Tapestry5HowToReadSymbolsFromPropertiesFile

Motivation

Tapestry already supports configuration symbols using system properties (e.g. -Dkey=value) on the command line and one can contribute to ApplicationDefaults or FactoryDefaults in one's AppModule or you could provide configuration symbols as init parameters in your web.xml. But I neither wanted to supply all my configuration on the command line nor did I want to hard-code it into my application nor did I want to have a web.xml bloated with configuration. In these cases I'd have to restart the web container or redeploy my application in order for configuration changes to take effect. A properties file would be much nicer as I could change that, reload the context and have my configuration changes in effect.

Background

In Tapestry, configuration symbols can be accessed in services, pages and components through SymbolProviders which are contributed to the SymbolSource service. If you look up a symbol, the SymbolSource service queries an ordered list of SymbolProviders for that symbol. The first provider takes precedence over the following SymbolProviders, so it is possible to overwrite configuration symbols in some other place. For example the built-in ApplicationDefaults SymbolProvider takes precedence over FactoryDefaults so that you can contribute to the ApplicationDefaults service and thus can overwrite values set in FactoryDefaults. Similarily system properties, i.e. properties supplied on the command line with the -Dkey=value syntax, overwrite both ApplicationDefaults and FactoryDefaults.

Tapestry ships with 4 SymbolProviders of which 2 are in use by default:

- SingleKeySmbolProvider takes 2 constructor arguments (a key and a value String) and maps a single key to a value
- ServletContextSymbolProvider makes init parameters in your web.xml accessible as symbols
- MapSymbolProvider internally stores key-value pairs in a map. FactoryDefaults and ApplicationDefaults are MapSymbolProviders.
- SystemPropertiesSymbolProvider makes system properties (coming from your JVM) available as symbols

Implementation

I wanted to have a SymbolProvider that looks for configuration symbols in a properties file either located somewhere on the filesystem or in the classpath. Additionally symbol lookups should be case insensitive (as everywhere else in Tapestry). I therefore wrote a PropertiesFileSymbolProvider that implements SymbolProvider:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.Map;
import java.util.Properties;

import org.apache.tapestry5.ioc.services.SymbolProvider;
import org.apache.tapestry5.ioc.util.CaseInsensitiveMap;
import org.slf4j.Logger;

public class PropertiesFileSymbolProvider implements SymbolProvider
{

    private final Map<String, String> propertiesMap = new CaseInsensitiveMap<String>();

    /**
     * Instantiate a new PropertiesFileSymbolProvider using a given resource name
     *
     * @param logger the logger to log error messages to
     * @param resourceName the name of the resource to load
     * @param classPath whether to look on the classpath or filesystem
     */
    public PropertiesFileSymbolProvider(Logger logger, String resourceName, boolean classPath)
    {
        try
        {
            InputStream in;
```

```
        if (classPath)
        {
                in = this.getClass().getClassLoader().getResourceAsStream(resourceName);
                if(in == null) {
                        in = ClassLoader.getSystemResourceAsStream(resourceName);
                }

            // ClassLoader.getSystemResourceAsStream() returns null if
            // the resource cannot be found on the classpath
            if (in == null)
                throw new FileNotFoundException();
        }
        else
            in = new FileInputStream(resourceName);

        initialize(logger, in);

    } catch (FileNotFoundException e)
    {
        String msg = "Could not find '" + resourceName + "'";

        logger.error(msg);

        throw new IllegalArgumentException(msg, e);
    }
}

/**
 * Instantiate a PropertiesFileSymbolProvider using a given InputStream
 *
 * @param logger the logger
 * @param in an InputStream representing the resource
 */
public PropertiesFileSymbolProvider(Logger logger, InputStream in)
{
    initialize(logger, in);
}

/**
 * Instantiate a PropertiesFileSymbolProvider from a given URL.
 *
 * @param logger the logger
 * @param url an URL to open
 */
public PropertiesFileSymbolProvider(Logger logger, URL url)
{
    try
    {
        initialize(logger, url.openStream());
    } catch (IOException e)
    {
        String msg = "IOException while opening URL '" + url + "': " + e.getMessage();

        logger.error(msg);

        throw new IllegalArgumentException(msg, e);
    }
}

private void initialize(Logger logger, InputStream in)
{
    Properties properties = new Properties();

    try
    {
        properties.load(in);
        propertiesMap.putAll(properties);
    } catch (IOException e)
    {
        String msg = "IOEception while loading properties: " + e.getMessage();
```

```
            logger.error(msg);

            throw new IllegalArgumentException(msg, e);
        }
    }

    public String valueForSymbol(String arg0)
    {
        return propertiesMap.get(arg0);
    }
}
```

Tying it together

Next you have to define one (or more) instances of PropertiesFileSymbolProvider as a service in your AppModule. In the following example I'm defining two services: ClasspathPropertiesFileSymbolProvider looks for the file test.properties in the classpath and FilesystemPropertiesFileSymbolProvider looks for the file test2.properties somewhere in the file system. The constructor for the PropertiesFileSymbolProvider takes 3 arguments: the first is a Logger object which is provided by Tapestry. The second is the name of the resource to look for and the third indicates whether to look in the classpath (true) or on the file system (false).

```
// make configuration from 'test.properties' on the classpath available as symbols
public PropertiesFileSymbolProvider buildClasspathPropertiesFileSymbolProvider(Logger logger)
{
    return new PropertiesFileSymbolProvider(logger, "test.properties", true);
}

// make configuration from 'test2.properties' on the filesystem available as symbols
public PropertiesFileSymbolProvider buildFilesystemPropertiesFileSymbolProvider(Logger logger)
{
    return new PropertiesFileSymbolProvider(logger, "src/main/webapp/WEB-INF/test2.properties", false);
}
```

The last step is to contribute those two services to the SymbolSource service so that we can access our configuration symbols from our services, pages and components.

```
public static void contributeSymbolSource(OrderedConfiguration<SymbolProvider> configuration,

        @InjectService("ClasspathPropertiesFileSymbolProvider")

        SymbolProvider classpathPropertiesFileSymbolProvider,

        @InjectService("FilesystemPropertiesFileSymbolProvider")

        SymbolProvider filesystemPropertiesFileSymbolProvider)
{
    configuration.add("ClasspathPropertiesFile", classpathPropertiesFileSymbolProvider, "after:
SystemProperties", "before:ApplicationDefaults");

    configuration.add("FilesystemPropertiesFile", filesystemPropertiesFileSymbolProvider, "after:
ClasspathPropertiesFile", "before:ApplicationDefaults");
}
```

The order of your SymbolProviders is important (before: and after: syntax). The SymbolProviders for the default SymbolSource service are ordered System Properties, ApplicationDefaults, FactoryDefaults. Thus system properties override contributions to the ApplicationDefaults service which in turn override contributions to the FactoryDefaults service. If you don't want your configuration symbols being overriden by any of the pre-defined SymbolProviders you have to make sure to insert them in the correct place. In my case I wanted the properties from the file on the classpath to be overridable only by system properties and the properties from the file on the filesystem be overridable by system properties and the file on the classpath. Thus my order is SystemProperties, ClasspathPropertiesFile, FilesystemPropertiesFile, ApplicationDefaults, FactoryDefaults.

Using symbols in pages, components and services

To use configuration symbols in your pages and components you just have to inject them. A simple symbol value can be injected with:

```
@Inject @Symbol(value="foo")
private String foo;
```

The foo string will then contain the value of the foo configuration symbol. If for example your properties file contained a line `foo=bar`, then the foo string would be "bar". If you want to expand a symbol in a string, you have to use the Value annotation:

```
@Inject @Value(value="${root}/something/${somedir}")
private String directory
```

If you had a properties file whith the lines `root=/var/lib` and `somedir=foo` than the directory string would contain "/var/lib/something/foo". Tapestry tries to find a symbol whose name matches the content inside $ and expands the string whith that symbol's value.

For information on how to use symbols inside your services, please refer to http://tapestry.apache.org/tapestry5/tapestry-ioc/symbols.html.