Задание по программированию: Клиент для отправки метрик

В крупных проектах, с большим количеством пользователей, необходимо тщательно наблюдать за всеми процессами, происходящими в нем. Информация о процессах может быть представлена различными численными показателями, например: количество запросов к вашему приложению, время ответа вашего сервиса на каждый запрос, количество пользователей в сутки и другие. Эти различные численные показатели мы будем называть метриками.

Для сбора, хранения и отображения подобных метрик существуют готовые решения, например **Graphite, InfluxDB**. Мы в рамках курса разработаем свою систему для сбора и хранения метрик, основанную на клиент-серверной архитектуре.

На этой неделе мы начнем с разработки клиента для отправки и получения метрик. На следующей неделе, в качестве финального задания, вам будет предложено реализовать сервер для хранения метрик.

Протокол взаимодействия

Прежде, чем приступить к описанию задания, рассмотрим протокол взаимодействия, по которому будет происходить обмен данными между клиентом и сервером.

Клиент и сервер взаимодействуют между собой по простому текстовому протоколу через TCP сокеты. Текстовый протокол имеет главное преимущество – наглядность, можно просматривать диалог взаимодействия клиентской и серверной стороны без использования дополнительных инструментов.

Общий формат запросов и ответов.

Протокол поддерживает два вида запросов к серверу со стороны клиента:

- отправка данных для сохранения их на сервере

- получения сохраненных данных

Общий формат запроса клиента:

```
<команда> <данные запроса><\n>
```

где:

- **<команда>** - команда сервера (команда может принимать одно из двух значений: **put** — сохранить данные на сервере, **get** — вернуть сохраненные данные с сервера),

- **<данные запроса>** - данные запроса (их формат мы подробно разберем ниже в примере),

- **<\n>** - символ переноса строки.

Обратим ваше внимание на пробел между командой и данными запроса и его отсутствием между данными и символом перевода на новую строку.

Общий формат ответов сервера:

```
<статус ответа><\n><данные ответа><\n\n>
```

где:

- **<статус ответа>** - статус выполнения команды, допустимы два варианта: **«ok»** - команда успешно выполнена на сервере и **«error»** - выполнение команды завершилось ошибкой

- **<данные ответа>** - не обязательное поле (формат ответа и случаи его отсутствия будут рассмотрены в примере ниже)

- **<\n\n>** - два символа переноса строки.

Обратите внимание, что статус ответа и данные ответа разделены символом перевода строки **<\n>**.

Пример взаимодействия сервера и клиента.

Для наглядности рассмотрим протокол взаимодействия между клиентом и сервером на конкретном примере. В примере мы будем, собирать метрики с данными о работе операционной системы: **cpu** (загрузка процессора), **usage** (потребление памяти), **disk_usage** (потребление места на жестком диске), **network_usage** (статистика сетевых интерфейсов). Такие данные могут понадобится для контроля загрузки серверов и прогноза по расширению парка железа компании - проще говоря для мониторинга.

Какие данные мы будем сохранять?

Для каждой метрики (**<key>**) мы будем хранить данные о ее значениях (**<value>**) и времени, когда производилось измерение (**<timestamp>**) . Поскольку, в реальной жизни серверов может быть несколько, необходимо различать данные полученные от разных серверов (в нашем примере имеются в наличии два сервера **palm** и **eardrum**). Договоримся об именовании **<key>**, в примере мы будем определять их по правилу:

```
<название сервера>.<название метрики><название сервера>.<название

    метрики>
```

Примеры названий метрик: **"palm.cpu", "eardrum.memory"**.

Таким образом на сервере по каждому ключу будет сохранятся список данных конкретных измерений (пара: значение, время измерения).

Запросы клиента.

Рассмотрим пример отправки на сервер данных для сохранения. Пусть у нас имеются данные измерений - загрузка процессора **«cpu»** на сервере **"palm"** во время **1150864247** была равна **23.7** процента. Строка запроса в этом случае будет иметь вид:

```
put palm.cpu 23.7 1150864247\n
```

В запросе на сохранение мы можем передать данные только об одном измерении.

Чтобы получить с сервера данные, сохраненные по ключу «palm.cpu», необходимо в данных запроса просто передать имя ключа:

```
get palm.cpu\n
```

Для случая, когда необходимо получить все хранимые на сервере данные, в качестве ключа используется символ звездочки **«*»**. Пример строки запроса:

```
get *\n
```

Ответы сервера.

Допустим, что на сервере хранятся данные:

```
key          | value | timestamp

-----------------------------------

"palm.cpu"    |  2.0  | 1150864247

"palm.cpu"    |  0.5  | 1150864248

"eardrum.cpu" |  3.0  | 1150864250
```

Тогда в ответ на запрос о получении данных по ключу **"palm.cpu"** сервер отправит строку:

```
ok\npalm.cpu 2.0 1150864248\npalm.cpu 0.5 1150864248\n\n
```

Данные ответа содержат данные о каждой сохраненной записи с ключом **"palm.cpu"** (метрика, значение, временная метка разделенные пробелом), которые разделены символом перевода строки **«\n»**.

Строка ответа сервера на запрос о получении всех хранящихся на сервере данных (в качестве ключа передано **«*»**) в нашем случае будет таким:

```
ok\npalm.cpu 2.0 1150864247\npalm.cpu 0.5 1150864248\neardrum.cpu 3.0

    1150864250\n\n
```

В случаях:

- когда в запросе на получение данных передан не существующий ключ

- успешного выполнения команды сохранения данных **put**

сервер отправляет клиенту строку со статусом «ok» и пустым полем с данными ответа:

```
ok\n\n
```

Если в параметре запроса переданы не валидные данные (например: нарушен формат запроса, ошибочная команда или значения value и timestamp не могут быть приведены к необходимому типу данных) сервер отправляет строку со статусом ответа **«error»** и данными ответа **«wrong command»**:

```
error\nwrong command\n\n
```

Реализация клиента.

Необходимо реализовать класс **Client**, в котором будет инкапсулировано соединение с сервером, клиентский сокет и методы для получения (**get**) и отправки (**put**) метрик на сервер. Отправка и получение данных в методах **get** и **put** должна быть реализована в соответствии с протоколом, описанным выше. В конструктор класса Client должна передаваться адресная пара хост и порт, а также необязательный аргумент **timeout** (имеющий значение по умолчанию - **None**). Соединение с сервером устанавливается при создании экземпляра класса Client и не должно разрываться между запросами.

Пример создания объекта клиента и отправки запросов на сервер:

```
>>> from solution import Client

>>> client = Client("127.0.0.1", 8888, timeout=15)

>>> client.put("palm.cpu", 0.5, timestamp=1150864247)

>>> client.put("palm.cpu", 2.0, timestamp=1150864248)

>>> client.put("palm.cpu", 0.5, timestamp=1150864248)

>>> client.put("eardrum.cpu", 3, timestamp=1150864250)

>>> client.put("eardrum.cpu", 4, timestamp=1150864251)

>>> client.put("eardrum.memory", 4200000)

>>> print(client.get("*"))
```

Метод put.

Метод **put** принимает в качестве параметров: название метрики, численное значение и необязательный именованный параметр **timestamp**. Если пользователь вызвал метод **put** без аргумента **timestamp**, то клиент автоматически должен подставить значение временной отметки, полученное с помощью вызова **int(time.time()).**

Метод **put** не возвращает ничего в случае успешной отправки и выбрасывает пользовательское исключение **ClientError** в случае не успешной.

Метод get.

Метод **get** принимает в качестве параметра имя метрики, значения которой мы хотим получить. В качестве имени метрики можно использовать символ **«*»**, о котором мы упоминали в описании протокола.

Метод **get** возвращает словарь с метриками (смотрите пример ниже) в случае успешного получения ответа от сервера и выбрасывает исключение **ClientError** в случае не успешного.

Клиент получает данные от сервера в текстовом виде, метод **get** должен обработать строку ответа и вернуть словарь с полученными ключами с сервера. Значением ключей в словаре является список кортежей:

```
[(timestamp1, metric_value1), (timestamp2, metric_value2), …]
```

Значение **timestamp** и **metric_value** должны быть преобразованы соответственно к типам **int** и **float**. Список должен быть отсортирован по значению **timestamp** (по возрастанию).

Пример возвращаемого значения при успешном вызове **client.get("palm.cpu")**:

```
{
  'palm.cpu': [
    (1150864247, 0.5),
    (1150864248, 0.5)
  ]
}
```

Обратите внимание, что сервер хранит данные с максимальным разрешением в одну секунду. Это означает, что если в одну и ту же секунду отправить две одинаковые метрики, то будет сохранено только одно значение, которое было обработано последним. Все остальные значения будут перезаписаны. По этой причине запрос по ключу **"palm.cpu"** вернул данные двух измерений.

Пример возвращаемого значения при успешном вызове **client.get("*")**:

```
{
  'palm.cpu': [
    (1150864247, 0.5),
    (1150864248, 0.5)
  ],
  'eardrum.cpu': [
    (1150864250, 3.0),
    (1150864251, 4.0)
  ],
  'eardrum.memory': [
    (1503320872, 4200000.0)
  ]
}
```

Если в ответ на get-запрос сервер вернул положительный ответ **"ok\n\n"**, но без данных (то есть данных по запрашиваемому ключу нет), то метод **get** клиента должен вернуть пустой словарь.

Итак, в качестве решения вам необходимо предоставить модуль с реализованным в нем классом **Client**, пользовательским исключением **ClientError**. В классе **Client** должны быть доступны методы **get** и **put** с описанной выше сигнатурой. При вызове методов **get** и **put** клиент должен посылать сообщения в TCP-соединение с сервером (в соответствии с описанным текстовым протоколом), получать ответ от сервера и возвращать словарь с данными, в формате описанном выше.

Примечание.

Не смотря на то, что на этой неделе вы изучали асинхронность, клиент должен быть синхронным. Не расстраивайтесь, если вы хотели попробовать свои силы в написании асинхронного кода, на следующей неделе вам представится такая возможность.