asphalt-exceptions

Release 1.0.0.post1

Nov 26, 2017

Contents

1 Configuration **3**
1.1 Multiple backends 3

2 Reporting exceptions **5**

3 Extending asphalt-exceptions **7**
3.1 Writing new reporter backends 7
3.2 Writing extras providers 7

4 Version history **9**

This Asphalt framework component provides a pluggable means to send exception reports to external services. Optionally, it can also install itself as the default handler for exceptions occurring in the event loop.

The following backends are provided out of the box:

- Sentry
- Raygun
- Standard library logging

Plugins can also be written to provide context specific custom data for each backend.

CHAPTER 1

Configuration

You will need to install the library with the appropriate extras in order to use the backend you want. For example, to use the Sentry backend, do this:

```
pip install asphalt-exceptions[sentry]
```

The minimal configuration for Sentry integration would be like this:

```
components:
  exceptions:
    backend: sentry
    dsn: https://
↪d8e5bbc3aca4bccfc09690fd4cae45a3:47d2ecdd2bec818861db1db62cafd8d4@sentry.io/111111
```

And if you want to use Raygun instead:

```
components:
  exceptions:
    backend: raygun
    api_key: your_api_key_here
```

1.1 Multiple backends

You are not limited to using just one backend. To configure multiple backends, you can do this:

```
components:
  exceptions:
    reporters:
      sentry:
        dsn: https://
↪d8e5bbc3aca4bccfc09690fd4cae45a3:47d2ecdd2bec818861db1db62cafd8d4@sentry.io/111111
      raygun:
        api_key: your_api_key_here
```

Consult the API documentation of each backend class for details on the configuration options.

CHAPTER 2

Reporting exceptions

When an exception is caught, the typical course of action is to log it:

```
async with Context() as ctx:
    ...
    try:
        do_something()
    except Exception:
        logger.exception('Tried to do something but it failed :(')
```

To take advantage of the exception reporters configured with this component, all you have to do is call `report_exception()` instead:

```
from asphalt.exceptions import report_exception


async with Context() as ctx:
    ...
    try:
        do_something()
    except Exception:
        report_exception(ctx, 'Tried to do something but it failed :(')
```

This will not only log the exception as usual, but also send it to any external services represented by the configured exception reporter backends.

The `ctx` argument is required in order for the function to find the configured exception reporter resources. Additionally, it looks up plugins matching the fully qualified class name of the context object to provide additional information to each exception reporter backend.

CHAPTER 3

Extending asphalt-exceptions

3.1 Writing new reporter backends

To support new exception reporting services, you can subclass the `ExceptionReporter` class. You just need to implement the `report_exception()` method.

If you want your exception reporter to be available as a backend for `ExceptionReporterComponent`, you need to add the corresponding entry point for it. Suppose your exception reporter class is named `MyExceptionReporter` and it lives in the package `foo.bar.myreporter` and you want to give it the alias `myreporter`, then add this line to your project's `setup.py` under the `entry_points` argument in the `asphalt.exceptions.reporters` namespace:

```
setup(
    # (...other arguments...)
    entry_points={
        'asphalt.exceptions.reporters': [
            'myreporter = foo.bar.myreporter:MyExceptionReporter'
        ]
    }
)
```

Or in `setup.cfg`:

```
[options.entry_points]
asphalt.exceptions.reporters =
    myreporter = foo.bar.myreporter:MyExceptionReporter
```

3.2 Writing extras providers

If you want to provide backend specific extra data for exception reporting, you can do so by subclassing `ExtrasProvider` and adding one or more instances of it as resources to the context.

For example, if you wanted to provide extra data for Sentry about your custom context (`MyContext`), you could do write a provider like this:

```
from asphalt.exceptions.api import ExtrasProvider
from asphalt.exceptions.reporters.sentry import SentryExceptionReporter


class MyExtrasProvider(ExtrasProvider):
    def get_extras(ctx, reporter):
        if isinstance(ctx, MyContext) and isinstance(reporter,␣
↪SentryExceptionReporter):
            return {
                'time_spent': 1265,
                'data': {
                    'user': {'email': 'foo@example.org'}
                },
                'tags': {'site': 'example.org'},
                'extra': {'foo': 'bar'}
            }
```

And then during the startup of your component:

```
from asphalt.exceptions.api import ExtrasProvider


class MyComponent(Component):
    ...
    async def start(ctx):
        ...
        ctx.add_resource(MyExtrasProvider(), types=[ExtrasProvider])
```

CHAPTER 4

Version history

This library adheres to Semantic Versioning.

1.0.0 (2017-11-26)

- Initial release
- API reference