面向对象式设计：

```python
class Checker(object):
    def check(self, context):
        pass

class EmailChecker(Checker):
    def check(self, context):
        if not context['email'].lower() == 'zanxiaofeng@163.com':
            return "Email is error"

class AccessCodeChecker(Checker):
    def check(self, context):
        if not context['access_code'].lower().startswith('wg'):
            return "AccessCode is error"

class ComposedChecker(Checker):
    def __init__(self, *checkers):
        self.checkers = checkers

    def check(self, context):
        for checker in self.checkers:
            result = checker.check(context)
            return result if result else None

class EmailAndAccessCodeChecker(ComposedChecker):
    def __init__(self):
        super().__init__(EmailChecker(), AccessCodeChecker())

checker = EmailAndAccessCodeChecker()
assert checker.check({'email': '', 'access_code': 'wg_123'}) == "Email is error"
assert checker.check({'email': 'zanxiaofeng@163.com', 'access_code': 'wg_123'}) is None
```

面向函数式设计：

```python
def email_check(context):
    if not context['email'].lower() == 'zanxiaofeng@163.com':
        return "Email is error"



def access_code_check(context):
    if not context['access_code'].lower().startswith('wg'):
        return "AccessCode is error"



def execute_checks(context, *checks):
    for check in checks:
        result = check(context)
        if result:
            return result



def email_and_access_code_check(context):
    return execute_checks(context, email_check, access_code_check)

assert email_and_access_code_check({'email': '', 'access_code': 'wg_123'}) == "Email is er
assert email_and_access_code_check({'email': 'zanxiaofeng@163.com', 'access_code': 'wg_123
```