Сообщить об ошибке.

Модуль mmap в Python, ускорение операций ввода-вывода

Увеличение производительности файловых операций ввода-вывода

Модуль mmap обеспечивает ввод и вывод файла с отображением памяти (I/O), что позволяет использовать функциональные возможности операционной системы более низкого уровня для чтения файлов. Такое поведение может значительно улучшить производительность кода, требующего большого количества файловых операций ввода-вывода.

Memory mapping - это метод, использующий низкоуровневые API-интерфейсы операционной системы для загрузки файла непосредственно в память компьютера. Подход mmap жертвует использованием памяти ради скорости, что классически называется компромиссом между пространством и временем. При отображении файла в память не требуется больше физической памяти, чем при традиционном подходе. Операционная система очень умная. Она будет лениво загружать данные по запросу, подобно тому, как работают генераторы Python.

Содержание:

Типы компьютерной памяти.

Чтобы лучше понять, как отображение в память повышает производительность, а также как и когда можно использовать модуль mmap для увеличения производительности, сначала разберемся как работает память в операционной системе.

Каждый тип памяти может вступить в игру, когда будет использоваться модуль mmap. Рассмотрим каждый из них.

Физическая (physical) память.

Физическая память обычно находится на картах, подключенных к материнской плате компьютера и представляет собой объем энергозависимой памяти, доступной для использования программами во время работы. Физическую память не следует путать с хранилищем, таким как жесткий диск или твердотельный диск.

Виртуальная (virtual) память.

Виртуальная память - это способ управления памятью. Операционная система начинает использовать виртуальную память в том случае, если физической памяти не хватает на какие-то операции ввода-вывода. За кулисами, для имитации дополнительной оперативной памяти, система использует часть энергонезависимого хранилища, такого как твердотельный диск. Для этого операционная система должна поддерживать сопоставление между физической памятью и виртуальной памятью. Каждая операционная система использует свой собственный сложный алгоритм для сопоставления адресов виртуальной памяти с физическими с помощью структуры данных, называемой таблицей страниц.

Модуль mmap использует *VIRTUAL* память для загрузки очень больших файлов, даже если содержимое файла больше физической памяти.

Разделяемая (shared) память.

Разделяемая память - это еще одна технология, которая позволяет нескольким программам одновременно обращаться к одним и тем же данным. Разделяемая память может быть очень эффективным способом обработки данных в программе, использующей потоковую или многопроцессорную обработку данных.

Модуль mmap использует *SHARED* память для эффективного обмена большими объемами данных между несколькими процессами в Python, потоками и задачами, которые выполняются одновременно.

Производительность mmap на примере чтения файла.

Подход "Memory mapping" немного сложнее, чем типичный файловый ввод-вывод, так как требует создания объекта mmap. Но это небольшое изменение может привести к значительному повышению производительности при чтении файла размером всего в несколько мегабайт.

Использование mmap очень похоже на традиционный способ чтения файла с несколькими небольшими изменениями:

- Открыть файл с помощью open() недостаточно. Дополнительно нужно использовать класс mmap.mmap(), чтобы сообщить операционной системе, о необходимости отображения файла в ОЗУ.
- Необходимо убедиться, что режим, который использует функция open(), совместим с mmap.mmap(). Режим по умолчанию для open() предназначен только для чтения, но режим по умолчанию для mmap.mmap() предназначен для чтения и записи. Таким образом, для mmap режим открытии файла нужно указывать явно.
- Необходимо выполнять все операции чтения и записи с использованием объекта mmap вместо стандартного файлового объекта, возвращаемого функцией open().

```python
# test.py
import mmap

def open_io(filename):
    """Чтение файла традиционным способом"""
    with open(filename, mode="r", encoding="utf8") as fp:
        fp.read()

def mmap_io(filename):
    """Чтение файла с использованием модуля `mmap`"""
    with open(filename, mode="r", encoding="utf8") as fp:
        with mmap.mmap(fp.fileno(), length=0, access=mmap.ACCESS_READ) as mobj:
            mobj.read()
```

Сохраним код выше в файл test.py и запустим его в интерактивном режиме: $python3 -i test.py.

```
>>> import timeit
# здесь используется текстовый файл, объемом 2,6 Mb,
# расположенный на быстром накопителе NVMe
>>> filename = '/path/to/file.txt'
# Чтение файла традиционным способом
>>> timeit.repeat(
...     "open_io(filename)",
...     repeat=3,
...     number=1,
...     setup="from __main__ import open_io, filename")
# [0.005494390999956522, 0.004731214001367334, 0.0036292479999247007]

# Чтение файла с использованием модуля `mmap`
>>> timeit.repeat(
...     "mmap_io(filename)",
...     repeat=3,
...     number=1,
...     setup="from __main__ import mmap_io, filename")
# [0.0013046439999016002, 0.0010477209980308544, 0.0010247729987895582]
```

Код измеряет количество времени, необходимое для чтения всего 2,6-мегабайтного файла с использованием обычного файлового ввода-вывода и файлового ввода-вывода с использованием модуля mmap. Как можно видеть, подход *Memory mapping* занимает в 5 раз меньше по сравнению с обычным подходом чтения файла. Обратите внимание, что улучшение производительности будет еще больше при чтении с обычного диска SATA или SSD и особенно больших файлов.

Примечание. Эти результаты были получены с использованием OS lunux и Python 3.8, так как подход *Memory mapping* сильно зависит от реализации операционной системы.

API, предоставляемый объектом mmap, очень похож на традиционный файловый объект, за исключением одной дополнительной возможности: для объекта mmap можно использовать срезы так же просто, как для строковых объектов!

Описание класса mmap.mmap().

Можно использовать объекты MMAP в большинстве мест, где ожидается использование bytearray; например, можно использовать модуль re для поиска в отображенном в памяти файле. Также можно изменить один байт, выполнив MMAP[index] = 97, или изменить подпоследовательность, назначив срезу: MMAP[i1:i2] = b'...'. Можно читать и записывать данные, начиная с текущей позиции, и передвигаться MMAP.seek() по файлу в разные позиции.

Файл с создается конструктором mmap.mmap(), который отличается в Unix и Windows. В любом случае необходимо указать дескриптор файла, открытого для обновления. Аргумент fileno (дескриптор файла) можно извлечь из файлового объекта, полученного при открытии файла функцией open(), использовав метод file.fileno(). В противном случае можно открыть файл с помощью функции os.open(), которая напрямую возвращает дескриптор файла (по завершении файл все равно должен быть закрыт).

Примечание. Если нужно создать отображение памяти для доступного для записи буферизованного файла, то необходимо сначала выполнить команду mobj.flush() файла. Это необходимо для того, чтобы локальные модификации буферов действительно были доступны для сопоставления.

Для версий конструктора для Unix и Windows аргумент access может быть указан как необязательный ключевой аргумент. Аргумент access принимает одно из четырех значений: mmap.ACCESS_READ (только для чтение, невозможно изменить данные в памяти), mmap.ACCESS_WRITE (сквозная запись), mmap.ACCESS_COPY (можно изменять в памяти, не записывает изменения на диск, даже если вызывается flush()) или mmap.ACCESS_DEFAULT (используется режим, указанный по умолчанию аргументе prot). Аргумент access можно использовать как в Unix, так и в Windows.

Если access не указан, то в Windows модуль mmap возвращает объект со сквозной записью. Начальные значения памяти для всех трех типов доступа access берутся из открытого файла и должны соответствовать (например, open() без указания mode откроет файл только для чтения, а режим mmap.mmap() без указания access - сквозная запись, в результате будет ошибка TypeError).

Изменено в версии Python 3.7: Добавлена константа mmap.ACCESS_DEFAULT.

Чтобы отобразить анонимную память, в качестве номера файла вместе с длиной следует передать -1.

mmap.mmap(fileno, length, tagname=None, access=ACCESS_DEFAULT[, offset]):

Версия mmap.mmap() для Windows сопоставляет длину байтов из файла, указанного дескриптором файла fileno, и создает объект mmap.

Если аргумент length больше, чем текущий размер файла, то файл расширяется до указанной длины length, чтобы содержать дополнительные байты. Если аргумент length равен 0, то максимальная длина памяти будет равна текущему размеру файла, за исключением того, что если файл пуст, то Windows выдает исключение (нельзя создать пустое отображение в Windows).

Аргумент tagname, если он указан, а не None, представляет собой строку, задающую имя тега для сопоставления в памяти. Windows позволяет иметь много разных сопоставлений с одним и тем же файлом. Если указывается имя существующего тега, то этот тег открывается, в противном случае создается новый тег с этим именем. Если этот аргумент опущен или имеет значение None, то сопоставление создается без имени. Отказ от использования тега tagname поможет обеспечить переносимость кода между Unix и Windows.

Аргумент offset может быть указан как неотрицательное целочисленное смещение. Ссылки mmap будут относиться к смещению от начала файла. Аргумент offset по умолчанию равен 0, он также должен быть кратен mmap.ALLOCATIONGRANULARITY.

mmap.mmap(fileno, length, flags=MAP_SHARED, prot=PROT_WRITE|PROT_READ, access=ACCESS_DEFAULT[, offset]):

Версия mmap.mmap() для Unix сопоставляет длину байтов из файла, указанного файловым дескриптором fileno, и возвращает объект mmap.

Если длина length равна 0, то при вызове mmap(), максимальная длина памяти будет равна текущему размеру файла.

Аргумент flags указывает характер отображения. mmap.MAP_PRIVATE создает частное сопоставление с копированием при записи, поэтому изменения содержимого объекта mmap будут частными для этого процесса, а mmap.MAP_SHARED создает сопоставление, совместно используемое всеми другими процессами, отображающими те же области файла. Значение по умолчанию: mmap.MAP_SHARED. Некоторые системы имеют дополнительные возможные флаги.

Аргумент prot, если он указан, то дает желаемую защиту памяти. Два наиболее полезных значения: mmap.PROT_READ и mmap.PROT_WRITE указывают, что страницы могут быть прочитаны или записаны. По умолчанию PROT_READ|PROT_WRITE.

Аргумент access может быть указан вместо flags и prot как необязательный ключевой аргумент. Ошибочно указывать оба флага, prot и access. Смотрите описание как использовать аргумента access выше.

Аргумент offset может быть указан как неотрицательное целочисленное смещение. Ссылки mmap будут относиться к смещению от начала файла. offset по умолчанию равен 0, и должен быть кратным mmap.ALLOCATIONGRANULARITY, который в системах Unix равен mmap.PAGESIZE.

Чтобы обеспечить достоверность созданного сопоставления памяти, файл, указанный дескриптором fileno, автоматически синхронизируется внутри с физическим резервным хранилищем в macOS и OpenVMS.

Методы объекта MMAP.

Важно понимать, что объект MMAP - это область памяти, представленная в байтах, и следовательно не поддерживает методы, доступные байтовым строкам.

- MMAP.close() закрывает объект MMAP,
- MMAP.closed проверяет закрытие файла,
- MMAP.find() возвращает наименьший индекс в объекте,
- MMAP.flush() сбрасывает изменения, внесенные в копию файла,
- MMAP.madvise() рекомендации ядру относительно области памяти,
- MMAP.move() копирует количество байтов,
- MMAP.read() возвращает до n байтов,
- MMAP.read_byte() возвращает байт в текущей позиции,
- MMAP.readline() возвращает одну строку, начиная с текущей позиции,
- MMAP.resize() изменяет размер памяти и базового файла,
- MMAP.rfind() возвращает наивысший индекс в объекте,
- MMAP.seek() устанавливает текущую позицию файла,
- MMAP.size() возвращает длину файла,
- MMAP.tell() текущая позиция указателя файла,
- MMAP.write() записывает байты в память с текущей позиции,
- MMAP.write_byte() записывает целочисленный байт в память.

MMAP.close():

Метод MMAP.close() закрывает объект MMAP. Последующие вызовы других методов объекта приведут к возникновению исключения ValueError. Это не приведет к закрытию открытого файла..

MMAP.closed:

Свойство MMAP.closed проверяет закрытие файла, True если файл закрыт.

MMAP.find(sub[, start[, end]]):

Метод MMAP.find() возвращает наименьший индекс в объекте, в котором найдена подпоследовательность sub, так что sub содержится в диапазоне [start, end]. Необязательные аргументы start и end интерпретируются как в нотации среза. Возвращает -1 при сбое.

MMAP.flush([offset[, size]]):

Метод MMAP.flush() сбрасывает изменения, внесенные в копию файла в памяти обратно на диск. Без использования этого вызова нет никакой гарантии, что изменения будут записаны обратно до того, как объект будет уничтожен. Если указаны смещение offset и размер size, то на диск будут сброшены только изменения в заданном диапазоне байт, в противном случае будет сброшен весь экстент сопоставления. Смещение offset должно быть кратным mmap.PAGESIZE или mmap.ALLOCATIONGRANULARITY.

В случае успеха метод возвращает None. При сбое вызова возникает исключение.

Изменено в Python 3.8: ранее в случае успеха возвращалось ненулевое значение, ноль возвращался при ошибке под Windows. В Unix, в случае успеха возвращалось нулевое значение. При ошибке возникло исключение.

MMAP.madvise(option[, start[, length]]):

Метод MMAP.madvise() отправляет option в качестве рекомендаций ядру относительно области памяти, начинающейся с start и увеличивающейся в байтах length. Аргумент option должен быть одной из констант MADV_*, доступных в системе. Если start и length опущены, то охватывается все отображение. В некоторых системах (включая Linux) значение start должно быть кратно размеру mmap.PAGESIZE.

Доступность: Системы с системным вызовом madvise().

Новое в Python 3.8.

MMAP.move(dest, src, count):

Метод MMAP.move() копирует количество байтов count, начиная со смещения src, в целевой индекс dest. Если mmap был создан с ACCESS_READ, то вызовы move вызовут исключение TypeError.

MMAP.read([n]):

Метод MMAP.read() возвращает до n байтов, начиная с текущей позиции в файле. Если аргумент опущен, None или отрицательный, то возвращаются все байты от текущей позиции файла до конца сопоставления. Указатель в файле перемещается на позицию после возвращенных байтов.

MMAP.read_byte():

Метод MMAP.read_byte() возвращает байт в текущей позиции файла в виде целого числа и увеличивает позицию указателя в файле на 1.

MMAP.readline():

Метод MMAP.readline() возвращает одну строку, начиная с текущей позиции файла и до следующей новой строки. Указатель файла перемещается до позиция после байтов, которые были возвращены.

MMAP.resize(newsize):

Метод MMAP.resize() изменяет размер памяти и базового файла, если он есть. Если объект MMAP был создан с помощью ACCESS_READ или ACCESS_COPY, то изменение размера карты вызовет исключение TypeError.

MMAP.rfind(sub[, start[, end]]):

Метод MMAP.rfind() возвращает наивысший индекс в объекте, где найдена подпоследовательность sub, такая, что sub содержится в диапазоне [start, end]. Необязательные аргументы start и end интерпретируются как в нотации среза. Возвращает -1 при сбое.

MMAP.seek(pos[, whence]):

Метод MMAP.seek() устанавливает текущую позицию файла. Аргумент whence необязателен и по умолчанию равен os.SEEK_SET или 0 (абсолютное позиционирование файла). Другие значения: os.SEEK_CUR или 1 (поиск относительно текущей позиции) и os.SEEK_END или 2 (поиск относительно конца файла).

MMAP.size():

Метод MMAP.size() возвращает длину файла, которая может быть больше, чем размер отображаемой в памяти области..

MMAP.tell():

Метод MMAP.tell() возвращает текущую позицию указателя файла.

MMAP.write(bytes):

Метод MMAP.write() записывает байты в память с текущей позиции указателя файла и возвращает количество записанных байтов (никогда не меньше, чем len(bytes), т.к. в случае сбоя записи будет вызвано значение ValueError). Указатель файла передвигается в позицию после записанных байтов. Если MMAP был создан с ACCESS_READ, то запись в него вызовет исключение TypeError.

Изменено в Python 3.6: теперь возвращается количество записанных байтов.

MMAP.write_byte(byte):

Метод MMAP.write_byte() записывает целочисленный байт в память с текущей позиции указателя файла. Позиция в файле увеличивается на 1. Если объект MMAP был создан с ACCESS_READ, то запись в него вызовет исключение TypeError.

Примеры использования модуля mmap.

Отображение памяти наиболее полезно для чтения файлов, но также можно использовать его для записи файлов. API mmap для записи файлов очень похож на обычный файловый ввод-вывод, за исключением нескольких отличий.

```
>>> import mmap
>>> text = 'text text text text text'
>>> with open('fname.txt', mode='w', encoding='utf-8') as fp:
...     mm_io = mmap.mmap(fp.fileno(), length=0, access=mmap.ACCESS_WRITE)
...     mm_io.write(text)
...     mm_io.flush()
...     mm_io.close()

# Traceback (most recent call last):
#   File "<stdin>", line 2, in <module>
# ValueError: cannot mmap an empty file
```

Этот код вызывает исключение ValueError, т.к. функция open() при mode='w' создаст пустой файл. Модуль mmap не позволяет отображать в память пустые файлы. Это разумно, потому что концептуально, пустой отображаемый в память файл является просто буфером памяти, следовательно объект в памяти не нуждается.

Обычно отображение памяти используется в режиме чтения или чтения-изменения-записи/дозаписи. В следующем примере показан простой способ использования mmap:

```
import mmap

# сначала запишем что нибудь в файл
with open("hello.txt", "wb") as fp:
    fp.write(b"Hello Python!\n")

with open("hello.txt", "r+b") as fp:
    # сопоставляем с памятью файл, length=0 означает весь файл
    mm = mmap.mmap(fp.fileno(), length=0, access=mmap.ACCESS_WRITE)
    # читать содержимое можно стандартными файловыми методами
    print(mm.readline())
    # b"Hello Python!\n"

    # считывание содержимого с помощью нотации среза
    print(mm[:5])
    # b"Hello"

    # ВНИМАНИЕ, при обновлении содержимого,
    # новый контент должен иметь такой же размер!
    mm[6:] = b" world!\n"
    # читаем снова, используя стандартные методы работы с файлами
    mm.seek(0)
    print(mm.readline())
    # b"Hello  world!\n"

    # закрываем сопоставление
    mm.close()
```

Примечание. Почему при изменении важно, чтобы новый контент имел такой же размер? Напомним еще раз, что объект MMAP - это область памяти представленная в байтах и если данные изменяются, то они затираются (НЕ ВСТАВЛЯЮТСЯ) вместо старых! Следовательно, если новые данные будут больше, тех которые надо изменить, то затрется часть данных, которые изменять не надо. Если новые данные будут меньше, то часть старых данных останется.

Модуль mmap также можно использовать в качестве диспетчера контекста в операторе with:

```
import mmap

with mmap.mmap(-1, 13) as mm:
    mm.write(b"Hello world!")
```

Дозапись данных в конец файла, при использовании модуля mmap.

Примечание. Если осуществляется запись в конец файла, то не забываем выделять для ОЗУ дополнительную область, объемом, равную дозаписываемым данным. Это можно сделать в момент создания объекта mmap, задав length (!размер в байтах) больший чем размер открываемого файла на величину записываемых данных или при помощи метода MMAP.resize(). Если этого не сделать, то будет получено исключение ValueError: data out of range.

```
import mmap

# запишем начальные данные в файл
with open("hello.txt", "w") as fp:
    fp.write('Hello  world!\n')

# добавляемый текст
text = "Привет Мир!\n"
# переводим в байты
add_text = bytes(text, encoding='utf-8')
# или
add_text = text.encode('utf-8')

with open("hello.txt", "r+b") as fm:
    with mmap.mmap(fm.fileno(), length=0, access=mmap.ACCESS_WRITE) as mm:
        # длина прочитанных данных в байтах
        size = mm.size()
        # перемещаем указатель файла в конец
        # (на величину прочитанных данных)
        mm.seek(size)
        # длина добавляемых данных
        diff = len(add_text)
        # увеличиваем выделенную память
        # на величину добавляемых данных
        mm.resize(size + diff)
        # записываем новые данные
        mm.write(add_text)
        # сбрасываем данные из памяти в файл
        mm.flush()

# смотрим, что получилось
with open("hello.txt", encoding='utf-8') as fp:
    print(fp.read())

# Hello  world!
# Привет Мир!
```

Вставка/замена данных в файле, при использовании модуля mmap.

Чтобы вставить/заменить (в буквальном смысле) какие-то данные в определенном месте оперативной памяти, то нужно сначала увеличить/уменьшить выделенную память на величину изменений при помощи метода MMAP.resize() (если не изменять, то будет исключение ValueError: data out of range), запомнить в переменную оставшуюся часть данных (при помощи среза), затем, с нужной позиции затереть старые данные новыми, и наконец с конца позиции новых данных добавить оставшиеся данные.

Для вычисления величины, на которую нужно менять выделенную память, например, при замене слова *"мир"* на *"привет"*, можно использовать функцию len(): len('привет'.encode()) - len('мир'.encode()), т.е. увеличить на 6 байт.

```
import mmap

# сначала запишем что нибудь в файл
with open("test.txt", "w") as fp:
    fp.write("Привет миропорядок!")

# меняем слово
text = 'миро'.encode('utf-8')
# на слово
repl_text = 'привет '.encode('utf-8')

with open("test.txt", "r+b") as fp:
    with mmap.mmap(fp.fileno(), length=0, access=mmap.ACCESS_WRITE) as mm:
        # текущий размер файла
        current_size = mm.size()
        # индекс начала заменяемого слова
        index_text = mm.find(text)
        # длина заменяемого слова
        len_text = len(text)
        # индекс начала данных, которые нужно "подвинуть"
        index_data = index_text + len_text
        # запоминаем данные, которые нужно "подвинуть"
        data = mm[index_data:]
        # вычисляем на сколько измениться область памяти
        diff = len(repl_text) - len_text
        # изменяем область памяти
        mm.resize(current_size + diff)
        # перемещаем курсор на начало заменяемых данных
        mm.seek(index_text)
        # пишем новые данные + оставшиеся данные
        mm.write(repl_text + data)
        # сбрасываем все в файл
        mm.flush()

# смотрим, что получилось
with open("test.txt", encoding='utf-8') as fp:
    print(fp.read())

# Привет привет порядок!
```

Кстати, искать нужные подстроки в отображении файла в памяти также можно при помощи регулярных выражений, например функцией re.search(), объект поиска которой сразу дает начало и конец совпадения в виде кортежа или по отдельности.

```
>>> mm = bytes('Привет миропорядок!', encoding='utf-8')
>>> textsearch = bytes('миро', encoding='utf-8')
>>> import re
>>> match = re.search(textsearch, memo_map)
>>> match.span()
# (13, 21)
>>> match.start()
# 13
>>> match.end()
# 21
```

Для вставки/удаления данных в определенной области памяти, алгоритм действий будет похожим. Только при удалении данных или замене меньшим объемом данных, изменение размера отображаемой/выделенной памяти делается в последнюю очередь.

Анонимная область памяти для обмена данными между процессами.

В следующем примере показано, как создать анонимную память и обмениваться данными между родительским и дочерним процессами:

```
import mmap
import os

mm = mmap.mmap(-1, 13)
mm.write(b"Hello world!")

pid = os.fork()

if pid == 0:  # В дочернем процессе
    mm.seek(0)
    print(mm.readline())
    mm.close()
```

Для обмена данными между процессами лучше конечно использовать специально предназначенный для этих целей класс SharedMemory(), который создает общую защищенную область памяти с авторизацией процессов по ее имени.

Константы MAP_*.

Это различные флаги, которые можно передать в конструктор mmap.mmap(). Обратите внимание, что некоторые параметры могут отсутствовать в некоторых системах.

- mmap.MAP_SHARED;
- mmap.MAP_PRIVATE;
- mmap.MAP_DENYWRITE;
- mmap.MAP_EXECUTABLE;
- mmap.MAP_ANON;
- mmap.MAP_ANONYMOUS;
- mmap.MAP_POPULATE;

Изменено в Python 3.10: Добавлена константа MAP_POPULATE.

Константы MADV_*.

Новое в Python 3.8. Эти параметры могут быть переданы в mmap.madvise(). Обратите внимание, что некоторые параметры могут отсутствовать в некоторых системах.

- mmap.MADV_NORMAL;
- mmap.MADV_RANDOM;
- mmap.MADV_SEQUENTIAL;
- mmap.MADV_WILLNEED;
- mmap.MADV_DONTNEED;
- mmap.MADV_REMOVE;
- mmap.MADV_DONTFORK;
- mmap.MADV_DOFORK;
- mmap.MADV_HWPOISON;
- mmap.MADV_MERGEABLE;
- mmap.MADV_UNMERGEABLE;
- mmap.MADV_SOFT_OFFLINE;
- mmap.MADV_HUGEPAGE;
- mmap.MADV_NOHUGEPAGE;
- mmap.MADV_DONTDUMP;
- mmap.MADV_DODUMP;
- mmap.MADV_FREE;
- mmap.MADV_NOSYNC;
- mmap.MADV_AUTOSYNC;
- mmap.MADV_NOCORE;
- mmap.MADV_CORE;
- mmap.MADV_PROTECT;
- mmap.MADV_FREE_REUSABLE;
- mmap.MADV_FREE_REUSE;