Spinneret Documentation

Release 0.1.2

Jonathan Jacobs

March 05, 2015

Contents

Release v0.1.2.

CHAPTER 1

What is this?

Spinneret is a collection of higher-level utility classes and functions to make writing complex Twisted Web applications far simpler, it is designed to easily integrate with existing Twisted Web projects for things like the improved `IResource` implementations.

CHAPTER 2

Why is this one different?

While I think Klein is a fantastic library—and a terrific source of inspiration—there are some fundamental aspects I disagree with, not to mention Spinneret includes a number of other utilities to make other aspects of Twisted Web development easier. However, using Spinneret to enhance your Klein (or any other Twisted Web) application is not only trivial and perfectly reasonable but also encouraged!

CHAPTER 3

Installation

```
$ pip install txspinneret
```

Or to install the latest development version:

```
$ pip install git+git://github.com/jonathanj/txspinneret
```

CHAPTER 4

Documentation

4.1 URL Routing

Often it is desirable to describe a resource hierarchy by matching URL segments in the request, this is commonly referred to as "URL routing".

A Python-based Domain Specific Language is used to specify and match routing paths, string literal components are matched for structure while plain callable components match segment values and are stored by name for use in the handler, assuming all the components match; this makes it trivial to create new functions to match path components.

In order to promote better reuse of resources—by composing and nesting them—it is only possible to specify relative routes.

4.1.1 Router basics

`Router.route` will match a URL route exactly, meaning that every route component must match the respective segment in the URL path; eg. `route('foo')` will only match a relative URL path of exactly one segment that must be the string `foo`.

`Router.subroute` will partially match a URL route, meaning that once every route component has matched its respective segment in the URL path the route will be a match, regardless of whether there are URL path segments left over. This is useful for the case where you wish to match enough of the URL to know that you should delegate to another resource.

A route handler may return any of the values that `ISpinneretResource.locateChild` supports.

Routes are intended to be used as method decorators and may be stacked to have multiple routes serviced by the same handler.

Call `Router.resource` to produce an `IResource` suitable for composing with other parts of Spinneret or Twisted Web.

4.1.2 Special routes

There are two routes—particularly in the case of nested routers—that may not be obvious at first: The null root and the empty route.

Assuming we had the following hierarchy:

```
class Root(object):
    router = Router()

    @router.subroute('bar')
    def bar(self, request, params):
        return SubRoot().router.resource()


class SubRoot(object):
    router = Router()
```

In the case of a request for the resource at `/bar/` we can match that by declaring a route in `SubRoot` with `@router.route('/')` or `@router.route('')` (the empty route.) If the request was instead for the resource at `/bar` (note the absence of the trailing `/`) we can match that with `@router.route()` (the null route.)

4.1.3 Matcher basics

`txspinneret.route` contains some basic matchers such as `Any` (which is a synonym for `Text`) and `Integer`. These matchers are simple factory functions that take some parameters and produce a `callable` that takes the `IRequest` and the segment being matched, as `bytes`, returning a 2-`tuple` of the parameter name and the processed matched value (or `None` if there is no match.) Writing your own matchers to suit your needs is encouraged.

4.1.4 Reducing router resource boilerplate

When using routers as resources (such as when nesting routers) it becomes common to write `return SomeRouter(...).router.resource()`. The `routedResource` decorator can be used to wrap a router class into a callable that returns a resource.

4.1.5 An example router

```
from collections import namedtuple
from twisted.web.static import Data
from txspinneret.route import Router, Any, routedResource

@routedResource
class UserResource(object):
    router = Router()

    def __init__(self, user):
        self.user = user

    def getFriend(self, name):
        return self.user.friends[name]

    @router.route('/')
    def name(self, request, params):
        return Data(self.user.name, 'text/plain')

    @router.subroute('friend', Any('name'))
    def friend(self, request, params):
        return UserResource(self.getFriend(params['name']))

def start():
    User = namedtuple(b'User', ['name', 'friends'])
    bob = User('bob', {})
    chuck = User('chuck', {'bob': bob})
    default = User('default', {'bob': bob, 'chuck': chuck})
    return UserRouter(default)
```

(Source: `user_router.py`)

Putting this in a file called `user_router.py` and running `twistd -n web --class=user_router.start` you'll find it reacts as below:

```
$ curl http://localhost:8080/
default
$ curl http://localhost:8080/friend/chuck/friend/bob/
bob
```

4.2 Query Arguments

The tedious process of processing query arguments usually follows this pattern:

- Check for the existence of the argument;
- Check that the argument has at least one value;
- Convert the argument from its text representation into something more useful.

4.2.1 Parsing query arguments

This small set of utility functions can relieve some of that pain, for example assume the query string is `count=1&fields=a,b,c&includeHidden=yes&start=1399473753&end=&flags=FOO&flags=BAR`, Twisted Web parses this into the following `IRequest.args` result:

```
{'count':         ['1'],
 'fields':        ['a,b,c'],
 'includeHidden': ['yes'],
 'start':         ['1399473753'],
 'end':           [],
 'flags':         ['FOO', 'BAR']}
```

Instead of performing all the necessary checks and transformations yourself this could be parsed with the following:

```
from txspinneret import query as q
from twisted.python.constants import Names, NamedConstant

class FlagConstants(Names):
    FOO = NamedConstant()
    BAR = NamedConstant()

flag = lambda value: FlagConstants.lookupByName(q.Text(value))
q.parse({
    'count':         q.one(q.Integer),
    'fields':        q.one(q.Delimited),
    'includeHidden': q.one(q.Boolean),
    'start':         q.one(q.Timestamp),
    'end':           q.one(q.Timestamp),
    'flags':         q.many(flag)}, request.args)
```

Which would produce the result:

```
{'count':         1,
 'end':           None,
 'fields':        [u'a', u'b', u'c'],
 'start':         datetime.datetime(2014, 5, 7, 16, 42, 33),
 'flags':         [<FlagConstant=FOO>, <FlagConstant=BAR>],
 'includeHidden': True}
```

4.3 Resource Utilities

A collection of higher-level Twisted Web resource utilities, suitable for use with any existing `IResource` implementations.

4.3.1 More featureful resources

`ISpinneretResource` is cut-down version of `IResource` that allows child location (via `ISpinneretResource.locateChild`) and rendering (via the normal `render_GET`, `render_POST`, etc. methods) to return a 2-`tuple` of firstly any of the following:

- `bytes`, in the same way that `IResource` does;
- An object that can be adapted to either `IResource` or `IRenderable`;
- A `URLPath` instance, to indicate an HTTP redirect;
- Or a `Deferred` that results in any of the above values.

And secondly, a `list` of remaining path segments to be processed.

`ISpinneretResource` implementations may be adapted to `IResource` via `SpinneretResource`, to produce a resource suitable for use with Twisted Web.

4.3.2 Negotiating resources based on `Accept`

When building an API, in particular, you may want to negotiate the resource that best fits what the client is willing to accept, as specified in the `Accept` header; enter `ContentTypeNegotiator`. For example: If the client indicates it accepts, in order, `application/xml` and `application/json` and your service supports JSON and HTML, you are able to deliver a result that the client finds acceptable. It is also possible to allow falling back to a default representation in the case where negotiations fail.

An example of supporting JSON and HTML might be:

```
import json
from twisted.web.resource import Resource
from twisted.web.template import Element, TagLoader, tags
from txspinneret.interfaces import INegotiableResource, ISpinneretResource
from txspinneret.resource import ContentTypeNegotiator
from zope.interface import implementer

@implementer(INegotiableResource)
class FooJSON(Resource):
    contentType = 'application/json'
    acceptTypes = ['application/json', 'application/javascript']

    def __init__(self, obj):
        self.obj = obj
        Resource.__init__(self)

    def render_GET(self, request):
        return json.dumps(self.obj)

class FooElement(Element):
    loader = TagLoader(tags.h1('Try accepting JSON!'))

@implementer(INegotiableResource, ISpinneretResource)
class FooHTML(object):
    contentType = 'text/html'
    acceptTypes = ['text/html']

    def render_GET(self, request):
        return FooElement()

def start():
    data = {'name': 'Bob'}
    resource = ContentTypeNegotiator(
        [FooHTML(), FooJSON(data)],
        fallback=True)
    resource.isLeaf = True
    return resource
```

(Source: `negotiator.py`)

Putting this in a file called `negotiator.py` and running `twistd -n web --class=negotiator.start` will create a resource where performing an HTTP `GET` on `/` with a `text/html` `Accept` header (or no `Accept` header) results in an HTML page, while an `Accept` header of `application/json` results in a JSON response:

```
$ curl -H 'Accept: application/json' http://localhost:8080/
{"name": "Bob"}
```

While any other content type results in the HTML page.

4.4 API documentation

4.4.1 API Documentation

Resource routing

URL routing for Twisted Web resources.

A Python-based Domain Specific Language is used to specify and match routing paths, string literal components are matched for structure while plain callable components match segment values and are stored by name for use in the handler, assuming all the components match; this makes it trivial to create new functions to match path components.

`route` is used to match a URL exactly (the number of route components must match the number of URL path segments) while `subroute` is used to match a URL prefix (the specified route components must match the respective segments in a URL path, additional segments are used in child resource location as normal.)

`Router` is an `IResource` that allows decorating methods as route or subroute handlers.

Members

class txspinneret.route.**Router**

Bases: object

URL routing.

Router is designed to be used as a Python descriptor using Router.route or Router.subroute to decorate route handlers, for example:

```
class Users(object):
    router = Router()

    @router.route('name')
    def name(self, request, params):
        # ...
```

Route handlers can return any value supported by ISpinneretResource.locateChild.

Calling Router.resource will produce an IResource.

__weakref__

list of weak references to the object (if defined)

resource()

Create an IResource that will perform URL routing.

route(**components*)

See txspinneret.route.route.

This decorator can be stacked with itself to specify multiple routes with a single handler.

subroute(**components*)

See txspinneret.route.subroute.

This decorator can be stacked with itself to specify multiple routes with a single handler.

txspinneret.route.**Any**(*name*, *encoding=None*)

Match a route parameter.

Any is a synonym for Text.

Parameters

- **name** (bytes) – Route parameter name.
- **encoding** (bytes) – Default encoding to assume if the Content-Type header is lacking one.

Returns callable suitable for use with route or subroute.

txspinneret.route.**Text**(*name*, *encoding=None*)

Match a route parameter.

Any is a synonym for Text.

Parameters

- **name** (bytes) – Route parameter name.
- **encoding** (bytes) – Default encoding to assume if the Content-Type header is lacking one.

Returns callable suitable for use with route or subroute.

`txspinneret.route.`**`Integer`**`(`*name*, *base=10*, *encoding=None*`)`

Match an integer route parameter.

Parameters

- **name** (`bytes`) – Route parameter name.
- **base** (`int`) – Base to interpret the value in.
- **encoding** (`bytes`) – Default encoding to assume if the `Content-Type` header is lacking one.

Returns `callable` suitable for use with `route` or `subroute`.

`txspinneret.route.`**`route`**`(`**components*`)`

Match a request path exactly.

The path components are always matched relative to their parent is in the resource hierarchy, in other words it is only possible to match URIs nested more deeply than the parent resource.

Parameters **components** (`iterable` of `bytes` or `callable`) – Iterable of path components, to match against the request, either static strings or dynamic parameters. As a convenience, a single `bytes` component containing `/` may be given instead of manually separating the components. If no components are given the null route is matched, this is the case where `segments` is empty.

Return type 2-`tuple` of `dict` keyed on `bytes` and `list` of `bytes`

Returns Pair of parameter results, mapping parameter names to processed values, and a list of the remaining request path segments. If there is no route match the result will be `None` and the original request path segments.

`txspinneret.route.`**`subroute`**`(`**components*`)`

Partially match a request path exactly.

The path components are always matched relative to their parent is in the resource hierarchy, in other words it is only possible to match URIs nested more deeply than the parent resource.

If there are more request path segments than components the match may still be successful, the remaining path segments are returned in the second part of the result.

Parameters **components** (`iterable` of `bytes` or `callable`) – Iterable of path components, to match against the request, either static strings or dynamic parameters. As a convenience, a single `bytes` component containing `/` may be given instead of manually separating the components. If no components are given the null route is matched, this is the case where `segments` is empty.

Return type 2-`tuple` of `dict` keyed on `bytes` and `list` of `bytes`

Returns Pair of parameter results, mapping parameter names to processed values, and a list of the remaining request path segments. If there is no route match the result will be `None` and the original request path segments.

`txspinneret.route.`**`routedResource`**`(`*f*, *routerAttribute='router'*`)`

Decorate a router-producing callable to instead produce a resource.

This simply produces a new callable that invokes the original callable, and calls `resource` on the `routerAttribute`.

If the router producer has multiple routers the attribute can be altered to choose the appropriate one, for example:

```
class _ComplexRouter(object):
    router = Router()
    privateRouter = Router()

    @router.route('/')
```

```
        def publicRoot(self, request, params):
            return SomethingPublic(...)

        @privateRouter.route('/')
        def privateRoot(self, request, params):
            return SomethingPrivate(...)

    PublicResource = routedResource(_ComplexRouter)
    PrivateResource = routedResource(_ComplexRouter, 'privateRouter')
```

Parameters

- **f** (`callable`) – Callable producing an object with a `Router` attribute, for example, a type.
- **routerAttribute** (`str`) – Name of the `Router` attribute on the result of calling `f`.

Return type `callable`

Returns Callable producing an `IResource`.

Query arguments

Utility functions for processing query arguments.

The task of processing query arguments is made easy by deciding whether you want exactly `one` or `many` results and then composing that with the expected argument type, such as `Integer`, `Text`, `Boolean`, etc.; for example:

`one(Integer)`

Produces a callable that takes a list of strings and produces an integer from the first value, or `None` if the list was empty or the first value could not be parsed as an integer.

`many(Boolean)`

Produces a callable that takes a list of strings and produces a list of booleans.

Members

`txspinneret.query.`**`parse`**(*expected*, *query*)

Parse query parameters.

Parameters

- **expected** (`dict` mapping `bytes` to `callable`) – Mapping of query argument names to argument parsing callables.
- **query** (`dict` mapping `bytes` to `list` of `bytes`) – Mapping of query argument names to lists of argument values, this is the form that Twisted Web's `IRequest.args` value takes.

Return type `dict` mapping `bytes` to `object`

Returns Mapping of query argument names to parsed argument values.

`txspinneret.query.`**`one`**(*func*, *n=0*)

Create a callable that applies `func` to a value in a sequence.

If the value is not a sequence or is an empty sequence then `None` is returned.

Parameters

- **func** (`callable`) – Callable to be applied to each result.
- **n** (`int`) – Index of the value to apply `func` to.

`txspinneret.query.`**`many`**(*func*)
Create a callable that applies `func` to every value in a sequence.

If the value is not a sequence then an empty list is returned.

Parameters **func** (`callable`) – Callable to be applied to the first result.

`txspinneret.query.`**`Text`**(*value*, *encoding=None*)
Parse a value as text.

Parameters

- **value** (`unicode` or `bytes`) – Text value to parse
- **encoding** (`bytes`) – Encoding to treat `bytes` values as, defaults to `utf-8`.

Return type `unicode`

Returns Parsed text or `None` if `value` is neither `bytes` nor `unicode`.

`txspinneret.query.`**`Integer`**(*value*, *base=10*, *encoding=None*)
Parse a value as an integer.

Parameters

- **value** (`unicode` or `bytes`) – Text value to parse
- **base** (`unicode` or `bytes`) – Base to assume `value` is specified in.
- **encoding** (`bytes`) – Encoding to treat `bytes` values as, defaults to `utf-8`.

Return type `int`

Returns Parsed integer or `None` if `value` could not be parsed as an integer.

`txspinneret.query.`**`Float`**(*value*, *encoding=None*)
Parse a value as a floating point number.

Parameters

- **value** (`unicode` or `bytes`) – Text value to parse.
- **encoding** (`bytes`) – Encoding to treat `bytes` values as, defaults to `utf-8`.

Return type `float`

Returns Parsed float or `None` if `value` could not be parsed as a float.

`txspinneret.query.`**`Boolean`**(*value*, *true=(u'yes', u'1', u'true')*, *false=(u'no', u'0', u'false')*, *encoding=None*)
Parse a value as a boolean.

Parameters

- **value** (`unicode` or `bytes`) – Text value to parse.
- **true** (`tuple` of `unicode`) – Values to compare, ignoring case, for `True` values.
- **false** (`tuple` of `unicode`) – Values to compare, ignoring case, for `False` values.
- **encoding** (`bytes`) – Encoding to treat `bytes` values as, defaults to `utf-8`.

Return type `bool`

Returns Parsed boolean or `None` if `value` did not match `true` or `false` values.

`txspinneret.query.`**`Delimited`**(*value*, *parser=<function Text at 0x7f5aa7214758>*, *delimiter=u', '*, *encoding=None*)

Parse a value as a delimited list.

Parameters

- **value** (`unicode` or `bytes`) – Text value to parse.
- **parser** (`callable` taking a `unicode` parameter) – Callable to map over the delimited text values.
- **delimiter** (`unicode`) – Delimiter text.
- **encoding** (`bytes`) – Encoding to treat `bytes` values as, defaults to `utf-8`.

Return type `list`

Returns List of parsed values.

`txspinneret.query.`**`Timestamp`**(*value*, *_divisor=1.0*, *tz=FixedOffset(0, 0)*, *encoding=None*)

Parse a value as a POSIX timestamp in seconds.

Parameters

- **value** (`unicode` or `bytes`) – Text value to parse, which should be the number of seconds since the epoch.
- **_divisor** (`float`) – Number to divide the value by.
- **tz** (`tzinfo`) – Timezone, defaults to UTC.
- **encoding** (`bytes`) – Encoding to treat `bytes` values as, defaults to `utf-8`.

Return type `datetime.datetime`

Returns Parsed datetime or `None` if `value` could not be parsed.

`txspinneret.query.`**`TimestampMs`**(*value*, *encoding=None*)

Parse a value as a POSIX timestamp in milliseconds.

Parameters

- **value** (`unicode` or `bytes`) – Text value to parse, which should be the number of milliseconds since the epoch.
- **encoding** (`bytes`) – Encoding to treat `bytes` values as, defaults to `utf-8`.

Return type `datetime.datetime`

Returns Parsed datetime or `None` if `value` could not be parsed.

Utility resources

A collection of higher-level Twisted Web resources, suitable for use with any existing `IResource` implementations.

`SpinneretResource` adapts an `ISpinneretResource` to `IResource`.

`ContentTypeNegotiator` will negotiate a resource based on the `Accept` header.

Members

class `txspinneret.resource.`**`SpinneretResource`**(*wrappedResource*)

Bases: `twisted.web.resource.Resource`

Adapter from `ISpinneretResource` to `IResource`.

Parameters **wrappedResource** (`ISpinneretResource`) – Spinneret resource to wrap in an `IResource`.

class `txspinneret.resource.`**`ContentTypeNegotiator`**(*handlers*, *fallback=False*)
Bases: `twisted.web.resource.Resource`

Negotiate an appropriate representation based on the `Accept` header.

Rendering this resource will negotiate a representation and render the matching handler.

Parameters

- **handlers** (`iterable` of `INegotiableResource` and either `IResource` or `ISpinneretResource`.) – Iterable of negotiable resources, either `ISpinneretResource` or `IResource`, to use as handlers for negotiation.
- **fallback** (`bool`) – Fall back to the first handler in the case where negotiation fails?

class `txspinneret.resource.`**`NotAcceptable`**
Bases: `twisted.web.resource.Resource`

Leaf resource that renders an empty body for `406 Not Acceptable`.

Initialize.

class `txspinneret.resource.`**`NotFound`**
Bases: `twisted.web.resource.NoResource`

Leaf resource that renders a page for `404 Not Found`.

Interfaces

interface `txspinneret.interfaces.`**`INegotiableResource`**
Bases: `zope.interface.Interface`

Resource used for content negotiation.

The implementation should be adaptable to `IResource`.

`acceptTypes` **= <zope.interface.interface.Attribute object at 0x7f5aa6a3a950>**
`list` of `bytes` indicating the content types this resource is capable of accepting.

`contentType` **= <zope.interface.interface.Attribute object at 0x7f5aa6a3a8d0>**
`bytes` indicating the content type of this resource when rendered.

interface `txspinneret.interfaces.`**`ISpinneretResource`**
Bases: `zope.interface.Interface`

Spinneret resource.

Spinneret resources may additionally have methods like `render_GET` (to handle a `GET` request) `render_POST` etc., like `IResource`, that may return the same types of objects as `ISpinneretResource.locateChild`.

Adaptable to `IResource`.

`locateChild`(*request*, *segments*)
Locate another object which can be adapted to `IResource`.

Parameters

- **request** (`IRequest`) – Request.
- **segments** (`sequence` of `bytes`) – Sequence of strings giving the remaining query segments to resolve.

Return type 2-`tuple` of `IResource`, `IRenderable` or `URLPath` and a `sequence` of `bytes`

Returns Pair of an `IResource`, `IRenderable` or `URLPath` and a sequence of the remaining path segments to be process, or a `Deferred` containing the aforementioned result.

CHAPTER 5

Contributing

5.1 Contributing to Spinneret

Spinneret is an open source project that encourages community contributions to all aspects:

- Code patches;
- Documentation improvements;
- Bug reports;
- Code reviews for contributed patches.

5.1.1 Code

- Propose all code changes via a pull request in the GitHub repository;
- Use Twisted's coding standard;
- Ensure codes changes have unit tests and good coverage;
- New features should have examples and documentation;
- Add yourself to AUTHORS.

5.1.2 Documentation

- The header order:

```
========
Header 1
========

Header 2
========

Header 3
--------

Header 4
~~~~~~~~
```

- Perform at least basic spelling checks;

- Use gender-neutral language (singular they is great!);

5.1.3 Reviewing

All code that is merged into Spinneret must be reviewed by at least one person other than an author of the change.

While perfection is a noble goal it often leads to an idea that improvement without achieving perfection is not an improvement. Improvements need only be that, improvements. Glyph wrote a compelling email on this topic, it's worth reading if you're a reviewer *or* a contributor.

Python Module Index