Web Method

One of many ways to handle an HTTP request is by writing a method on your URL-bound classes. In Stapler-speak, these methods are called "web methods".

Web methods are public instance methods that has the name that starts with "do", for example:

```
class MyRootAction {
  ...

  @RequirePOST
  public HttpResponse doStart() {
    start();
    return HttpResponses.redirectTo(".");
  }
}
```

Such a method would handle an HTTP request sent to `/.../start` (again, see the reference for the exact routing rules)

Parameters

Web methods can define parameters. When Stapler invokes your web method, it needs your help to figure out what you expect in those parameters, and there are several ways to do this.

Firstly, if the type of a parameter is one of the following "well-recognized" types, Stapler would instantly know what to do:

- StaplerRequest/`HttpServletRequest` — the request object
- StaplerResponse/`HttpServletResponse` — the response object

Secondly, you can place parameter injection annotations to instruct Jenkins what you want to see in that parameter. Unlike typical Java programming, parameter names are significant.

- @**Header**: requests that the value of the request HTTP header be injected.

  ```
  public HttpResponse doStart(@Header String referer) { ... }
  ```

- @**QueryParameter**: requests that the value of the request parameter be injected. This includes a submitted form and a query parameter.
- @**AncestorInPath**: Stapler will call `StaplerRequest.findAncestor(Class)` and inject the obtained object. This object is the nearest "ancestor" of the specified type in the current URL.

> Parameter injection annotations are extensible. One can define a custom injection parameter by using the @InjectedParameter] meta annotation, which is how all these annotations are defined.

The type of the injected parameter can be anything, and Apache Commons Beanutils is used to convert the incoming value into the appropriate Java type your method requests.

Return value and exception

If a web method returns, either normally through the return statement or abnormally through a thrown exception, Stapler checks if the return value / exception implements HttpResponse interface. If so, this object is expected to render a response via its `generateResponse` method.

Jenkins defines a number of high-level HTTP response classes that implements this interface, such as AutoCompletionCandidates, Context Menu, ListBoxModel, FormValidation, TrendChart, etc.

There's also HttpResponses that provides a number of static methods that help you create typical HTTP responses, such as redirect, errors, serving static files, etc.

Interceptor Annotations

Web methods can have "interceptor annotations", which decorates your web method by adding additional processing before and after your method gets invoked. Interceptor annotations to web methods are like servlet fileters to servlets.

Jenkins defines several built-in interceptor annotations:

- @**RequirePOST**: aborts with "400 bad request" if the request is not POST. This is how we protect endpoints that can update states in Jenkins, with crumb.
- @**RespondSuccess**: used on a web method that returns "void" so that when the method returns normally "200 success" will be returned as a response.

Interceptor annotations are extensible through InterceptorAnnotation.

> Parameter injection, return value/exception response rendering, and interceptor annotations help you reduce the HTTP dependency in your model objects. This tends to make code easier and more easily testable.

@WebMethod

Normally, a name of a web method determines how the request is routed. For example, `doEatPizza()` would be mapped to `.../eatPizza`. But you can explicitly specify the URL name by using WebMethod annotation. For example, the following code maps `.../abc` and `.../xyz.xml` to the same method.

```
@WebMethod("abc","xyz.xml")
HttpResponse doMonkey(...) {
  ...
}
```