How *lift* processes HTTP requests:

lift implements a Servlet. When the Servlet's "service" method gets called, *lift* does the following:

- If the request is a GET, *lift* looks to see if the file name suffix is "png", "js", "css", "jpg", "ico", "gif", "tiff", "jpeg" and that the file exists. If so, *lift* serves the static file.

- *lift* iterates over functions added with Servlet.appendEarly This allows, for example, setting the req.setCharacterEncoding("UTF-8")

- *lift* recursively pattern matches rewrite rules defined with Servlet.`addRewriteBefore` and Servlet.`addRewriteAfter` The rewrites are partial functions (case statements) that allow you to pattern match the URI, a list of the path elements, the HTTP request, and the request type and return a re-written URI, path, and parameters. This pattern is recursively applied. So:
  `case (_, "foo":: bar :: _, _, _) => ("/foo", List("foo"), TreeMap("param" -> bar))`
  Will find all requests that start with "foo" and turn the second part of the path into a parameter named "param"

- *lift* then attempts to match patterns with patterns defined with Servlet.`addDispatchBefore` and `Servlet.addDispatchAfter` If any patterns are matched, *lift* creates a thread-local "S" state and invokes the function. The response from the function is matched in the following way:

  - Response instance – return it

  - NodeSeq – treat as XHTML, fix the references (img, a, form, etc. tags) and generate a Response with type "text/html" and code 200

  - XmlResponse – create a Response with type "text/xml" and code 200

  - Otherwise return a "404 Not Found"

  - Return the response

- Find/create a Session Actor for the current Servlet session

- Drain any pending items from the current thread's inbox

- Set the timeout for the request (10 seconds for normal, 100 seconds for AJAX request)

- Send an "AskSessionToRender" message to the Session and wait for the maximum time for a response. If no valid response is returned, response with a 404. If the session sends a "Response" process it and return it as the HTTP response.

The Session handles an “AskSessionToRender” in the following way:

- Set up “S” with the Request and any notices posted by the prior page service
- Process each of the parameters. The parameter name is matched to the table of Functions. If the parameter name matches a function, the function is invoked with the list of parameter values.
- The template for the given request is located:
  - Path elements starting with “.”, “_” or containing “-hidden” are discarded.
  - The servlet context is searched for a file that matches the request URI (as modified by rewrite) with the following attempted suffixes: "", "html", "xhtml", "htm"
  - If the template is not found, class names are constructed based on the first element in the path:
    - packages defined by Helpers.addToPackages plus
    - lift.app.view
    - e.g. /foo_bar/baz -> myapp.view.foo_bar, myapp.view.FooBar, lift.app.view.foo_bar, lift.app.view.FooBar
  - If a class is found, the second element in the path (“baz” in the above example) is located and invoked. If the method returns a NodeSeq, that is used as the template.
  - If the first path element is not supplied, “default_template” is used. If the second path element is not supplied, “render” is used.
  - The code method for template creation may not be supported in the future.
- Once a template is located and loaded, the template is processed by locating and substituting the following tags:
  - <lift:surround [with=”template-name”] [at=”location”]> A template is located based on the value of with attribute (defaults to /templates-hidden/Default) and places the child nodes at the location of the <lift:bind name=”at”/> If the at attribute is not specified, “main” is assumed.
  - <lift:embed what=”template-to-embed”/> locate a template (using the template location rules, but relaxing the restriction on “.”, “_”, and “-hidden”) and place the template at the current location
  - <lift:snippet type=”name:method”/> A snippet of code is located and executed (note that snippets are not processed for AJAX requests.)

- The class name for the snippet is defined by "name". *lift* loos for a class with that name (both as-is, and camel cased) in "lift.app.snippet", "net.liftweb.builtin.snippet" as well as any packages specified in Helpers.addToPackages (with .snippet appended). Once the class is located...

- The method is invoked. If no "method" is defined, "render" is used as the default. If a method with a single scala.xml.Group exists (e.g., render(xhtml: Group)) that method is invoked with the child node of the snippet tag. Otherwise a no-parameter method is invoked.

- Once the template has been rendered, the session looks for any <lift:controller> tags. If none exist, the Session "fixes" the XHTML (fixes references in img, form, etc. tags) and sends the XHTML back to the servlet to be served.

- If a RedirectException is thrown during the above processing, a Redirect is sent back to the servlet.

- If the template contains a Controller, a Page Actor is located/created for the URI and the Page is sent an AskRenderPage message

-