メインエージェント開発

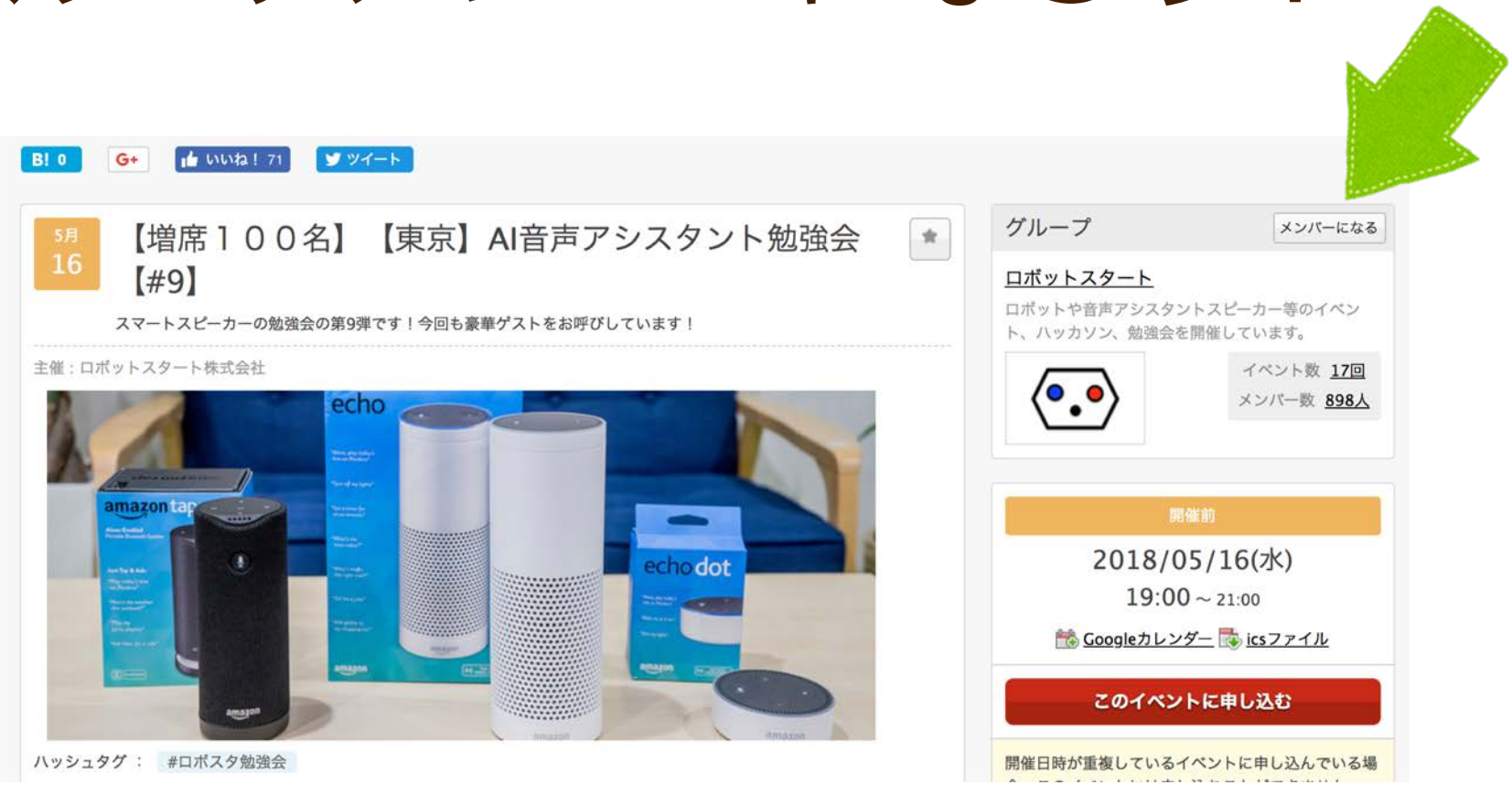
project SEBASTIEN

ロボットスタート
robot start inc.

ロボスタについて

「ロボスタ」で検索。

グループメンバーになろう！



右上の「メンバーになる」を押してください！

イベント情報が配信されます！

#ロボスタ勉強会

写真OKです。拡散してね！

ロボットスタート
robot start inc.

はじめに

ロボットエバンジェリスト
スマートスピーカーエバンジェリスト

にしだ　かんすけ
西田　寛輔

とのさまラボ代表。

ロボスタでは、ロボットエバンジェリスト／スマートスピーカーエバンジェリストとして活動中。

シブヤ経済新聞
Minkei Network

ロボットスタート
robot start inc.

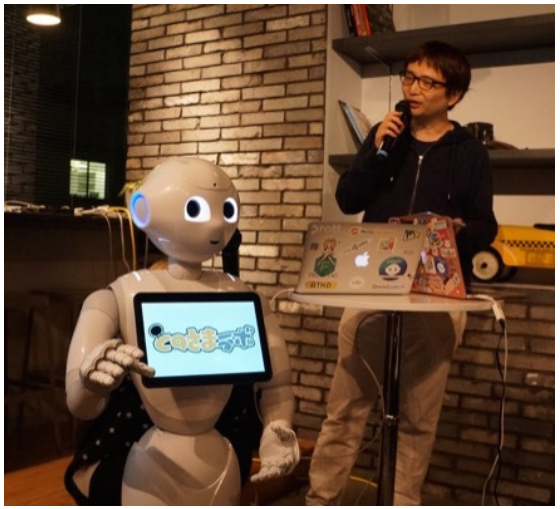
西田　寛輔
Kansuke Nishida

ロボットスタート株式会社
robot start inc.

ロボットエバンジェリスト
スマートスピーカーエバンジェリスト
Robot Evangelist
Smart Speaker Evangelist

Email : nishida@robotstart.co.jp
FB : http://fb.me/tonosamart
Tel / Fax : 03-6822-9601
〒153-0064
東京都目黒区下目黒 2-20-28 東信目黒ビル 7 階
URL : http://robotstart.co.jp/

エバンジェリスト！



- ロボット情報メディア「ロボスタ」ライター
- 各種勉強会講師、イベント登壇、ハッカソン審査員など
- ヒトとロボットの音楽ユニット「mirai capsule」
- スタジオALTA認定「CRAZY TOKYO」
- その他、ロボットアプリ作成、コンサルなど

メインエージェント

SEBASTIEN

メインエージェントって？

メイン
エージェント

- 端末やアプリなどの「メイン」となって応答を行うエージェント
- ドコモのシステム基盤を利用することにより、簡単に開発が可能
- あらかじめ登録してあるシナリオで動作する「シナリオ対話」形式
- エキスパートエージェントを呼び出すことが出来る

こんなキャラが作れます

SAMPLE

消火栓ねこ

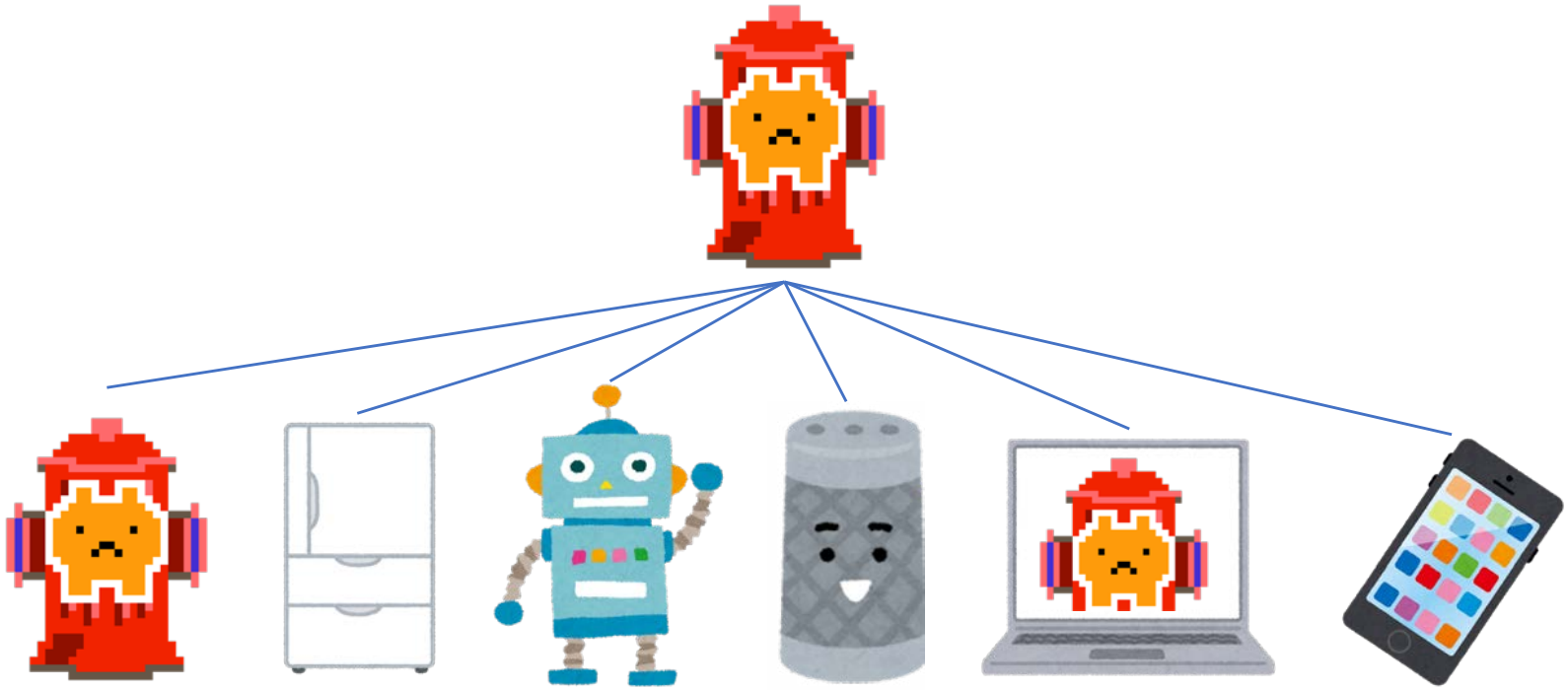
消火栓とねこが合体した
ゆるキャラ

口癖は「～なの。」

ずらすとヘルメットを隠して
いる時がある。

- 豊富な音声を使って、オリジナルのキャラクターを作成できるほか、声優さんの声でオリジナルボイス（音声合成）の作成が可能（有料）。
- 会話内容や口癖などをAIMLで自由に作成可能。
※個性を抑えたアシスタントキャラも作成可能です。

さまざまなハードに搭載できます

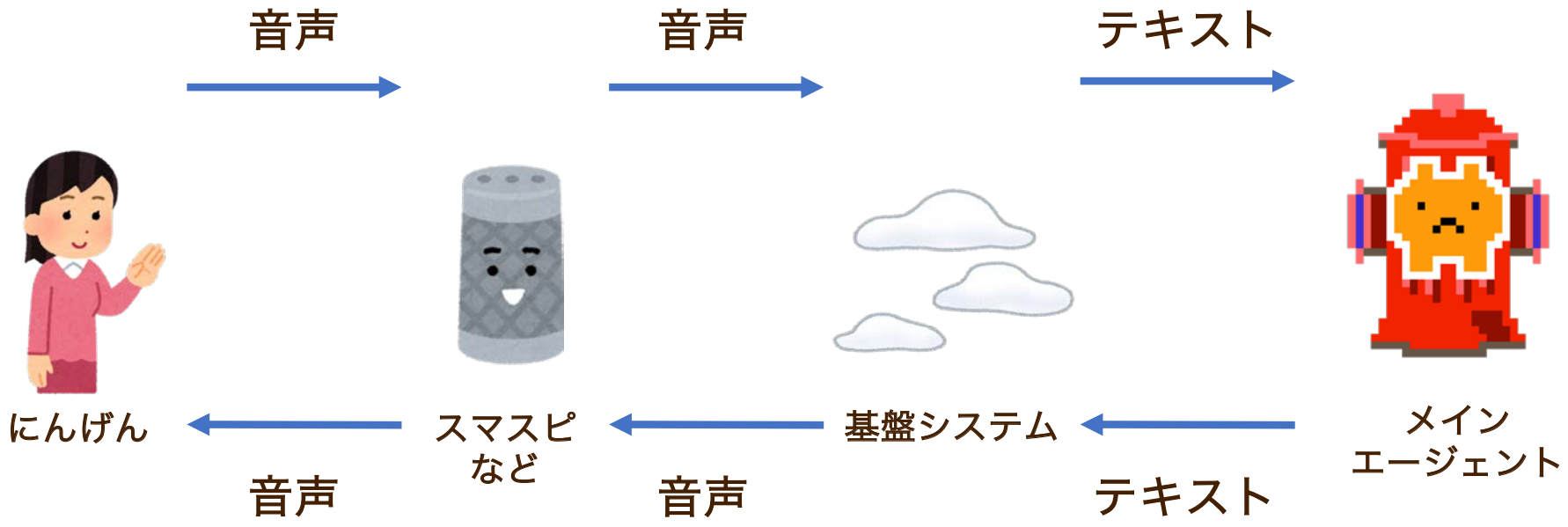


- ハードウェア（ロボット、スマートスピーカー、家電製品など）
- ウェブサービス（チャットボット、バーチャルユーチューバーなど）
- ソフトウェア（スマホアプリ、パソコンアプリなど）

※クラウドにシステムがあるため、さまざまな用途で利用可能です。

簡単な仕組み

音声合成を使用して返答する場合の例



- ユーザーが発話した音声を基盤システムがテキストに変換
- テキストに対する応答処理をAIMLで作成すればOK

AIMLとは？

会話”シナリオ"（ルール）を作ることができる、XMLを応用したマークアップ言語。

もともとは、カーネギーメロン大学のRichard WallaceやOSSコミュニティが、開発したもので、A.L.I.C.E. という 雑談会話ボット などに採用されています。

ドコモでは、AIMLの仕様を参考に、xAIML（Extended Expressive AIML）という独自の仕様を定義しています。

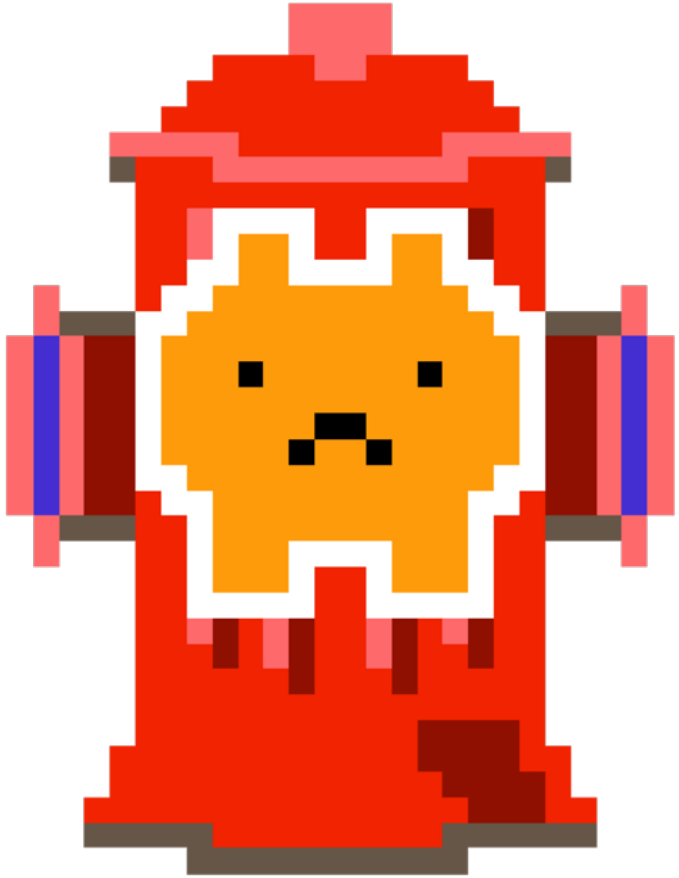
※この資料では表記をAIMLに統一しています。

シナリオ対話の特徴

- 流れを想定した”シナリオ“を作成しておくことで、ユーザーの会話を引き出す、システム主導の会話をすることが出来る。
- あらかじめ返答内容を確定できる。（想定外のことを話さない。）
- シナリオをあらかじめ用意しておく必要がある。そのため、想定外の臨機応変な会話はできない。

消火栓ねこ Ver.1.1

まずは作ってみよう



ファイルを準備しよう

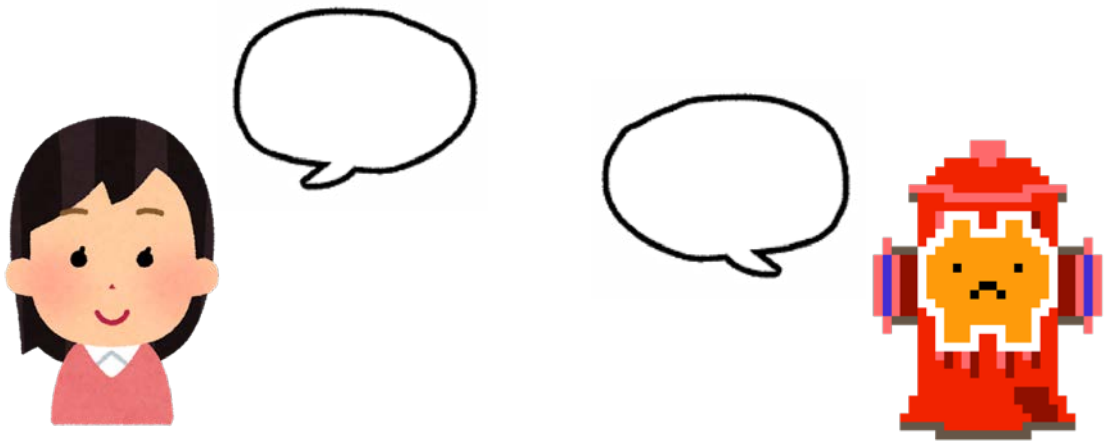
- AIMLは、拡張子が「.aiml」のXMLファイルです。
- UTF-8で保存する必要があります。
- 拡張子を除いたファイル名が、そのままシナリオのIDになります。命名には注意してください（日本語不可、違う名前のファイルをアップすると別のシナリオになるので注意）。

ファイル
https://robotstart.co.jp/event/20180612/
にある、「fireplug_cat_te.aiml」

このファイルを
fireplug_cat.aiml にリネームして、アップロードしてください。

一問一答

対話文脈に関係なくユーザがAと言ったらシステムがBと返す例です。



```
<category>
    <pattern>おはよう</pattern>
    <template>おはようなの！</template>
 </category>
```

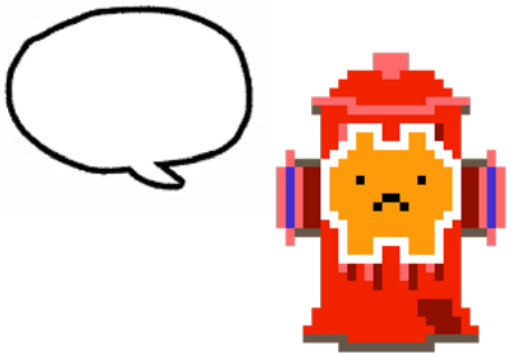
かいせつ

```
<category>
    <pattern>おはよう</pattern>
    <template>おはようなの！</template>
 </category>
```

<category>で囲んだところが１個の会話になります。
ユーザーの発話が<pattern>にマッチしたら、<template>の内容を話します。

これが、一番基本的なパターンです。

ちなみに、「おはよう」の完全一致ではなく
「ゆらぎ吸収」で、ある程度柔軟に判定されます。

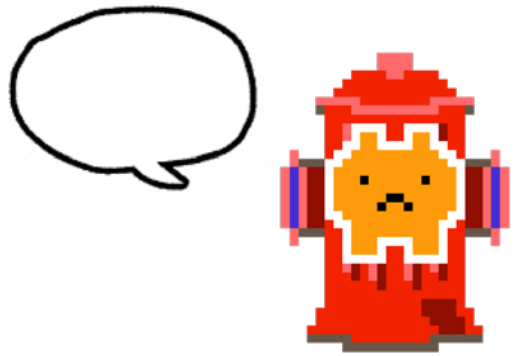


かいせつ

```
<category>
    <pattern>*</pattern>
    <template>よくわからなかったなの。</template>
  </category>
```

<pattern>に*と書いてあります。これは、ワイルドカードとよばれるもので「１形態素以上の任意文字列」を表します。

「おはよう」にマッチングしなかったのは、全部ここに行きます。



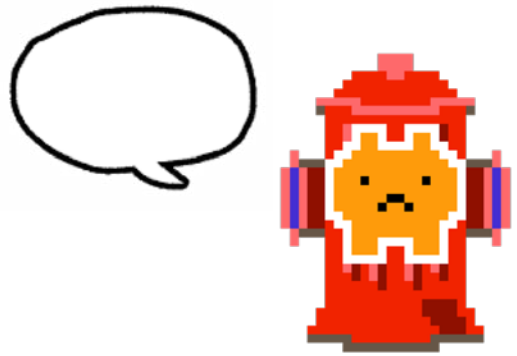
やってみよう

```
<!-- 一問一答 -->
  <category>
    <pattern>おはよう</pattern>
    <template>おはようなの！</template>
  </category>
  <category>
    <pattern>こんばんは</pattern>
    <template>こんばんはなの</template>
  </category>
```

「こんばんは」に対する応答を追加してみましょう！

ためしてみよう

実機で試してみましょう！



fireplug_cat_01.aiml

声を変えよう

- このままだと、「消火栓ねこ」のイメージとは違うので声を変えることにします。
- 音声は、エキスパートエージェントの編集画面などでテストしたり、名称を調べることが可能です。
- 今回は、「 aoi 」を使います。

fireplug_cat_02.aiml

やってみよう

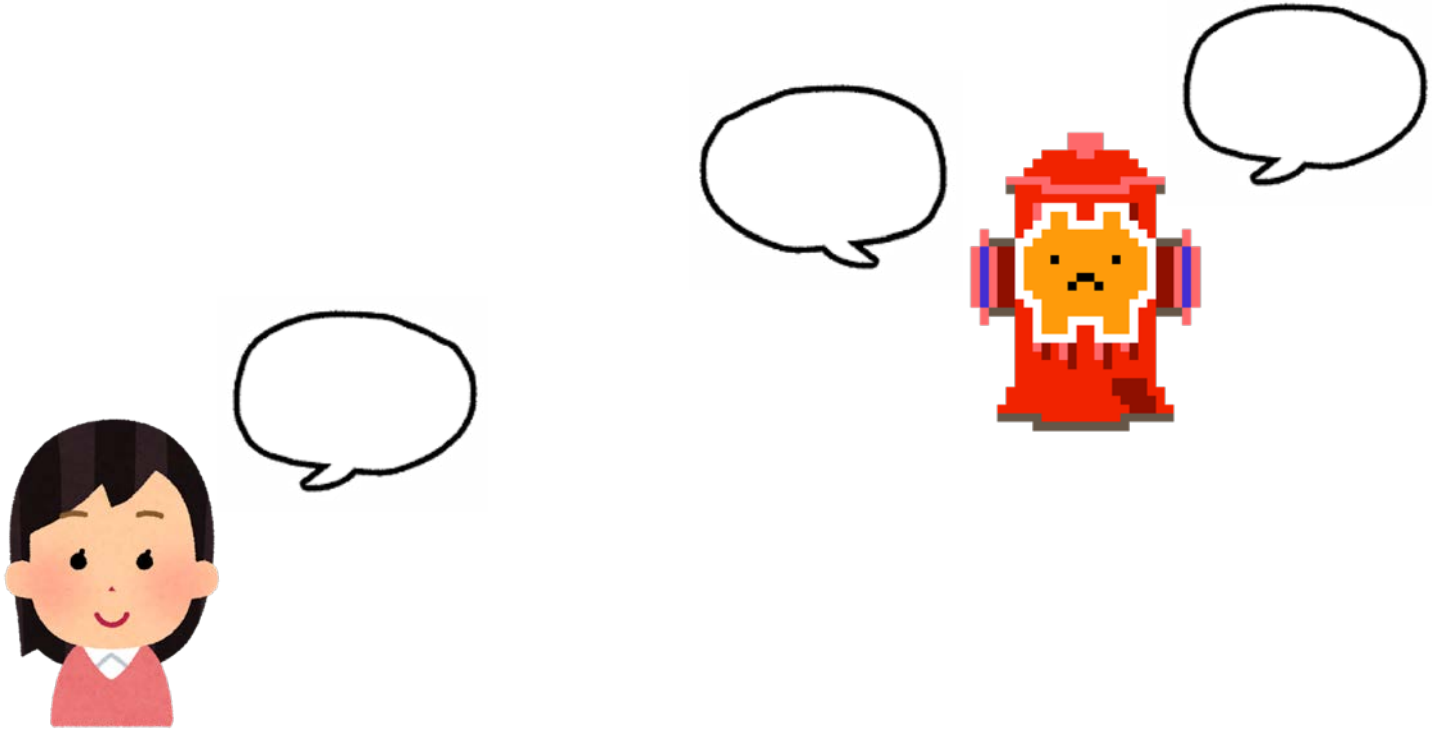
```
<aiml version="2.0.0" xmlns="http://www. ...
<!-- 音声の設定 -->
  <topic name="_">
    <category>
      <pattern>_</pattern>
      <template>
        <command>{"version":"1.0.1", "speaker":"aoi",
"option":{}}</command>
        <srai>
          <get name="input"/>
        </srai>
      </template>
    </category>
  </topic>
```

これはまあ、こういうものだと覚えてください。

fireplug_cat_02.aiml

ためしてみよう

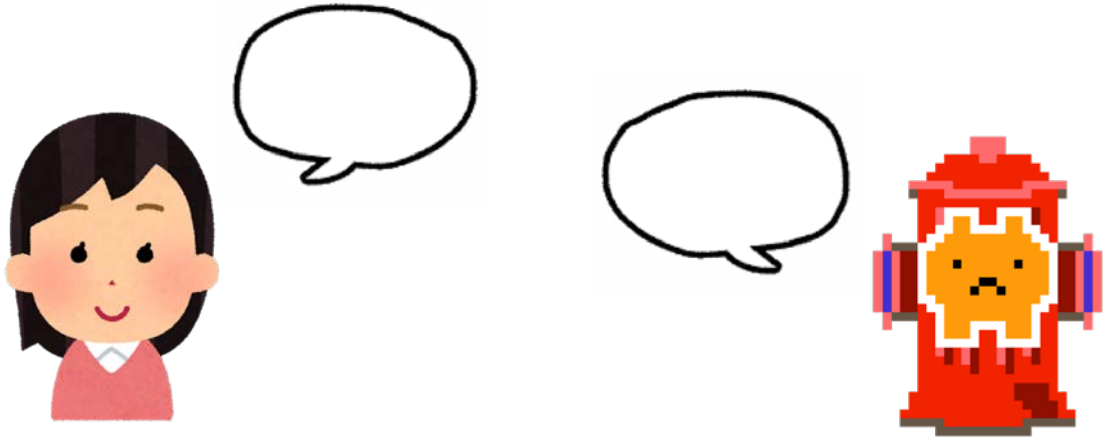
実機で試してみましょう



fireplug_cat_02.aiml

対話文脈

対話文脈を定義することがもできます。
同じ「嫌い」でも、その前の質問の内容に沿った返答が可能です。



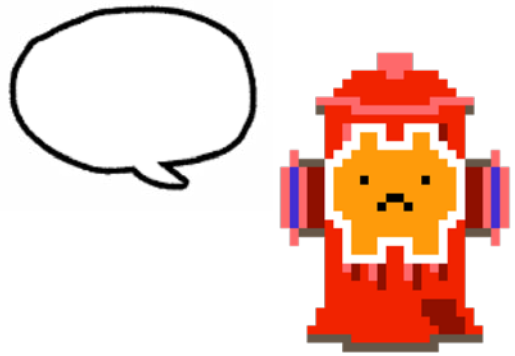
```
<category>
   <pattern>嫌い</pattern>
   <that>じゃあ質問するね、私のこと「好き」なの？「嫌い」なの？</that>
   <template>やっぱり、消火栓は嫌いなのかなの・・・。</template>
 </category>
```

fireplug_cat_03.aiml

かいせつ

```
<!-- 質問してみて -->
  <category>
    <pattern>質問してみて？</pattern>
    <template>じゃあ質問するね、私のこと「好き」なの？「嫌い」なの？
</template>
  </category>
```

まず、一問一答の会話を作成します。
次に、この会話に対して、どういう返答をするかを定義します。



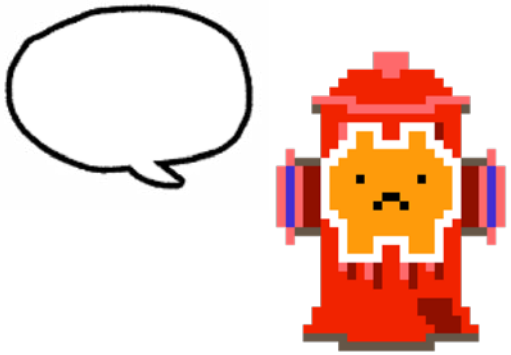
fireplug_cat_03.aiml

かいせつ

```
<category>
    <pattern>好き</pattern>
    <that>じゃあ質問するね、私のこと「好き」なの？「嫌い」なの？</that>
    <template>嬉しいなの。すごく嬉しいなの。</template>
  </category>
```

<that>で、直前の会話の内容を指定します。

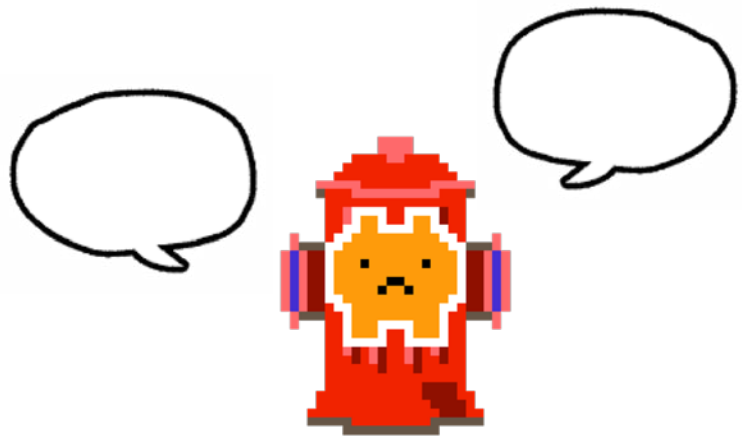
この例では、「じゃあ質問するね、私のこと「好き」なの？「嫌い」なの？ 」が直前の会話かつ、<pattern>が「好き」だった場合の返答を定義しています。



fireplug_cat_03.aiml

ためしてみよう

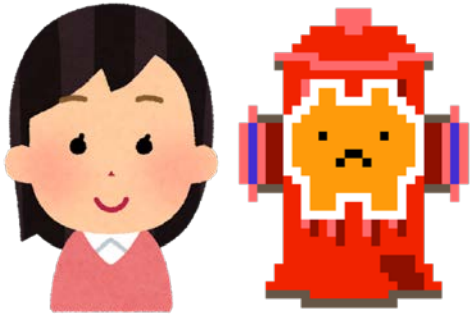
実機で試してみましょう



fireplug_cat_03.aiml

対話文脈（ID指定）

対話文脈をIDで指定することもできます。



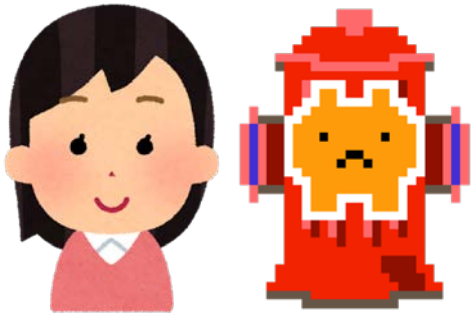
```
<!-- IDバージョン -->
  <category>
    <pattern>働いていますか？</pattern>
    <template id="work">私は、働いているなの！ あなたは、働いていますか？</template>
  </category>
  <category>
    <pattern>働いている</pattern>
    <pattern>働いてる</pattern>
    <pattern>はい</pattern>
    <that id="work" />
    <template>それはいいなの。働いたお金で食べる「さかな」は美味しいなの。</template>
  </category>
```

fireplug_cat_04.aiml

複数パターン

複数のパターンを指定できます。
同じ意味でも違う言い方がある場合に使います。

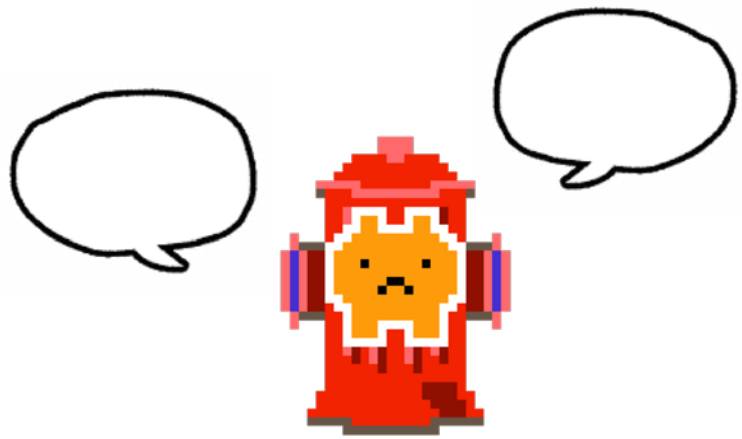
```
<category>
    <pattern>働いている</pattern>
    <pattern>働いてる</pattern>
    <pattern>はい</pattern>
    <that id="work" />
    <template>それはいいなの。働いたお金で食べる「さかな」は美味しいなの。
</template>
  </category>
```



fireplug_cat_04.aiml

ためしてみよう

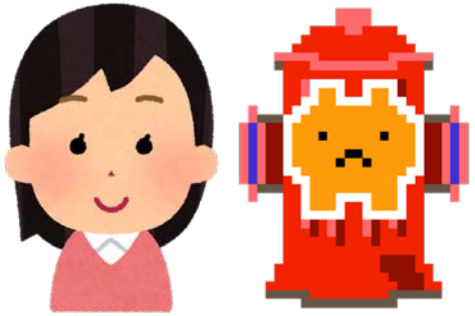
実機で試してみましょう



fireplug_cat_04.aiml

マッチした文字列を呼び出す

＊の内容を喋らせたりすることができます。



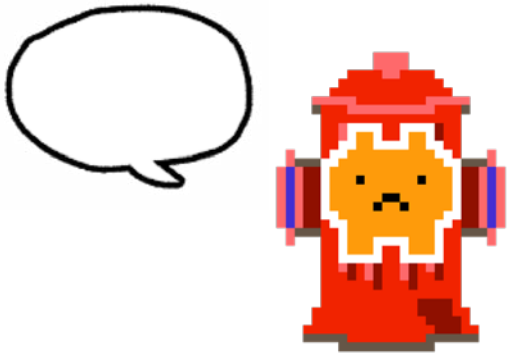
```
<!-- 好きな食べ物 -->
  <category>
    <pattern>好きな食べ物</pattern>
    <template id="like">私は、ちっちゃい魚が好きなの！ あなたは、なにが好きですか？</template>
  </category>
  <category>
    <pattern>*</pattern>
    <that id="like" />
    <template>あなたは、<star />が好きなのね。</template>
  </category>
```

fireplug_cat_05.aiml

かいせつ

```
<category>
    <pattern>*と*</pattern>
    <that id="vegetables" />
    <template>あなたは<star index="1" />と<star index="2" />が好きなのね。
</template>
  </category>
```

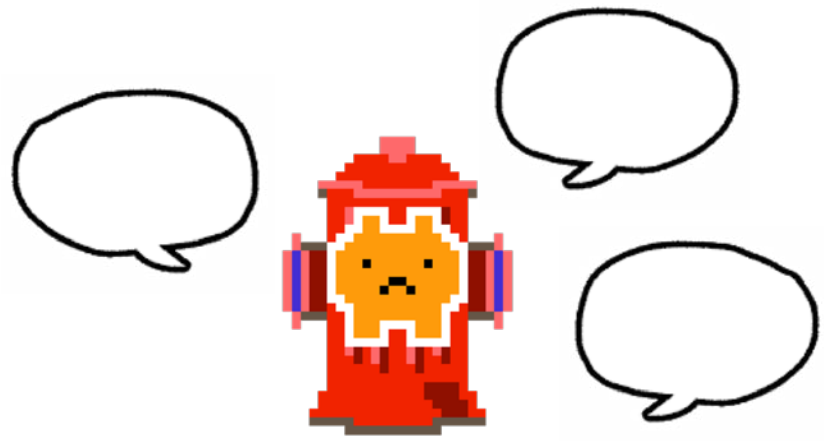
複数ある場合は、indexを指定して取得できます。



fireplug_cat_05.aiml

ためしてみよう

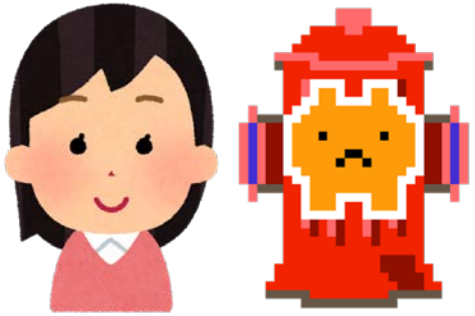
実機で試してみましょう



fireplug_cat_05.aiml

変数を保存する

set nameで、値を保存できます。



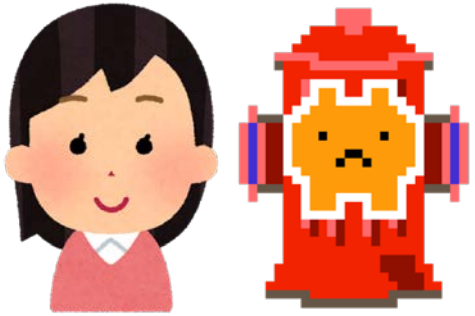
```
<category>
    <pattern>*</pattern>
    <that id="remember" />
    <template>
      <star />をおぼえるなの。<think>
      <set name="remember">
        <star />
      </set>
    </think>
  </template>
</category>
```

※rememberに「*」の内容を保存します。

fireplug_cat_06.aiml

変数を利用する

get nameで、値を利用できます。



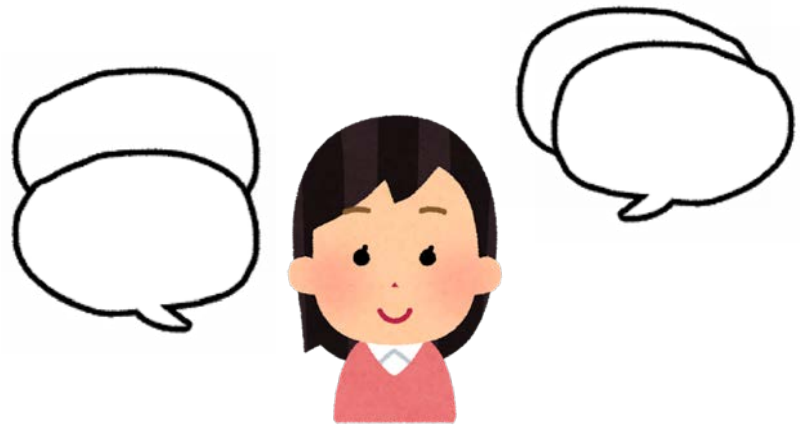
```
<category>
    <pattern>教えて</pattern>
    <template>
      <condition name="remember">
        <li value="undefined">おしえてもらってないなの。「覚えて」って言ってくれたら覚えるのなの。</li>
        <li>ちゃーんとおぼえてますなの。「<get name="remember" />」ですよね。</li>
      </condition>
    </template>
  </category>
```

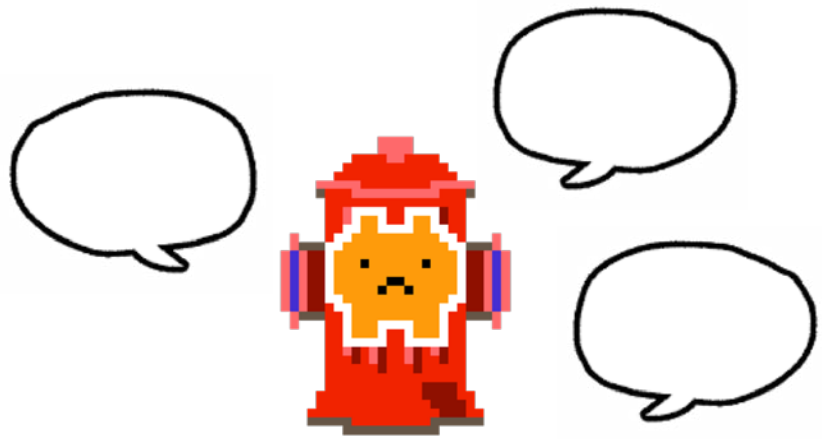
※rememberが「 undefined 」かどうかで処理を分けます。

fireplug_cat_06.aiml

ためしてみよう

実機で試してみましょう



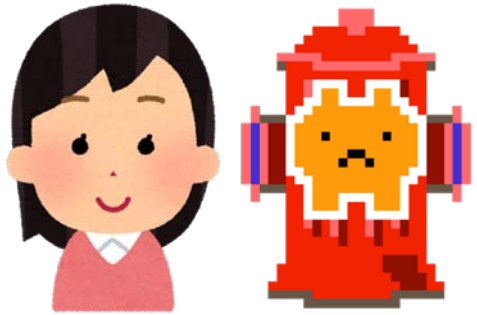


fireplug_cat_06.aiml

トピックの指定

<topic>で作られたブロックで場面を分けることが可能です。
（ゲームで言うシーンみたいな感じ）
topic名を指定すると、指定を解除するまではその名称のtopic内の
category だけが呼ばれるようになります。

```
<!-- トピックへの移動 -->
  <category>
    <pattern>鑑定*</pattern>
    <template>
      <think>
        <set name="topic">assess</set>
      </think>
      <srai>#init</srai>
    </template>
  </category>
```



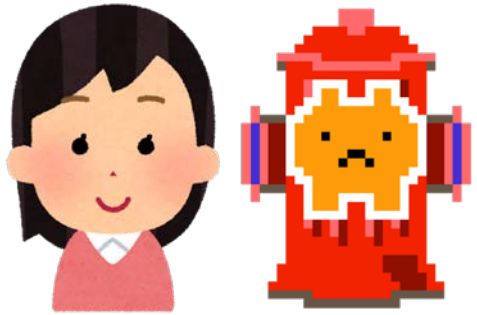
fireplug_cat_07.aiml

sraiによる遷移①

sraiを使うと、ユーザーの発話内容を指定して、再度AIMLを実行できます。

この場合は、topic名assessにセットした後、#initというユーザー発話がされたことになります。

```
<!-- トピックへの移動 -->
  <category>
    <pattern>鑑定*</pattern>
    <template>
      <think>
        <set name="topic">assess</set>
      </think>
      <srai>#init</srai>
    </template>
  </category>
```



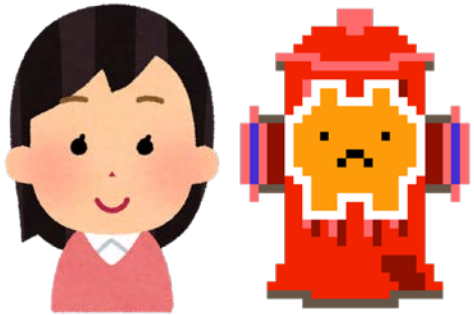
fireplug_cat_07.aiml

sraiによる遷移②

topic名assessで、#initというユーザー発話がされると、これが実行されます。

※initだと本当に発話されちゃうかもしれないので#をつけています。
（チャットボットとかだと入力される可能性があります。）

```
<!-- トピック -->
  <topic name="assess">
    <category>
      <pattern>#init</pattern>
      <template id="question">それじゃあ、鑑定するなの。鑑定してほしいものを言ってみて！</template>
    </category>
```

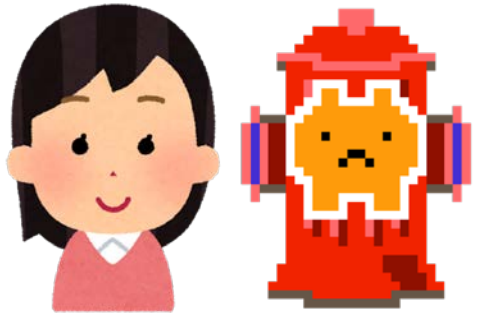


fireplug_cat_07.aiml

thinkタグ

喋らせたくないけど、なにかさせたいときなどに使います。
タグの中身は実行されるけど、喋りません。
変数の設定や、感情のメモなどにつかいます。

```
<!-- トピックへの移動 -->
  <category>
    <pattern>鑑定*</pattern>
    <template>
      <think>
        <set name="topic">assess</set>
      </think>
      <srai>#init</srai>
    </template>
  </category>
```



fireplug_cat_07.aiml

randomタグ

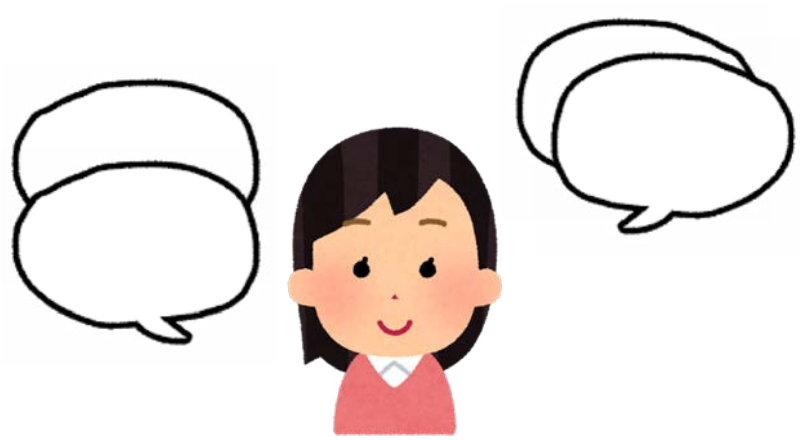
liのどれかをランダムに話させることができます。

```
<category>
    <pattern>*</pattern>
    <template>
      <random>
        <li>ふむふむ。私の鑑定によると・・・これは、良い「<star />」なの。</li>
        <li>ふむふむ。私の鑑定によると・・・普通の「<star />」なの。</li>
        <li>ふむふむ。私の鑑定によると・・・残念ながら、微妙な「<star />」なの。</li>
      </random>
      <srai>#help</srai>
    </template>
  </category>
```

fireplug_cat_07.aiml

ためしてみよう

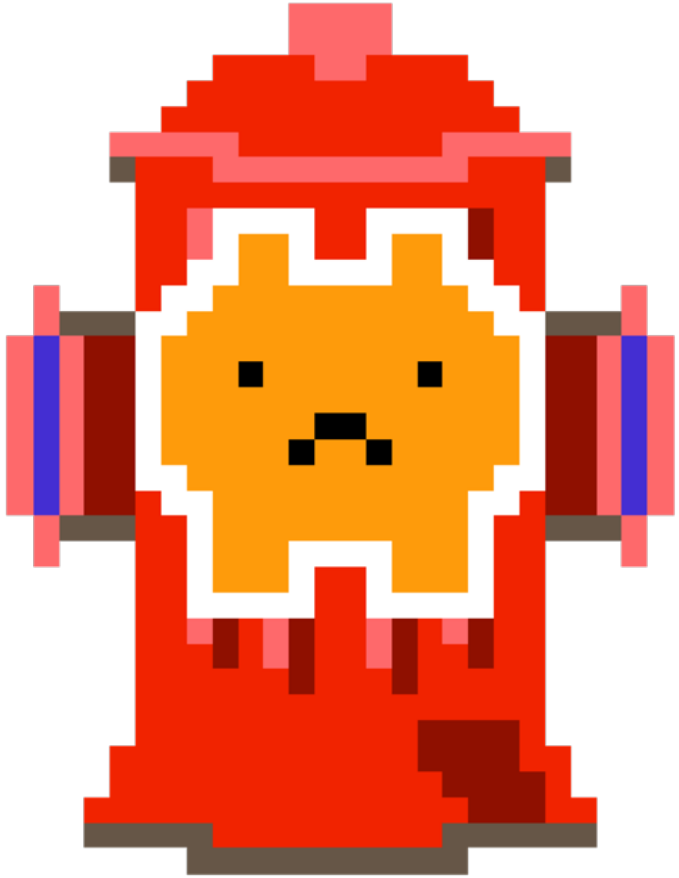
実機で試してみましょう



fireplug_cat_07.aiml

消火栓ねこ完成！

まずは作ってみよう



そのほかのAIMLの特徴

これらのシナリオは、機能毎に作成して管理することが可能です。
（オブジェクト指向言語のクラスのような感じ）

別のシナリオから他のシナリオを呼び出したりもできるので、
シナリオの共通化や、再利用も可能です。

パラメータを設定することで、*で名詞や食べ物の名前だけを取得することも
できます。（辞書を用意することも可能）

パターンのマッチングレベルを変更可能です。完全一致、よみがな一致など。

変数（のようなもの）を各種用意しています。
会話の内容を一時的や、永続的に記憶したりすることが可能です。

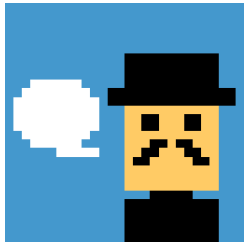
さらにセバスチャンなら・・・

ほかのAIアシスタントのアプリのような感覚で、
エキスパートエージェントを呼ぶことができます。

他社製のエキスパートエージェントだけでなく、
自社製のエキスパートエージェントも呼べるので、
便利な機能は、エキスパートエージェントにして配布するのもありかも。

本当に作りたい「メイン」のキャラクターだけに注力できます。

メイン
エージェント



エキスパート
エージェント

実際に作り始めるには・・・？

メインエージェントを作り始めるには、ドコモとの契約が必要（有料）
まずは、AIエージェント特務チーム“AI Geeks”に連絡するといいみたい！

エージェント自体に興味がある方は、エキスパートエージェントを
開発してみましょう！（無料）

AIMLに興味を持ったら・・・？

xAIML　Nav

Quickstart
Overview
Reference

about xAIML

Summary: xAIMLとは？

Table of Contents

xAIMLとは

AIML（Artificial Intelligence Markup Language）をベースにドコモが仕様拡張を行った記述言語であり、"x"には、ExtendedやExpressiveの意味が含まれています。 ドコモの自然対話プラットフォーム上で対話システムを構築する際に使用します。
AIMLとは、自然言語ソフトウェアエージェント構築のためのXMLを応用した記述言語です。フリーソフトウェアコミュニティによって開発されました。

Reference

xAIMLで実現したいこと

人とシステムとの自然な対話（言葉のキャッチボール）を実現します。

基本構成は２つ

・「ユーザ発話」と「システム発話」の２つ組の構成。

・「システム発話」と「ユーザ発話」また「システム発話」の３つ組の構成。

この２つの構成で成り立っています。

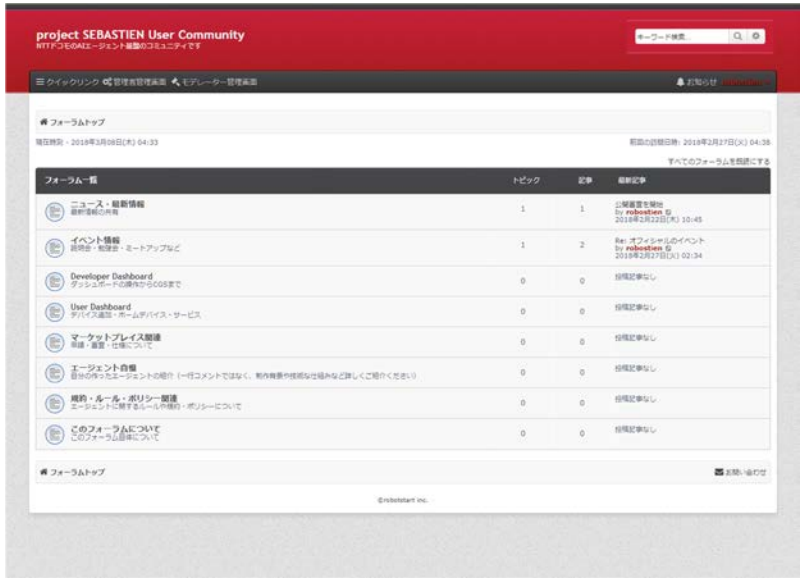
Example

https://xaiml.sebastien.ai/

project SEBASTIEN User Community

https://sebastien.robotstart.jp/

- セバスチャンの開発や導入など困ったことがあったら質問しよう！



SEBASTIEN

アンケートお願いします！

- スマホから回答できるので、ぜひお願いします！

https://tinyurl.com/y7cl9bg7