方法__init__()中会根据点号对 var 进行切割，将其转成元组后赋给 lookups 属性，结果如下：

```
>>> from django.template.base import Variable
>>> var = Variable(var='person.name')
>>> var.lookups
('person', 'name')
>>> var.literal is None
True
```

从上面的初始化过程可以知道，对于不用渲染的结果，比如数值字符串、带引号的字符串等，将直接赋给 Variable 对象的 literal 属性。而对于需要根据 context 值进行渲染的，则将相关变量信息保存在 lookup 属性中，最后统一由 resolve()方法渲染得到结果。resolve()方法的实现源码如下：

```
# 源码位置：django/template/base.py
# ……

class Variable:
    # ……

    def resolve(self, context):
        if self.lookups is not None:
            # 继续调用_resolve_lookup()方法渲染变量结果
            value = self._resolve_lookup(context)
        else:
            # 如果 self.lookups 的值为 None，则表示不需要渲染
            value = self.literal
        if self.translate:
            #当 var 字符串以'_('开头且以')'结尾时，self.translate 值会被设置为 True
            # 通常情况下不会执行这里的代码
            is_safe = isinstance(value, SafeData)
            msgid = value.replace('%', '%%')
            msgid = mark_safe(msgid) if is_safe else msgid
            if self.message_context:
                return pgettext_lazy(self.message_context, msgid)
            else:
                return gettext_lazy(msgid)
        # 最终返回 value 结果，即渲染的最终结果
        return value

    # ……

# ……
```

resolve()方法的处理逻辑非常简单，根据前面的分析可知，当 Variable 对象的 lookups 属性值不为空时，需要结合 context 中的 key-value 值渲染模板变量，对于变量或者多级变量的情况，最终调用_resolve_lookup()方法渲染结果。_resolve_lookup()方法的实现源码如下：