Comparing Struts 1 and 2

Feature	Struts 1	Struts 2
Action classes	Struts 1 requires Action classes to extend an abstract base class. A common problem in Struts 1 is programming to abstract classes instead of interfaces.	An Struts 2 Action *may* implement an `Action` interface, along with other interfaces to enable optional and custom services. Struts 2 provides a base ActionSupport class to implement commonly used interfaces. Albeit, the Action interface is **not** required. Any POJO object with a `execute` signature can be used as an Struts 2 Action object.
Threading Model	Struts 1 Actions are singletons and must be thread-safe since there will only be one instance of a class to handle all requests for that Action. The singleton strategy places restrictions on what can be done with Struts 1 Actions and requires extra care to develop. Action resources must be thread-safe or synchronized.	Struts 2 Action objects are instantiated for each request, so there are no thread-safety issues. (In practice, servlet containers generate many throw-away objects per request, and one more object does not impose a performance penalty or impact garbage collection.)
Servlet Dependency	Struts 1 Actions have dependencies on the servlet API since the HttpServletRequest and HttpServletResponse is passed to the `execute` method when an Action is invoked.	Struts 2 Actions are not coupled to a container. Most often the servlet contexts are represented as simple Maps, allowing Actions to be tested in isolation. Struts 2 Actions can still access the original request and response, if required. However, other architectural elements reduce or eliminate the need to access the HttpServetRequest or HttpServletResponse directly.
Testability	A major hurdle to testing Struts 1 Actions is that the `execute` method exposes the Servlet API. A third-party extension, Struts TestCase, offers a set of mock object for Struts 1.	Struts 2 Actions can be tested by instantiating the Action, setting properties, and invoking methods. Dependency Injection support also makes testing simpler.
Harvesting Input	Struts 1 uses an ActionForm object to capture input. Like Actions, all ActionForms must extend a base class. Since other JavaBeans cannot be used as ActionForms, developers often create redundant classes to capture input. DynaBeans can used as an alternative to creating conventional ActionForm classes, but, here too, developers may be redescribing existing JavaBeans.	Struts 2 uses Action properties as input properties, eliminating the need for a second input object. Input properties may be rich object types which may have their own properties. The Action properties can be accessed from the web page via the taglibs. Struts 2 also supports the ActionForm pattern, as well as POJO form objects and POJO Actions. Rich object types, including business or domain objects, can be used as input/output objects. The ModelDriven feature simplifies taglb references to POJO input objects.
Expression Language	Struts 1 integrates with JSTL, so it uses the JSTL EL. The EL has basic object graph traversal, but relatively weak collection and indexed property support.	Struts 2 can use JSTL, but the framework also supports a more powerful and flexible expression language called "Object Graph Notation Language" (OGNL).
Binding values into views	Struts 1 uses the standard JSP mechanism for binding objects into the page context for access.	Struts 2 uses a "ValueStack" technology so that the taglibs can access values without coupling your view to the object type it is rendering. The ValueStack strategy allows reuse of views across a range of types which may have the same property name but different property types.
Type Conversion	Struts 1 ActionForm properties are usually all Strings. Struts 1 uses Commons-Beanutils for type conversion. Converters are per-class, and not configurable per instance.	Struts 2 uses OGNL for type conversion. The framework includes converters for basic and common object types and primitives.
Validation	Struts 1 supports manual validation via a `validate` method on the ActionForm, or through an extension to the Commons Validator. Classes can have different validation contexts for the same class, but cannot chain to validations on sub-objects.	Struts 2 supports manual validation via the `validate` method and the XWork Validation framework. The Xwork Validation Framework supports chaining validation into sub-properties using the validations defined for the properties class type and the validation context.
Control Of Action Execution	Struts 1 supports separate Request Processors (lifecycles) for each module, but all the Actions in the module must share the same lifecycle.	Struts 2 supports creating different lifecycles on a per Action basis via Interceptor Stacks. Custom stacks can be created and used with different Actions, as needed.

See Also

- Matt Raible wrote (mid 2005) an interesting whitepaper where he compared various web frameworks. You can view the PDF here:
  - https://equinox.dev.java.net/framework-comparison/WebFrameworks.pdf
  - http://www.virtuas.com/files/osl-jwf-01.pdf

Next: Migration Strategies