typedjsonrpc

Release 0.4.0+0.gd63dfe7.dirty

January 14, 2016

Contents

Contents:

CHAPTER 1

typedjsonrpc

typedjsonrpc is a decorator-based JSON-RPC library for Python that exposes parameter and return types. It is influenced by Flask JSON-RPC but has some key differences:

typedjsonrpc...

- allows return type checking
- focuses on easy debugging

These docs are also available on Read the Docs.

1.1 Using typedjsonrpc

1.1.1 Installation

Use pip to install typedjsonrpc:

```
$ pip install typedjsonrpc
```

1.1.2 Project setup

To include typedjsonrpc in your project, use:

```
from typedjsonrpc.registry import Registry
from typedjsonrpc.server import Server

registry = Registry()
server = Server(registry)
```

The registry will keep track of methods that are available for JSON-RPC. Whenever you annotate a method, it will be added to the registry. You can always use the method `rpc.describe()` to get a description of all available methods. `Server` is a WSGI compatible app that handles requests. `Server` also has a development mode that can be run using `server.run(host, port)`.

1.1.3 Example usage

Annotate your methods to make them accessible and provide type information:

```
@registry.method(returns=int, a=int, b=int)
def add(a, b):
    return a + b

@registry.method(returns=str, a=str, b=str)
def concat(a, b):
    return a + b
```

The return type *has* to be declared using the `returns` keyword. For methods that don't return anything, you can use either `type(None)` or just `None`:

```
@registry.method(returns=type(None), a=str)
def foo(a):
    print(a)

@registry.method(returns=None, a=int)
def bar(a):
    print(5 * a)
```

You can use any of the basic JSON types:

JSON type	Python type
string	basestring (Python 2), str (Python 3)
number	int, float
null	None
boolean	bool
array	list
object	dict

Your functions may also accept `*args` and `**kwargs`, but you cannot declare their types. So the correct way to use these would be:

```
@registry.method(a=str)
def foo(a, *args, **kwargs):
    return a + str(args) + str(kwargs)
```

To check that everything is running properly, try (assuming `add` is declared in your main module):

```
$ curl -XPOST http://<host>:<port>/api -d @- <<EOF
{
    "jsonrpc": "2.0",
    "method": "__main__.add",
    "params": {
        "a": 5,
        "b": 7
    },
    "id": "foo"
}
EOF

{
    "jsonrpc": "2.0",
    "id": "foo",
    "result": 12
}
```

Passing any non-integer arguments into `add` will raise a `InvalidParamsError`.

1.1.4 Batching

You can send a list of JSON-RPC request objects as one request and will receive a list of JSON-RPC response objects in return. These response objects can be mapped back to the request objects using the `id`. Here's an example of calling the `add` method with two sets of parameters:

```
$ curl -XPOST http://<host>:<port>/api -d @- <<EOF
[
    {
        "jsonrpc": "2.0",
        "method": "__main__.add",
        "params": {
            "a": 5,
            "b": 7
        },
        "id": "foo"
    }, {
        "jsonrpc": "2.0",
        "method": "__main__.add",
        "params": {
            "a": 42,
            "b": 1337
        },
        "id": "bar"
    }
]
EOF

[
    {
        "jsonrpc": "2.0",
        "id": "foo",
        "result": 12
    }, {
        "jsonrpc": "2.0",
        "id": "bar",
        "result": 1379
    }
]
```

1.1.5 Debugging

If you create the registry with the parameter `debug=True`, you'll be able to use werkzeug's debugger. In that case, if there is an error during execution - e.g. you tried to use a string as one of the parameters for `add` - the response will contain an error object with a `debug_url`:

```
$ curl -XPOST http://<host>:<port>/api -d @- <<EOF
{
    "jsonrpc": "2.0",
    "method": "__main__.add",
    "params": {
        "a": 42,
        "b": "hello"
    },
    "id": "bar"
}
EOF
```

```
{
    "jsonrpc": "2.0",
    "id": "bar",
    "error": {
        "message": "Invalid params",
        "code": -32602,
        "data": {
            "message": "Value 'hello' for parameter 'b' is not of expected type <type 'int'>.",
            "debug_url": "/debug/1234567890"
        }
    }
}
```

This tells you to find the traceback interpreter at `<host>:<port>/debug/1234567890`.

1.1.6 Logging

The registry has a default logger in the module `typedjsonrpc.registry` and it logs all errors that are not defined by `typedjsonrpc`. You can configure the logger as follows:

```
import logging
logger = logging.getLogger("typedjsonrpc.registry")
# Do configuration to this logger
```

1.1.7 HTTP status codes

Since typedjsonrpc 0.4.0, HTTP status codes were added to the responses from the `typedjsonrpc.server.Server` class. This is to improve the usage of typedjsonrpc over HTTP. The following chart are the satus codes which are returned:

Condition	Batched	Status code
Success	Y	200
	N	200
All notifications	Y	204
	N	204
ParseError or InvalidRequestError	Y	200
	N	400
MethodNotFoundError	Y	200
	N	404
All other errors	Y	200
	N	500

1.2 Additional features

1.2.1 Customizing type serialization

If you would like to serialize custom types, you can set the `json_encoder` and `json_decoder` attributes on `Server` to your own custom `json.JSONEncoder` and `json.JSONDecoder` instance. By default, we use the default encoder and decoder.

1.2.2 Adding hooks before the first request

You can add functions to run before the first request is called. This can be useful for some special setup you need for your WSGI app. For example, you can register a function to print debugging information before your first request:

```
import datetime

from typedjsonrpc.registry import Registry
from typedjsonrpc.server import Server

registry = Registry()
server = Server(registry)

def print_time():
    now = datetime.datetime.now()
    print("Handling first request at: {}".format(now))

server.register_before_first_request(print_time)
```

1.2.3 Accessing the HTTP request from JSON-RPC methods

In some situations, you may want to access the HTTP request from your JSON-RPC method. For example, you could need to perform logic based on headers in the request. In the *typedjsonrpc.server* module, there is a special *typedjsonrpc.server.current_request* attribute which allows you to access the HTTP request which was used to call the current method.

> **Warning:** `current_request` is implemented as a thread-local. If you attempt to call `Server.wsgi_app` from `Registry.method`, then `current_request` *will be overriden in that thread.*

Example:

```
from typedjsonrpc.server import current_request

@registry.method(returns=list)
def get_headers():
    return list(current_request.headers)
```

1.2.4 Disabling strictness of floats

`typedjsonrpc` by default will only accept floats into a *float* typed parameter. For example, if your function were this:

```
import math

@registry.method(returns=int, x=float)
def floor(x):
    return int(math.floor(x))
```

and your input were this:

```
{
    "jsonrpc": "2.0",
    "method": "floor",
    "params": {
        "x": 1
```

```
    },
    "id": "foo"
}
```

You would get an invalid param error like this:

```
{
    "error": {
        "code": -32602,
        "data": {
            "debug_url": "/debug/4456954960",
            "message": "Value '1' for parameter 'x' is not of expected type <type 'float'>."
        },
        "message": "Invalid params"
    },
    "id": "foo",
    "jsonrpc": "2.0"
}
```

This can actually frequently come up when you use a JSON encoder. A JSON encoder may choose to write the float `1.0` as an integer `1`. In order to get around this, you can manually edit the JSON or set `strict_floats` to `False` in your *typedjsonrpc.registry.Registry*.

CHAPTER 2

API Reference

2.1 Errors

Error classes for typedjsonrpc.

exception typedjsonrpc.errors.**Error**(*data=None*)
Base class for all errors.

New in version 0.1.0.

as_error_object()
Turns the error into an error object.

New in version 0.1.0.

exception typedjsonrpc.errors.**InternalError**(*data=None*)
Internal JSON-RPC error.

New in version 0.1.0.

static from_error(*exc_info*, *json_encoder*, *debug_url=None*)
Wraps another Exception in an InternalError.

Parameters **exc_info** (*(type, object, traceback)*) – The exception info for the wrapped exception

Return type *InternalError*

New in version 0.1.0.

Changed in version 0.2.0: Stringifies non-JSON-serializable objects

exception typedjsonrpc.errors.**InvalidParamsError**(*data=None*)
Invalid method parameter(s).

New in version 0.1.0.

exception typedjsonrpc.errors.**InvalidRequestError**(*data=None*)
The JSON sent is not a valid request object.

New in version 0.1.0.

exception typedjsonrpc.errors.**InvalidReturnTypeError**(*data=None*)
Return type does not match expected type.

New in version 0.1.0.

exception typedjsonrpc.errors.**MethodNotFoundError**(*data=None*)
The method does not exist.

New in version 0.1.0.

exception typedjsonrpc.errors.**ParseError**(*data=None*)
Invalid JSON was received by the server / JSON could not be parsed.

New in version 0.1.0.

exception typedjsonrpc.errors.**ServerError**(*data=None*)
Something else went wrong.

New in version 0.1.0.

typedjsonrpc.errors.**get_status_code_from_error_code**(*error_code*)
Returns the status code for the matching error code.

New in version 0.4.0.

2.2 Method Info

Data structures for wrapping methods and information about them.

class typedjsonrpc.method_info.**MethodInfo**
An object wrapping a method and information about it.

Attribute name Name of the function

Attribute method The function being described

Attribute signature A description of the types this method takes as parameters and returns

describe()
Describes the method.

Returns Description

Return type dict[str, object]

description
Returns the docstring for this method.

Return type str

params
The parameters for this method in a JSON-compatible format

Return type list[dict[str, str]]

returns
The return type for this method in a JSON-compatible format.

This handles the special case of `None` which allows `type(None)` also.

Return type str | None

class typedjsonrpc.method_info.**MethodSignature**
Represents the types which a function takes as input and output.

Attribute parameter_types A list of tuples mapping strings to type with a specified order

Attribute return_type The type which the function returns

static **create**(*parameter_names*, *parameter_types*, *return_type*)
Returns a signature object ensuring order of parameter names and types.

Parameters

- **parameter_names** (*list[str]*) – A list of ordered parameter names
- **parameter_types** (*dict[str, type]*) – A dictionary of parameter names to types
- **return_type** (*type*) – The type the function returns

Return type *MethodSignature*

2.3 Parameter Checker

Logic for checking parameter declarations and parameter types.

typedjsonrpc.parameter_checker.**check_return_type**(*value*, *expected_type*, *strict_floats*)
Checks that the given return value has the correct type.

Parameters

- **value** (*object*) – Value returned by the method
- **expected_type** (*type*) – Expected return type
- **strict_floats** (*bool*) – If False, treat integers as floats

typedjsonrpc.parameter_checker.**check_type_declaration**(*parameter_names*, *parameter_types*)
Checks that exactly the given parameter names have declared types.

Parameters

- **parameter_names** (*list[str]*) – The names of the parameters in the method declaration
- **parameter_types** (*dict[str, type]*) – Parameter type by name

typedjsonrpc.parameter_checker.**check_types**(*parameters*, *parameter_types*, *strict_floats*)
Checks that the given parameters have the correct types.

Parameters

- **parameters** (*dict[str, object]*) – List of (name, value) pairs of the given parameters
- **parameter_types** (*dict[str, type]*) – Parameter type by name.
- **strict_floats** (*bool*) – If False, treat integers as floats

typedjsonrpc.parameter_checker.**validate_params_match**(*method*, *parameters*)
Validates that the given parameters are exactly the method's declared parameters.

Parameters

- **method** (*function*) – The method to be called
- **parameters** (*dict[str, object]* | *list[object]*) – The parameters to use in the call

2.4 Registry

Logic for storing and calling jsonrpc methods.

class typedjsonrpc.registry.**Registry**(*debug=False*, *strict_floats=True*)
The registry for storing and calling jsonrpc methods.

Attribute debug Debug option which enables recording of tracebacks

Attribute tracebacks Tracebacks for debugging

New in version 0.1.0.

__init__(*debug=False*, *strict_floats=True*)

Parameters

- **debug** (*bool*) – If True, the registry records tracebacks for debugging purposes
- **strict_floats** (*bool*) – If True, the registry does not allow ints as float parameters

Changed in version 0.4.0: Added strict_floats option

describe()
Returns a description of all the methods in the registry.

Returns Description

Return type dict[str, object]

New in version 0.1.0.

dispatch(*request*)
Takes a request and dispatches its data to a jsonrpc method.

Parameters **request** (*werkzeug.wrappers.Request*) – a werkzeug request with json data

Returns json output of the corresponding method

Return type str

New in version 0.1.0.

json_decoder = <json.decoder.JSONDecoder object>
The JSON decoder to use. Defaults to json.JSONDecoder

New in version 0.1.0.

Changed in version 0.2.0: Changed from class to instance

json_encoder = <json.encoder.JSONEncoder object>
The JSON encoder to use. Defaults to json.JSONEncoder

New in version 0.1.0.

Changed in version 0.2.0: Changed from class to instance

method(*returns*, ***parameter_types*)
Syntactic sugar for registering a method

Example:

```
>>> registry = Registry()
>>> @registry.method(returns=int, x=int, y=int)
... def add(x, y):
...     return x + y
```

Parameters

- **returns** (*type*) – The method's return type
- **parameter_types** (*dict[str, type]*) – The types of the method's parameters

New in version 0.1.0.

register(*name*, *method*, *method_signature=None*)
Registers a method with a given name and signature.

Parameters

- **name** (*str*) – The name used to register the method
- **method** (*function*) – The method to register
- **method_signature** (*MethodSignature | None*) – The method signature for the given function

New in version 0.1.0.

2.5 Server

Contains the Werkzeug server for debugging and WSGI compatibility.

class typedjsonrpc.server.**Server**(*registry*, *endpoint='/api'*)
A basic WSGI-compatible server for typedjsonrpc endpoints.

Attribute registry The registry for this server

New in version 0.1.0.

Changed in version 0.4.0: Now returns HTTP status codes

__init__(*registry*, *endpoint='/api'*)

Parameters

- **registry** (typedjsonrpc.registry.Registry) – The JSON-RPC registry to use
- **endpoint** (*str*) – The endpoint to publish JSON-RPC endpoints. Default "/api".

register_before_first_request(*func*)
Registers a function to be called once before the first served request.

Parameters **func** (*() -> object*) – Function called

New in version 0.1.0.

run(*host*, *port*, ***options*)
For debugging purposes, you can run this as a standalone server.

> **Warning: Security vulnerability**
> This uses `DebuggedJsonRpcApplication` to assist debugging. If you want to use this in production, you should run `Server` as a standard WSGI app with uWSGI or another similar WSGI server.

New in version 0.1.0.

wsgi_app(*environ*, *start_response*)
A basic WSGI app

class typedjsonrpc.server.**DebuggedJsonRpcApplication**(*app*, ***kwargs*)
A JSON-RPC-specific debugged application.

This differs from DebuggedApplication since the normal debugger assumes you are hitting the endpoint from a web browser.

A returned response will be JSON of the form: `{"traceback_id": <id>}` which you can use to hit the endpoint `http://<host>:<port>/debug/<traceback_id>`.

New in version 0.1.0.

> **Warning: Security vulnerability**
> This should never be used in production because users have arbitrary shell access in debug mode.

__init__(*app*, ***kwargs*)

Parameters

- **app** (typedjsonrpc.server.Server) – The wsgi application to be debugged
- **kwargs** – The arguments to pass to the DebuggedApplication

debug_application(*environ*, *start_response*)
Run the application and preserve the traceback frames.

Parameters

- **environ** (*dict[str, object]*) – The environment which is passed into the wsgi application
- **start_response** (*(str, list[(str, str)]) -> None*) – The start_response function of the wsgi application

Return type generator[str]

New in version 0.1.0.

handle_debug(*environ*, *start_response*, *traceback_id*)
Handles the debug endpoint for inspecting previous errors.

Parameters

- **environ** (*dict[str, object]*) – The environment which is passed into the wsgi application
- **start_response** (*(str, list[(str, str)]) -> NoneType*) – The start_response function of the wsgi application
- **traceback_id** (*int*) – The id of the traceback to inspect

New in version 0.1.0.

typedjsonrpc.server.**current_request = <LocalProxy unbound>**
A thread-local which stores the current request object when dispatching requests for *Server*.

Stores a `werkzeug.wrappers.Request`.

New in version 0.2.0.

CHAPTER 3

Release Notes

3.1 0.4.0

This update includes a few new features around debugging.

3.1.1 Features

- Added automatic logging of user-created errors thrown during runtime
- Added HTTP status codes to *typedjsonrpc.server.Server* based on JSON-RPC protocol-based errors
- Added flag to *typedjsonrpc.registry.Registry* for the type system to allow integers when a field accepts floats

3.2 0.3.0

This is a small update to better handle JSON encoding errors.

3.2.1 Bugfixes

- Any exceptions thrown by a custom `json.JSONEncoder` will be reencoded after the exception has been thrown. JSON-RPC will not return a response if the custom encoder cannot encode the exception.

3.3 0.2.0

This is a small update from the last release based on usage.

3.3.1 Features

- Added ability to access the current request in method call
- Allowed more flexibility in JSON serialization

3.3.2 Bugfixes

- Exceptions which are not JSON-serializable are now converted to strings using `repr()` rather than failing serialization

3.3.3 Breaking changes

- *typedjsonrpc.registry.Registry.json_encoder* and *typedjsonrpc.registry.Registry.json_deco* are now instances rather than class objects

3.4 0.1.0

Initial Release

CHAPTER 4

Indices and tables

- genindex
- modindex

t

Symbols

A

C

D

E

F

G

H

I

J

M

P

R

S

T

V

W