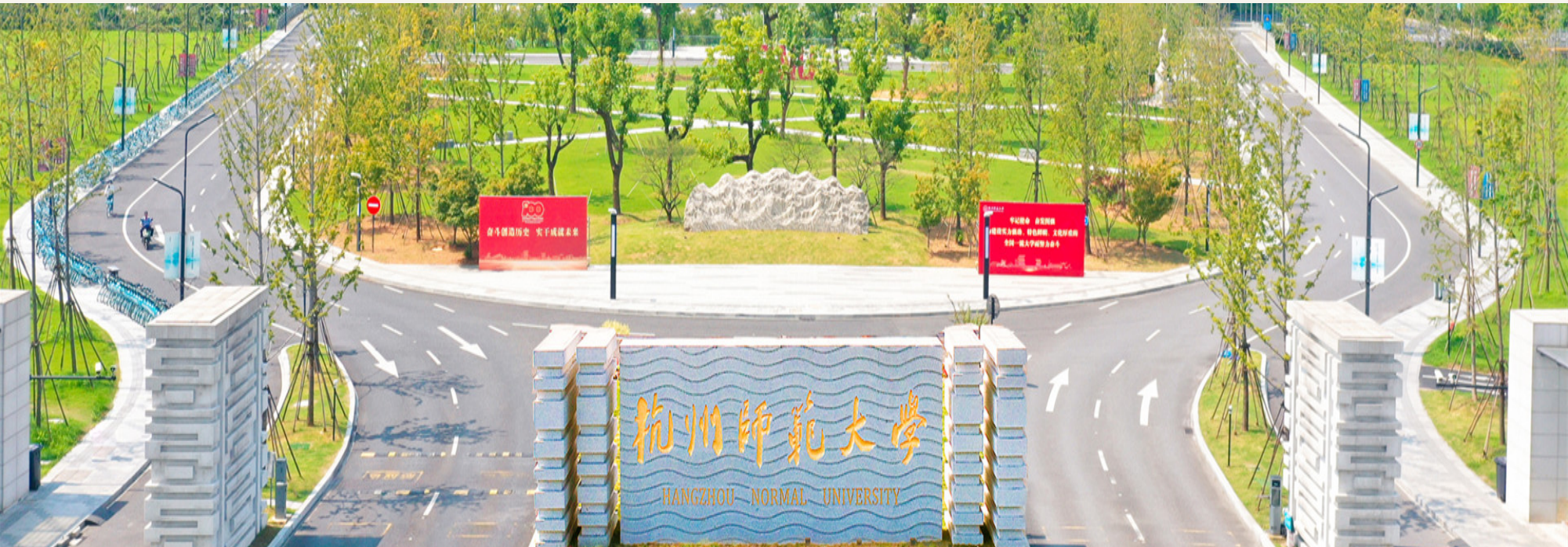
第六章 - 集合 (编程练习)

张建章

阿里巴巴商学院

杭州师范大学

2025-09



代码示例

- 集合是Python中的一种非序列可变数据类型，存储多个无序元素；
- 集合中的元素是唯一的，即不存在重复值，这使其非常适合执行诸如去重、成员测试等操作；
- 集合中的每个元素必须是不可变的类型，如，整数、字符串等；
- 集合支持多种集合运算，如并集、交集、差集和对称差集。

创建集合有两种主要方法：使用花括号 `{}` 直接定义集合，或使用内置的 `set()` 函数。这两种方法均可用于构建一个包含唯一元素的集合。

1. **使用花括号定义集合**：将元素放在花括号内并用逗号分隔即可。

```
fruits = {"apple", "banana", "cherry"}
print(fruits)  # 输出: {'apple', 'banana', 'cherry'}
```

集合不允许重复元素，如在定义时加入重复值，它们将被自动去除。

2. 使用 `set()` 函数创建集合

适合需要从其他可迭代对象（如列表或元组）转换为集合的情况。可以将一个可迭代对象作为参数传递给 `set()` 函数来创建集合：

```
numbers = set([1, 2, 3, 3, 4])
print(numbers)  # 输出: {1, 2, 3, 4}
```

列表中的重复值 `3` 被移除，最终得到一个只包含唯一元素的集合。

注意：使用花括号创建空集合并不可行，因为 `{}` 默认创建一个空字典。若要创建一个空集合，必须使用 `set()` 函数：

```
empty_set = set()
print(empty_set)  # 输出: set()
```

集合的基本操作包括并集、交集、差集和对称差集，这些操作可以通过运算符或集合的方法来实现。这些集合操作为数据分析和大数据管理中的数据去重、交叉比对及合并操作提供了高效工具。

1. **并集（Union）**

并集操作返回两个集合中所有不重复的元素。可以使用 `|` 运算符或 `union()` 方法：

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 | set2)  # 输出: {1, 2, 3, 4, 5}
print(set1.union(set2))  # 输出: {1, 2, 3, 4, 5}
```

运算符 `|` 和方法 `union()` 功能相同，但 `union()` 方法可以接受其他可迭代对象（如列表）。

2. 交集（Intersection）

返回两个集合中的公共元素。可以使用 `&` 运算符或 `intersection()` 方法，两种方式均支持同时对多个集合进行操作。

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
print(set1 & set2)  # 输出: {3, 4}
print(set1.intersection(set2))  # 输出: {3, 4}
```

3. 差集（Difference）

差集操作返回一个集合中存在但另一个集合中不存在的元素。可以使用 `-` 运算符或 `difference()` 方法，差集运算结果依赖集合的顺序。

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5}
print(set1 - set2)  # 输出: {1, 2}
print(set1.difference(set2))  # 输出: {1, 2}
```

4. 对称差集（Symmetric Difference）

对称差集操作返回两个集合中不重复的元素。可以使用`^`运算符或`symmetric_difference()`方法：

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
print(set1 ^ set2)  # 输出: {1, 2, 5, 6}
print(set1.symmetric_difference(set2))  # 输出: {1, 2, 5, 6}
```

该操作适用于查找在两个集合中互不重复的元素。

代码示例

集合提供了多种内置方法来操作和管理数据，这些方法在数据分析、数据清洗和集合运算等场景中非常有用。

1. add()

`add()` 方法将一个元素添加到集合中，如果元素已经存在，则不会有任何变化，此方法适合在动态数据处理中逐个添加元素。

```
my_set = {1, 2, 3}
my_set.add(4)
print(my_set)  # 输出: {1, 2, 3, 4}
```

2. update()

`update()` 方法用于将其他可迭代对象（如列表、元组、字典等）的元素添加到当前集合中。如果添加的元素在集合中已存在，则该元素只会出现一次，重复的会被忽略。

```
# 创建一个包含水果名称的集合
fruits = {'apple', 'banana'}
# 定义一个包含新水果的列表
new_fruits = ['cherry', 'date', 'apple']
# 使用 update() 方法将新水果添加到集合中
fruits.update(new_fruits)
```

在上述示例中，`new_fruits` 列表中的元素被添加到 `fruits` 集合中。由于集合不允许重复元素，`apple` 在添加时被忽略。

2. remove() 和 discard()

从集合中删除指定的元素。如果该元素存在于集合中，则将其移除；如果元素不存在，`discard()` 不会进行任何操作，`remove()` 会引发 `KeyError` 异常。

```
# 创建一个包含水果名称的集合
fruits = {'apple', 'banana', 'cherry'}

# 移除存在的元素 'banana'
fruits.remove('banana')

# 尝试移除不存在的元素 'orange'
fruits.remove('orange')  # 引发 KeyError 异常

fruits.discard('orange')  # 不进行任何操作，也不会引发异常
```

使用 `discard()` 更为安全，因为可以避免潜在的错误。

3. pop()

`pop()` 方法用于从集合中移除并返回一个任意元素。由于集合是无序的，因此无法预测 `pop()` 方法会移除哪个元素。如果集合为空，调用 `pop()` 方法将引发 `KeyError` 异常。

```
# 创建一个包含水果名称的集合
fruits = {'apple', 'banana', 'cherry'}

# 使用 pop() 方法移除并返回一个任意元素
removed_fruit = fruits.pop()
print(f"被移除的水果: {removed_fruit}")
print(f"剩余的水果集合: {fruits}")
```

4. issubset() 和 issuperset()

`issubset()` 和 `issuperset()` 分别用于检查一个集合是否为另一个集合的子集或超集：

```
set1 = {1, 2}
set2 = {1, 2, 3}
print(set1.issubset(set2))   # 输出: True
print(set2.issuperset(set1))   # 输出: True
```

在数据分析中，这些方法可以帮助快速验证集合之间的包含关系。

5. clear()

`clear()` 方法清空集合中的所有元素，用于重置集合。

```
my_set = {1, 2, 3}
my_set.clear()
print(my_set)   # 输出: set()
```

`set` 和 `frozenset` 都是用来存储无序、唯一元素的集合数据类型，但 `set` 是可变的，而 `frozenset` 是不可变的。

`set`：适合在需要动态修改集合内容的场景下使用，当数据集合需要频繁更新或元素需要动态增删时，`set` 提供了灵活性。

`frozenset`：由于不可变性，它是哈希的，这使得它可以作为字典的键或其他集合的元素。

```
# 使用set
mutable_set = {1, 2, 3}
mutable_set.add(4)
print(mutable_set)  # 输出: {1, 2, 3, 4}

# 使用frozenset
immutable_frozenset = frozenset([1, 2, 3])
print(immutable_frozenset)  # 输出: frozenset({1, 2, 3})
# 试图修改frozenset将会引发错误
immutable_frozenset.add(4)  # AttributeError: 'frozenset' object
↪  has no attribute 'add'
```

THE END