docs.jinkan.org

沙箱 — Jinja2 2.7 documentation

24-30 分钟

Jinja2 沙箱用于为不信任的代码求值。访问不安全的属性和方法是被禁止的。

假定在默认配置中 *env* 是一个 `SandboxedEnvironment` 实例，下面的代码展示了它如何工作:

```
>>> env.from_string("{{ func.func_code }}").render(func=lambda:None)
u''
>>> env.from_string("{{ func.func_code.do_something
}}").render(func=lambda:None)
Traceback (most recent call last):
  ...
SecurityError: access to attribute 'func_code' of 'function' object is
unsafe.
```

API¶

class `jinja2.sandbox.SandboxedEnvironment`([*options*])¶

The sandboxed environment. It works like the regular environment but tells the compiler to generate sandboxed code. Additionally subclasses of this environment may override the methods that tell the runtime what attributes or functions are safe to access.

If the template tries to access insecure code a `SecurityError` is raised. However also other exceptions may occour during the rendering so the caller has to ensure that all exceptions are catched.

`call_binop`(*context*, *operator*, *left*, *right*)¶

For intercepted binary operator calls (`intercepted_binops()`) this function is executed instead of the builtin operator. This can be used to fine tune the behavior of certain operators.

New in version 2.6.

`call_unop`(*context*, *operator*, *arg*)¶

For intercepted unary operator calls (`intercepted_unops()`) this function is executed instead of the builtin operator. This can be used to fine tune the behavior of certain operators.

New in version 2.6.

`default_binop_table` = *{'//': <built-in function floordiv>, '%': <built-in function mod>, '+':*

<built-in function add>, '': <built-in function mul>, '-': <built-in function sub>, '/': <built-in function truediv>, '**': <built-in function pow>}*¶

default callback table for the binary operators. A copy of this is available on each instance of a sandboxed environment as `binop_table`

`default_unop_table` = *{'+': <built-in function pos>, '-': <built-in function neg>}*¶

default callback table for the unary operators. A copy of this is available on each instance of a sandboxed environment as `unop_table`

`intercepted_binops` = *frozenset([])*¶

a set of binary operators that should be intercepted. Each operator that is added to this set (empty by default) is delegated to the `call_binop()` method that will perform the operator. The default operator callback is specified by `binop_table`.

The following binary operators are interceptable: `//`, `%`, `+`, `*`, `-`, `/`, and `**`

The default operation form the operator table corresponds to the builtin function. Intercepted calls are always slower than the native operator call, so make sure only to intercept the ones you are interested in.

New in version 2.6.

`intercepted_unops` = *frozenset([])*¶

a set of unary operators that should be intercepted. Each operator that is added to this set (empty by default) is delegated to the `call_unop()` method that will perform the operator. The default operator callback is specified by `unop_table`.

The following unary operators are interceptable: `+`, `-`

The default operation form the operator table corresponds to the builtin function. Intercepted calls are always slower than the native operator call, so make sure only to intercept the ones you are interested in.

New in version 2.6.

`is_safe_attribute`(*obj*, *attr*, *value*)¶

The sandboxed environment will call this method to check if the attribute of an object is safe to access. Per default all attributes starting with an underscore are considered private as well as the special attributes of internal python objects as returned by the `is_internal_attribute()` function.

`is_safe_callable`(*obj*)¶

Check if an object is safely callable. Per default a function is considered safe unless the *unsafe_callable* attribute exists and is True. Override this method to alter the behavior, but this won’t affect the *unsafe* decorator from this module.

class `jinja2.sandbox.ImmutableSandboxedEnvironment`([*options*])¶

Works exactly like the regular *SandboxedEnvironment* but does not permit modifications on the builtin mutable objects *list*, *set*, and *dict* by using the `modifies_known_mutable()` function.

exception `jinja2.sandbox.SecurityError`(*message=None*)¶

Raised if a template tries to do something insecure if the sandbox is enabled.

`jinja2.sandbox.unsafe`(*f*)¶

Marks a function or method as unsafe.

```
@unsafe
def delete(self):
    pass
```

`jinja2.sandbox.is_internal_attribute`(*obj*, *attr*)¶

Test if the attribute given is an internal python attribute. For example this function returns *True* for the *func_code* attribute of python objects. This is useful if the environment method `is_safe_attribute()` is overridden.

```
>>> from jinja2.sandbox import is_internal_attribute
>>> is_internal_attribute(lambda: None, "func_code")
```

```
True
>>> is_internal_attribute((lambda x:x).func_code, 'co_code')
True
>>> is_internal_attribute(str, "upper")
False
```

`jinja2.sandbox.modifies_known_mutable`(*obj*, *attr*)¶

This function checks if an attribute on a builtin mutable object (list, dict, set or deque) would modify it if called. It also supports the "user"-versions of the objects (*sets.Set*, *UserDict.** etc.) and with Python 2.6 onwards the abstract base classes *MutableSet*, *MutableMapping*, and *MutableSequence*.

```
>>> modifies_known_mutable({}, "clear")
True
>>> modifies_known_mutable({}, "keys")
False
>>> modifies_known_mutable([], "append")
True
>>> modifies_known_mutable([], "index")
False
```

If called with an unsupported object (such as unicode) *False* is returned.

```
>>> modifies_known_mutable("foo", "upper")
False
```

提示

Jinja2 沙箱自己并没有彻底解决安全问题。特别是对 web 应用，你必须晓得用户可能用任意 HTML 来创建模板，所以保证他们不通过注入 JavaScript 或其它更多方法来互相损害至关重要（如果你在同一个服务器上运行多用户）。

同样，沙箱的好处取决于配置。我们强烈建议只向模板传递非共享资源，并且使用某种属性白名单。

也请记住，模板会抛出运行时或编译期错误，确保捕获它们。

运算符拦截¶

New in version 2.6.

为了性能最大化， Jinja2 会让运算符直接条用类型特定的回调方法。这意味着，通过重载 `Environment.call()` 来拦截是不可能的。此外，由于运算符的工作方式，把运算符转换为特殊方法不总是直接可行的。比如为了分类，至少一个特殊方法存在。

在 Jinja 2.6 中，开始支持显式的运算符拦截。必要时也可以用于自定义的特定运算符。为了拦截运

算符，需要覆写 `SandboxedEnvironment.intercepted_binops` 属性。当需要拦截的运算符被添加到这个集合， Jinja2 会生成调用 `SandboxedEnvironment.call_binop()` 函数的字节码。对于一元运算符，必须替代地使用 *unary* 属性和方法。

`SandboxedEnvironment.call_binop` 的默认实现会使用 `SandboxedEnvironment.binop_table` 来把运算符标号翻译成执行默认运算符行为的回调。

这个例子展示了幂（ `**` ）操作符可以在 Jinja2 中禁用:

```
from jinja2.sandbox import SandboxedEnvironment


class MyEnvironment(SandboxedEnvironment):
    intercepted_binops = frozenset(['**'])

    def call_binop(self, context, operator, left, right):
        if operator == '**':
            return self.undefined('the power operator is unavailable')
        return SandboxedEnvironment.call_binop(self, context,
                                               operator, left, right)
```

确保始终调入 super 方法，即使你不拦截这个调用。 Jinja2 内部会调用这个方法来对表达式求值。