Architecture

Jenkins is primarily a set of Java classes that model the concepts of a build system in a straight-forward fashion (and if you are using Jenkins, you've seen most of those already). There are classes like Project, Build, that represents what the name says. The root of this object model is Hudson, and all the other model objects are reachable from here.

Then there are interfaces and classes that model code that performs a part of a build, such as SCM for accessing source code control system, Ant for performing an Ant-based build, Mailer for sending out e-mail notifications.

Stapler

Those Jenkins classes are bound to URLs by using Stapler. The singleton `Hudson` instance is bound to the context root (e.g. "/") URL, and the rest of the objects are bound according to their reachability from this root object. Stapler uses reflection to recursively determine how to process any given URL. A few examples of how the URL `/foo/bar` could be processed:

- A `getFoo(String)` is defined on the `Jenkins` object, and Stapler passes `bar` as a parameter. The object returned has a method called `doIndex(…)` that gets called and renders the response.
- `getFoo()` or `doFoo()` are defined. They return an object that has a `getBar` or `doBar` method. The object returned from that has an associated `index.jelly` or `index.groovy` view.
- `getFoo()` or `doFoo()` are defined. The object return has a view named `bar.jelly` or `bar.groovy` defined.

A number of additional ways to handle requests exist, but these are the most common.

As a real-world example, there's the `Jenkins#getJob(String)` method. So the URL `/job/foo/` will be bound to the object returned by `Jenkins#getJob("foo")` (which would be a `Project` named `foo`.

Additionally, objects can implement one of two interfaces to further control how Stapler processes URLs:

- `StaplerProxy` allows delegating the processing of a URL to another object. So, for `/foo/bar`, if `getFoo()` returns an object `x` that implements `StaplerProxy`'s `getTarget()` method, Stapler will call `x.getTarget()` and continue using that to process the rest of the URL (`bar`). This has the **highest priority** among all possible URL processing options.
- `StaplerFallback` allows delegating the processing of a URL to another object, similar to `StaplerProxy`, but has the **lowest priority** among all possible URL processing options.

For more information on how Stapler binds (or staples) a Java Object Model to a URL hierarchy, see the Stapler Reference Documentation

Views

Jenkins' model objects have multiple "views" that are used to render HTML pages about each object. Jenkins uses Jelly as the view technology (which is somewhat similar to JSP+JSTL) Views are like methods and each of them work against a particular class. So the views are organized according to classes that they belong to, just like methods are organized according to classes that they belong to. Again, see the Stapler project for more about how this works.

Taglibs

Jenkins defines a few Jelly tag libraries to encourage views to have the common theme. For example, one of them defines tags that form the basic page layout of Jenkins, another one defines tags that are used in the configuration pages, and so on.

Persistence

Jenkins uses the file system to store its data. Directories are created inside `$JENKINS_HOME` in a way that models the object model structure. Some data, like console output, are stored just as plain text file, some are stored as Java property files. But the majority of the structured data, such as how a project is configured, or various records of the build, are persisted by using XStream.

This allows object state to be persisted relatively easily (including those from plugins), but one must pay attention to what's serialized in XML, and take measures to preserve backward compatibility. For example, in various parts of Jenkins you see the `transient` keyword (which instructs XStream not to bind the field to XML), fields left strictly for backward compatibility, or re-construction of in-memory data structure after data is loaded.

More persistence topics

- Evolving data structure
- Hint on retaining backward compatibility
- XStream Tips

Plugins

Jenkins' object model is extensible (for example, one can define additional SCM implementations, provided that they implement certain interfaces), and it supports the notion of "plugins," which can plug into those extensibility points and extend the capabilities of Jenkins.

Jenkins loads each plugin into a separate class loader to avoid conflicts. Plugins can then participate to the system activities just like other Jenkins built-in classes do. They can participate in XStream-based persistence, they can provide "views" by Jelly, they can provide static resources like images, and from users, everything works seamlessly --- there's no distinction between functionalities that are built-in vs those from plugins.