Result Configuration

When an action class method completes, it returns a String. The value of the String is used to select a result element. An action mapping will often have a set of results representing different possible outcomes. A standard set of result tokens are defined by the `ActionSupport` base class.

Predefined result names

```
String SUCCESS = "success";
String NONE    = "none";
String ERROR   = "error";
String INPUT   = "input";
String LOGIN   = "login";
```

Of course, applications can define other result tokens to match specific cases.

Returning `ActionSupport.NONE` (or `null`) from an action class method causes the results processing to be skipped. This is useful if the action fully handles the result processing such as writing directly to the HttpServletResponse OutputStream.

Result Elements

The result element has two jobs. First, it provides a logical name. An `Action` can pass back a token like "success" or "error" without knowing any other implementation details. Second, the result element provides a result type. Most results simply forward to a server page or template, but other Result Types can be used to do more interesting things.

Intelligent Defaults

Each package may set a default result type to be used if none is specified in a result element. If one package extends another, the "child" package can set its own default result, or inherit one from the parent.

Setting a default Result Type

```
<result-types>
   <result-type name="dispatcher" default="true"
                class="org.apache.struts2.dispatcher.ServletDispatcherResult" />
</result-types>
```

If a `type` attribute is not specified, the framework will use the default `dispatcher` type, which forwards to another web resource. If the resource is a JavaServer Page, then the container will render it, using its JSP engine.

Likewise if the `name` attribute is not specified, the framework will give it the name "success".

Using these intelligent defaults, the most often used result types also become the simplest.

Result element without defaults

```
<result name="success" type="dispatcher">
    <param name="location">/ThankYou.jsp</param>
</result>
```

A Result element using some defaults

```
<result>
    <param name="location">/ThankYou.jsp</param>
</result>
```

The `param` tag sets a property on the Result object. The most commonly-set property is `location`, which usually specifies the path to a web resources. The `param` attribute is another intelligent default.

Result element using more defaults

```
<result>/ThankYou.jsp</result>
```

Mixing results with intelligent defaults with other results makes it easier to see the "critical path".

Multiple Results

```
<action name="Hello">
    <result>/hello/Result.jsp</result>
    <result name="error">/hello/Error.jsp</result>
    <result name="input">/hello/Input.jsp</result>
</action>
```

A special 'other' result can be configured by adding a result with name="*". This result will only be selected if no result is found with a matching name.

'*' Other Result

```
<action name="Hello">
    <result>/hello/Result.jsp</result>
    <result name="error">/hello/Error.jsp</result>
    <result name="input">/hello/Input.jsp</result>
    <result name="*">/hello/Other.jsp</result>
</action>
```

The name="*" is **not** a wildcard pattern, it is a special name that is only selected if an exact match is not found.

In most cases if an action returns an unrecognized result name this would be a programming error and should be fixed.

Multiple names

It is possible to define multiple names for the same result:

```
<action name="save">
    <result>success.jsp</result>
    <result name="error, input">input-form.jsp</result>
</action>
```

Such functionality was added in Struts 2.5

Global Results

Most often, results are nested with the action element. But some results apply to multiple actions. In a secure application, a client might try to access a page without being authorized, and many actions may need access to a "logon" result.

If actions need to share results, a set of global results can be defined for each package. The framework will first look for a local result nested in the action. If a local match is not found, then the global results are checked.

Defining global results

```
<global-results>
    <result name="error">/Error.jsp</result>
    <result name="invalid.token">/Error.jsp</result>
    <result name="login" type="redirectAction">Logon!input</result>
</global-results>
```

For more about results, see Result Types.

Dynamic Results

A result may not be known until execution time. Consider the implementation of a state-machine-based execution flow; the next state might depend on any combination of form input elements, session attributes, user roles, moon phase, etc. In other words, determining the next action, input page, etc. may not be known at configuration time.

Result values may be retrieved from its corresponding Action implementation by using EL expressions that access the Action's properties, just like the Struts 2 tag libraries. So given the following Action fragment:

FragmentAction implementation

```
private String nextAction;

public String getNextAction() {
    return nextAction;
}
```

you might define a result like this:

FragmentAction configuration

```
<action name="fragment" class="FragmentAction">
    <result name="next" type="redirectAction">${nextAction}</result>
</action>
```

If a `FragmentAction` method returns "next" the actual *value* of that result will be whatever is in `FragmentAction`'s `nextAction` property. So `nextAction` may be computed based on whatever state information necessary then passed at runtime to "next"'s `redirectAction`.

See Parameters in configuration results for an expanded discussion.

Returning Result Objects

Instead of configuring results and returning the name, it is possible to return a result object:

```
public Result runAction() {
        ServletDispatcherResult result = new ServletDispatcherResult();
        result.setLocation("input-form.jsp");
        return result;
}
```