irbis

Выпуск 0.2a1

Alexey Mironov <amironov73@gmail.com>

мая 29, 2023

Содержание:

Пакет `irbis` представляет собой фреймворк для создания клиентских приложений для системы автоматизации библиотек ИРБИС64 на языке Python.

Глава 1

Введение. Установка. Примеры программ

1.1 Введение

Пакет `irbis` представляет собой фреймворк для создания клиентских приложений для системы автоматизации библиотек ИРБИС64 на языке Python.

Пакет не содержит неуправляемого кода и не требует `irbis64_client.dll`. Успешно работает на 32-битных и 64-битных версиях операционных систем Windows, Linux и Mac OS X.

Основные возможности пакета:

- Поиск и расформатирование записей.
- Создание и модификация записей, сохранение записей в базе данных на сервере.
- Работа с поисковым словарем: просмотр термов и постингов.
- Администраторские функции: получение списка пользователей, его модификация, передача списка на сервер, создание и удаление баз данных.
- Импорт и экспорт записей в формате ISO 2709 и в обменном формате ИРБИС.

Поддерживаются Python, начиная с версии 3.6, и сервер ИРБИС64, начиная с 2014. Более ранние версии Python будут выдавать ошибки, т. к. в пакет использует конструкции, присущие Python 3.6. Аналогично обстоит дело и с более ранними версиями сервера ИРБИС64.

1.2 Установка

irbis загружен в централизованный репозиторий пакетов PyPI, поэтому можно установить его с помощью стандартного клиента pip, входящего в поставку Python:

```
pip install irbis --user --upgrade
```

или

```
python -m pip install irbis --user --upgrade
```

(оба способа эквивалентны).

Здесь --user означает установку только для текущего пользователя (без этого ключа установка будет выполняться для всех пользователей и может потребовать администраторских прав), а --upgrade - обновление пакета при необходимости. Если уже установлена последняя версия пакета, то pip просто сообщит об этом и завершит работу.

Также можно установить пакет, скачав необходимые файлы с репозитория GitHub: https://github.com/amironov73/PythonIrbis

Кроме того, доступны ночные dist-сборки на AppVeyor: https://ci.appveyor.com/project/AlexeyMironov/pythonirbis/build/artifacts

1.3 Примеры программ

Ниже прилагается пример простой программы. Сначала находятся и загружаются 10 первых библиографических записей, в которых автором является А. С. Пушкин. Показано нахождение значения поля с заданным тегом и подполя с заданным кодом. Также показано расформатирование записи в формат brief.

```
import irbis

# Подключаемся к серверу
client = irbis.Connection()
client.parse_connection_string('host=127.0.0.1;port=6666;' +
    'database=IBIS;user=librarian;password=secret;')
client.connect()

if not client.connected:
    print('Невозможно подключиться!')
    exit(1)

# Ищем все книги, автором которых является А. С. Пушкин
# Обратите внимание на двойные кавычки в тексте запроса
found = client.search('"A=ПУШКИН$"')
print(f'Найдено записей: {len(found)}')

# Чтобы не распечатывать все найденные записи, отберем только 10 первых
for mfn in found[:10]:
    # Получаем запись из базы данных
    record = client.read_record(mfn)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    # Извлекаем из записи интересующее нас поле и подполе
    title = record.fm(200, 'a')
    print('Заглавие:', title)

    # Форматируем запись средствами сервера
    description = client.format_record(irbis.BRIEF, mfn)
    print('Биб. описание:', description)

    print()  # Добавляем пустую строку

# Отключаемся от сервера
client.disconnect()
```

В следующей программе создается и отправляется на сервер 10 записей. Показано добавление в запись полей с подполями.

```
import irbis

# Подключаемся к серверу
client = irbis.Connection()
client.parse_connection_string('host=127.0.0.1;port=6666;' +
                               'database=IBIS;user=1;password=1;')
client.connect()

if not client.connected:
    print('Невозможно подключиться!')
    exit(1)

for i in range(10):
    # Создаем запись
    record = irbis.Record()

    # Наполняем её полями: первый автор
    record.add(700) \
        .add('a', 'Миронов') \
        .add('b', 'А. В.') \
        .add('g', 'Алексей Владимирович')

    # заглавие
    record.add(200) \
        .add('a', f'Работа с ИРБИС64: версия {i}.0') \
        .add('e', 'руководство пользователя')

    # выходные данные
    record.add(210) \
        .add('a', 'Иркутск') \
        .add('c', 'ИРНИТУ') \
        .add('d', '2018')

    # рабочий лист
    record.add(920, 'PAZK')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    # Отсылаем запись на сервер.
    # Обратно приходит запись, обработанная AUTOIN.GBL
    client.write_record(record)
    print(record)  # распечатываем обработанную запись
    print()

# Отключаемся от сервера
client.disconnect()
```

Глава 2

Класс Connection

Класс `Connection` - «рабочая лошадка». Он осуществляет связь с сервером и всю необходимую перепаковку данных из клиентского представления в сетевое.

Экземпляр клиента создается конструктором:

```
import irbis

client = irbis.Connection()
```

При создании клиента можно указать (некоторые) настройки:

```
import irbis

client = irbis.Connection(host='irbis.rsl.ru', port=5555, username='ninja')
```

Можно задать те же настройки с помощью полей *host*, *port* и т. д.:

```
import irbis

client = irbis.Connection()
client.host = 'irbis.rsl.ru'
client.port = 5555
```

Поле	Тип	Назначение	Значение по умолчанию
host	str	Адрес сервера	„127.0.0.1“
port	int	Порт	6666
username	str	Имя (логин) пользователя	None
password	str	Пароль пользователя	None
database	str	Имя базы данных	„IBIS“
workstation	str	Тип АРМа (см. таблицу ниже)	„C“

Типы АРМов

Обозначение	Тип	Константа
„R“	Читатель	READER
„C“	Каталогизатор	CATALOGER
„M“	Комплектатор	ACQUISITIONS
„B“	Книговыдача	CIRCULATION
„K“	Книгообеспеченность	PROVISION
„A“	Администратор	ADMINISTRATOR

Обратите внимание, что адрес сервера задается строкой, так что может принимать как значения вроде `192.168.1.1`, так и `irbis.yourlib.com`.

Тип рабочей станции лучше задавать константой:

```
import irbis

client = irbis.Connection()
client.host = '8.8.8.8'
client.workstation = irbis.ADMINISTRATOR
```

Если какой-либо из вышеперечисленных параметров не задан явно, используется значение по умолчанию.

Имейте в виду, что логин и пароль пользователя не имеют значения по умолчанию, поэтому должны быть заданы явно.

2.1 Подключение к серверу и отключение от него

Только что созданный клиент еще не подключен к серверу. Подключаться необходимо явно с помощью метода `connect`, при этом можно указать параметры подключения:

```
import irbis

client = irbis.Connection()
client.connect('192.168.1.2', 6666, 'librarian', 'secret')
if not client:
    print('Подключиться не удалось')
```

Отключаться от сервера можно двумя способами: во-первых, с помощью метода `disconnect`:

```
client.disconnect()
```

во-вторых, с помощью контекста, задаваемого блоком `with`:

```
import irbis

with irbis.Connection(host='192.168.1.3') as client:
    client.connect(username='itsme', password='secret')

    if not client:
        print('Подключиться не удалось')
        exit(1)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    # Выполняем некие действия.
    # По выходу из блока отключение от сервера произойдет автоматически.
```

При подключении клиент получает с сервера INI-файл с настройками, которые могут понадобиться в процессе работы:

```
import irbis

client = irbis.Connection()
ini = client.connect()
format_menu_name = ini.get_value('Main', 'FmtMnu', 'FMT31.MNU')
```

Полученный с сервера INI-файл также хранится в поле `ini_file`.

Повторная попытка подключения с помощью того же экземпляра `Connection` игнорируется. При необходимости можно создать другой экземпляр и подключиться с его помощью (если позволяют клиентские лицензии). Аналогично игнорируются повторные попытки отключения от сервера.

Проверить статус «клиент подключен или нет» можно с помощью преобразования подключения к типу `bool`:

```
import irbis

client = irbis.Connection()

# спустя 300 строк кода
if not client:
    # В настоящий момент мы не подключены
    return
```

Вместо индивидуального задания каждого из полей `host`, `port`, `username`, `password` и `database` предпочтительнее использовать метод `parse_connection_string`:

```
import irbis

client = irbis.Connection()
client.parse_connection_string('host=192.168.1.4;port=5555;username=itsme;
↪password=secret;db=RDR;')
client.connect()
# выполняем какие-нибудь действия
client.disconnect()
```

Подключенный клиент может сформировать строку подключения (с помощью метода `to_connection_string`), которую можно использовать для другого подключения:

```
import irbis

client1 = irbis.Connection()
client1.connect('host', 6666, 'librarian', 'secret')
# выполняем какие-нибудь действия
connection_string = client1.to_connection_string()
client1.disconnect()
client2 = irbis.Connection()
client2.parse_connection_string(connection_string)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
client2.connect()
# выполняем какие-нибудь действия
client2.disconnect()
```

2.2 Многопоточность

Клиент написан в наивном однопоточном стиле, поэтому не поддерживает одновременный вызов методов из разных потоков.

Для одновременной отсылки на сервер нескольких команд необходимо создать соответствующее количество экземпляров подключений (если подобное позволяет лицензия сервера).

2.3 Подтверждение подключения

`irbis` не посылает самостоятельно на сервер подтверждений того, что клиент все еще подключен. Этим должно заниматься приложение, например, по таймеру.

Подтверждение посылается серверу методом `nop`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
client.nop()
client.disconnect()
```

2.4 Чтение записей с сервера

Для индивидуального чтения записи с сервера применяется метод `read_record`.

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
mfn = 123
record = client.read_record(mfn)
print(record.status)
client.disconnect()
```

Для группового чтения записей с сервера применяется метод `read_records`.

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
mfns = [12, 34, 56]
records = client.read_records(mfns)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
for record in records:
    print(record.status)
client.disconnect()
```

Методы для работы записями в клиентском представлении (доступ к полям/подполям, добавление/удаление полей и т. д.) см. в следующей главе.

2.5 Сохранение записи на сервере

Вновь созданную либо модифицированную запись можно сохранить на сервере с помощью метода `write_record`. Сначала покажем, как выполняется модификация записи:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
mfn = 123
record = client.read_record(mfn)
# Добавляем в запись поле 300 (общее примечание)
record.add(300, 'Примечание к записи')
# Сохраняем запись обратно на сервер
client.write_record(record)
client.disconnect()
```

Теперь попробуем создать запись «с нуля» и сохранить её в базе данных:

```
import irbis

SF = irbis.SubField # для краткости

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')

# Создаём запись и наполняем её полями
record = irbis.Record()
record.add(200, SF('a', 'Заглавие'))
record.add(700, SF('a', 'Фамилия автора'))

# Отправляем запись на сервер
# Запись попадёт в текущую базу данных
client.write_record(record)

client.disconnect()
```

2.6 Удаление записей

Удаление записи заключается в установке её статуса `LOGICALLY_DELETED`. Для этого применяется метод `delete_record`, принимающий MFN записи:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
mfn = 123 # MFN записи, подлежащей удалению
client.delete_record(mfn)
client.disconnect()
```

После удаления запись становится логически удалённой, т. е. физически она присутствует в файле документов и может быть считана клиентом, однако исключается из поиска. Логически удалённую запись в любой момент можно восстановить с помощью метода `undelete_record`, также принимающего MFN записи.

Администратор может провести на сервере так называемую реорганизацию файла документов, в процессе которой логически удалённые записи будут исключены из мастер-файла (однако, за ними сохранится MFN). Такие записи не могут быть прочитаны и/или восстановлены клиентом.

2.7 Поиск записей

Поиск записей в ИРБИС осуществляется двумя способами:

1. Так называемый «прямой поиск», выполняемый по автоматически поддерживаемому поисковому индексу (словарю). Используется специальный синтаксис, описанный на странице http://sntnarciss.ru/irbis/spravka/pril01701001000.htm
2. Так называемый «последовательный поиск», заключающийся в последовательном переборе записей. Для него используется другой синтаксис, описанный на странице http://sntnarciss.ru/irbis/spravka/pril01701002000.htm.

2.7.1 Прямой поиск

Обратите внимание, что при прямом поиске мы заключаем искомые термины в дополнительные двойные кавычки, это требование сервера ИРБИС64 (ведь термины могут включать в себя пробелы и специальные символы, и без кавычек сервер не сможет определить конец одного термина и начало другого).

Вот как выглядит правильно сформулированное поисковое выражение:

```
search_expression = '"A=ПУШКИН$" * ("T=СКАЗКИ$" + "T=ПОВЕСТИ$")'
```

Количество найденных записей по данному запросу:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
# Находим книги, автором которых является Пушкин
```

(continues on next page)

(продолжение с предыдущей страницы)

```
count = client.search_count('"A=ПУШКИН$"')
print(f"Всего книг Пушкина в фонде: {count}")
client.disconnect()
```

Имейте в виду, что сервер ИРБИС64 возвращает записи в произвольном порядке! Чаще всего этот порядок совпадает с порядком, в котором записи были введены в базу данных. Поэтому сортировать записи должен сам клиент.

Обычный поиск с помощью метода `search` выдаёт не более 32 тыс. найденных записей (это ограничение сервера ИРБИС64):

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
# Получаем MFN книг, автором которых является Пушкин
found = client.search('"A=ПУШКИН$"')
# Распечатываем список найденных MFN
print('Найдены MFN:', ', '.join(found))
client.disconnect()
```

Метод `search_all` выдаёт все найденные записи, в т. ч. если их много больше 32 тыс. *Осторожно! Этот метод может занять много времени и ресурсов как сервера, так и клиента!*

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
# Получаем MFN книг, у которых есть хотя бы один автор
found = client.search_all('"A=$"')
# Распечатываем список найденных MFN
print('Найдены MFN:', ', '.join(found))
client.disconnect()
```

Можно объединить поиск с одновременным считыванием записей, применив метод `search_read`. *Осторожно! Этот метод может занять много времени и ресурсов как сервера, так и клиента!* Устанавливайте разумное значение параметра `limit` при вызове этого метода.

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
# Считываем первые 5 найденных записей для книг,
# автором которых является Пушкин
found = client.search_read('"A=ПУШКИН$"', 5)
# Распечатываем заглавия найденных книг:
for record in found:
    print(record.fm(200, 'a'))
client.disconnect()
```

Поиск с одновременным расформатированием записей осуществляется методм `search_format`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
# Расформатируем первые 5 найденных записей для книг,
# автором которых является Пушкин
found = client.search_format('"A=ПУШКИН$"', '@brief', 5)
# Распечатываем результаты форматирования:
for line in found:
    print(line)
client.disconnect()
```

2.7.2 Последовательный поиск

Последовательный поиск можно выполнить при помощи метода `search_ex`, принимающий в себя спецификацию поискового запроса `SearchParameters`. Часто последовательный поиск проводят по результатам предварительного прямого поиска. В терминах `SearchParameters` это означает задание значения поля `sequential` (выражение для последовательного поиска) вместе (согласованно) со значением поля `expression` (выражение для прямого поиска).

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
params = irbis.SearchParameters()
params.database = 'IBIS' # По какой базе ищем
params.expression = '"A=ПУШКИН$"' # Прямой поиск (по словарю)
params.number = 10 # Выдать не больше 10 записей
params.format = '@brief' # Форматирование найденных записей
# Последовательнсый поиск среди отобранных по словарю записей
params.sequential = "if v200^a:'Сказки' then '1' else '0' fi"
found = client.search_ex(params)
for line in found:
    record = client.read_record(line.mfn)
    print(record.fm(200, 'a'))
    # Получаем расформатированную запись
    print(line.description)
```

2.8 Работа с текстовыми файлами на сервере

Сначала необходимо упомянуть об используемой сервером ИРБИС64 спецификации имён файлов. Эта спецификация выглядит так:

```
Расположение . База . ИмяФайла
```

где `расположение` - число, означающее место, где сервер должен искать файл:

- 0 – общесистемный путь (директория, в которую установлен сервер ИРБИС64), чаще всего `C:\IRBIS64`.
- 1 – путь размещения сведений о базах данных сервера ИРБИС64, чаще всего `C:\IRBIS64\DATAI`.

- 2 – путь на мастер-файл базы данных. Для базы данных IBIS чаще всего выглядит так: `C:\IRBIS64\DATAI\IBIS`.
- 3 – путь на словарь базы данных. Чаще всего совпадает с предыдущим путём.
- 10 – путь на параметрию базы данных. Чаще всего совпадает с предыдущим путём.

Для расположений `0` и `1` имя базы данных указывать не нужно, т. к. оно не имеет смысла. Такие пути выглядят так:

- `0..RC.MNU` - меню с римскими цифрами, хранится рядом с сервером ИРБИС64.
- `1..dbnam2.mnu` - меню со списком баз данных, доступных АРМ «Каталогизатор», хранится в папке `DATAI`.

Для остальных расположений между двух точек указывают имя базы данных. Примеры:

- `2.IBIS.brief.pft` - формат краткого библиографического описания для базы данных `IBIS`.
- `2.RDR.email.pft` - формат электронной почты для базы данных `RDR`.
- `2.CMPL.KP.MNU` - меню с каналами поступления для базы данных `CMPL`.

Для путей, больших или равных 2, сервер сначала ищет файл в директории, заданной в PAR-файле, и, если не находит там, то пытается найти файл с тем же именем в папке `Deposit`.

Обратите внимание! Сервер ИРБИС64 под Windows игнорирует регистр символов в спецификации имён файлов!

Чаще всего клиенты считывают с сервера следующие текстовые файлы:

- форматы (имеют расширение PFT) для формирования каталожных карточек и списков литературы,
- меню (имеют расширение MNU),
- иерархические меню (имеют расширение TRE),
- INI-файлы со сценариями поиска,
- рабочие листы ввода (имеют расширения WS и WSS).

Однако, ничего не мешает клиентам получать с сервера и любые другие текстовые и двоичные файлы, необходимые им для работы.

Текстовый файл можно получить с сервера с помощью метода `read_text_file`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
# Считываем формат краткого библиографического
# описания для базы IBIS
brief = client.read_text_file('2.IBIS.brief.pft')
print('BRIEF:', brief)
client.disconnect()
```

Обратите внимание! Если сервер не может найти указанный файл, либо не может получить доступ к этому файлу (недостаточно прав), он возвращает строку нулевой длины!

Для считывания с сервера меню, иерархических справочников и других специфических текстовых файлов имеются соответствующие методы, описанные в главе 4.

Сохранить текстовый файл на сервере можно с помощью метода `write_text_file`. Имейте в виду, что текстовые файлы на сервере хранятся, как правило, в кодировке CP1251, так что все символы, не имеющие представления в данной кодировке, будут утеряны при сохранении.

Для получения с сервера двоичных файлов (например, изображений) используется метод `read_binary_file`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
# Считываем GIF-файл с бегущим ирбисом,
# хранящийся рядом с сервером ИРБИС64
running = client.read_binary_file('0..irbis.gif')
# Сохраняем в локальный файл
with open('bars.gif', 'wb') as f:
    f.write(running)
client.disconnect()
```

Получить список файлов на сервере можно с помощью метода `list_files`. В него передаётся перечень (может состоять из одного файла) спецификаций файлов, которые нас интересуют. Разрешается использовать метасимволы „?“ и „*“, имеющие тот же смысл, что и в командной строке Windows. Метод возвращает массив имён (не спецификаций!) найденных файлов.

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
# Получаем список форматных файлов для базы IBIS
formats = client.list_files('2.IBIS.')
# Распечатываем полученный список файлов:
print(formats)
client.disconnect()
```

2.9 Поддержка асинхронности

```
import irbis

async def do_async_stuff():
    result = await connection.connect_async()
    if not result:
        print('Failed to connect')
        return

    print('Connected')

    max_mfn = await connection.get_max_mfn_async()
    print(f"Max MFN={max_mfn}")

    text = await connection.format_record_async('@brief', 1)
    print(text)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    await connection.nop_async()
    print('NOP')

    record = await connection.read_record_async(1)
    print(record)

    text = await connection.read_text_file_async('dn.mnu')
    print(text)

    count = await connection.search_count_async('К=бетон')
    print(f'Count={count}')

    found = await connection.search_async('К=бетон')
    print(found)

    await connection.disconnect_async()
    print('Disconnected')

#=============================================

connection = irbis.Connection()
connection.host = 'localhost'
connection.username = 'librarian'
connection.password = 'secret'
connection.database = 'IBIS'

irbis.init_async()

irbis.irbis_event_loop.run_until_complete(do_async_stuff())

irbis.close_async()
```

Глава 3

Классы Record, Field и SubField

Классы `Record`, `Field` и `SubField` предназначены для создания и манипуляции записями, полями и подполями соответственно в клиентском представлении. Они также отвечают за перекодирование записей из серверного представления в клиентское и обратно.

3.1 Record

Каждый экземпляр класса `Record` соответствует одной записи в базе данных ИРБИС и имеет следующие атрибуты:

- **database** – имя базы данных, из которой загружена данная запись. Для вновь созданных записей `None`. После отправки записи на сервер поле `database` выставляется автоматически.
- **mfn** – порядковый номер записи в базе данных. Для вновь созданных записей 0. После отправки записи на сервер поле `mfn` выставляется автоматически. Диапазон значений для MFN: от 1 до 4294967295. Обратите внимание, при реорганизации базы данных MFN записи может измениться!
- **status** – состояние записи: удалена, отсутствует и т. д. Для вновь созданных записей 0. После отправки записи на сервер поле `status` выставляется автоматически. Обратите внимание, при реорганизации базе данных логически удалённые записи могут пропасть из неё.
- **version** – номер версии записи. Для вновь созданных записей 0. При каждой отправке записи на сервер поле `version` выставляется автоматически. Поле *version* используется сервером ИРБИС64 для отслеживания конфликтов одновременного обновления записей несколькими клиентами. Обратите внимание, при реорганизации базе данных её версия может измениться!
- **fields** – запись содержит произвольное количество полей. Технически их может быть 0, но на практике это означает сбой системы.

При отправке записи на сервер ИРБИС64 тот отсылает обратно клиенту эту же запись со всеми модификациями, которые были проведены над ней сценарием `autoin.gbl`. При этом могут измениться любые поля записи, а также её статус и версия.

Атрибуты в виде таблицы:

Поле	Тип	Назначение
database	str	Имя базы данных, из которой загружена данная запись.
mfn	int	Номер записи в мастер-файле.
status	int	Статус записи: логически удалена, отсутствует (см. ниже).
version	int	Номер версии записи.
fields	list	Список полей записи.

Статус записи: набор флагов

Имя	Число	Значение
LOGICALLY_DELETED	1	Логически удалена (может быть восстановлена)
PHYSICALLY_DELETED	2	Физически удалена (не может быть восстановлена)
ABSENT	4	Отсутствует
NON_ACTUALIZED	8	Не актуализирована
LAST	32	Последняя версия записи
LOCKED	64	Запись заблокирована на ввод
AUTOIN_ERROR	128	Ошибка в Autoin.gbl.
FULL_TEXT_NOT_ACTUALIZED	256	Полный текст не актуализирован.

Класс `Record` содержит следующие методы:

- **def __init__(self, *fields)** – конструктор, создаёт новый экземпляр записи в памяти клиента. Можно указать поля, которыми будет наполнена запись.
- **def add(self, tag, value = None, *subfields) -> Record** – добавляет поле (возможно, со значением и подполями) к записи. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def add_non_empty(self, tag, value) -> Record** – добавляет поле, если его значение не пустое. Возвращает *self*, поэтому метод может использоваться в цепочечных вызовах.
- **def all(self, tag) -> List[Field]** – возвращает список всех полей с указанной меткой.
- **all_as_dict(self, tag: int) -> List[dict]** – список полей с указанной меткой, каждое поле в виде словаря «код - значение». Если указанные поля не найдены, возвращается пустой список.
- **def clear(self) -> Record** – очистка записи (удаление всех полей).
- **def clone(self) -> Record** – создание клона, т. е. полной копии записи. Поля, если присутствуют, тоже клонируются. Возвращает клон записи.
- **def encode(self) -> List[str]** – кодирование записи в серверное представление. Инфраструктурный метод, парный к *parse*.
- **def fm(self, tag, code = „“) -> str** – возвращает значение первого поля с указанной меткой (или значение подполя с указанным кодом). Если задан пустой код, то возвращается значение поля до первого разделителя. Если задан код „*“, то возвращается значение до первого разделителя либо значение первого по порядку подполя (смотря, что есть в поле). Если поле с указанной меткой не найдено или нет нужного подполя, возвращается `None`.
- **def fma(self, tag, code = „“) -> List[str]** – возвращает список значений полей с указанной меткой (или значения подполей с указанным кодом). Пустые значения в список не включаются. Если задан пустой код, то возвращаются значения полей до первого разделителя. Если задан код *, то возвращаются значения до первого разделителя либо значения первого по порядку подполя (смотря, что есть в поле). Если нужного подполя в просматриваемом поле нет, в список ничего не добавляется. Если полей с заданной меткой не найдено, возвращается пустой список.
- **def first(self, tag) -> Optional[Field]** – возвращает первое поле с указанной меткой либо `None`.

- **def first_as_dict(self, tag: int) -> dict** – возвращает первое из полей с указанной меткой в виде словаря «код - значение». Если указанное поле не найдено, возвращается пустой словарь.
- **def have_field(self, tag) -> bool** – выясняет, есть ли в записи поле с указанной меткой.
- **def insert_at(self, index, tag, value = None) -> Field** – вставка поля в указанной позиции. Возвращает вставленное поле.
- **def is_deleted(self) -> bool** – удалена ли запись?
- **def parse(self, text)** – разбор текстового представления записи (в серверном формате). Инфраструктурный метод, парный к `encode`.
- **def remove_at(self, index) -> Record** – удаляет поле в указанной позиции. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def remove_field(self, tag) -> Record** – удаляет поля с указанной меткой. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def reset(self) -> Record** – сбрасывает состояние записи, отвязывая её от базы данных. Поля при этом остаются нетронутыми. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def set_field(self, tag, value) -> Record** – устанавливает значение первого повторения указанного поля. Если указанное значение пустое, поле удаляется из записи. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def set_subfield(self, tag, code, value) -> Record** – устанавливает значение подполя в первом повторении указанного поля. Если указанное значение пустое, подполе удаляется из поля. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def __iter__(self)** – возможность итерироваться по полям.
- **def __iadd__(self, other)** – возможность использовать оператор „+=“ для добавления полей.
- **def __isub__(self, other)** – возможность использовать оператор „-=“ для удаления полей.
- **def __getitem__(self, item)** – возможность получения значения полей с помощью обращения по индексу.
- **def __setitem__(self, key, value)** – возможность присваивания значений полей с помощью обращения по индексу.
- **def __len__(self)** – количество полей в записи.
- **def __bool__(self)** – проверка, не пустая ли запись.
- **def __str__(self)** – получение текстового представления записи.

```
import irbis

SF = irbis.SubField

record = irbis.Record()
record.add(700, SF('a', 'Миронов')) \
    .add('b', 'А. В.') \
    .add('g', 'Алексей Владимирович')
record.add(200, SF('a', 'Заглавие книги')) \
    .add('e', 'Подзаголовочные сведения')
print(f"Заглавие: {record.fm(200, 'a')}")
print(record)
```

3.2 Field

Поле записи характеризуется числовой меткой в диапазоне от 1 до 2147483647 (на практике встречаются коды от 1 до 9999) и содержит значение до первого разделителя (опционально) и произвольное количество подполей (см. класс `SubField`).

Стандартом MARC у полей предусмотрены также два односимвольных индикатора, но ИРБИС вслед за ISIS их не поддерживает.

Кроме того, стандарт MARC предусматривает т. наз. «фиксированные» поля с метками от 1 до 9 включительно, которые не должны содержать ни индикаторов, ни подполей, но имеют строго фиксированную структуру. ИРБИС такие поля обрабатывает особым образом только в ситуации импорта/экспорта в формат ISO2709, в остальном же он их трактует точно так же, как и прочие поля (которые стандарт называет полями переменной длины).

Стандартом MARC предусмотрены метки в диапазоне от 1 до 999, все прочие являются самодеятельностью ИРБИС. Поля с нестандартными метками не могут быть выгружены в формат ISO2709.

Хотя технически поле может содержать одновременно и значение до первого разделителя, и подполя, но стандартом такая ситуация не предусмотрена, на практике она означает сбой. В стандарте MARC поле содержит либо значение либо подполя.

Начиная с версии 2018, ИРБИС64 резервирует метку 2147483647 для поля GUID - уникального идентификатора записи.

Порядок подполей в поле важен, т. к. на этот порядок завязана обработка т. наз. «вложенных полей».

Стандартом MARC предусмотрено, что внутри поля могут повторяться подполя с одинаковым кодом, однако, ИРБИС вслед за ISIS очень ограниченно поддерживает эту ситуацию (см. форматный выход `&umarci`).

Класс `Field` имеет следующие атрибуты:

Поле	Тип	Назначение
tag	int	Тег поля
value	str	Значение поля до первого разделителя
subfields	list	Список подполей

Класс `Field` содержит следующие методы:

- **def __init__(self, tag = 0, value = None, *subfields)** – конструктор, создаёт новый экземпляр поля в памяти клиента. Можно указать подполя, которыми будет наполнено поле.
- **def add(self, code, value = „“) -> Field** – добавление подполя с указанным кодом (и, возможно, значением) к записи. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def add_non_empty(self, code, value) -> Field** – добавление подполя с указанным кодом при условии, что значение поля не пустое. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def all(self, code) -> List[SubField]** – возвращает список всех подполей с указанным кодом.
- **def all_values(self, code) -> List[str]** – возвращает список значений всех подполей с указанным кодом. Пустые значения подполей в список не включаются.
- **def assign_from(self, other)** – присваивание от другого поля. Значение данного поля становится равным значению другого поля. В данное поле помещаются клоны подполей из другого поля. Метка поля не меняется.

- **def clear(self) -> Field** – очистка поля. Удаляются все подполя и значение до первого разделителя. Возвращает `self`, что позволяет использовать метод в цепочечных вызовах.
- **def clone(self) -> Field** – создание клона, т. е. полной копии поля. Подполя, если присутствуют, тоже клонируются. Возвращает клон поля.
- **def first(self, code) -> Optional[SubField]** – находит первое подполе с указанным кодом, возвращает найденное подполе или `None`.
- **def first_value(self, code) -> Optional[str]** – находит первое подполе с указанным кодом, возвращает значение найденного подполя или `None`.
- **def get_embedded_fields(self) -> List[Field]** – получение списка встроенных полей.
- **def get_value_or_first_subfield(self) -> Optional[str]** – выдаёт значение для ^*.
- **def have_subfield(self, code) -> bool** – выясняет, есть ли подполе с указанным кодом.
- **def insert_at(self, index: int, code value) -> Field** – вставляет подполе в указанную позицию. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def parse(self, line)** – разбор текстового представления поля (в серверном формате). Инфраструктурный метод.
- **def remove_at(self, index) -> Field** – удаляет подполе в указанной позиции. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def remove_subfield(self, code) -> Field** – удаляет все подполя с указанным кодом. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def replace_subfield(self, code, old_value, new_value) -> Field** – заменяет значение подполя с указанным кодом. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def set_subfield(self, code, value) -> Field** – устанавливает значение первого повторения подполя с указанным кодом. Если `value==None`, подполе удаляется. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def text(self) -> str** – Текстовое представление поля без кода.
- **def to_dict(self) -> dict** – Выдает словарь «код - значение подполя».
- **def __iter__(self)** – возможность итерироваться по подполям.
- **def __iadd__(self, other)** – возможность использовать оператор „+=“ для добавления подполей.
- **def __isub__(self, other)** – возможность использовать оператор „-=“ для удаления подполей.
- **def __getitem__(self, item)** – возможность получения значения подполей с помощью обращения по индексу.
- **def __setitem__(self, key, value)** – возможность присваивания значений подполей с помощью обращения по индексу.
- **def __len__(self)** – количество подполей в поле.
- **def __bool__(self)** – проверка, не пустое ли поле.
- **def __str__(self)** – получение текстового представления поля.

```
import irbis

field = irbis.Field(700)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
field.add('a', 'Миронов').add('e', 'A. B.')
field.add('g', 'Алексей Владимирович')
print(field)
```

3.3 SubField

Подполе характеризуется односимвольным кодом (как правило алфавитно-цифровым `A-Z, 0-9`, но бывают подполя с экзотическими кодами вроде `!`, `(` и др.) и содержит строковое значение (технически может быть пустым, но на практике пустое значение означает сбой).

Коды подполей не чувствительны к регистру. Как правило, ИРБИС приводит коды к верхнему регистру, но это не точно. :)

ИРБИС трактует код подполя „*“ как «значение до первого разделителя либо значение первого по порядку подполя» (смотря по тому, что присутствует в записи).

Поле	Тип	Назначение
code	str	Код подполя (односимвольный!)
value	str	Значение подполя

- **def __init__(self, code = „\0“, value = None)** – конструктор, создаёт новый экземпляр подполя в памяти клиента.
- **def assign_from(self, other)** – присваивание от другого поля: код и значение берутся от другого подполя.
- **def clone(self) -> SubField** – клонирование, т. е. создание точной копии подполя. Возвращает клон подполя.
- **def __bool__(self)** – проверка, не пустое ли подполе.
- **def __str__(self)** – получение текстового представления подполя.

```
import irbis

subfield = irbis.SubField('a', 'Подполе A')
print(subfield)
```

3.4 Класс RawRecord

Запись с нераскодированными полями/подполями. Класс определён в `irbis.ext` и содержит следующие поля:

Поле	Тип	Назначение
database	str	Имя базы данных, из которой загружена данная запись. Для вновь созданных записей `None`
mfn	int	Номер записи в мастер-файле. Для вновь созданных записей 0
status	int	Статус записи: логически удалена, отсутствует (аналогично `Record`)
version	int	Номер версии записи
fields	list	Список полей записи в нераскодированном виде (просто строки).

Определены следующие методы:

- **def __init__(self, *fields)** – конструктор, создаёт новый экземпляр записи в памяти клиента. Можно указать поля, которыми будет наполнена запись.
- **def clear(self) -> RawRecord** – очистка записи (удаление всех полей).
- **def clone(self) -> RawRecord** – создание клона, т. е. полной копии записи. Поля, если присутствуют, тоже клонируются. Возвращает клон записи.
- **def encode(self) -> List[str]** – кодирование записи в серверное представление. Инфраструктурный метод, парный к `parse`.
- **def is_deleted(self) -> bool** – удалена ли запись?
- **def parse(self, text)** – разбор текстового представления записи (в серверном формате). Инфраструктурный метод, парный к `encode`.
- **def remove_at(self, index) -> Record** – удаляет поле в указанной позиции. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def reset(self) -> Record** – сбрасывает состояние записи, отвязывая её от базы данных. Поля при этом остаются нетронутыми. Возвращает `self`, поэтому метод может использоваться в цепочечных вызовах.
- **def __iter__(self)** – возможность итерироваться по полям.
- **def __len__(self)** – количество полей в записи.
- **def __bool__(self)** – проверка, не пустая ли запись.
- **def __str__(self)** – получение текстового представления записи.

Загрузить сырую запись с сервера можно с помощью метода `read_raw_record`, сохранить на сервере можно с помощью метода `write_raw_record`.

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
record = client.read_raw_record(123)
record.fields.append('300#Комментарий к записи')
client.write_raw_record(record)
client.disconnect()
```

Глава 4

Прочие (вспомогательные) классы и функции

4.1 FoundLine

Строка найденной записи, может содержать результат расформатирования найденной записи. Содержит два поля:

Поле	Тип	Значение
mfn	int	MFN найденной записи
description	str	Результат расформатирования записи (опционально)

Пример применения см. раздел `SearchParameters` в данной главе.

4.2 MenuFile и MenuLine

Файл меню (справочника) `MenuFile` состоит из пар строк `MenuLine`.

`MenuLine` содержит следующие поля:

Поле	Тип	Значение
code	str	Условный код (должен быть непустым)
comment	str	Комментарий к коду (может быть пустым)

`MenuFile` содержит единственное поле `entries` - список элементов типа `MenuLine`. Определены следующие методы:

- **def get_entry(self, code: str) -> Optional[MenuEntry]** – получение элемента по коду. Внимание! Если в меню присуствует несколько элементов с одинаковым кодом, возвращается первый найденный.

- **def get_value(self, code: str, default_value: Optional[str] = None) -> Optional[str]** – получение комментария по коду (задаётся также значение по умолчанию, которое возвращается, когда соответствующий элемент не найден).
- **def parse(self, lines: List[str]) -> None** – разбор текстового представления меню (справочника).
- **def save(self, filename: str) -> None** – сохранение меню в локальный текстовый файл.

Загрузить меню (справочник) из локального текстового файла можно с помощью функции `load_menu`:

```
import irbis

menu = irbis.load_menu(r'C:\path\file.mnu')
print(menu.get_value('a', '???'))
```

Загрузить меню (справочник) с сервера ИРБИС64 можно с помощью функции `read_menu`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
menu = client.read_menu('3.IBIS.ii.mnu')
value = menu.get_value('1')
print(f"Value is {value}")
client.disconnect()
```

4.3 IniFile, IniSection и IniLine

INI-файл, состоящий из секций, которые в свою очередь состоят из строк вида «ключ=значение».

Клиент получает свой INI-файл при подключении к серверу. Он хранится в свойстве `ini_file` класса `Connection`.

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
ini = client.ini_file
dbnnamecat = ini.get_value('Main', 'DBNNAMECAT')
print(f"DBNNAMECAT={dbnnamecat}")
client.disconnect()
```

Также можно прочитать произвольный INI-файл с сервера ИРБИС64 с помощью метода `read_ini_file`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
ini = client.read_ini_file('3.RDR.KO.INI')
number = ini.get_value('SEARCH', 'ItemNumb')
print(f"Число элементов={number}")
client.disconnect()
```

4.4 TreeFile и TreeNode

TRE-файл – древовидный текстовый справочник. Состоит из узлов, каждый из которых может быть либо узлом самого верхнего уровня, либо дочерним по отношению к узлу более высокого уровня. Уровень узла определяется величиной отступа, с которым соответствующая строка записана в файле справочника.

Класс `TreeNode` соответствует узлу дерева. Содержит следующие поля:

Поле	Тип	Назначение
children	list	Список дочерних узлов (может быть пустым).
value	str	Текстовое значение узла (не может быть пустым).
level	int	Уровень узла (0 = узел самого верхнего уровня).

Класс `TreeFile` описывает TRE-файл в целом. Содержит следующие поля:

Поле	Тип	Назначение
roots	list	Список узлов самого верхнего уровня (корневых).

Прочитать древовидный справочник из текстового файла можно с помощью функции `load_tree_file`:

```
import irbis

tree = irbis.load_tree_file(r'C:\IRBIS64\Datai\IBIS\ii.tre')
print(tree.roots[0].value)
```

Загрузить TRE-файл с сервера ИРБИС64 можно с помощью функции `read_tree_file`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
tree = client.read_tree_file('2.IBIS.II.tre')
print(tree.roots[0].value)
client.disconnect()
```

4.5 DatabaseInfo

Информация о базе данных ИРБИС. Класс содержит следующие поля:

Поле	Тип	Назначение
name	str	Имя базы данных (непустое).
description	str	Описание в произвольной форме (может быть пустым).
max_mfn	int	Максимальный MFN.
logically_deleted	list	Перечень MFN логически удалённых записей (может быть пустым).
physically_deleted	list	Перечень MFN физически удалённых записей (может быть пустым).
nonactualized	list	Перечень MFN неактуализированных записей (может быть пустым).
database_locked	bool	Флаг: база заблокирована на ввод.
read_only	bool	Флаг: база доступна только для чтения.

Получение информации о конкретной базе данных (заполняются только поля `max_mfn`, `logically_deleted`, `physically_deleted`, `nonactualized`, `database_locked`):

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
info = client.get_database_info('IBIS')
print(f"Удалённых записей: {len(info.logically_deleted)}")
client.disconnect()
```

Получить список баз данных, доступных для данного АРМ, можно с помощью метода `list_databases` (заполняются только поля `name`, `description`, `read_only`).

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
databases = client.list_databases('1..dbnam2.mnu')
for db in databases:
    print(f"{db.name} => {db.description}")
client.disconnect()
```

4.6 ProcessInfo

Информация о запущенном на ИРБИС-сервере процессе.

4.7 VersionInfo

Информация о версии ИРБИС-сервера.

4.8 ClientInfo

Информация о клиенте, подключенном к серверу ИРБИС (не обязательно о текущем).

4.9 UserInfo

Информация о зарегистрированном пользователе системы (по данным `client_m.mnu`). Определены следующие поля:

Поле	Тип	Назначение
number	str	Номер по порядку в списке.
name	str	Логин пользователя.
password	str	Пароль.
cataloger	str	Доступность АРМ «Каталогизатор».
reader	str	Доступность АРМ «Читатель».
circulation	str	Доступность АРМ «Книговыдача».
acquisitions	str	Доступность АРМ «Комплектатор».
provision	str	Доступность АРМ «Книгообеспеченность».
administrator	str	Доступность АРМ «Администратор».

Если строка доступа к АРМ пустая, то доступ пользователя к соответствующему АРМ запрещен.

Получить список зарегистрированных в системе пользователей можно с помощью метода `list_users`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
users = client.list_users()
for user in users:
    print(f"{user.name} => {user.password}")
client.disconnect()
```

Обновить список зарегистрированных пользователей можно с помощью метода `update_user_list`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
users = client.list_users()
checkhov = irbis.UserInfo()
checkhov.number = str(len(users))
checkhov.name = 'Чехов'
checkhov.password = 'Каштанка'
checkhov.cataloger = 'irbisc_chekhov.ini'
users.append(checkhov)
client.update_user_list(users)
client.disconnect()
```

4.10 TableDefinition

Данные для метода `print_table`.

4.11 ServerStat

Статистика работы ИРБИС-сервера.

4.12 PostingParameters

Параметры для запроса постингов с сервера. Содержит следующие поля:

Поле	Тип	Значение
database	str	Имя базы данных
first	int	Номер первого постинга (нумерация с 1)
fmt	str	Опциональный формат
number	int	Количество затребуемых постингов
terms	list	Список терминов, для которых требуются постинги

Получить список постингов с сервера можно с помощью функции `read_postings`. Класс `PostingParameters` предоставляет возможность тонко настроить эту функцию:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
params = irbis.PostingParameters()
params.database = 'IBIS'  # Имя базы данных
params.first = 1  # Постинги, начиная с первого
params.number = 10  # Требуем до 10 постингов
params.terms = ['K=БЕТОН']  # Термины
postings = client.read_postings(params)
for posting in postings:
    print(f"MFN={posting.mfn}, TAG={posting.tag}, OCC={posting.occurrence}")
client.disconnect()
```

4.13 TermParameters

Параметры для запроса терминов с сервера. Содержит следующие поля:

Поле	Тип	Значение
database	str	Имя базы данных
number	int	Количество затребуемых терминов
reverse	bool	Выдавать термины в обратном порядке?
start	str	Стартовый термин
format	str	Опциональный формат

Получить список терминов с сервера можно с помощью функции `read_terms`. Класс `TermParameters` предоставляет возможность тонко настроить эту функцию:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
params = irbis.TermParameters()
params.database = 'IBIS'  # Имя базы данных
params.number = 10  # Требуем выдать до 10 терминов
params.reverse = True  # В обратном порядке
params.start = 'К=БЕТОН'
terms = client.read_terms(params)
for term in terms:
    print(f"{term.text} => {term.count}")
client.disconnect()
```

4.14 TermInfo

Информация о термине поискового словаря. Содержит всего два поля:

Поле	Тип	Значение
count	int	Количество постингов (вхождений) термина в поисковом словаре
text	str	Собственно значение термина

Имейте в виду, что термин может входить в одну и ту же запись несколько раз, и все эти вхождения будут отражены в словаре.

Получить список терминов с сервера можно с помощью функции `read_terms`.

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
terms = client.read_terms(('К=БЕТОН', 10))
for term in terms:
    print(f"{term.text} => {term.count}")
client.disconnect()
```

4.15 TermPosting

Постинг (вхождение) термина в поисковом индексе. Содержит следующие поля:

Поле	Тип	Значение
mfn	int	MFN записи
tag	int	Метка поля
occurrence	int	Повторение поля
count	int	Позиция в поле
text	str	Опциональный результат расформатирования

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
postings = client.read_postings('K=БЕТОН')
for posting in postings:
    print(f"MFN={posting.mfn}, TAG={posting.tag}, OCC={posting.occurrence}")
client.disconnect()
```

4.16 SearchParameters

Параметры для поиска записей (методы `search` и `search_ex`). Содержит следующие поля:

Поле | Тип | Значение по умолчанию | Назначение ———== ==-== ==————-== ==——— database | str | None | Имя базы данных (опционально) expression | str | None | Выражение для поиска по словарю (быстрый поиск) first | int | 1 | Индекс первой из возвращаемых записей format | str | None | Опциональный формат для найденных записей max_mfn | int | 0 | Максимальный MFN для поиска (опционально) min_mfn | int | 0 | Минимальный MFN для поиска (опционально) number | int | 0 | Количество возвращаемых записей (0 = все) sequential | str | None | Выражение для последовательного поиска (медленный поиск)

Если имя базы данных не задано, подразумевается текущая база данных, к которой подключен клиент.

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
params = irbis.SearchParameters()
params.database = 'IBIS'  # По какой базе щем
params.expression = '"A=ПУШКИН$"'  # Поиск по словарю
params.number = 10  # Выдать не больше 10 записей
params.format = '@brief'  # Форматирование найденных записей
# Последовательнсый поиск среди отобранных по словарю записей
params.sequential = "if v200^a:'Сказки' then '1' else '0' fi"
found = client.search_ex(params)
for line in found:
    record = client.read_record(line.mfn)
    print(record.fm(200, 'a'))
    # Получаем расформатированную запись
    print(line.description)
```

4.17 SearchScenario

Сценарий поиска. Содержит следующие поля:

Поле	Тип	Значение
name	str	Наименование поискового атрибута (автор, заглавие и т. п.)
prefix	str	Префикс соответствующих терминов в поисковом словаре (может быть пустым)
type	int	Тип словаря для соответствующего поиска
menu	str	Имя файла справочника (меню)
old	str	Имя формата (без расширения)
correction	str	Способ корректировки по словарю
truncation	bool	Исходное положение переключателя «усечение»
hint	str	Текст подсказки/предупреждения
mod_by_dic_auto	str	Параметр пока не задействован
logic	int	Применимые логические операторы
advance	str	Правила автоматического расширения поиска на основе авторитетного файла или тезауруса
format	str	Имя формата показа документов

Нестандартные сценарии поиска можно загрузить с сервера с помощью метода `read_search_scenario`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
scenarios = client.read_search_scenario('2.IBIS.SEARCH.INI')
print(f"Всего сценариев поиска: {len(scenarios)}")
for scenario in scenarios:
    print(f"{scenario.name} => {scenario.prefix}")
client.disconnect()
```

Стандартный сценарий поиска содержится в INI-файле, полученном клиентом с сервера при подключении:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
scenarios = irbis.SearchScenario.parse(client.ini_file)
print(f"Всего сценариев поиска: {len(scenarios)}")
for scenario in scenarios:
    print(f"{scenario.name} => {scenario.prefix}")
client.disconnect()
```

4.18 ParFile

PAR-файл – содержит пути к файлам базы данных ИРБИС. Определены следующие поля:

Поле	Тип	Значение
xrf	str	Путь к XRF-файлу
mst	str	Путь к MST-файлу
cnt	str	Путь к CNT-файлу
n01	str	Путь к N01-файлу
n02	str	В ИРБИС64 не используется
l01	str	Путь к L01-файлу
l02	str	В ИРБИС64 не используется
ifp	str	Путь к IFP-файлу
any	str	В ИРБИС64 не используется
pft	str	Путь к PFT-файлам
ext	str	Путь к полнотекстовым файлам

Как правило, все поля, кроме `ext`, имеют одно и то же значение, т. к. вся база данных, кроме полнотекстовых файлов, хранится в одной и той же директории.

Загрузить PAR-файл из локального текстового файла можно с помощью функции `load_par_file`:

```
import irbis

par = irbis.load_par_file(r'C:\IRBIS64\DataI\IBIS.par')
# Получаем путь к MST-файлу
print(par.mst)
```

Загрузить PAR-файл с сервера ИРБИС64 можно с помощью функции `read_par_file`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
par = client.read_par_file('1..IBIS.par')
# Получаем путь к MST-файлу
print(par.mst)
client.disconnect()
```

4.19 OptFile и OptLine

OPT-файл – файл оптимизации рабочих листов и форматов показа.

Типичный OPT-файл выглядит так:

```
920
5
PAZK  PAZK42
PVK   PVK42
SPEC  SPEC42
J     !RPJ51
```

(continues on next page)

(продолжение с предыдущей страницы)

```
NJ    !NJ31
NJP   !NJ31
NJK   !NJ31
AUNTD AUNTD42
ASP   ASP42
MUSP  MUSP
SZPRF SZPRF
BOUNI BOUNI
IBIS  IBIS
+++++ PAZK42
*****
```

Класс `OptLine` представляет одну строку в OPT-файле. Содержит следующие поля.

Поле	Тип	Значение
pattern	str	Шаблон для имени рабочего листа (см. ниже).
worksheet	str	Имя соответствующего WS-файла (без расширения).

Шаблон для имени может содержать символ „+“, означающий «любой символ, в том числе его отсутствие».

Класс `OptFile` представляет OPT-файл в целом. Содержит следующие поля.

Поле	Тип	Значение
lines	list	Список строк (`OptLine`).
length	int	Длина шаблона в символах.
tag	int	Метка поля в записи, хранящего имя рабочиего листа.

Определены следующие методы:

- **def parse(self, text)** – разбор текстового представления OPT-файла.
- **def resolve_worksheet(self, tag: str) -> Optional[str]** – поиск имени WS-файла для указанного значения (например, «SPEC»). Если соответствующего имени не найдено, возвращается `None`.
- **def save(self, filename)** – сохранение в текстовый файл с указанным именем.

Прочитать OPT-файл из локального файла можно с помощью функции `load_opt_file`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
opt = irbis.load_opt_file(r"C:\IRBIS64\Datai\IBIS\WS31.opt")
record = client.read_record(123)
worklist = record.fm(opt.tag)
ws_name = opt.resolve_worksheet(worklist)
print(f"WS name: {ws_name}")
client.disconnect()
```

Загрузить OPT-файл с сервера можно с помощью функции `read_opt_file`:

```
import irbis

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
opt = client.read_opt_file('2.IBIS.WS31.opt')
record = client.read_record(123)
worklist = record.fm(opt.tag)
ws_name = opt.resolve_worksheet(worklist)
print(f"WS name: {ws_name}")
client.disconnect()
```

4.20 GblStatement и GblSettings

Классы для глобальной корректировки базы данных.

4.21 ClientQuery

Клиентский запрос. Инфраструктурный класс.

4.22 ServerResponse

Ответ сервера. Инфраструктурный класс.

Глава 5

Построитель запросов

Класс `Search` представляет собой построитель запросов в синтаксисе прямого поиска ИРБИС64. В нём имеются следующие статические методы:

- **def all() -> Search** – отбор всех документов в базе. Фактически строит запрос `I=$`.
- **def equals(prefix, *values) -> Search** – поиск по совпадению с одним из перечисленных значений. Возвращает построитель запросов для последующих цепочечных вызовов.

Экземплярные методы:

- **def and_(self, *items) -> Search** – логическое И. Возвращает построитель запросов для последующих цепочечных вызовов.
- **def not_(self, text) -> Search** – логическое НЕ. Возвращает построитель запросов для последующих цепочечных вызовов.
- **def or_(self, *items) -> Search** – логическое ИЛИ. Возвращает построитель запросов для последующих цепочечных вызовов.
- **def same_field_(self, *items) -> Search** – логический оператор «в том же поле». Возвращает построитель запросов для последующих цепочечных вызовов.
- **def same_repeat_(self, *items) -> Search** – логический оператор «в том же повторении поля». Возвращает построитель запросов для последующих цепочечных вызовов.

Кроме того, предоставляются следующие функции, значительно упрощающие формирование запроса:

Функция	Поиск по
author	автору
bbk	индексу ББК
document_kind	виду документа
keyword	ключевым словам
language	языку текста
magazine	заглавию журнала
mhr	месту хранения
number	инвентарному номеру
place	месту издания (городу)
publisher	издательству
rzn	разделу знаний
subject	предметной рубрике
title	заглавию
udc	индексу УДК
year	году издания

Пример применения построителя запросов:

```
import irbis
from irbis.builder import author, title

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
expression = author('ПУШКИН$').and_(title('СКАЗКИ$'))
found = client.search_count(expression)
print(f"Найдено: {found}")
client.disconnect()
```

Глава 6

Экспорт/импорт

Модуль `irbis` содержит функции, необходимые для экспорта и импорта записей в как в текстовом формате, принятом в системах ИРБИС32 и ИРБИС64, так и в международном формате ISO2709.

6.1 Текстовый экспорт/импорт

Текстовый обменный формат специфичен для ИРБИС, он не поддерживается другими библиотечными системами. Вот как выглядит типичный файл в обменном формате ИРБИС:

```
#920: SPEC
#102: RU
#101: rus
#606: ^AХУДОЖЕСТВЕННАЯ ЛИТЕРАТУРА (ПРОИЗВЕДЕНИЯ)
#919: ^Arus^N02  ^KPSBO^Gca
#60: 10
#961: ^ZДА^AШукшин^BВ. М.^GВасилий Макарович
#210: ^D1998
#610: РУССКАЯ ЛИТЕРАТУРА
#610: РАССКАЗЫ
#1119: 7ef4c9af-f1d3-4adc-981b-5012463155a1
#900: ^B03^C11a^Xm^Ta
#215: ^A528^3в пер.
#200: ^VКн. 3^ACтранные люди
#10: ^A5-86150-048-7^D80
#907: ^CПК^A20180613^BNovikovaIA
#461: ^CСобрание сочинений : в 6 кн.^FB. М. Шукшин^GНадежда-1^H1998^Z1998^XШукшин,␣
↪Василий Макарович^DМосква^U1
#903: -051259089
#910: ^A0^B1759089^DФ104^U2018/45^Y45^C20180607
#905: ^21^D1^J1^S1
*****
```

Функция read_text_record определена следующим образом:

```
def read_text_record(stream: TextIO) -> Optional[Record]:
    pass
```

Она принимает в качестве аргумента текстовый поток, считывает запись и возвращает её. Если достигнут конец потока, возвращается None. Вот как можно использовать эту функцию:

```
import irbis

filename ='data/records.txt'
with open(filename, 'rt', encoding='utf-8') as stream:
    while True:
        # Считываем все записи из файла
        record = irbis.read_text_record(stream)
        if not record:
            break
        # и распечатываем заглавия
        print(record.fm(200, 'a') or '(null)')
```

Обратите внимание, что функция принимает любой вид текстового потока: не только файло, но и сокет, консоль, массив символов в оперативной памяти и т. д.

Функция write_text_record, обратная read_text_record, определена следующим образом:

```
def write_text_record(stream: TextIO, record: Record) -> None:
    pass
```

В качестве аргумента она принимает текстовый поток, в который разрешена запись. Это может быть как файл, так и сокет, консоль, вообще любой объект, реализующий протокол текстового вывода. Вот как можно использовать данную функцию:

```
import irbis
import tempfile

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
# Находим и загружаем с сервера 5 сказок Пушкина
found = client.search_read('"A=ПУШКИН$" * "T=СКАЗКИ$"', 5)
client.disconnect()

# Создаём временный текстовый файл
with tempfile.NamedTemporaryFile(mode='wt',
    encoding='utf-8', delete=False) as stream:
    # Сохраняем найденные записи в файле
    for record in found:
        irbis.write_text_record(stream, record)
    # Не забываем записать признак окончания
    stream.write(irbis.STOP_MARKER)
```

6.2 Формат ISO2709

```
import irbis

filename ='data/records.iso'
with open(filename, 'rb', encoding='utf-8') as stream:
    while True:
        # Считываем все записи из файла
        record = irbis.read_iso_record(stream)
        if not record:
            break
        # и распечатываем заглавия
        print(record.fm(200, 'a') or '(null)')
```

```
import irbis
import tempfile

client = irbis.Connection()
client.connect('host', 6666, 'librarian', 'secret')
# Находим и загружаем с сервера 5 сказок Пушкина
found = client.search_read('"A=ПУШКИН$" * "T=СКАЗКИ$"', 5)
client.disconnect()

# Создаём временный текстовый файл
with tempfile.NamedTemporaryFile(mode='wb',
    encoding='utf-8', delete=False) as stream:
    # Сохраняем найденные записи в файле
    for record in found:
        irbis.write_iso_record(stream, record)
```

Глава 7

Указатели и таблицы

- genindex
- modindex
- search