grpc-interceptor

Dan Hipschman

Jul 19, 2020

CONTENTS

CHAPTER

ONE

REFERENCE

1.1 grpc_interceptor.base

Base class for server-side interceptors.

class grpc_interceptor.base.**Interceptor**

Base class for server-side interceptors.

To implement an interceptor, subclass this class and override the intercept method.

abstract intercept(*method: Callable*, *request: Any*, *context: grpc.ServicerContext*, *method_name: str*) → Any

Override this method to implement a custom interceptor.

You should call method(request, context) to invoke the next handler (either the RPC method implementation, or the next interceptor in the list).

Parameters

- **method** – Either the RPC method implementation, or the next interceptor in the chain.
- **request** – The RPC request, as a protobuf message.
- **context** – The ServicerContext pass by gRPC to the service.
- **method_name** – A string of the form "/protobuf.package.Service/Method"

Returns This should generally return the result of method(request, context), which is typically the RPC method response, as a protobuf message. The interceptor is free to modify this in some way, however.

intercept_service(*continuation*, *handler_call_details*)

Implementation of grpc.ServerInterceptor.

This is not part of the Interceptor API, but must have a public name. Do not override it, unless you know what you're doing.

1.2 grpc_interceptor.exception_to_status

ExceptionToStatusInterceptor catches GrpcException and sets the gRPC context.

class grpc_interceptor.exception_to_status.**ExceptionToStatusInterceptor**
An interceptor that catches exceptions and sets the RPC status and details.

ExceptionToStatusInterceptor will catch any subclass of GrpcException and set the status code and details on the gRPC context.

intercept(*method: Callable*, *request: Any*, *context: grpc.ServicerContext*, *method_name: str*) → Any
Do not call this directly; use the interceptor kwarg on grpc.server().

1.3 grpc_interceptor.exceptions

Exceptions for ExceptionToStatusInterceptor.

See https://grpc.github.io/grpc/core/md_doc_statuscodes.html for the source of truth on status code meanings.

exception grpc_interceptor.exceptions.**Aborted**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)
The operation was aborted.

Typically this is due to a concurrency issue such as a sequencer check failure or transaction abort. See the guidelines on other exceptions for deciding between FAILED_PRECONDITION, ABORTED, and UNAVAILABLE.

exception grpc_interceptor.exceptions.**AlreadyExists**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)
The entity that a client attempted to create already exists.

E.g., a file or directory that a client is trying to create already exists.

exception grpc_interceptor.exceptions.**Cancelled**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)
The operation was cancelled, typically by the caller.

exception grpc_interceptor.exceptions.**DataLoss**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)
Unrecoverable data loss or corruption.

exception grpc_interceptor.exceptions.**DeadlineExceeded**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)
The deadline expired before the operation could complete.

For operations that change the state of the system, this error may be returned even if the operation has completed successfully. For example, a successful response from a server could have been delayed long.

exception grpc_interceptor.exceptions.**FailedPrecondition**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)
The operation failed because the system is in an invalid state for execution.

For example, the directory to be deleted is non-empty, an rmdir operation is applied to a non-directory, etc. Service implementors can use the following guidelines to decide between FAILED_PRECONDITION, ABORTED, and UNAVAILABLE: (a) Use UNAVAILABLE if the client can retry just the failing call. (b) Use ABORTED if the client should retry at a higher level (e.g., when a client-specified test-and-set fails, indicating the client should restart a read-modify-write sequence). (c) Use FAILED_PRECONDITION if the client should not retry until the system state has been explicitly fixed. E.g., if an "rmdir" fails because the directory is non-empty, FAILED_PRECONDITION should be returned since the client should not retry unless the files are deleted from the directory.

exception grpc_interceptor.exceptions.**GrpcException**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)

Base class for gRPC exceptions.

Generally you would not use this class directly, but rather use a subclass representing one of the standard gRPC status codes (see: https://grpc.github.io/grpc/core/md_doc_statuscodes.html for the official list).

status_code
A grpc.StatusCode other than OK. The only use case for this is if gRPC adds a new status code that isn't represented by one of the subclasses of GrpcException. Must not be OK, because gRPC will not raise an RpcError to the client if the status code is OK.

details
A string with additional informantion about the error.

exception grpc_interceptor.exceptions.**Internal**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)

Internal errors.

This means that some invariants expected by the underlying system have been broken. This error code is reserved for serious errors.

exception grpc_interceptor.exceptions.**InvalidArgument**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)

The client specified an invalid argument.

Note that this differs from FAILED_PRECONDITION. INVALID_ARGUMENT indicates arguments that are problematic regardless of the state of the system (e.g., a malformed file name).

exception grpc_interceptor.exceptions.**NotFound**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)

Some requested entity (e.g., file or directory) was not found.

Note to server developers: if a request is denied for an entire class of users, such as gradual feature rollout or undocumented whitelist, NOT_FOUND may be used. If a request is denied for some users within a class of users, such as user-based access control, PERMISSION_DENIED must be used.

exception grpc_interceptor.exceptions.**OutOfRange**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)

The operation was attempted past the valid range.

E.g., seeking or reading past end-of-file. Unlike INVALID_ARGUMENT, this error indicates a problem that may be fixed if the system state changes. For example, a 32-bit file system will generate INVALID_ARGUMENT if asked to read at an offset that is not in the range [0,2^32-1], but it will generate OUT_OF_RANGE if asked to read from an offset past the current file size. There is a fair bit of overlap

between FAILED_PRECONDITION and OUT_OF_RANGE. We recommend using OUT_OF_RANGE (the more specific error) when it applies so that callers who are iterating through a space can easily look for an OUT_OF_RANGE error to detect when they are done.

exception grpc_interceptor.exceptions.**PermissionDenied**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)

The caller does not have permission to execute the specified operation.

PERMISSION_DENIED must not be used for rejections caused by exhausting some resource (use RESOURCE_EXHAUSTED instead for those errors). PERMISSION_DENIED must not be used if the caller can not be identified (use UNAUTHENTICATED instead for those errors). This error code does not imply the request is valid or the requested entity exists or satisfies other pre-conditions.

exception grpc_interceptor.exceptions.**ResourceExhausted**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)

Some resource has been exhausted.

Perhaps a per-user quota, or perhaps the entire file system is out of space.

exception grpc_interceptor.exceptions.**Unauthenticated**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)

The request does not have valid authentication credentials for the operation.

exception grpc_interceptor.exceptions.**Unavailable**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)

The service is currently unavailable.

This is most likely a transient condition, which can be corrected by retrying with a backoff. Note that it is not always safe to retry non-idempotent operations.

exception grpc_interceptor.exceptions.**Unimplemented**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)

The operation is not implemented or is not supported/enabled in this service.

exception grpc_interceptor.exceptions.**Unknown**(*details: Optional[str] = None*, *status_code: Optional[grpc.StatusCode] = None*)

Unknown error.

For example, this error may be returned when a Status value received from another address space belongs to an error space that is not known in this address space. Also errors raised by APIs that do not return enough error information may be converted to this error.

CHAPTER

TWO

LICENSE

MIT License

Copyright (c) 2020 Dan Hipschman

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The primary aim of this project is to make Python gRPC interceptors simple. The Python `grpc` package provides service interceptors, but they're a bit hard to use because of their flexibility. The `grpc` interceptors don't have direct access to the request and response objects, or the service context. Access to these are often desired, to be able to log data in the request or response, or set status codes on the context.

The secondary aim of this project is to keep the code small and simple. Code you can read through and understand quickly gives you confidence and helps debug issues. When you install this package, you also don't want a bunch of other packages that might cause conflicts within your project. Too many dependencies also slow down installation as well as runtime (fresh imports take time). Hence, a goal of this project is to keep dependencies to a minimum. The only dependency is the `grpc` package.

The `grpc_interceptor` package provides the following:

- An `Interceptor` base class, to make it easy to define your own service interceptors.
- An `ExceptionToStatusInterceptor` interceptor, so your service can raise exceptions that set the gRPC status code correctly (rather than the default of every exception resulting in an `UNKNOWN` status code). This is something for which pretty much any service will have a use.

CHAPTER

THREE

INSTALLATION

```
$ pip install grpc-interceptor
```

CHAPTER

FOUR

USAGE

To define your own interceptor (we can use `ExceptionToStatusInterceptor` as an example):

```
from grpc_interceptor.base import Interceptor

class ExceptionToStatusInterceptor(Interceptor):

    def intercept(
        self,
        method: Callable,
        request: Any,
        context: grpc.ServicerContext,
        method_name: str,
    ) -> Any:
        """Override this method to implement a custom interceptor.

        You should call method(request, context) to invoke the
        next handler (either the RPC method implementation, or the
        next interceptor in the list).

        Args:
            method: The next interceptor, or method implementation.
            request: The RPC request, as a protobuf message.
            context: The ServicerContext pass by gRPC to the service.
            method_name: A string of the form
                "/protobuf.package.Service/Method"

        Returns:
            This should generally return the result of
            method(request, context), which is typically the RPC
            method response, as a protobuf message. The interceptor
            is free to modify this in some way, however.
        """
        try:
            return method(request, context)
        except GrpcException as e:
            context.set_code(e.status_code)
            context.set_details(e.details)
            raise
```

Then inject your interceptor when you create the `grpc` server:

```
interceptors = [ExceptionToStatusInterceptor()]
server = grpc.server(
    futures.ThreadPoolExecutor(max_workers=10),
```

(continues on next page)

(continued from previous page)

```
    interceptors=interceptors
)
```

To use `ExceptionToStatusInterceptor`:

```
from grpc_interceptor.exceptions import NotFound

class MyService(my_pb2_grpc.MyServiceServicer):
    def MyRpcMethod(
        self, request: MyRequest, context: grpc.ServicerContext
    ) -> MyResponse:
        thing = lookup_thing()
        if not thing:
            raise NotFound("Sorry, your thing is missing")
        ...
```

This results in the gRPC status status code being set to `NOT_FOUND`, and the details `"Sorry, your thing is missing"`. This saves you the hassle of catching exceptions in your service handler, or passing the context down into helper functions so they can call `context.abort` or `context.set_code`. It allows the more Pythonic approach of just raising an exception from anywhere in the code, and having it be handled automatically.

CHAPTER

FIVE

LIMITATIONS

These are the current limitations, although supporting these is possible. Contributions or requests are welcome.

- `Interceptor` currently only supports unary-unary RPCs.
- The package only provides service interceptors.

PYTHON MODULE INDEX

g

INDEX