Knihovna SpeechCloud.dialog

Jan Švec <honzas@ntis.zcu.cz>

Verze:

- 2019-04-26
  počáteční dokumentace SpeechCloud.dialog API a SpeechCloud API
- 2019-05-03
  dokumentace API pro obousměrnou komunikaci s klienty
  příklad demonstrující nízkoúrovňové SpeechCloud API
  servování statických souborů z dialogového manažeru
- 2019-05-14
  dokumentace API pro přístup k SLU výsledkům
- 2019-06-07
  dokumentace zahájení session a SIP rozhraní
- 2019-07-11
  přidání parametru weight u definice gramatik
- 2019-07-15
  parametr text u synthesize může být seznam
- 2019-08-06
  metoda dialogového manažeru end_session()
- 2019-08-10
  metody synthesize_and_wait_for_*_result()
- 2020-02-12
  metody dialogového manažeru start_session() a end_session() jsou asynchronní

SpeechCloud.dialog

Před instalací

Pro zprovoznění SpeechCloud.dialog dialogového manažeru budete potřebovat zřízenou aplikaci v platformě SpeechCloud. Tuto aplikaci je možné vytvořit pouze úpravou nastavení platformy a tudíž pro vytvoření SpeechCloud aplikace kontaktujte autora tohoto dokumentu.

SpeechCloud aplikace je identifikována svým názvem, tento název může mít lidsky čitelný tvar, ale i tvar v podobě např. UUID apod. Předpokládejme aplikace s názvem `numbers`, pak jsou dostupné následující URL:

- Testovací SpeechCloud stránka na URL:
  https://cak.zcu.cz:9444/index.html?numbers

- Dokumentace nízkoúrovňového API včetně seznamu metod a událostí na URL: https://cak.zcu.cz:9443/v1/schema/wss/numbers/?format=docson
- Konfigurační SpeechCloud URL (není nutné pro SpeechCloud.dialog): https://cak.zcu.cz:9443/v1/speechcloud/numbers

Součástí konfigurace SpeechCloud aplikace je i případné WebSocket URL dialogového manažera, na které se má platforma SpeechCloud připojit při zahájení dialogu. Pro zřízení SpeechCloud aplikace je tedy nutné vědět IP adresu a TCP port, na kterém je toto WebSocket URL dostupné.

Instalace

Instalace v cloudovém prostředí stratus.zcu.cz

Instalace DEB balíčků Python 3.7

```
sudo apt-get install -t buster python3.7 python3-virtualenv
```

Vytvoření Python 3.7 virtualenv a jeho instalace

```
mkdir SpeechCloud
cd SpeechCloud
python3 -m virtualenv -p python3 site
. site/bin/activate
pip install tornado PyEventEmitter
```

Použití knihovny

Základní kostra aplikace

Mějme základní kostru aplikace uloženou v souboru example_base.py:

```
from dialog import SpeechCloudWS, Dialog
import logging

class ExampleDialog(Dialog):
    async def main(self):
        pass

if __name__ == '__main__':
    logging.basicConfig(format='%(asctime)s %(levelname)-10s %(message)s',level=logging.DEBUG)

    SpeechCloudWS.run(ExampleDialog, address="0.0.0.0", port=8888)
```

Kostra aplikace spustí dialogový manažer definovaný třídou ExampleDialog jako SpeechCloud WebSocket server. Pro spuštění dialogového manažeru je nutné aktivovat virtuální prostředí Pythonu a následně spustit hlavní soubor:

```
. site/bin/activate
python example_base.py
```

Tento server poslouchá na všech IP adresách daného stroje a na portu 8888. Je-li IP veřejná IP adresa stroje např. 147.228.121.10, pak WebSocket URL, na kterém je dostupný dialogový manažer je

```
http://147.228.121.10:8888/ws
```

Správnou funkcionalitu je možné ověřit příkazem curl, kde se pokusíme přistoupit na tuto URL pomocí HTTP GET:

```
curl http://147.228.121.10:8888/ws
```

Jako výstup příkazu curl by se mělo objevit:

```
Can "Upgrade" only to "WebSocket".
```

A dialogový manažer by měl do konzole vypsat podobnou zprávu:

```
2019-04-26 12:14:14,082 WARNING    400 GET /ws (147.228.47.63) 0.78ms
2019-04-26 12:14:14,082 DEBUG      Can "Upgrade" only to "WebSocket".
```

Toto chování je správně, neboť dialogový manažer na výše zmíněné adrese podporuje pouze WebSocket protokol, na který je přepnuto po počátečním HTTP požadavku.

Po připojení na testovací SpeechCloud stránku s propojenou SpeechCloud aplikací (zde pro ukázku https://cak.zcu.cz:9444/index.html?F54AC144-8AE0-4315-98B0-B47C4CA1B580) je možné očekávat následující výpis:

[SIP] - closed
ASR ready
[LIB] - Events: sc_error, slu_entities, asr_set_grammar_ok, asr_input_processed, asr_signal, asr_recognizing, tts_started, asr_paused, asr_offline_started, sc_activated, asr_offline_finished, asr_offline_error, asr_first_speech, asr_full_buffers, asr_set_grammar_error, dm_display, slu_set_grammars_done, asr_initialized, asr_audio_record, asr_result, tts_done, asr_ready
[LIB] - Methods: asr_offline_push_data, asr_process_text, asr_switch_engine, asr_offline_stop, asr_recognize_data, asr_set_grammar_uri, asr_offline_start, asr_pause, asr_set_grammar, tts_synthesize, asr_recognize, slu_set_grammars, sc_start_session, tts_stop
SpeechCloud model URI https://cak.zcu.cz:9443/v1/speechcloud/F54AC144-8AE0-4315-98B0-B47C4CA1B580
JSON schema URI https://cak.zcu.cz:9443/v1/schema/wss/F54AC144-8AE0-4315-98B0-B47C4CA1B580/0
Session started fBTgQ8VqXUAEGXtktPBt4i
[WS] - session started id=fBTgQ8VqXUAEGXtktPBt4i
[WS] - connected
[SIP] - registered
[SIP] - client id=5868d5f4-b96e-436f-a878-3b9d9782cc17

A v konzoli dialogového manažeru se objeví výpis:

```
2019-04-26 12:21:01,855 INFO       101 GET /ws (147.228.125.15) 0.59ms
2019-04-26 12:21:01,855 INFO       Creating dialog manager as instance of <class
'__main__.ExampleDialog'>
2019-04-26 12:21:05,287 INFO       Canceling dialog manager task (WebSocket closed)
```

Vidíme, že se SpeechCloud připojil na dialogový manažer a že dialog skončil (zpráva [SIP] - closed) sotva byl iniciován (zpráva ASR ready). To je díky tomu, že třída dialogového manažeru nemá definované chování (metodu main).

Dialogový manažer

Dialogový manažer implementovaný pomocí knihovny SpeechCloud.dialog je potomek třídy Dialog a je implementován pomocí asynchronního programování. Asynchronní programování je ideální prostředek v okamžiku, kdy váš program komunikuje prostřednictvím sítě a často čeká na různé události. To že programový kód používá asynchronní programování je přítomnost klíčových slov `async` a `await`. Tato klíčová slova v Pythonu umožňují před řízení centrální programové smyčce v okamžiku, kdy váš program čeká na nějakou událost. Více informací naleznete například v tutoriálu https://naucse.python.cz/lessons/intro/async/ .

Protože dialogový manažer čeká neustále na mnoho událostí (vypršení časovače, výsledek rozpoznávání, konec syntézy) a zároveň dialogový manažer běží distribuovaně (mimo platformu SpeechCloud), tak je použití asynchronního programování vhodná cesta, jak komunikaci organizovat.

Dialogový manažer je oděděn od třídy `Dialog` a může využívat dvě různá rozhraní a zároveň je může i kombinovat:

- nízkoúrovňové SpeechCloud API pomocí objektu `self.sc`, který má každý potomek třídy `Dialog`.
- vysokoúrovňového SpeechCloud.dialog API, které obaluje nízkoúrovňové SpeechCloud API a nabízí pohodlné funkce obalující typicky volání několika SpeechCloud API metod a událostí.

SpeechCloud.dialog API

Syntéza řeči

```
await self.synthesize(text, **kwargs)
await self.synthesize_and_wait(text, **kwargs)
```

Metody pro syntézu řeči z textu, povinný parametr `text` obsahuje text, který se má sesyntetizovat. Při volání je nutné použít klíčové slovo `await`, neboť se komunikuje po síti s platformou SpeechCloud. Varianta `synthesize_and_wait` čeká na skončení syntézy řeči. Je možné předat další parametry:

- `voice` - hlas, kterým se text přečte (např. “Iva210”, “Jiri210” apod.)
- `gain` - zesílení, jaké se aplikuje na syntetizovaný prompt.

Pro urychlení syntézy řeči je možné syntetizovaný prompt rozdělit na více částí odpovídajících jednotlivým větám a ty poté předat jako seznam v parametru `text`.

Rozpoznávání řeči

```
result = await self.recognize_and_wait_for_asr_result(timeout=5.)
```

Metoda aktivuje rozpoznávání řeči a počká na výsledek (klíčové slovo **await**). Metodě je možné předat hodnotu `timeout` v sekundách pro časovač, po jehož uplynutí se vrátí výsledek **None**, pokud uživatel do této doby nezačal komunikovat. Hodnota **timeout = None** znamená čekání na výsledek rozpoznávání bez časového omezení.

Vrácená struktura obsahuje následující prvky:

- `word_1best` - rozpoznaná posloupnost jako řetězec, slova jsou oddělená mezerami
- `word_array` - rozpoznaná slova jako pole řetězců
- `word_conf` - pole float čísel o stejné délce jako `word_array`, pro každé slovo obsahuje confidence skóre (číslo 0 až 1 vyjadřující, jak moc rozpoznávač věří v danou hypotézu).
- `word_times` - pole o stejné délce jako `word_array`, obsahuje vnořená pole o délce 2 určující začátek a konec odpovídajícího slova (v sekundách od začátku promluvy).

Porozumění řeči

Pro využití porozumění řeči je nutné nejprve nadefinovat sémantické entity, které budou pro porozumění použity, k tomu slouží metoda:

```
await self.define_slu_grammars(grm)
```

přičemž `grm` je seznam specifikací jednotlivých gramatik. Specifikaci gramatik je možné napsat ručně ve formátu, v jakém jej očekává SpeechCloud metoda slu_set_grammars(), nebo je možné použít SpeechCloud.dialog funkci, která slouží pro pohodlné vygenerování odpovídající gramatiky:

```
grm = self.grammar_from_dict(entity, grm_dict, weight=1.0)
```

Tato funkce slouží pro vytvoření sémantické entity s názvem `entity`, která je popsána pomocí terminálů v `grm_dict`. Název `entity` je řetězec, který musí splňovat podmínky na to být identifikátorem (malá a velká písmena, číslice, podtržítka). Slovník grm_dict má jako klíče Python objekty a jako hodnoty řetězce nebo množiny, popř. seznamy řetězců. Hodnoty slovníku určují slovní realizaci toho kterého klíče, jinými slovy definují terminální symboly gramatiky použité pro porozumění. Parametr `weight` umožňuje specifikovat multiplikativní váhu (prioritu) dané sémantické entity vůči ostatním. Vyšší váha znamená, že tato sémantická entita bude mít při parsování přednost před ostatními entitatmi.

Tato metoda běží lokálně v dialogovém manažeru, proto se nepoužívá klíčové slovo await. Například:

```
grm = self.grammar_from_dict("numbers", {1: "jedna",
                                         2: {"dva", "dvě"},
```

```
                                        3: "tři",
                                        4: "čtyři",
                                        5: "pět",
                                        6: "šest",
                                        7: "sedm",
                                        8: "osm",
                                        9: "devět"})
```

Pro určení slovní realizace je možné kombinovat jak samotné řetězce, tak jejich množiny/seznamy (viz příklad výše, číslo 2). Výsledná sémantická entita `numbers` je detekována po zavolání `await self.define_slu_grammars(grm)`. Návratová hodnota metody `grammar_from_dict` je typu seznam, je proto možné pomocí operátoru + spojit více definic sémantických entit za sebe.
Po aktivaci gramatik porozumění řeči metodou `define_slu_grammars` je možné použít metodu

```
result = await self.recognize_and_wait_for_slu_result(timeout=5.)
```

Tato metoda má obdobné chování jako `recognize_and_wait_for_asr_result`, ale čeká na výsledek z porozumění řeči. Význam parametru `timeout` je stejný.

Reprezentace výsledku porozumění řeči

Výsledek porozumění řeči je reprezentován instancí třídy SLU`Result`. Tato třída je vrácena z metody `recognize_and_wait_for_slu_result` a podporuje testování na logickou pravdu/nepravdu, čímž je možné detekovat, zda je výsledek prázdný (tj. vypršel `timeout`). Příklad:

```
result = await self.recognize_and_wait_for_slu_result(timeout=5.)
if result:
    print(“Recognized entities: “, result.entities)
else:
    print(“no-input timeout”)
```

Instance třídy SLUResult obsahuje následující atributy:

- `result.asr_result` - odpovídající výsledek rozpoznávání řeči, pro který bylo provedeno porozumění řeči
- `result.entity_1best` - první nejlepší posloupnost sémantických entit (instance třídy `EntityMap`)
- `result.entity_types` - seznam typů sémantických entit
- `result.entities` - slovník sémantických entit dělený podle typů sémantických entit (klíče jsou typy sémantických entit jako řetězce, hodnoty pak instance třídy `EntityMap`)
- pro každý typ sémantické entity zkrácený alias, například pro typ entity `numbers` jsou následující objekty ekvivalentní:
    - `result.entities[“numbers”]`

    - `result.numbers`

Třída `EntityMap` slouží ke sdružení detailů pro jednotlivé sémantické entity. Tato třída má rozhraní read-only dictionary, klíče jsou sémantické entity reprezentované jako Python object, hodnoty pak seznam instancí třídy `Entity`. Jednotlivé operace, atributy a metody podporované třídou `EntityMap` jsou:

- `entity_map[value]` - seznam detailů entity `value` (např. `entity_map[1]`)
- `value in entity_map` - bool hodnota indikující zda byla rozpoznána entita `value`
- `entity_map.keys()` - iterátor všech rozpoznaných entit
- `entity_map.values()` - iterátor všech detailů rozpoznaných entit
- `entity_map.first` - první rozpoznaná entita
- `entity_map.last` - poslední rozpoznaná entita
- `entity_map.all` - seznam všech rozpoznaných entit

Pro každou sémantickou entitu je možné získat její detaily jako instanci třídy `Entity`, ta má atributy:

- `value` - hodnota sémantické entity (Python object)
- `entity_type` - typ sémantické entity (string)
- `conf` - skóre důvěry v tuto sémantickou entitu (float)
- `begin` - index počátku sémantické entity
- `end` - index konce sémantické entity

Příklad

Předpokládejme, že uživatel pronesl slova “dva pět pět sedm” v dialogu, který měl definovány následující sémantické entity:

```
grm = self.grammar_from_dict("numbers", {1: "jedna",
                                         2: {"dva", "dvě"},
                                         3: "tři",
                                         4: "čtyři",
                                         5: "pět",
                                         6: "šest",
                                         7: "sedm",
                                         8: "osm",
                                         9: "devět"})
await self.define_slu_grammars(grm)
result = await self.recognize_and_wait_for_slu_result(timeout=5.)
```

- Test, zda uživatel pronesl alespoň jednu číslici:
  `if result.entities[“numbers”]: ....`

  nebo zkráceně:
  `if result.numbers: …`

- Test, zda uživatel řekl číslici 8:
  `if 8 in result.entities[“numbers”]: ...`

nebo zkráceně:
```
if 8 in result.numbers: …
```

- Test, zda uživatel řekl alespoň 4 číslice:
  ```
  if len(result.entities[“numbers”]) >= 4: …
  ```

  nebo zkráceně:
  ```
  if len(result.numbers) >= 4: …
  ```

  Lze rovněž otestovat, pomocí atributu `entity_types`:
  ```
  if result.entity_types.count(“numbers”) >= 4: …
  ```

- Seznam všech pronesených číslic:
  ```
  result.entities[“numbers”].all   # [2, 5, 5, 7]
  ```

  nebo zkráceně:
  ```
  result.numbers.all   # [2, 5, 5, 7]
  ```

- První pronesená číslice:
  ```
  result.entities[“numbers”].first   # 2
  ```

  nebo zkráceně:
  ```
  result.numbers.first   # 2
  ```

- Poslední pronesená číslice:
  ```
  result.entities[“numbers”].last   # 7
  ```

  nebo zkráceně:
  ```
  result.numbers.last   # 7
  ```

Bylo-li pro porozumění použito více typů sémantických entit, je možné zjistit jejich pořadí v rámci první nejlepší hypotézy:

- Hodnoty všech sémantických entit:
  ```
  result.entity_1best.all    # [2, 5, 5, 7]
  ```

- První a poslední hodnota sémantické entity:
  ```
  result.entity_1best.first
  result.entity_1best.last
  ```

- Detaily všech entit s hodnotou 5:
  ```
  result.entities[“numbers”][5]
  ```

  nebo zkráceně:
  ```
  result.numbers[5]
  ```

- Detaily všech entit v 1-best hypotéze:
  `result.entity_1best.values()`

- Míry důvěry všech entit v 1-best hypotéze:
  `[e.conf for e in result.entity_1best.values()]`

Zastaralý přístup k rozpoznaným entitám

Z historických důvodů lze k první, poslední a všem entitám přistupovat i pomocí následujících atributů:

- `result.first.numbers` - je první rozpoznaná entity typu `numbers`
- `result.last.numbers` - je poslední rozpoznaná entity typu `numbers`
- `result.all.numbers` - jsou všechny rozpoznané entity typu `numbers`

Tento způsob již není podporován a bude odstraněn v budoucích verzích knihovny SpeechCloud.dialog.

Syntéza řeči následovaná rozpoznáváním

Knihovna SpeechCloud.dialog podporuje zřetězení syntetizovaného promptu a rozpoznávání řeči pomocí dvojice metod:

```
await self.synthesize_and_wait_for_asr_result(text,
                                               timeout=None,
                                               **kwargs)
await self.synthesize_and_wait_for_slu_result(text,
                                               timeout=None,
                                               **kwargs)
```

Tyto metody sesyntetizují `text` (popř. texty, pokud `text` je seznam) a následně aktivují rozpoznávání řeči. Dále se čeká na výsledek z ASR nebo SLU maximálně timeout sekund (pokud je `timeout` None, pak se čeká bez omezení). Význam dalších předaných parametrů, viz dokumentace metody `synthesize()`.

Volání:

```
result = await self.synthesize_and_wait_for_asr_result(
                            text="Prosím, mluvte...", timeout=5.)
```

je ekvivalentní dvojici volání:

```
await self.synthesize_and_wait(text="Prosím, mluvte...")
result = await self.recognize_and_wait_for_asr_result(timeout=5.)
```

Strukturování dialogu

Pro správné strukturování programového kódu je možné použít volání dílčích částí dialogů naprogramovaných v dílčích asynchronních metodách, kód pak vypadá následovně:

```
class ExampleDialog(Dialog):
    async def main(self):
        await self.welcome()
        await self.question1()
        await self.question2()

    async def welcome(self):
        await self.synthesize_and_wait("Dobrý den.")

    async def question1(self):
        await self.synthesize_and_wait("Bylo vám již 18 let?")
        await self.recognize_and_wait_for_asr_result(timeout=5)

    async def question2(self):
        pass
```

Zahájení session a dialogu

Dialog je zahájen připojením všech stran (klient, SpeechCloud worker, dialogový manažer) do jedné session. Toto propojení je signalizováno událostí `sc_start_session`. Její standardní obsluha ve třídě `Dialog` nastaví tyto atributy:

- `schema_uri` - URI JSON schematu, podle kterého probíhá aktuální session
- `session_id` - jedinečný identifikátor aktuální session
- `session_uri` - URI se záznamem událostí vyměněných v rámci session

Bezprostředně po obdržení události `sc_start_session` jsou ve třídě `Dialog` nastaveny výše zmíněné atributy a následně je zavolána metoda `start_session` (pozor, tato metoda není asynchronní). Tato metoda je vhodným místem pro navázání handlerů událostí pomocí volání `self.sc.on("event_name", handler)`.

Schéma aktuální session

Veškerá komunikace se SpeechCloudem po WebSocketu je validována oproti JSON Schematu. Toto schema je verzováno a je dostupné na URI, které je zasláno komunikujícím stranám při inicializaci session. Schema je zároveň možné vyrenderovat pomocí DOCSON tak, aby poskytlo human-readable dokumentaci komunikace.

Příklady:

- https://cak.zcu.cz:9443/v1/schema/wss/numbers/10 - JSON Schema
- https://cak.zcu.cz:9443/v1/schema/wss/numbers/10?format=docson - DOCSON dokumentace

Poznámka:
Standardní implementace třídy Dialog vypíše URI schématu komunikace pomocí debug zprávy do svého loggeru.

Záznam události v rámci session

Komunikace se SpeechCloudem je ukládána do databáze pod jedinečným ID dané session. Tato komunikace je přístupná již po inicializaci session. URI je dostupné v různých formátech specifikovaných HTTP parametrem format:

- `format=json`, `format=yaml` - formáty JSON, resp. YAML
- `format=json.html`, `format=yaml.html` - formáty JSON, resp. YAML renderované do HTML se zvýrazněním syntaxe a proklikávatelnými odkazy

Příklady:

- https://cak.zcu.cz:9443/v1/session/4WciTfJ26DZL7mXenRm3JJ?format=yaml.html
- https://cak.zcu.cz:9443/v1/session/4WciTfJ26DZL7mXenRm3JJ?format=json
- https://cak.zcu.cz:9443/v1/session/4WciTfJ26DZL7mXenRm3JJ

Poznámka:
Standardní implementace třídy Dialog vypíše ID session a URI záznamu komunikace pomocí debug zprávy do svého loggeru.

Ukončení dialogu

Dialog je ukončen skončením běhu metody `main`, popř. výjimkou, která v kódu nebyla odchycena.

Po ukončení dialogu, ať již ze strany dialogového manažeru, tak ze strany uživatele, je uzavřen řídící WebSocket a zavolána metoda **asynchronní** `end_session`. Tato metoda již nemůže komunikovat se SpeechCloudem (řídící WebSocket je již uzavřen) a všechny výjimky vzniklé v této metodě jsou pouze zalogovány a dále ignorovány.

Čekání v dialogu

Pro čekání určitou definovanou dobu je možné použít funkci z modulu `asyncio` opět prefixovanou klíčovým slovem await:

```
await asyncio.sleep(0.3)
```

Obousměrná komunikace s HTML klientem

Pro multi-modální dialogy, popř. pro různé GUI aplikace řízené dialogovým manažerem je potřeba API pro obousměrnou komunikaci. SpeechCloud.dialog nabízí trojici metod pro zasílání libovolný dat serializovatelných do JSON formátu:

```
await self.send_message(message)
```

slouží pro zaslání libovolné datové zprávy `message` z dialogového manažera do klientské aplikace. Pro odchycení v klientské HTML aplikace lze použít (předpokládáme, že `speechCloud` je objekt reprezentující inicializované spojení na SpeechCloud):

```
speechCloud.on('dm_receive_message', function (msg) {
    console.log(msg.data);
});
```

Pro příjem zpráv asynchronně, kdykoli v průběhu dialogu, je možné přepsat metodu třídy Dialog:

```
def on_receive_message(self, data):
    print(data)
```

Tento přístup je vhodný pro asynchronní nastavování různých parametrů a stavu dialogového manažeru kdykoli v průběhu dialogu.

Pro synchronní příjem zpráv v rámci dialogu pak slouží metoda `pop_message()`:

```
recv = await self.pop_message(timeout=5)
```

Metodě lze předat parametr `timeout`, který nastaví, jak dlouho se bude čekat na přijetí zprávy. Hodnota None značí neomezené čekání. V případě, že vyprší `timeout`, metoda vrátí hodnotu None. Tato metoda je vhodná pro získání zpětné vazby od uživatele v průběhu dialogu, například po vyžádání kliknutí na tlačítko.

Pro odeslání z HTML klienta se v JavaScriptu použije:

```
speechCloud.dm_send_message({...});
```

SpeechCloud API

Události a metody

SpeechCloud interně rozlišuje mezi událostmi (které jsou generované platformou) a metodami (které jsou naopak posílány platformě). Metody mohou být používány přímo ve SpeechCloud.dialog pomocí atributu `self.sc`, například implementace vysokoúrovňové metody `synthesize_and_wait` by vypadala následovně:

```
async def synthesize_and_wait(self, text):
    await self.sc.tts_synthesize(text=text)
    await self.sc.tts_done()
```

Důležité je, že veškeré parametry metodám jsou předávány jako keyword arguments (pojmenované argumenty), tj. je nutné je prefixovat názvem parametru.

Pro navázání handleru na libovolnou SpeechCloud událost se používá opět objekt `self.sc`, který poskytuje PyEventEmitter API (https://pypi.org/project/PyEventEmitter/).)

Příklad

Příklad, kde je použito volání jak SpeechCloud API metody, tak navázání handleru na událost (uložen v souboru example_low_level.py):

```
class ExampleDialog(Dialog):
    def on_asr_signal(self, level, speech):
        print(level, speech)

    async def main(self):
        self.sc.on("asr_signal", self.on_asr_signal)
        await self.sc.asr_recognize()
        await asyncio.sleep(10)
        await self.sc.asr_pause()
```

SIP rozhraní

Je-li telefonní hovor směrovaný ze SIP účtu pomocí rozhraní SpeechCloud.SIP, pak lze použít další funkcionalitu, která poskytuje údaje o SIP spojení a umožňuje manipulaci se SIPovým hovorem.

Získání údajů o SIP spojení

Po inicializaci SpeechCloud session pomocí rozhraní SpeechCloud.SIP odešle toto rozhraní událost `sip_call_info`, která má v parametrech uvedeny následující údaje:

- `call_id_string` - identifikace SIPového hovoru
- `remote_contact`, `remote_uri` - identifikace protistrany SIPového hovoru
- `local_contact`, `local_uri` - identifikace SpeechCloud.SIP rozhraní v SIPovém hovoru

Příklad
Pro získání informací o hovoru pomocí SpeechCloud.SIP je nutné v potomkovi třídy `Dialog` předefinovat metodu **asynchronní** `start_session` a v ní navázat handler pro událost `sip_call_info`, neboť tato událost je vyvolána bezprostředně po zahájení session. Příklad níže vypíše hodnotu parametru `remote_contact`, pokud bude zavolán pomocí SpeechCloud.SIP:

```
class ExampleDialog(Dialog):
    def print_sip_call_info(self, **kwargs):
        print(kwargs[“remote_contact”])

    async def start_session(self):
        self.sc.on("sip_call_info", self.print_sip_call_info)

    async def main(self):
        ...
```

Přepojení SIP hovoru

Příchozí SIPový hovor lze pomocí SpeechCloud.SIP přepojit (SIP attended transfer) na jiné SIP URI. Toho lze dosáhnout voláním metody `sc.sip_call_transfer` s parametrem `target_uri`, který obsahuje cílové URI, příklad:

```
class SIPTransfer(Dialog):
    async def main(self):
        await self.synthesize_and_wait(text=f"Přepojuji váš hovor.")
        await self.sc.sip_call_transfer(target_uri="sip:...")

        ...
```

Spuštění SpeechCloud.dialog aplikace

Dialogový manažer (v příkladu níže třídu `ExampleDialog`), který je potomkem třídy `Dialog` lze spustit pomocí jednoduchého wrapperu:

```
SpeechCloudWS.run(ExampleDialog,
                  address="0.0.0.0",
                  port=8888,
                  static_path="./static")
```

Wrapper sestaví instanci třídy `tornado.web.Application` a spustí ji tak, aby poslouchala na dané IP adrese (hodnota `"0.0.0.0"` znamená všechna rozhraní) a daném TCP portu (8888). Zároveň bude na URL http://adresa:8888/static poskytovat statický obsah z adresáře `./static`. Pokud není třeba poskytovat statický obsah, pak se parametr `static_path` neuvede.

HTTPS proxy pro SpeechCloud aplikace

Pro zpřístupnění hlasových služeb z webového prohlížeče musí být webová stránka načtená pomocí HTTPS (HTTP nad TLS). Pro usnadnění vývoje SpeechCloud aplikací je možné využít vestavěné HTTPS proxy platformy SpeechCloud, která pro každou SpeechCloud aplikaci (např. `edu-hds-all`) s nastaveným WebSocket dialogovým manažerem zpřístupňuje jím servované statické soubory na doméně SpeechCloudu. Předpokládejme, že dialogový manažer pro aplikaci `edu-hds-all` běží na IP adrese 147.228.121.10. Zároveň při startu SpeechCloud.dialog aplikace byl předán parametr `static_path` ukazující na statické soubory dialogového manažeru (např. `index.html`). Tento statický soubor je dostupný na URL:

```
http://147.228.121.10/static/index.html
```

Platforma SpeechCloud pro něj zároveň vytvoří HTTPS proxy na adrese:

```
https://cak.zcu.cz:9444/edu-hds-all//index.html
```

Důležité! Celý řetězec v URL cestě až po dvojitá lomítka je název SpeechCloud aplikace (zde `edu-hds-all`), za zdvojenými lomítky následuje cesta k souboru, která je předána dialogovému manažeru. Obecně GET požadavek na URL:

```
https://cak.zcu.cz:9444/<app_id>//<cesta>
```

vybere SpeechCloud aplikaci `app_id`, pro tu nalezne odpovídající IP adresu a port a udělá proxy na URL dialogového manažeru pro aplikaci `app_id`:

```
http://IP_adresa:port/static/<cesta>
```