JsonRpc

A modern `Json-Rpc` implementation, compatible with `Json-Rpc 2.0` and `Json-Rpc X`, supports multiple network protocols and backend frameworks and supports bidirectional calls..

> 人间总有一两风，填我十万八千梦

安装

```
jsonrpc-pypip install async-jsonrpc
```

```
pdm add async-jsonrpc
```

特性

- ☑跨语言

    - [x] Python3
    - [ ] JavaScript
    - [ ] Go
    - [ ] C/C++
    - [ ] Java
    - [ ] Rust
- [x] 多种网络协议支持
    - [x] HTTP
    - [x] WebSocket
- [x] 兼容 Json-Rpc 2.0 规范
- [x] 兼容 Json-Rpc X 规范
- [x] 支持完备的链式调用
- [x] 自动 Json-Rpc 规范转换
- [x] 双向流式传输
- [ ] IDE 支持
- [ ] 分布式Server

使用

入门

Server 端

```
import asyncio
from jsonrpc import JsonRpc, AioServer

expose = {
    'add': lambda a, b: a + b,
    'sub': lambda a, b: a - b,
    'value': 'jsonrpc Server 0.0.1'
}

loop = asyncio.new_event_loop()

rpc = JsonRpc('/test', namespace=expose, loop=loop)

rpc.run_server(AioServer())
```

Client 端

```
import asyncio
from jsonrpc import JsonRpc, AioClient, logger

loop = asyncio.new_event_loop()

rpc = JsonRpc('/test', loop=loop)

async def main():
    await rpc.run_client(AioClient())
    res1 = await rpc.value
    res2 = await rpc.add(4, 4)

    logger.info(res1)
    logger.info(res2)

loop.run_until_complete(main())
loop.run_forever()
```

链式调用

Server 端

```
import asyncio
from jsonrpc import JsonRpc, AioServer

class number:

    def __init__(self) -> None:
        self.value = 0

    def add(self, i: int):
        self.value += i
        return self

    def sub(self, i: int):
        self.value -= i
        return self

expose = {
    'number': number
}

loop = asyncio.new_event_loop()

rpc = JsonRpc('/test', namespace=expose, loop=loop)

rpc.run_server(AioServer())
```

Client 端

```
import asyncio
from jsonrpc import JsonRpc, AioClient, logger

loop = asyncio.new_event_loop()

rpc = JsonRpc('/test', loop=loop)

async def main():
    await rpc.run_client(AioClient())
    res3 = await rpc.number().add(10).sub(10).value
    logger.info(res3)

loop.run_until_complete(main())
loop.run_forever()
```

反向调用

Server 端

```
import asyncio
from jsonrpc import JsonRpc, AioServer

expose = {'value': 'jsonrpc Server 0.0.1'}

loop = asyncio.new_event_loop()

rpc = JsonRpc('/test', namespace=expose, loop=loop)

async def repeat():
    ws = rpc._server.active_connections[0]
    res = await rpc.send_request_server(rpc.value, ws[0])
    return res

rpc.add_namespace('repeat', repeat)

rpc.run_server(AioServer())
```

Client 端

```
import asyncio
from jsonrpc import JsonRpc, AioClient, logger

expose = {'value': 'jsonrpc Client 0.0.1'}

loop = asyncio.new_event_loop()

rpc = JsonRpc('/test', namespace=expose, loop=loop)
```

```
async def main():
    await rpc.run_client(AioClient())
    res4 = await rpc.repeat()
    logger.info(res4)

loop.run_until_complete(main())
loop.run_forever()
```

流式调用

Server 端

```
import asyncio
from jsonrpc import JsonRpc, AioServer

async def async_generator(i = 10):
    for i in range(i):
        await asyncio.sleep(0.5)
        yield i

expose = {'async_generator': async_generator}

loop = asyncio.new_event_loop()

rpc = JsonRpc('/test', namespace=expose, loop=loop)

rpc.run_server(AioServer())
```

Client 端

```
import asyncio
from jsonrpc import JsonRpc, AioClient, logger

loop = asyncio.new_event_loop()

rpc = JsonRpc('/test', loop=loop)

async def main():
    await rpc.run_client(AioClient())

    async for i in rpc.async_generator(5): # type: ignore
        logger.info(i)

loop.run_until_complete(main())
loop.run_forever()
```