Injection

Injection is Tapestry's way of making a dependency – such as a resource, asset, component, block or service – available in a page, component, mixin or service class.

Injection is a key concept in Tapestry, and it is used in several different but related ways, listed below.

Related Articles

- Injection FAQ
- Injection in Detail
- Environmental Services
- Injection

Injection in Tapestry IOC Services

Main Article: Injection in Detail

The Tapestry IoC container makes use of injection primarily through constructors and via parameters to service builder methods.

Injection in Component Classes

For components, however, Tapestry takes a completely different tack: injection directly into component fields.

The @Inject annotation is used to identify fields that will contain injected services and other resources.

Tapestry allows for two kinds of injection:

- **Default injection**, where Tapestry determines the object to inject into the field based on its type.
- **Explicit injection**, where the particular service to be injected is specified.

In both cases, the field is transformed into a read only value. As elsewhere in Tapestry, this transformation occurs at runtime (which is very important in terms of being able to test your components). Attempting to update an injected field will result in a runtime exception.

In addition, there are a few special cases that are triggered by specific field types, or additional annotations, in addition, to @Inject, on a field.

Block Injection

For field type Block, the value of the Inject annotation is the id of the <t:block> element within the component's template. Normally, the id of the block is determined from the field name (after stripping out any leading "_" and "$" characters):

```
@Inject
private Block foo;
```

Where that is not appropriate, an @Id annotation can be supplied:

```
@Inject
@Id("bar")
private Block barBlock;
```

The first injection will inject the Block with id "foo" (as always, case is ignored). The second injection will inject the Block with id "bar".

Resource Injection

For a particular set of field types, Tapestry will inject a *resource* related to the component, such as its Locale.

A very common example occurs when a component needs access to its resources. The component can define a field of the appropriate type and use the `@Inject` annotation without a value:

```
@Inject
private ComponentResources resources;
```

Tapestry uses the type of the field, ComponentResources, to determine what to inject into this field.

The following types are supported for resources injection:

- **java.lang.String** – The complete id of the component, which incorporates the complete class name of the containing page and the nested id of the component within the page.
- **java.util.Locale** – The locale for the component (all components within a page use the same locale).
- **org.slf4j.Logger** – A Logger instance configured for the component, based on the component's class name. SLF4J is a wrapper around Log4J or other logging toolkits.

- **org.apache.tapestry5.ComponentResources** – The resources for the component, often used to generate links related to the component.
- **org.apache.tapestry5.ioc.Messages** – The component message catalog for the component, from which localized messages can be generated.

Asset Injection

Main Article: Assets

When the @Path annotation is also present, then the injected value (relative to the component) will be a localized asset.

```
@Inject
@Path("context:images/top_banner.png")
private Asset banner;
```

Symbols in the annotation value are expanded.

Service Injection

Here, a custom EmployeeService service is injected, but any custom or built-in service may be injected in the same way.

```
@Inject
private EmployeeService employeeService;
```

A large number of services are provided by Tapestry. See the following packages:

- Core Services
- AJAX Services
- Assets Services
- Dynamic Component Services
- JavaScript Services
- Link Transformation Services
- Message Services
- Component Metadata Services
- Page Loading Services
- Security Services
- Template Services
- Class Transformation Services
- Tapestry IOC Services
- Tapestry IOC Cron Services
- Kaptcha Services
- File Upload Services

Explicit Service Injection

Here, a specific object is requested. A @Service annotation is used to identify the service name.

```
@Inject
@Service("Request")
private Request request;
```

This is generally not necessary; you should always be able to identify the service to be injected just by type, not by explicit id. Explicit ids have the disadvantage of not being refactoring-safe: this won't happen with the Request service, but perhaps in your own code ... if you rename the service interface and rename the service id to match, your existing injections using the explicit service id will break.

Default Injection

When the type and/or other annotations are not sufficient to identify the object or service to inject, Tapestry falls back on two remaining steps. It assumes that the field type will be used to identify a service, by the service interface.

First, the object provider created by the Alias service is consulted. This object provider is used to disambiguate injections when there is more than one service that implements the same service interface.

Second, a search for a unique service that implements the interface occurs. This will fail if either there are no services that implement the interface, or there is more than one. In the latter case, you must disambiguate, either with a contribution to the Alias service, or by explicitly identifying the service with the @Service annotation.

Defining New Injection Logic

Anonymous injection is controlled by the InjectionProvider service. The configuration for this service is a chain of command for handling component injections.