Система META

Вы уже узнали про метод Utils.getKeysSortedByValue, в него передается Map и вариант сортировки (А-Я (false) или Я-А (true)), и метод возвращает ключи, отсортированные по значениям мапа.

```
List<String> lPartner = Utils.getKeysSortedByValue(hmPartner, false);
```

Вот еще несколько очень полезных функций, но уже из пакета appt.meta3:

`int pnInt = Util.s2i(pageNum);` - принимает на вход строку и преобразует ее в число, если не получилось - возвращает 0.

`String idTour = Util.s2s(req.getParameter("idTour"));` - принимает на вход строку и возвращает строку, если изначальная строка была null - после преобразования вернется пустая строка.

Так же есть методы s2l и s2d - они преобразуют Srting в long и double соответсвенно.

`int pnInt = Ob0.id2type(idOb);` - принимает на вход строковый id объекта и возвращает его тип.

Redis

В первом модуле (Сети и Протоколы) вы уже познакомились с этой in memory database и ее особенностями, теперь рассмотрим конкретное применение редиса в проектах на основе META и методами для работы с ним. Эти методы живут в классе appt.meta3.Ob3. Поскольку в системе может существовать много различных инстансов редиса для разных данных, для обращения к конкретному редису или дереву редисов используется формальное имя - "схема", существующая только в рамках конкретного проекта.

Работа с простыми ключами

В качестве ключа можно использовать строку или последовательность байт.

```
Ob3.puts(mains, REDIS_FREE_VOUCHERS, "bso_" + bl.id,
String.valueOf(numVoucher));
```

- Ob3.puts - кладет значение в формате строки или последовательности байт по переданному ключу в формате строки или последовательности байт во все мастера редисов схемы, если ключ не существует - он будет создан, если существует - будет обновлено его значение.
- REDIS_FREE_VOUCHERS - название редиса
- ключом является строка "bso_" + bl.id

```
byte[] K = ("main"+idTemplate+data).getBytes();
```

- получаем последовательность байт

```
Ob3.puts(mains, "lprice", "lprice", K, Ob0.toBytes(ob));
```

- lprice - название редиса и название коннекшена (для разведения пула можно передавать свои имена коннекшна, но использовать это нужно только, если вы понимаете, что делаете).
- ключом является последовательность байт K

В качестве значения может лежать строка, последовательность байт (используется в том числе для представления объектов Obb в редисе).

```
Ob3.puts(mains, REDIS_FREE_VOUCHERS, "bso_" + bl.id,
String.valueOf(numVoucher));
```

- в значении лежит строка String.valueOf(numVoucher)

```
Ob3.puts(mains, "lprice", "lprice", K, Ob0.toBytes(ob));
```

- Ob0.toBytes - перевод объекта в массив байт
- в значении лежит последовательность байт

Для работы с редисом потребуются следующие команды:

```
byte[] K = ("main"+idTemplate+data).getBytes();
```

- кодирует данную строку в последовательность байт

```
byte[] OB = Ob3.get(mains, "lprice", "lprice", K );
```

- lprice - это название редиса и имя коннекшена, они меняются в зависимости от редиса.

Функция возвращает значение в редисе, которое лежит по ключу K. Метод будет читать значение из ближайшего найденного редиса с указанной схемой, даже если это будет слейв.

Obb ob = Ob0.fromBytes(OB);

Ob0.fromBytes - перевод объекта из массива байт.

Ob3.puts(mains, "lprice", "lprice", K, Ob0.toBytes(ob));

Ob0.toBytes - перевод объекта в массив байт

Ob3.puts кладем данные в редис lprice, puts - нужен, чтобы положить значение по ключу.

Полная версия функции, которая кладет данные в редис:

Ob3.puts(ResourceBundle mains, String schem, byte[] key, byte[] val, long time, boolean nx, int expire, String oper);

Описание параметров:

par1: ResourceBundle (обязательный) Название файла с параметрами подключения

par2: schem - название редиса

par3: key - ключ

par4: val - значение

par5: time - показывает насколько "далеко" от текущего хоста искать машину с нужным редисом и нужным типом коннекшна. Чем параметр меньше, тем "ближе" должен быть хост. Самый близкий - искать только на себе, дальше - искать на соседях по стойке, на соседях по площадке, и далее на дальних площадках. Это все работает только в контексте распределенной системы, где известны расстояния (обычно это длина пинга) между площадками, датацентрами и т.п.
1000000000L - это значение используется для поиска вообще везде по всей системе, это не обычное его использование, просто самое безопасное, чтобы не было потери коннекшнов в софте, часто специально зажимается круг поиска из-за контекста задачи.

par6: nx - вызываем функцию setnx или нет, эта функция записывает строковое значение, если такого ключа ещё нет в базе данных. По умолчанию ставим false и вызываем функцию set, которая записывает строковое значение в переданный ключ. Если ключ до этого существовал, то он будет перезаписан.

par7: expire - время жизни ключа в секундах, по истечении которого ключ автоматически удаляется

par8: oper - C - когда нужен редис, в который можно писать

S - когда достаточен редис, из которого можно только читать

Важно знать, что ключи должны быть уникальными в рамках одной схемы.

Задания

1. Написать интерфейс, чтобы отображать данные из редиса. Зашли в интерфейс, написать откуда взяли данные, если в редисе нет ключа, написать, что заново положили ключ+значение(объект46) в редис и положить этот ключ+значение, или просто достали из редиса - написали об этом. В интерфейсе по id 46 надо отображать следующую информацию: 1000348 Название, 1000350 id номера, 1046222729 тип стоимости (0/1/2 - С/БНС/НС).Все кладем в редис rev, время жизни ключа = 3 минуты.(id для теста 104610001184, 104610000865, 104610000863, 104610000831, 104610000807, 104610000783, 104610000561, 104610000529, 104610000495)

2. Добавить любой ключ в редис и научиться его удалять.

3. Достать все объекты 36 типа и положить каждый из них в редис на час. Засечь время, за которое по 1 объекту достанете все из редиса. И Время для получения из базы тех же объектов по 1.

4. Какие данные, по вашему мнению, должны храниться в редисе?