *Gang Of Four* design patterns, the chain of responsibility.

With the chain of responsibility design pattern, a complex process is broken down into many individual steps. Each step is a *command* (see command pattern). A key part of this is that the commands are expected to implement some common interface. The commands are also carefully arranged into a specific order.

The process operates by working down the list of commands, and each command is given a chance to operate. In the ChainBuilder service, a command can terminate the process either by throwing an exception, or by returning true.

The return type of the command method does not have to be boolean: For object types, any non-null value short-circuits the process. For numeric type, any non-zero value. For void methods, only throwing an exception will short circuit the process.

Related Articles

- ChainBuilder Service
- ShadowBuilder Service
- IoC Cookbook - Patterns
- PipelineBuilder Service
- StrategyBuilder Service

Often, the command interface consists of a single method. When the command interface has multiple methods, each can be thought of as its own chain.

This is a useful pattern because it makes it very easy to *extend* a given process, simply by providing new commands and specifying where they fit into the overall process. Most often chain of command is combined with an ordered configuration to define what the list of commands are (and in what order they should execute).

ChainBuilder Service

Because this pattern is used so often inside Tapestry, a built-in service exists to create implementations of the pattern as needed. The ChainBuilder service takes care of all the work:

```
public interface ChainBuilder
{
  <T> T build(Class<T> commandInterface, List<T> commands);
}
```

All that generics parameterization just ensures that the command interface matches the items in the list, and confirms that a single instance of the command interface will be returned.

Invoking this method returns an object that encapsulates the chain of command for a particular interface and a particular list of commands implementing that interface.

This can be used inside a service builder method. Nothing says a service builder method just has to instantiate a class; it is only required to return an appropriate object. We can just let the ChainBuilder service create that object.

```
  public static MyChainService build(List<MyChainService> commands,
    @InjectService("ChainBuilder")
    ChainBuilder chainBuilder)
  {
     return chainBuilder.build(MyChainService.class, commands);
  }
```

Here, the behavior of the MyChainService is defined by its configuration: an ordered list of MyChainService commands that are contributed by one or more modules.

Internally, the ChainBuilder creates a new class that implements the service interface. The list of commands is converted into an array, which is used inside the service implementation (for maximum efficiency). Therefore, changing the list after creating the chain instance will not affect the chain instance's behavior.

ChainBuilder will reuse the fabricated class for any number of chains of the same command interface.