fieldmarshal

Stanis Trendelenburg

May 01, 2021

CONTENTS:

CHAPTER

ONE

API REFERENCE

fieldmarshal.**struct**(**args*, ***kw*)
Wrapper around `attr.s`.

Sets `slots=True` and `auto_attribs=True`. All other arguments are forwared to `attr.s`.

fieldmarshal.**field**(*name=None*, *omit=False*, *omit_if_none=False*, *marshal=None*, *unmarshal=None*, ***kw*)
Wrapper around `attr.ib` that accepts additional arguments.

Arguments that control marshalling are saved as an *Options* object in the fields metadata under the "fieldmarshal" key. See *Options* for the meaning of these parameters.

class fieldmarshal.**Hook**(*fn: Any*, *takes_args: bool = True*)
Container for marshal or unmarshal hooks.

Parameters

- **fn** (*callable*) – Marshal hook
- **takes_args** (*bool*) – Whether or not *fn* takes additional arguments

When *takes_args* is `True` (the default), additional arguments will be passed to the hook. The type of arguments depends on the type of hook. See *Registry.add_marshal_hook()* and *Registry.add_unmarshal_hook()* for details.

class fieldmarshal.**Options**(*name: str = None*, *omit: bool = False*, *omit_if_none: bool = False*, *marshal: Any = None*, *unmarshal: Any = None*)
Field options control how a field is marshalled/unmarshalled.

Parameters

- **name** (*str*) – Rename the field when marshalling/unmarshalling. Useful for example for JSON attribute names that are not valid Python identifiers.
- **omit** (*bool*) – Ignore the field for the purpose of marshalling/unmarshalling. The field value will neither be read nor written to. The field should also have a default value for the class to support unmarshalling.
- **omit_if_none** (*bool*) – Omit the field when marshalling if it's value is `None`. When unmarshalling, the field will be read as normal.
- **marshal** (*callable*) – A function to call to marshal the contents of this field. The function will be passed the field value as its only argument. This hook overrides other marshal hooks registered for the field's type.
- **unmarshal** (*callable*) – A function to call to unmarshal data for this field. The function will be passed the data being unmarshalled as its only argument. This hook overrides other unmarshal hooks registered for the field's type.

For `fieldmarshal` to recognize these options, put the object into the field's `metadata` dict under the "fieldmarshal" key, or use the *`field()`* helper in place of `attr.s`.

class fieldmarshal.**Registry**

Create a registry instance.

A registry is used for marshalling and unmarshalling objects, and for registering hooks for types that are not handled natively.

add_marshal_hook(*type_*, *fn*)

Add a custom marshal implementation for a type.

The hook can either be a function that takes one argument (the object being marshalled), or a *`Hook`* object, which can be used to opt-in to receive the registry instance as an additional argument.

The hook should return a JSON-compatible object. The hook will be called when an object of type *type_* is encountered when marshalling. The hook will also be used for instances of subclasses of *type_*, unless a more specific hook can be found.

add_unmarshal_hook(*type_*, *fn*)

Add a custom unmarshal implementation for a type.

*type_* can be a class or a concrete type from the `typing` module, such as `Union[list, str]`. The hook can either be a function that takes one argument (the object being unmarshalled), or a *`Hook`* object, which can be used to opt-in to receive the type the object is being unmarshalled to and the registry instance as additional arguments. The type passed this way is not necessarily the type the hook was registered for (it could be a subclass, for example).

The hook will be called when data needs to be marshalled to an object of type *type_*. If *type_* is a class, the hook will also be used for unmarshalling to subclasses of *type_*, unless a more specific hook can be found.

clear_cache()

Clear all caches of the registry.

This should not be necessary unless classes are modified at runtime.

lookup_marshal_impl(*cls*)

Return the marshal implementation objects of type *cls*.

lookup_unmarshal_impl(*cls*, *type_hint*)

Return the implementation and resolved type for unmarshalling data of type *cls* to an object of type *type_int*.

The resolved type is the same as *type_hint*, except for union types, where it is one of the members of the union.

marshal(*obj*)

Marshal an object to a JSON-compatible data structure.

The resulting data structure contains only objects of type `list`, `dict` (with string keys), `int`, `float`, `str`, `bool` or `NoneType` and can be converted to JSON without further modifications.

Raises `MarshalError` if an object is encountered that cannot be marshalled.

The reverse operation is *`unmarshal()`*.

marshal_json(*obj*)

Marshal an object to a JSON string.

Like *`marshal()`*, but converts the result to JSON.

unmarshal(*obj*, *type_hint*)

Unmarshal an object from a JSON-compatible data structure.

The data structure must contain only objects of type `list`, `dict` (with `str` keys), `int`, `float`, `str`, `bool` or `NoneType`, such as returned by *marshal()*.

type_hint specifies the type of object to create. This can be a class or a concrete type from the `typing` module, such as `List[int]`.

Raises `UnmarshalError` if the data cannot be unmarshalled to the desired type.

The reverse operation is *marshal()*.

unmarshal_json(*data*, *type_hint*)
Unmarshal an object from a JSON string.

Like *unmarshal()*, but accepts a data in JSON format.

fieldmarshal.**marshal**()

Standalone version of *Registry.marshal()* that uses the default registry.

fieldmarshal.**marshal_json**()

Standalone version of *Registry.marshal_json()* that uses the default registry.

fieldmarshal.**unmarshal**()

Standalone version of *Registry.unmarshal()* that uses the default registry.

fieldmarshal.**unmarshal_json**()

Standalone version of *Registry.unmarshal_json()* that uses the default registry.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX