


Cisco Spark, Tropeo REST, Pythonの概要

大園 通 / OHZONO TOHRU
Sr. Systems Engineer

DevNet Express [Tokyo]

March 15, 2017 • Tokyo, Japan

 DevNet
developer.cisco.com


Cisco Spark, Tropo REST, Pythonの概要

大園 通 / OHZONO TOHRU
Sr. Systems Engineer

本来は同列で並べるような
内容ではないが、
ゆるい関連性があるので、
順を追って説明します

DevNet Express [Tokyo]

March 15, 2017 • Tokyo, Japan

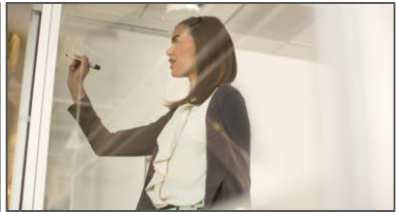
 DevNet
developer.cisco.com

はじめに

おことわり



本プレゼンでは、わかりやすさ重視で、
大胆に噛み砕いた説明を行います。
厳密さは多少犠牲にしているため、ご了承ください。



LEARN < CODE > INSPIRE





ようこそ
DevNet Expressへ！！





学んで



刺激を受けて
次につなげる



LEARN < CODE > INSPIRE



コードを
書いて



今日の過ごし方

楽しみ方は人それぞれですが...

内容的には、
プログラミングに
慣れていない方には
そこそこ難易度の高い
内容です

「動いた」を楽しむ

「動かない」を楽しむ

試行錯誤しながら、
大いに遊んでみましょう！

日本人的な読み方の例

読み方は人それぞれですが...

REST

レスト

JSON

ジェイソン

Python

パイソン

Tropo

トローポ

Git

ギット

GitHub

ギットハブ

「ジット」、「ジットハブ」と
読む人もいるので
文脈で判断

Tropo





Tropo – 主な機能の例



多くの国の電話網と連携可能



電話会議



SMSの送受信



Text to Speech



音声の録音



音声認識



Tropo – 主な機能の例



多

Tropoのクラウドプラットフォーム上で
スクリプト実行することで
Tropoの提供する様々な
機能にアクセスできます

電話会議



Speech

```
say("ようこそTropoの世界へ")  
conference("1337")
```



認識



Tropo – 主な機能の例



多

Tropoのクラウドプラットフォーム上で
スクリプト実行することで
Tropoの提供する様々な
機能にアクセスできます

電話会議



```
say("ようこそTropoの世界へ")  
conference("1337")
```

スクリプトによって、
動作や機能をガンガン
実装していきます！
開発者ありきな
プラットフォームといえる

認識





本日午後のハンズオンでは、
Tropoのハンズオンは実施対象外になります。
ハンズオンの資料自体はあります。

Cisco Spark



Cisco Spark コミュニケーションをよりシンプルに



音声
ビデオ

会議

ファイルの
共有

1:1またはチーム
でのテキスト
メッセージ

モバイル
デスクトップ
アプリ

デスクトップ
会議室の
ビデオ端末

Cisco Sparkプラットフォーム

統合されたユーザ体験



Message



Meeting



Call

オープンなプラットフォーム



Application Integration
API



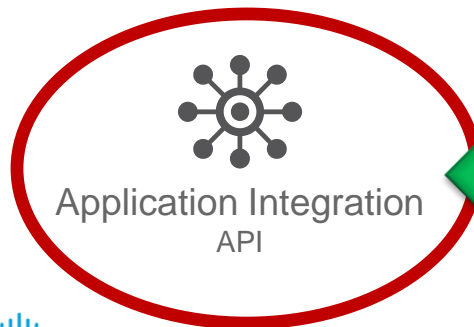
Partner Services
相互連携



Spark Hybrid Services
クラウド + オンプレ

Cisco Sparkプラットフォーム

統合されたユーザ体験



オー

今回は、この2つが
話題の中心！！



Call



Hybrid Services
クラウド+オンプレ

Cisco Spark - Message



Message

Demo

実際に見て、
触ってみるのが、
一番分かりやすい

Cisco Spark - Message

The screenshot displays the Cisco Spark messaging application interface. At the top, the recipient is identified as 'Tohru Ohzono' with a status of 'アクティブ' (Active). The main chat area is titled '開発部コミュニティ' (Development Community) with a subtitle 'DEVNET EXPRESS TOKYO 2017 DEMO'. The chat history shows a message from 'あなた' (You) at 11:01 asking 'あれ？この処理ってマズいね？' (Huh? This processing is a bit off, isn't it?). A code block follows, containing Java code for a loop that checks if a user is an administrator and then processes them. The code is as follows:

```
Java
1 for( final UserInfo user : ssp.getUsers() ) {
2     if( user.isAdmin() ) {
3         processUser( user );
4     }
5 }
```

The chat continues with a response from 'Yuko Yoshida' at 11:02 asking 'Tohru もしかして、getUsers()ってNull返すかも？' (Tohru, maybe getUsers() returns null?). The original sender replies at 11:04: 'Yuko うん。そもそもnull返すの良くないよ？' (Yuko, yes. It's not good to return null in the first place, is it?) and '空リストぶりーず。Collections.emptyList()とかね。' (Empty list, like Collections.emptyList() or something). The interface also shows a sidebar with various channels and a bottom input field for sending messages to the '開発部コミュニティ'.

Cisco Spark - Message

Cisco Spark

TO Tohru Ohzono
アクティブ

すべて +

開発部コミュニティ
DEVNET EXPRESS TOKYO 2017 DEMO

あなた 11:01
あれ？この処理ってマズいね？

```
Java
1 for( final UserInfo user : ssp.getUsers() ) {
2     if( user.isAdmin() ) {
3         processUser( user );
4     }
5 }
```

YY Yuko Yoshida 11:02
Tohru もしかして、getUsers()ってNull返すかも？
だったら、Nullチェック必要だねー

あなた 11:04
Yuko うん。そもそもnull返すの良くないよ？
空リストぶりーず。Collections.emptyList()とかね。
治してから、もう一回ブルクしてね。

+ 開発部コミュニティ にメッセージを書く

スペースとよばれる
仮想的な場所で
テキストやビデオを
使って
社内外の人達と
連携する

Cisco Spark - Message

The screenshot shows the Cisco Spark messaging application interface. At the top, it says "Cisco Spark" and "Tohru Ohzono". Below that, there's a search bar and a menu icon. The main content area shows a list of channels and messages. A red box highlights the channel list on the left. A blue speech bubble points to the channel list with the following text:

いろいろな議題などに
基づいて
様々なスペースが
できていく

The channel list includes:

- 開 DEVNET EXPRESS TOKYO 2017 D... 開発部コミュニティ 11:04
あなた: Yuko うん。そもそもnull返すの良...
- YY Yuko Yoshida Yesterday
Yuko: 今日中にして作ってシェアしときます。
- 一 DEVNET EXPRESS TOKYO 2017 D... 一般 Yesterday
あなた: 了解
- ITサポート相談室 Yesterday
スペース「ITサポート相談室」を開始します。
- 新 DEVNET EXPRESS TOKYO 2017 D... 新製品開発 Yesterday
スペース「新製品開発」を開始します。
- 勤 DEVNET EXPRESS TOKYO 2017 D... 勤怠連絡 Yesterday
スペース「勤怠連絡」を開始します。

At the bottom, there's a text input field: "+ 開発部コミュニティ にメッセージを書く" and a send button.

Cisco Spark - Message

The screenshot shows the Cisco Spark interface. At the top, it says "Cisco Spark" and "TO Tohru Ohzono". Below that, there's a search icon and a group chat header for "開発部コミュニティ" (Development Community) with the subtitle "DEVNET EXPRESS TOKYO 2017 DEMO". The chat history shows a message from "あなた" (You) at 11:01 asking "あれ?この処理ってマズいね?" (Isn't this process weird?). Below that is a code block:

```
1 for( final UserInfo user : users)
2     if( user.isAdmin() )
3         processUser(user);
4     }
5 }
```

Then, a message from "YY" (Yuko Yoshida) at 11:02 says "Tohru もしかして、getUsers()ってなかったら、Nullチェック必要だねー" (Tohru, maybe you need a null check for getUsers()). A red circle highlights the name "Yuko" in the message header. At the bottom, there's a button to "開発部コミュニティ にメッセージを書く" (Write message to Development Community).

大勢いるスペースでも
@mention(@メンション)機能を使って、
スペース内の特定の人に
話しかけることもできる。
この例では、“Yuko”に対して
スペース内で話しかけている。
(投稿内容はスペース内の全員に見える)

Cisco Spark - Message

The screenshot shows the Cisco Spark Message interface. At the top, it says "Cisco Spark" and "To: Tohru Ohzono". Below that, there's a search icon and a dropdown menu for "すべて". The main conversation is with "Yuko Yoshida". The message history shows:

- Yuko Yoshida: 今日中に作ってシェアします。 (16:41)
- あなた: 今日中に作ってシェアします。 (15:56) - Deleted message
- あなた: こんにちは！ (15:56)
- あなた: 明日の午後の会議だけど、資料準備できそう？ (16:40)
- Yuko Yoshida: 今日中に作ってシェアします。 (16:41)

The left sidebar shows a list of conversations, with the one for Yuko Yoshida highlighted by a red box. Other conversations include "開発部コミュニティ", "DEVNET EXPRESS TOKYO 2017 D...", "一般", "ITサポート相談室", "DEVNET EXPRESS TOKYO 2017 D...", "新製品開発", and "DEVNET EXPRESS TOKYO 2017 D...".

1対1でのやりとりも
できる

Sparkは何をつなぐか？

- いくつかの例 -



Sparkは何をつなぐか？

- いくつかの例 -



Sparkは何をつなぐか？

- いくつかの例 -



Sparkは何をつなぐか？ いくつかの例

さらにその先には、
別のアプリ、
監視カメラ、機械、乗り物
などなど

人々





Sparkは何をつなぐか？

- いくつかの例 -

インタフェースが
あるもの同士なら
可能性は無限大！！



人々



Sparkは何をつなぐか？

- いくつかの例 -

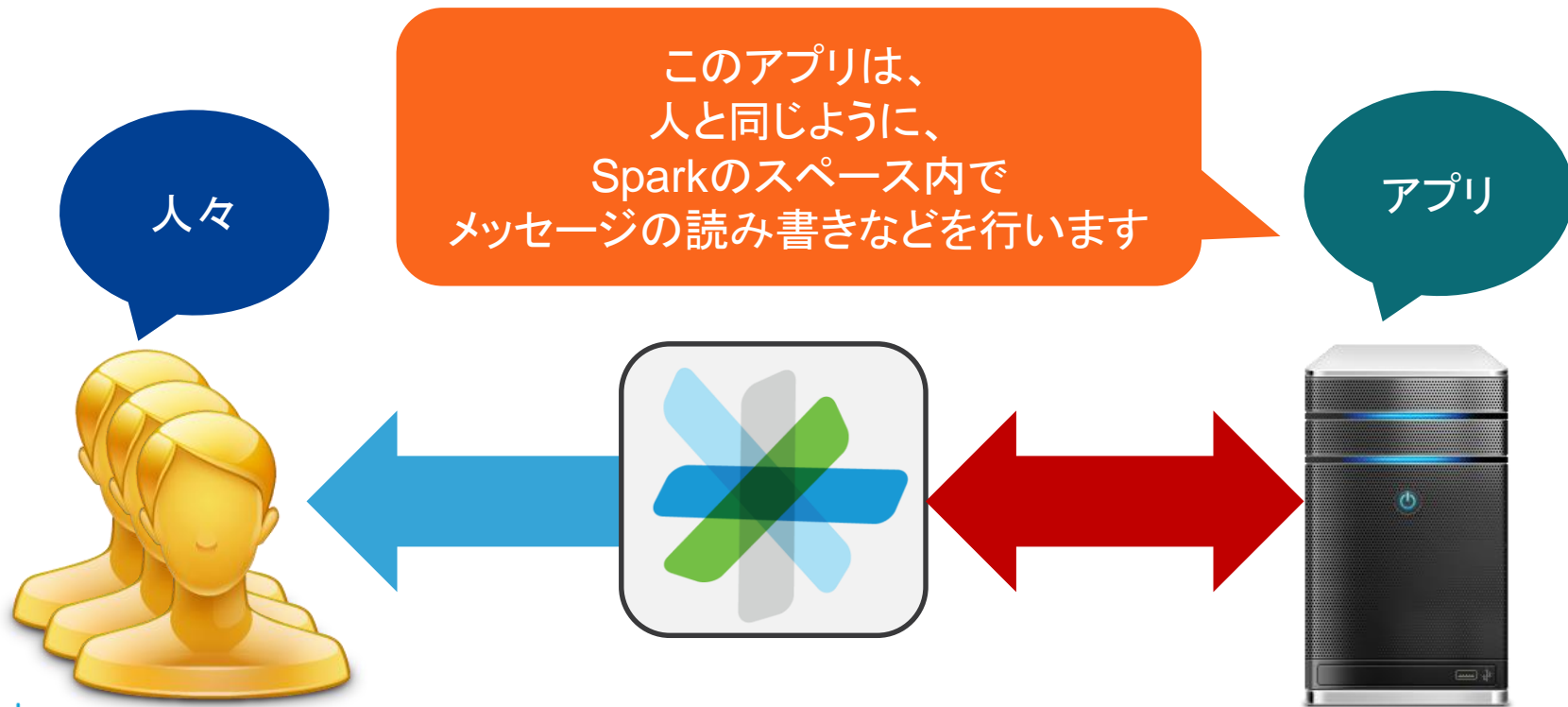


互いに
インターフェースさえあれば
人工衛星とだって
連携できる！



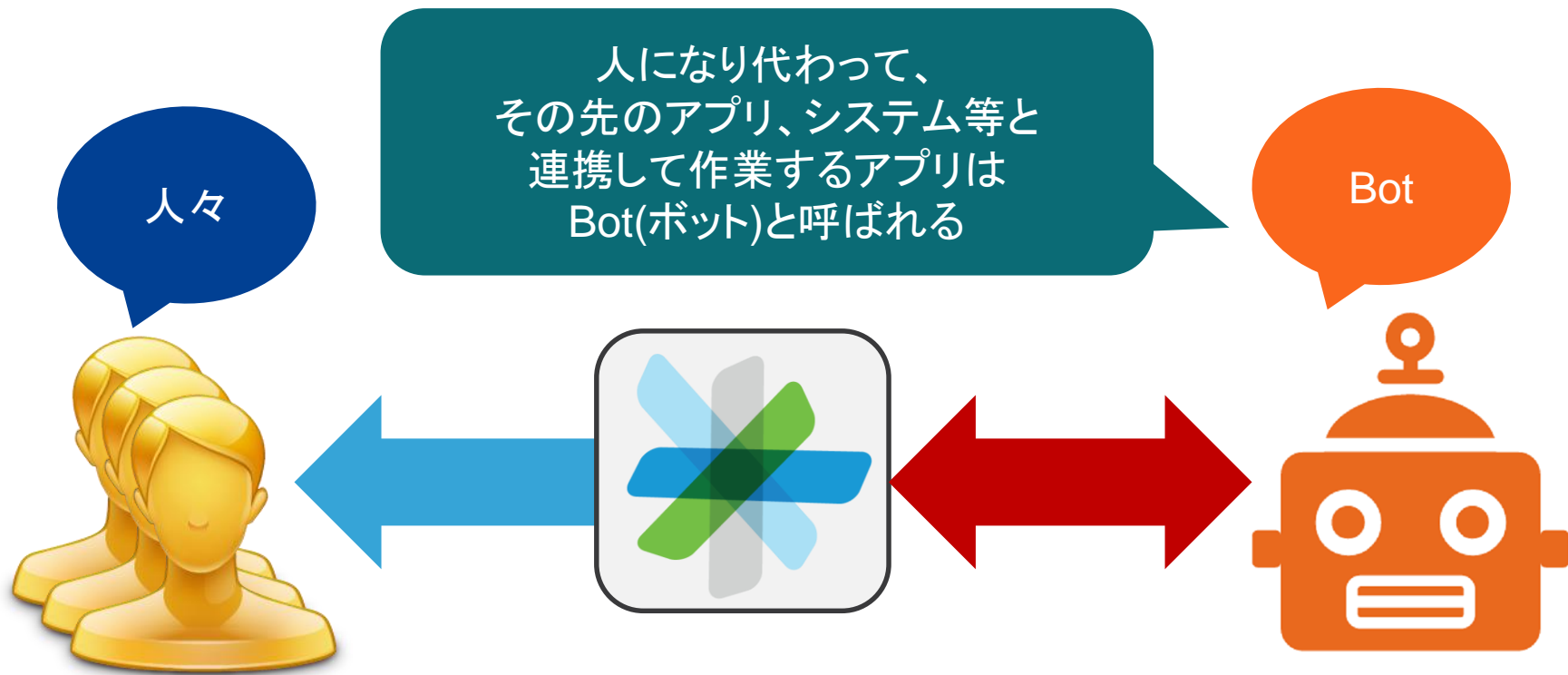
Sparkは何をつなぐか？

- いくつかの例 -



Sparkは何をつなぐか？

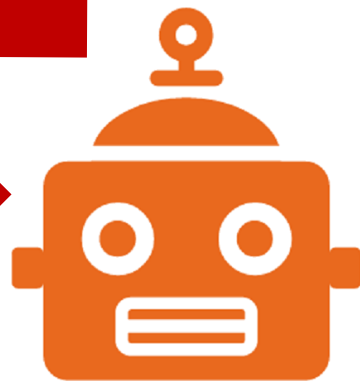
- いくつかの例 -



Sparkは何をつなぐか？

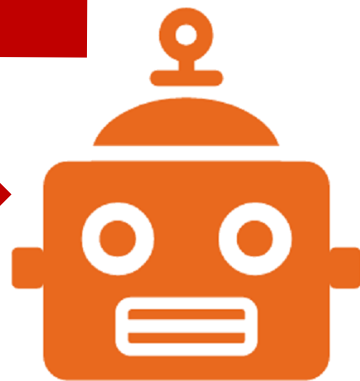
- いくつかの例 -

Botを介して
Sparkと様々な
ものがつながる
ことができる！！



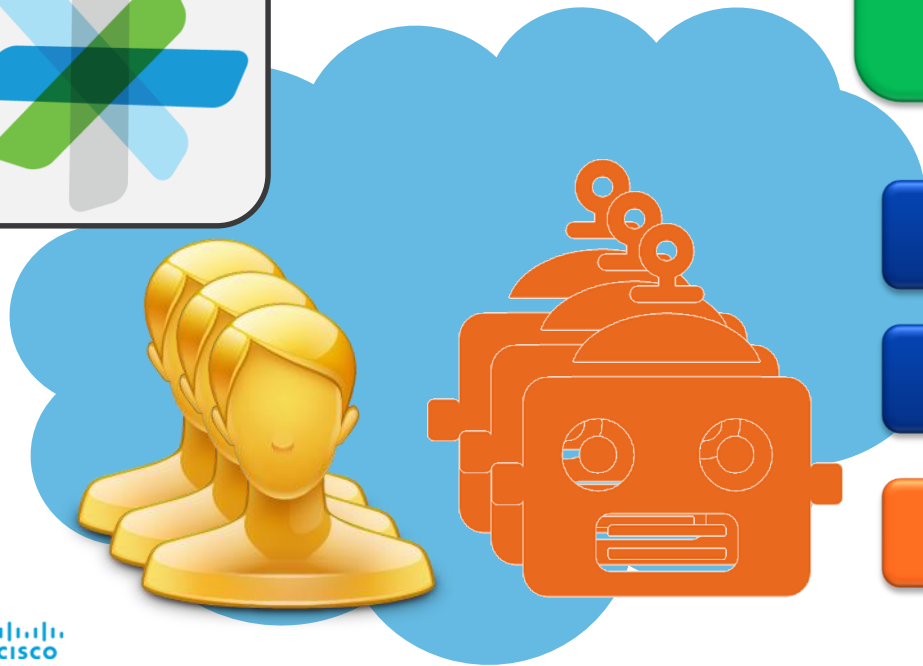
Sparkは何をつなぐか？ - いくつかの例 -

Botを介して
Sparkと様々な
ものがつながる
ことができる！！

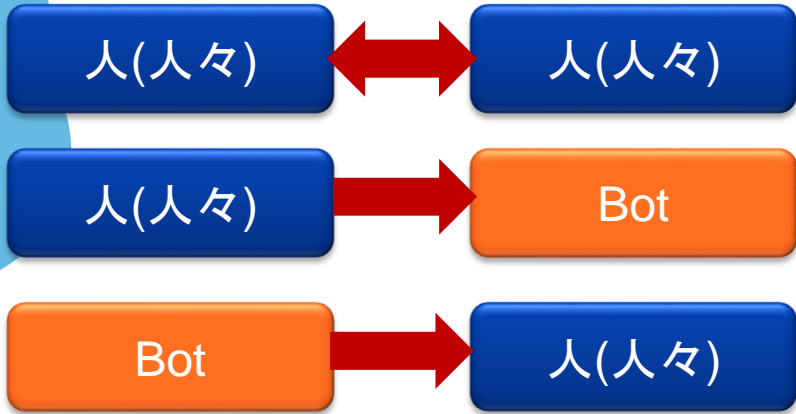


Sparkは何をつなぐか？

- いくつかの例 -



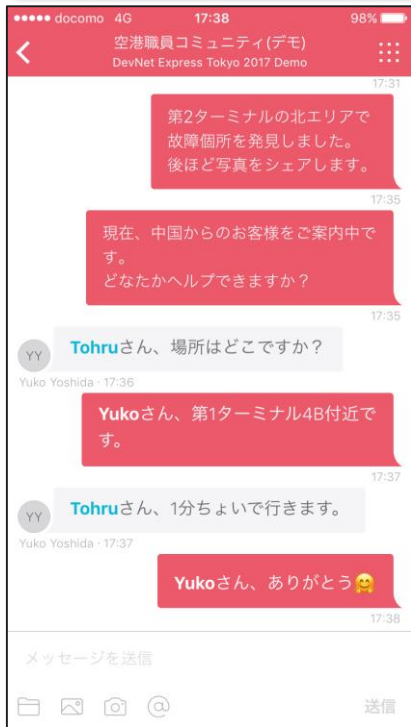
Sparkのスペース内で
これらが繋がらうことで
新しい価値を生み出せる
可能性がある





人(人々)、Bot、その先のシステム が繋がりが合う例

空港職員コミュニティスペースのコンセプトデモ

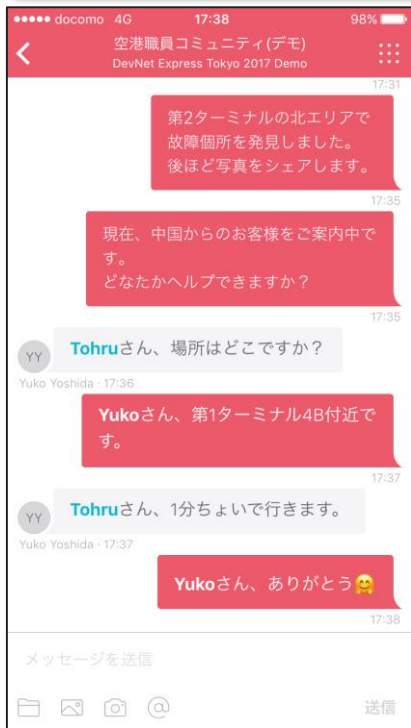


空港職員同士でのやりとり



人(人々)、Bot、その先のシステム が繋がりが合う例

空港職員コミュニティスペースのコンセプトデモ



お客様から、
“CSCO11便”の
運行情報を
質問された場合



人(人々)、Bot、その先のシステム が繋がりが合う例

空港職員コミュニティスペースのコンセプトデモ





人(人々)、Bot、その先のシステム が繋がりが合う例

空港職員コミュニティスペースのコンセプトデモ



同じスペースにいる
運行情報システムと連携するBotに
話しかけて、運行情報ゲット。
お客様に情報を伝える



人(人々)、Bot、その先のシステム が繋がりが合う例

空港職員コミュニティスペースのコンセプトデモ



お客様から、
忘れ物の
相談をされた場合



人(人々)、Bot、その先のシステム が繋がりが合う例

空港職員コミュニティスペースのコンセプトデモ





人(人々)、Bot、その先のシステム が繋がりが合う例

空港職員コミュニティスペースのコンセプトデモ



同じスペースにいる
紛失物検索システムと連携するBotに
話しかけて、登録情報を確認。
お客様をご案内



人(人々)、Bot、その先のシステム が繋がりが合う例

空港職員コミュニティスペースのコンセプトデモ



お客様ご案内ロボット
がお困り中



人(人々)、Bot、その先のシステム が繋がりが合う例

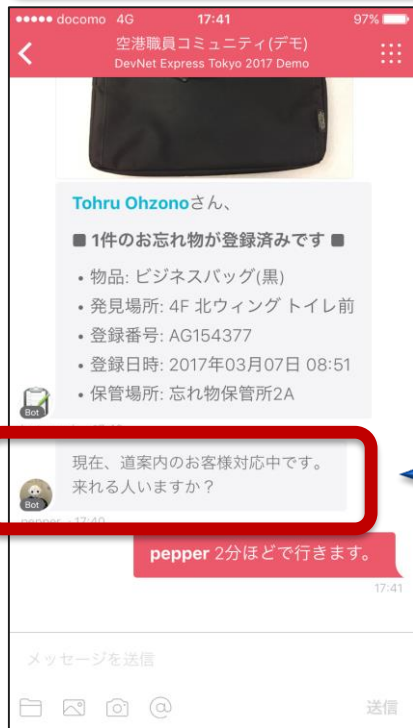
空港職員コミュニティスペースのコンセプトデモ





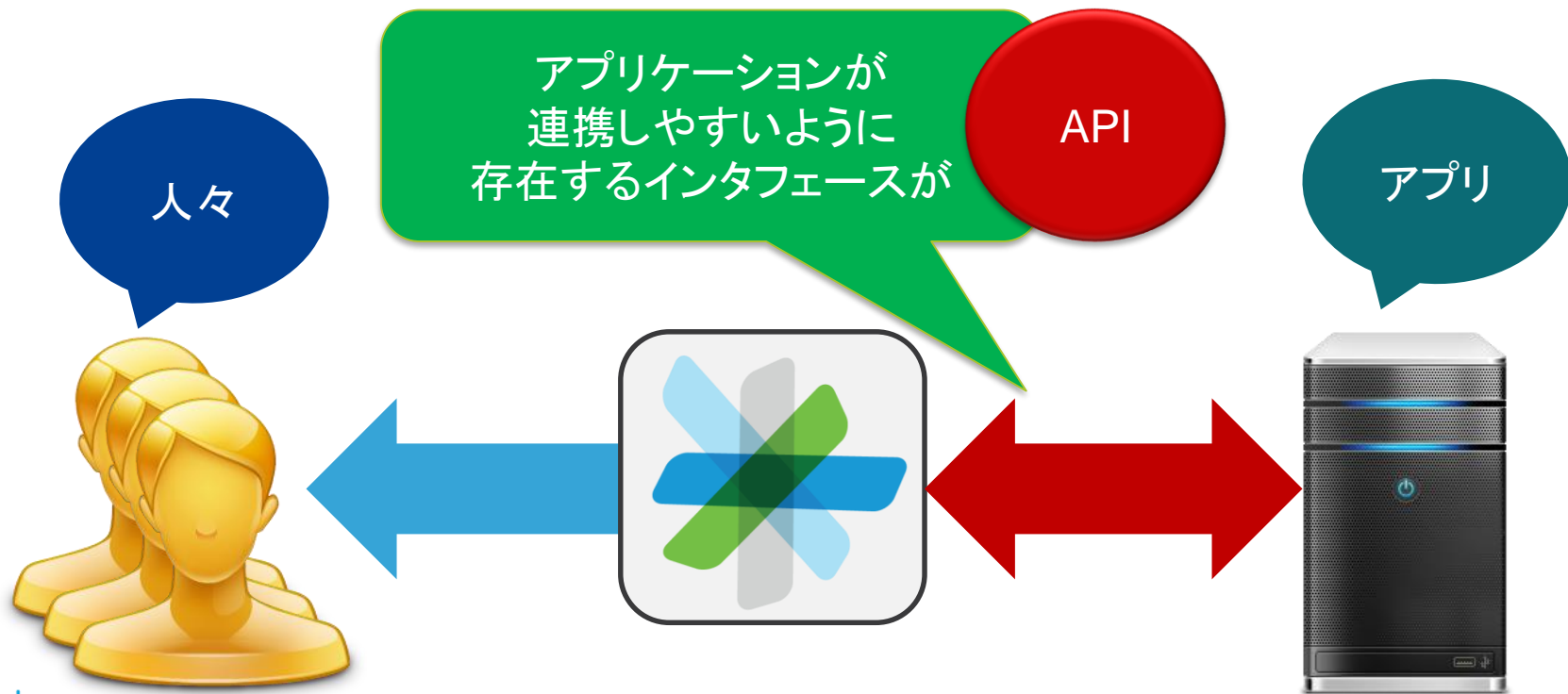
人(人々)、Bot、その先のシステム が繋がりが合う例

空港職員コミュニティスペースのコンセプトデモ

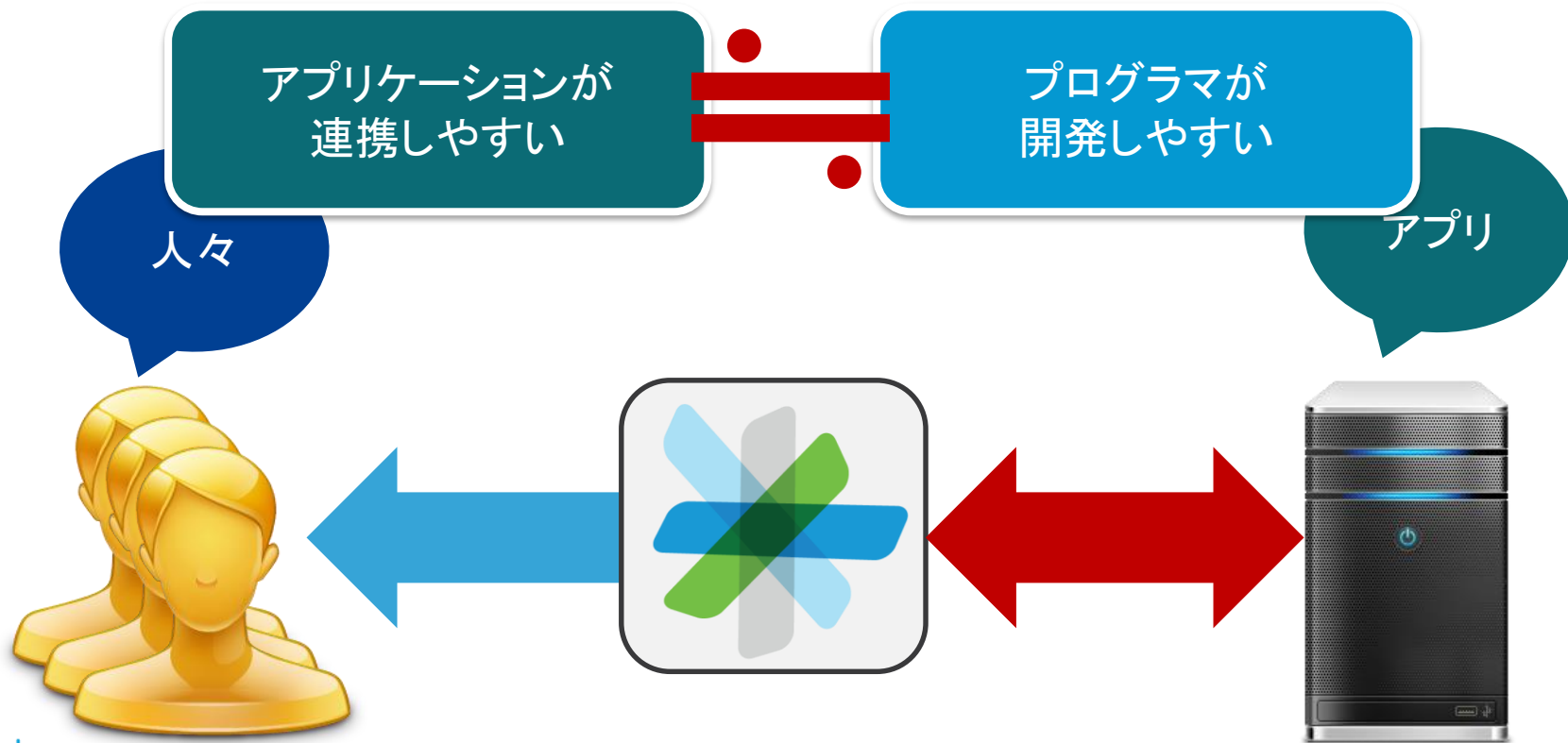


ロボットと連動したBot
が人に助けを請う

Spark API



Spark API



Spark API

Spark Message APIの機能の例

Spark上のメッセージ

投稿

リスト取得

削除

詳細取得

Spark上のスペース

作成

更新

リスト取得

詳細取得

削除

その他、メッセージが投稿された際などにイベントを受け取ったりも可能



Spark API

朗報！！

SparkのAPIを使って
アプリ開発を始めるのに
追加のコストは
必要ありません！

Sparkのユーザに
なった瞬間に、
開発者にもなれます！

Sparkの無償版の
機能に対応するAPIは、
やはり無償で使えます！

Spark For Developers

Cisco Sparkは、

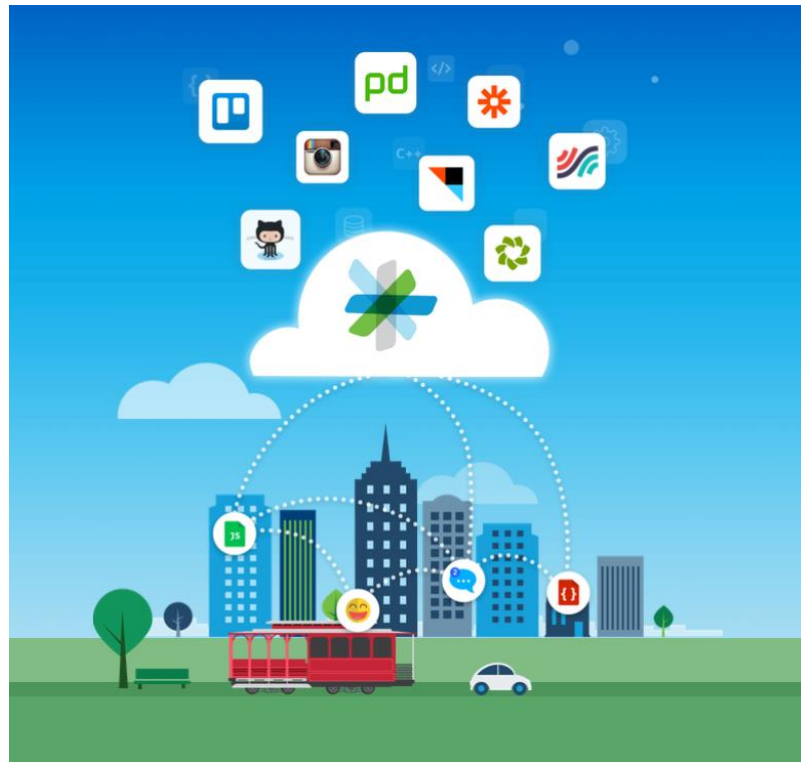
REST API

を提供しています

developer.ciscospark.com



Application Integration
APIs



REST API

REST API

REST API

多くの場合、

REST

というアーキテクチャ

の影響を多分に取り入れた

Web API

を指すことが多い

REST API

REST API

多くの場合、

REST

というアーキテクチャ

の影響を多分に取り入れた

Web API

を指すことが多い

厳密な話は抜きにして、
かなり噛み砕いた説明をします

REST API

REST API

REST

Web API

一般的には、
HTTP / HTTPSを利用した
APIを指す

HTTP

HTTP

まずは、HTTPの説明



今日は、
HTTP/1.1と呼ばれるバージョンの話です。
HTTP/2では、事情がまた異なります。

HTTP

```
https://server.example.com/path/contents?max=5
```

ネットワーク上のリソース(情報資源)の場所を表す
URLの例です

このURLが指し示す場所から
リソースを取得するアプリを
考えてみよう

HTTP

`https://server.example.com/path/contents?max=5`

`https`

`server.example.com`

`/path/contents`

`max=5`

適度に分解しました

HTTP

`https://server.example.com/path/contents?max=5`

`https`

`server.example.com`

`/path/contents`

`max=5`

スキーム名

ホスト名

パス

パラメータ
(Query)

HTTP

`https://server.example.com/path/contents?max=5`

`https`

`server.example.com`

`/path/contents`

`max=5`

TCP with TLSでコネクションを確立して、
その上でHTTPを使う

TCPコネクションを確立する相手は
server.example.com

HTTP

`https://server.example.com/path/contents?max=5`

https

server.example.com

/path/contents

max=5

TCP with TLSでコネクションが確立されて
通信が可能な状態になる



アプリ



server.example.com

HTTP

`https://server.example.com/path/contents?max=5`

https

server.example.com

/path/contents

max=5

このパス(場所)のリソースが欲しい

結果を絞りたい



アプリ



server.example.com

HTTP

```
https://server.example.com/path/contents?max=5
```

このようなテキストのリクエストを
確立されたTCPコネクション上で送る

```
path/contents
```

```
max=5
```

```
GET /path/contents?max=5 HTTP/1.1
```



アプリ



```
server.example.com
```

HTTP

`https://server.example.com/path/contents?max=5`

リクエストを受けたサーバのアプリは、
リソースをくれ(GET/取得)と
要求されていることが分かるし

`path/contents`

`max=5`

`GET /path/contents?max=5 HTTP/1.1`



アプリ



`server.example.com`

HTTP

`https://server.example.com/path/contents?max=5`

https

se

どのパス(場所)のリソースを
要求されているかもわかる

max=5

GET /path/contents?max=5 HTTP/1.1



アプリ



server.example.com

HTTP

`https://server.example.com/path/contents?max=5`

https

server.example.com

HTTP 1.1の手順と規約にしたがって
レスポンスを返せばいいこともわかる

GET /path/contents?max=5 HTTP/1.1



アプリ



server.example.com

HTTP

`https://server.example.com/path/contents?max=5`

`https`

`server.example.com`

要求された通りの
リソースを探したりすることで
レスポンスで返すことができる

`GET /path/contents?max=5 HTTP/1.1`



HTTP

`https://server.example.com/path/contents?max=5`

https

server

/path/contents

max=5

リクエストされたリソースを
このようなテキストのレスポンス
として返します

HTTP/1.1 200 OK

〇〇の秘密: 〇〇はxxだ!!



アプリ



server.example.com

HTTP

`https://server.example.com/path/contents?max=5`

`https`

`/path/contents`

`max=5`

HTTP 1.1の手順と規約にしたがった
レスポンスであることがわかる

HTTP/1.1 200 OK

〇〇の秘密: 〇〇はxxだ!!



アプリ



server.example.com

HTTP

`https://server.example.com/path/contents?max=5`

https

server.example.com

/path/contents

max=5

リソースが正常に
取得できていることもわかる

HTTP/1.1 200 OK

〇〇の秘密: 〇〇はxxだ!!



アプリ



server.example.com

HTTP

`https://server.example.com/path/contents?max=5`

https

server.example.com

/path/contents

max=5

リソースの中身が
取得できます

HTTP/1.1 200 OK

〇〇の秘密：〇〇はxxだ！！



アプリ



server.example.com

HTTP

`https://server.example.com/path/contents?max=5`

https

server

max=5

このように、
リクエストとレスポンスを
テキストでやりとりするのが、
HTTP/1.1の基本

リクエスト



アプリ



server.example.com

レスポンス

HTTP

実際には、
情報を受け取った側は、
これだけでは処理しづらいし、
拡張性もない



アプリ

```
HTTP/1.1 200 OK
```

```
〇〇の秘密: 〇〇はxxだ!!
```



server.example.com

HTTP

文字列の
エンコーディングは？

文字列情報は
何バイト
読み込めばいいの

文字列の情報しか
受け取れないの？



アプリ

```
HTTP/1.1 200 OK
```

```
〇〇の秘密： 〇〇はxxだ！！
```



server.example.com

HTTPヘッダ

リクエスト

と

レスポンス

それぞれに

ヘッダ

を付加することで、相手の処理しやすさと
拡張性を確保することができる



アプリ

```
HTTP/1.1 200 OK
```

```
〇〇の秘密： 〇〇はxxだ！！
```



server.example.com

HTTPレスポンスヘッダ

レスポンスにヘッダを付けた例

受信したアプリは、
このレスポンスを適切に処理
できるようになる

```
HTTP/1.1 200 OK  
Content-Length: 37  
Content-Type: text/plain; charset=utf-8
```

〇〇の秘密: 〇〇はxxだ!!



アプリ



server.example.com

HTTPレスポンスヘッダ

含まれるリソースの
本体(Body)は
単純なテキスト
であることがわかるし

```
HTTP/1.1 200 OK  
Content-Length: 27  
Content-Type: text/plain; charset=utf-8
```

〇〇の秘密: 〇〇はxxだ!!

本体(Body)



アプリ



server.example.com

HTTPレスポンスヘッダ

37バイト
読み込めば全部
読み込めることもわかる

```
HTTP/1.1 200 OK  
Content-Length: 37  
Content-Type: text/plain; charset=utf-8
```

〇〇の秘密: 〇〇はxxだ!!

本体(Body)



アプリ



server.example.com

HTTPレスポンスヘッダ

本体(Body)には
テキストだけではなく、
画像などのバイナリデータも
含まれるようになる

```
HTTP/1.1 200 OK  
Content-Length: 8254  
Content-Type: image/png
```

png画像のバイナリデータ...

本体(Body)



アプリ



server.example.com

HTTPリクエストヘッダ

```
GET /path/contents?max=5 HTTP/1.1  
Host: server.example.com  
Accept: text/plain
```



アプリ



リクエスト



server.example.com

HTTPリクエストヘッダ

リクエストにヘッダを付けた例

リクエスト時と同様に
リクエストを受信した側で、
適切に処理できるようになる

```
GET /path/contents?max=5 HTTP/1.1  
Host: server.example.com  
Accept: text/plain
```



アプリ



リクエスト



server.example.com

HTTPリクエストヘッダ

サーバ側に
ユーザの認証・認可情報を渡して
認証や認可処理を実行することもある

```
GET /path/contents?max=5 HTTP/1.1  
Host: server.example.com  
Accent: text/plain  
Authorization: Bearer NDM5MjYz.....
```



リクエスト



server.example.com

HTTPメソッド

リクエストにおいて、ここは

メソッド

と呼ばれて、
リソースに対する
操作を指定する

```
GET /path/contents?max=5 HTTP/1.1  
Host: server.example.com  
Accept: text/plain
```



アプリ



リクエスト



server.example.com

HTTPメソッド

DELETE

だと、

リソースの削除

を要求

```
DELETE /path/contents HTTP/1.1  
host: server.example.com  
Accept: text/plain
```



アプリ



リクエスト



server.example.com

HTTPメソッド

メソッド

の例

GET

POST

PUT

DELETE

今日は、これらの
メソッドが重要になるが
あとで説明



アプリ

リクエスト



server.example.com

HTTPレスポンスコード

レスポンスにおいて、ここは

ステータスコードと
その説明

になっていて
リソースへの操作
への成否等を表す

```
HTTP/1.1 200 OK  
〇〇の秘密: 〇〇はxxだ!!
```



アプリ



server.example.com

レスポンス

HTTPレスポンスコード

ステータスコードと
その説明

の例

200 OK

成功

204 No Content

成功、本文(Body)なし(本文見る必要なし)

404 Not Found

探したけど、リソースが見つからなかった



アプリ

```
HTTP/1.1 200 OK  
〇〇の秘密: 〇〇はxxだ!!
```



server.example.com

レスポンス

REST API

REST API

HTTPの仕組みの上で、さらに、
一定の規約、決まり事を設けることで、
クライアントとサーバ間で
一貫したルールをもって、やりとりを行う

REST API



実際に例で
考えるのが
手っ取り早いので

Cisco Spark君に
再登場してもらいます

REST APIの例



Sparkの
スペース内のメッセージをイメージします

Sparkクライアントでの呼び方

スペース / Space

Spark API上での名称

ルーム / Room

歴史的な経緯(割愛)で、
呼び方には差異がありますが、
本日の範囲では同じものを指すと理解しておく

REST APIの例



リソースを表すURLは例えばこんな感じ

```
https://api.ciscospark.com/v1/messages
```

REST APIの例



リソースを表すURLは例えばこんな感じ

`https://api.ciscopark.com/v1/messages`

APIを提供するサーバのアドレス

REST APIの例



リソースを表すURLは例えばこんな感じ

`https://api.ciscospark.com/v1/messages`

リソースのパス

Sparkのスペースの場合は、

`/v1/rooms`

など、リソースの種類によって異なる

REST APIの例



リソースを表すURLは例えばこんな感じ

`https://api.ciscopark.com/v1/messages`

拡張性や互換性のためなど、
パスにバージョンが含まれる場合も多い

REST APIの例



リソースを表すURLには、2パターンある

```
https://api.ciscospark.com/v1/messages
```

```
https://api.ciscospark.com/v1/messages/Y21zY29z...
```

REST APIの例



リソースを表すURLには、2パターンある

`https://api.ciscopark.com/v1/messages`

IDなし

`https://api.ciscopark.com/v1/messages, Y21zY29z...`

IDあり

REST APIの例



リソースを表すURLには、2パターンある

`https://api.ciscospark.com/v1/messages`

IDなし

`https://api.ciscospark.com/v1/messages/Y21zY29z...`

IDあり



IDあり、なしのパスが多段階でネストする場合があります。
Spark APIのように、ネストはないシンプルな構造の方が、
APIとしては美しい(と思います / 個人的な感想)

REST APIの例



<https://api.ciscopark.com/v1/messages/Y21zY29z...>



こいつを狙い撃ち

IDありの方は、
特定のメッセージを指す

このリソースは
世界にただ1つだけ
存在するのが原則

REST APIの例



```
https://api.ciscospark.com/v1/messages/Y21zY29z...
```

一般には、以下の操作が可能と考えられる。
GET/PUT/DELETEは
操作に対応するHTTPのメソッド

GET

特定のメッセージの取得

PUT

特定のメッセージの更新

DELETE

特定のメッセージの削除

REST APIの例



```
https://api.ciscospark.com/v1/messages/Y21zY29z...
```

Sparkは既存のメッセージの更新をサポートしないので、メッセージの操作にはPUTは利用しない (Sparkの機能から来る制限)

一般には、以下の操作が可能と考えられる。
GET/PUT/DELETEは
操作に対応するHTTPのメソッド

GET

特定のメッセージの取得

PUT

特定のメッセージの更新

DELETE

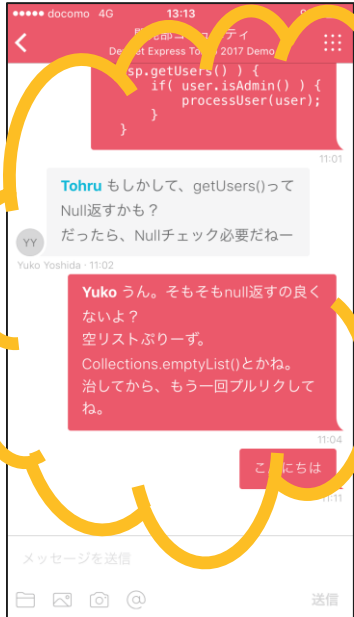
特定のメッセージの削除

REST APIの例



<https://api.ciscospark.com/v1/messages>

IDなしの方は、
一般のメッセージを指す
やや抽象的な概念



特定のどれかを
指すわけではない

REST APIの例



`https://api.ciscospark.com/v1/messages`

一般には、以下の操作が可能と考えられる。
GET/POST/PUT/DELETEは
操作に対応するHTTPのメソッド

GET

メッセージリストの取得

POST

新規メッセージの投稿

PUT

全てのメッセージの更新

DELETE

全てのメッセージの削除

REST APIの例



`https://api.ciscospark.com/v1/messages`

一般には、以下の操作が可能と考えられる。
GET/POST/PUT/DELETEは
操作に対応するHTTPのメソッド

Spark APIは複数の
メッセージに対する
一括更新、削除はサポート
しないので、
PUT/DELETEは利用しない
(Sparkの機能から来る制限)

GET

メッセージリストの取得

POST

新規メッセージの投稿

PUT

全てのメッセージの更新

DELETE

全てのメッセージの削除

REST APIの例



`https://api.ciscospark.com/v1/messages`

特定の1つの
メッセージに
限定されない

GET **メッセージリストの取得**

POST **新規メッセージの投稿**

PUT メッセージの更新

DELETE メッセージの削除

まだ存在しない
メッセージ

表にするとこんな感じ！！

REST APIの例

	IDなしのパス (例: .../messages)	IDありのパス (例: .../messages/Y2IzY29z...)
説明	該当パス上の一般的なリソースを表す	該当パスが表す特定のリソースを指す
GET	リソースのリストの取得	特定のリソースの詳細を取得
POST	リソースの新規作成	- (通常は使用しない)
PUT	リソース全体の更新	特定のリソースを変更する
DELETE	リソース全体の削除	特定のリソースを削除する

REST APIの例

	IDなしのパス (例: .../messages)	IDありのパス (例: .../messages/1)
説明	該当パス上の一般的なリソースを表す	特定のリソースを指す
GET	リソースのリストの取得	特定のリソースの詳細を取得
POST	リソースの新規作成	- (通常は使用しない)
PUT	リソース全体の更新	特定のリソースを変更する
DELETE	リソース全体の削除	特定のリソースを削除する

GETが最も安全な操作。
他の操作は、通常、リソースに
なんらかの変更がなされる



REST APIの例

	IDなしのパス (例: .../messages)	IDありのパス (例: .../messages/Y2IzY29z...)
説明	該当パス上の一般的なリソースを表す	該当パス上の一般的なリソースを表す
GET	リソースのリストの取得	リソースのリストの取得
POST	リソースの新規作成	リソースの新規作成
PUT	リソース全体の更新	特定のリソースを変更する
DELETE	リソース全体の削除	特定のリソースを削除する

これらの機能は
提供されていない場合も多い
※ 大きな変化をもたらす可能性が高く、
通常は、危険度が高い処理

REST APIの例

Sparkの特定のメッセージ
を取得する例で、
もう少しREST APIの中身に
踏み込んでみよう



Spark APIを
処理するサーバ



api.ciscospark.com

REST APIの例

ある特定の
メッセージを
取得したい！



Spark APIを
処理するサーバ



api.ciscopark.com

Sparkでメッセージを取得

```
https://api.ciscospark.com/v1/messages/Y21zY29z...
```

特定のメッセージリソースの取得なので、IDありのパスに

GET



アプリ



api.ciscospark.com

Sparkでメッセージを取得

`https://api.cisco`

こんな感じでSparkのAPIサーバに
リクエストを出します

```
GET /v1/messages/Y2lzY29z... HTTP/1.1
Host: api.ciscospark.com
Accept: application/json
Authorization: Bearer NDM5MjYz.....
Content-Type: application/json; encoding=utf-8
```



アプリ



`api.ciscospark.com`

Sparkでメッセージを取得

`https://api.ciscopark.com`

認証・認可のために
Botなどのトークン情報を渡しているので、
Spark APIサーバ側で、
Botなどの認証・認可等ができる

```
GET /v1/messages
Host: api.ciscopark.com
Accept: application/json
Authorization: Bearer NDM5MjYz.....
Content-Type: application/json; encoding=utf-8
```



アプリ



`api.ciscopark.com`

Sparkでメッセージを取得

`https://api.ciscopark.com/v1/messages/Y21zY29z...`

Sparkの
クラウドサーバと
連携して
メッセージを取得



`api.ciscopark.com`

Sparkでメッセージを取得

Spark APIサーバは、
こんな感じのレスポンスを返す

```
HTTP/1.1 200 OK
Content-Length: 578
Content-Type: application/json; charset=utf-8

{
  "id": "Y21zY29z...",
  "roomId": "Y21zY29zcGF...",
  (一部省略...)
  "text": "こんにちは",
  (一部省略...)
  "created": "2017-03-06T02:56:41.512Z"
}
```



アプリ



api.ciscospark.com

JSON

```
HTTP/1.1 200 OK
Content-Length: 578
Content-Type: application/json; charset=utf-8

{
  "id": "Y21zY29z...",
  "roomId": "Y21zY29zcGF...",
  "text": "こんにちは",
  "created": "2017-03-06T02:56:41.512Z"
}
```

JSON

```
HTTP/1.1 200 OK  
Content-Length: 576  
Content-Type: application/json; charset=utf-8
```

```
{  
  "id": "Y21zY29z...",  
  "roomId": "Y21zY29zcGF...",  
  "text": "こんにちは",  
  "created": "2017-03-06T02:56:41.512Z"  
}
```

リソースの中身の
本文(Body)の形式は何でもよいが

JSON

を利用するREST APIが多い

JSON

```
HTTP/1.1 200 OK
Content-Length: 578
Content-Type: application/json; charset=utf-8
```

```
{
  "id": "Y21zY29z...",
  "roomId": "Y21zY29zcGF...",
  "text": "こんにちは",
  "created": "2017-03-06T02:56:41.512Z"
}
```

これが

JSON

説明用に、
実際のSparkの
レスポンスから
一部省略している

JSON

```
HTTP/1.1 200 OK
Content-Length: 578
Content-Type: application/json; charset=utf-8
```

```
{
  "id": "Y21zY29z...",
  "roomId": "Y21zY29zcGF...",
  "text": "こんにちは",
  "created": "2017-03-06T02:56:41.512Z"
}
```

見た目以上に
多くの構成要素
で構成されている

メッセージID

存在するスペース

テキスト

作成日時

等々



JSON

JSONの例

```
{  
  "id": "Y21zY29z...",  
  "roomId": "Y21zY29zcGF...",  
  "text": "こんにちは",  
  "created": "2017-03-06T02:56:41.512Z"  
}
```

JSON

JSONの例

1つの情報の塊を、{}で囲んで
表現します。
この例では、Sparkのメッセージ情報
を表す情報の塊

```
{  
  "id": "Y21zY29z...",  
  "roomId": "Y21zY29zcGF...",  
  "text": "こんにちは",  
  "created": "2017-03-06T02:56:41.512Z"  
}
```


JSON

JSONの例

情報の中身の各メンバーは

```
{  
  "id": "Y21zY29z...",  
  "roomId": "Y21zY29zcGF...",  
  "text": "こんにちは",  
  "created": "2017-03-06T02:56:41.512Z"  
}
```

“キーとなる文字列”

:

値

のペアになっていて

,

で区切られる

JSON

JSONの例

```
{  
  "id": "Y21zY29z...",  
  "roomId": "Y21zY29zcGF...",  
  "text": "こんにちは",  
  "created": "2017-03-06T02:56:41.512Z"  
}
```

このメッセージは、

roomId

というキー名とペアになっている値の

"Y21zY29zcGF..."

という文字列のIDが
指し示すスペースにある

JSON

JSONの例

そのメッセージの文字列は、

text

というキー名とペアになっている値の

“こんにちは”

という文字列

```
{  
  "id": "Y21zY29z...",  
  "roomId": "Y21zY29zcGF...",  
  "text": "こんにちは",  
  "created": "2017-03-06T02:56:41.512Z"  
}
```

JSON

JSONの例

特定のメンバーの
キー名は不変

```
“roomId”: “Y217Y29zcGF...”,  
“text”: “こんにちは”,  
“created”: “2017-03-06T02:56:41.512Z”  
}
```

キーに対応する
値は対象のリソースに応じて
異なる

Sparkでスペース内のメッセージの一覧を取得

Sparkのスペースの
メッセージの一覧を取得する例も
見てみよう

Spark APIを
処理するサーバ



api.ciscospark.com

Sparkでスペース内のメッセージの一覧を取得

スペース内の
メッセージの
一覧を取得したい！



Spark APIを
処理するサーバ



api.ciscospark.com

Sparkでスペース内のメッセージの一覧を取得

`https://api.ciscospark.com/v1/messages`

メッセージリソースの一覧取得なので、IDなしのパスに

GET



アプリ



`api.ciscospark.com`

Sparkでスペース内のメッセージの一覧を取得

`https://api.ciscospark.com/v1/messages?roomId=Y21zY29zcGF...&max=2`

スペースは限定する必要があるので
パラメータとして指定します



`api.ciscospark.com`

Sparkでスペース内のメッセージの一覧を取得

```
https://api.ciscospark.com/v1/messages:roomId=Y21zY29zcGF...&max=2
```

パス自体にIDが含まれる場合との
違いに注意しよう。
※ ここでは、メッセージの一覧の取得条件を
パラメータで指定して限定している



api.ciscospark.com

Sparkでスペース内のメッセージの一覧を取得

`https://api.ciscospark.com/v1/messages?roomId=Y21zY29zcGF...&max=2`

リストする最大件数も指定したいので
パラメータをさらに追加します

パラメータを複数指定する場合は

&

で区切る



`api.ciscospark.com`

Sparkでスペース内のメッセージの一覧を取得

https://api.ciscospark.com

こんな感じでSparkのAPIサーバに
リクエストを出します

GF...&max=2

```
GET /v1/messages?roomId=Y21zY29zcGF...&max=2 HTTP/1.1
Host: api.ciscospark.com
Accept: application/json
Authorization: Bearer NDM5MjYz.....
Content-Type: application/json; encoding=utf-8
```



アプリ



api.ciscospark.com

Sparkでスペース内のメッセージの一覧を取得

```
https://api.ciscospark.com/v1/messages?roomId=Y21zY29zcGF...&max=2
```

Sparkの
クラウドサーバと
連携して
メッセージの一覧
を取得



api.ciscospark.com

Sparkでスペース内のメ

Spark APIサーバは、
こんな感じのレスポンスを返す。
(JSONをかなり省略しています)

`https://api.cis`

```
HTTP/1.1 200 OK
Content-Length: 578
Content-Type: application/json; charset=utf-8
```

`zcGF...&max=2`

```
{
  "items": [
    {
      "id": "Y2lzY29z...01",
      "roomId": "Y2lzY29zcGF...",
      (一部省略...)
      "text": "こんにちは",
      (以降省略...)
    }
  ]
}
```



アプリ



`api.ciscospark.com`

JSONの配列

```
{
  "items": [
    {
      "id": "Y21zY29z...01",
      "roomId": "Y21zY29zcGF...",
      "text": "こんにちは",
      "created": "2017-03-08T02:11:28.415Z"
    },
    {
      "id": "Y21zY29z...02",
      "roomId": "Y21zY29zcGF...",
      "text": "おはよう",
      "created": "2017-03-13T06:31:46.747Z"
    }
  ]
}
```

今回のJSONはこんな感じ。
(説明に必要な部分以外省略している)

JSONの配列

```
{  
  "items": [  
    {  
      "id": "Y21zY29z...01",  
      "roomId": "Y21zY29zcGF...",  
      "text": "こんにちは",  
      "created": "2017-03-08T02:11:28.415Z"  
    },  
    {  
      "id": "Y21zY29z...02",  
      "roomId": "Y21zY29zcGF...",  
      "text": "おはよう",  
      "created": "2017-03-13T06:31:46.747Z"  
    }  
  ]  
}
```

情報の塊が複数
含まれている

JSONの配列

```
{
  "items": [
    {
      "id": "Y21zY29z...01",
      "roomId": "Y21zY29zcGF...",
      "text": "こんにちは",
      "created": "2017-03-08T02:11:28.415Z"
    },
    {
      "id": "Y21zY29z...02",
      "roomId": "Y21zY29zcGF...",
      "text": "おはよう",
      "created": "2017-03-13T06:31:46.747Z"
    }
  ]
}
```

配列(リスト)は、[]で囲んで
表現します。
配列には複数の情報の塊や
複数の値を含めることができます

JSONの配列

```
{
  "items": [
    {
      "id": "Y21zY29z...01",
      "roomId": "Y21zY29zcGF...",
      "text": "こんにちは",
      "created": "2017-03-08T02:11:28.415Z"
    },
    {
      "id": "Y21zY29z...02",
      "roomId": "Y21zY29zcGF...",
      "text": "おはよう",
      "created": "2017-03-13T06:31:46.747Z"
    }
  ]
}
```

items

配列

というキー名とペアになっている値の

に複数のメッセージ情報が
含まれている

Sparkでスペースにメッセージを投稿

Sparkのスペースに
メッセージを投稿する場合を
考えてみよう

Spark APIを
処理するサーバ



api.ciscospark.com

Sparkでスペースにメッセージを投稿

`https://api.ciscospark.com/v1/messages`

メッセージリソースの一覧取得なので、IDなしのパスに

POST



アプリ



`api.ciscospark.com`

Sparkでスペースにメッセージを投稿

```
POST /v1/messages HTTP/1.1
Host: api.ciscospark.com
Accept: application/json
Authorization: Bearer NDM5MjYz.....
Content-Type: application/json; encoding=utf-8

{
  "roomId": "Y21zY29zcGF...",
  "text": "こんばんは"
}
```

こんな感じでSparkのAPIサーバに
リクエストを出します



アプリ



api.ciscospark.com

Sparkでスペースにメッセージを投稿

```
POST /v1/messages HTTP/1.1
Host: api.ciscospark.com
Accept: application/json
Authorization: Bearer NDM5MjYz.....
Content-Type: application/json; e
```

essages

```
{
  "roomId": "Y21zY29zcGF...",
  "text": "こんばんは"
}
```

投稿するので、
投稿対象のスペースのIDや、
投稿する文字列をJSONで
リクエストの本文(Body)の中で指定



アプリ



api.ciscospark.com

Sparkでスペースにメッセージを投稿

```
POST /v1/messages HTTP/1.1
Host: api.ciscospark.com
Accept: application/json
Authorization: Bearer NDM5MjYz.....
Content-Type: application/json; e
```

essages

```
{
  "roomId": "Y21zY29zcGF...",
  "text": "こんばんは"
}
```

この時点で、
メッセージにはIDは存在しない
(まだメッセージ自体存在しない)



アプリ



api.ciscospark.com

Sparkでス

投稿者は、
認証・認可情報として指定した
トークンに対応するBotなどになります

```
POST /v1/messages
Host: api.ciscopark.com
Accept: application/json
Authorization: Bearer NDM5MjYz.....
Content-Type: application/json; encoding=utf-8

{
  "roomId": "Y21zY29zcGF...",
  "text": "こんばんは"
}
```



アプリ



api.ciscopark.com

Sparkでスペースにメッセージを投稿

<https://api.ciscospark.com/v1/messages>

Sparkの
クラウドサーバと
連携して
メッセージを投稿



api.ciscospark.com

Sparkでスペースにメッセージ

Spark APIサーバは、
こんな感じのレスポンスを返す

```
HTTP/1.1 200 OK
Content-Length: 578
Content-Type: application/json; charset=utf-8

{
  "id": "Y21zY29z...03",
  "roomId": "Y21zY29zcGF...",
  (一部省略...)
  "text": "こんばんは",
  (一部省略...)
  "created": "2017-03-08T07:05:55.981Z"
}
```



アプリ



api.ciscospark.com

Sparkでスペースにメッセージを投稿

投稿されたメッセージを表すJSON。
メッセージの取得時と同じ形式。
この時点でメッセージのIDも出来上がっている。

```
{  
  "id": "Y21zY29z...03",  
  "roomId": "Y21zY29zcGF...",  
  (一部省略...)  
  "text": "こんばんは",  
  (一部省略...)  
  "created": "2017-03-08T07:05:55.981Z"  
}
```

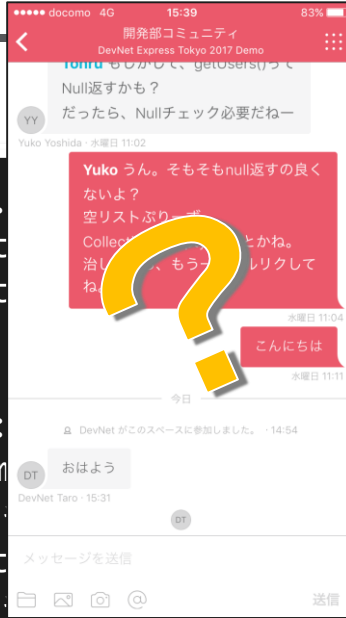


アプリ

api.ciscospark.com

Sparkでスペースを投稿

```
ht HTTP/1.  
Content  
Content  
  
{  
  "id":  
  "room  
  (一部  
  "text  
  (一部  
  "created": "2017-03-08T07:05:55.981Z"  
}
```



結果的に
Sparkクライアント上で
何が起きるかは
自分の目で
確かめてみよう！！



アプリ



api.ciscospark.com

アプリ？

Postman
というGUIベースの
Web APIの
テストクライアント

このアプリは
今日は何を使う？

Pythonの
実験用プログラムを
自分で編集して
実行します



api.ciscopark.com

REST APIのメリットの例 - プログラミング言語の視点から -

REST API

HTTPの仕組みの上で動作する

HTTPのやり取りができる言語ならREST APIが実行できる

ほとんどの言語でHTTPは利用できる

ほとんどの言語でREST APIが実行できる

REST APIのメリットの例 - プログラミング言語の視点から -

REST API

HTTPの仕組みの上で動作する

HTTPのやり取りができる言語ならREST APIが実行できる

ほとんどの言語でHTTPは利用できる

ほとんどの言語でREST APIが実行できる

もちろん
Pythonでも
できる

Python

Pythonとは

ハンズオンの
資料から抜粋

インタプリタ形式でオブジェクト指向の言語であり、
ダイナミックセマンティクスに対応した
高水準プログラミング言語です。

Pythonとは

ハンズオンの
資料から抜粋

なんじゃそりゃあ?!

インタプリタ形式でオブジェクト指向の言語であり、
ダイナミックセマンティクスに対応した
高水準プログラミング言語です。

Pythonとは

ハンズオンの
資料から抜粋

とりあえず、
さわってみて
雰囲気を
味わってみないと
何もはじまらない

インタプリタ形式でオブジェクト指向の言語であり、
ダイナミックセマンティクスに対応した
高水準プログラミング言語です。

Pythonとは

ハンズオンの
資料から抜粋

インタプリタ形式  の 

 プログラミング言語です。

Pythonとは

ハンズオンの
資料から抜粋

インタプリタ形式  の 

 プログラミング言語です。

今日はこの辺を
味わってみる

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')
```

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')
```

プログラミングとは
コンピュータに対して指示を与えていく作業です。
1つ1つの指示はごく単純

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')
```

プログラミングとは
コンピュータに対して指示を与えていく作業です。
1つ1つの指示はごく単純

プログラミング言語では、
その指示の与え方に決まり(文法)がある。
文法を守らないとエラーになる

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')
```

表示せよ

という指示

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')
```

表示する内容

‘ ’

または、

“ ”

で囲むと、文字列を表すことになる

Hello, Python!!

という文字列を

表示せよ

の指示になる

※ 実際には、「表示して改行せよ」

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')
```

これを実行すると、
コンピュータは何らかの方法で

Hello, Python!!

と表示する

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')
```

これを実行すると、
コンピュータは何らかの方法で

Hello, Python!!

と表示する

実行？
どうやって？

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')
```

例

こいつをファイルとして保存する

例えば、

hello.py

実行？
どうやって？

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')
```

例

こいつをファイルとして保存する

例えば、

hello.py

実行？
どうやって？

```
> python hello.py  
Hello, Python!!
```

コマンドラインなどから

pythonコマンド

にファイル名を渡して実行

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')
```

例

こいつをファイルとして保存する

例えば、

hello.py

指示した通りの文字列が
表示される

```
> python hello.py  
Hello, Python!!
```

コマンドラインなどから

pythonコマンド

にファイル名を渡して実行

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')
```

pythonコマンド

```
> python hello.py  
Hello, Python!!
```

環境によって、
コマンド名は異なる場合があります

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')
```

pythonコマンド

```
> python hello.py  
Hello, Python!!
```

コマンドを実行すると、
ファイルを読み込んで、
プログラミングされた指示を
1つずつ解釈しながら
実行していきます

このような特徴を持つ
プログラムの実行環境を

インタプリタ

と呼ぶ

簡単なPythonプログラムから始めてみる

```
print('Hello, Python!!')  
print('How are you?')  
print('See you again!!')
```

上から順番に指示した通りに
実行されるのが基本

変数

```
who = 'Tohru'  
print('My name is ' + who)
```

変数を使うと
値を覚えておいて、
後で参照できる

変数

```
who = 'Tohru'
```

適切な変数名を自分で
付けます

変数

= (イコール記号)

に続けて

```
who = 'Tohru'  
print('My name is ' + who)
```

代入したい値を指定している

この例では

Tohru

という文字列を代入

変数

```
who = 'Tohru'  
print('My name is ' + who)
```

代入された値を
ここで参照

この例では

Tohru

という文字列が代入
されている

変数

```
who = 'Tohru'  
print('My name is ' + who)
```

足し算の記号(演算子)を使って、
文字列を連結できる

変数

```
who = 'Tohru'  
print('My name is ' + who)
```




```
My name is Tohru
```

実行するとこうなる

変数

```
a = 2  
b = 3  
  
print(a + b)  
print(a - b)  
print(a * b)
```



では、これは？
(考えてみよう)

変数

```
a = 2  
b = 3
```

```
print(a + b)  
print(a - b)  
print(a * b)
```

‘ ’や” ”で囲まれていないので
これは文字列ではなく、
数値です

変数

```
a = 2  
b = 3
```

```
print(a + b)  
print(a - b)  
print(a * b)
```



足す(和)

引く(差)

かける(積)

を表す記号(演算子)

変数

```
a = 2  
b = 3  
  
print(a + b)  
print(a - b)  
print(a * b)
```



```
5  
-1  
6
```

実行するとこうなる

配列

```
arr = ['C', 'i', 'sco']  
who = arr[0] + arr[1] + arr[2]  
  
print('Hello, ' + who)
```

複数の値を
順番に格納できる
配列を
作ることができる

配列

[]で囲ってカンマ(,)で区切って、
その中に値を入れていきます

```
arr = ['C', 'i', 'sco']  
who = arr[0] + arr[1] + arr[2]  
print('Hello, ' + who)
```

C

i

sco

こんな感じで
順番に値が入っているイメージ

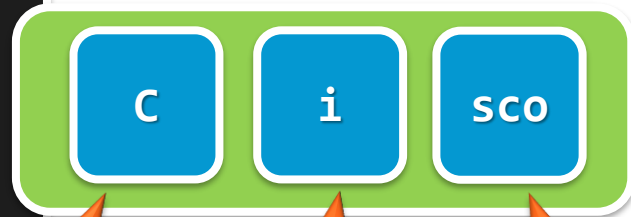
配列

```
arr = ['C', 'i', 'sco']  
who = arr[0] + arr[1] + arr[2]
```

アクセスするときは、

変数名[中身の番号]

でアクセスする



0番目

1番目

2番目

0番目から開始することに注意

配列

```
arr = ['C', 'i', 'sco']  
who = arr[0] + arr[1] + arr[2]  
print('Hello, ' + who)
```

順番に中身にアクセスして
連結しているので

配列

```
arr = ['C', 'i', 'sco']  
who = arr[0] + arr[1] + arr[2]  
  
print('Hello, ' + who)
```



```
Hello, Cisco
```

実行するところなる

配列の中身の変更

```
arr = ['C', 'i', 'sco']  
arr[0] = 'B'  
who = arr[0] + arr[1] + arr[2]  
  
print('Hello, ' + who)
```

配列の中身の変更

```
arr = ['C', 'i', 'sco']  
arr[0] = 'B'  
who = arr[0] + arr[1] + arr[2]  
  
print('Hello, ' + who)
```

配列は中身の追加、削除、変更などが
できます。

この例では、0番目を'B'に変更

配列の中身の変更

```
arr = ['C', 'i', 'sco']  
arr[0] = 'B'  
who = arr[0] + arr[1] + arr[2]  
  
print('Hello, ' + who)
```



```
Hello, Bisco
```

実行するとこうなる

タプル

```
arr = ('C', 'i', 'sco')  
arr[0] = 'B'  
who = arr[0] + arr[1] + arr[2]  
  
print('Hello, ' + who)
```

タプル

()で囲んで、カンマ(,)で区切ると
中身の変更できない配列(タプル)
を作ることができる

```
arr = ('C', 'i', 'sco')
arr[0] = 'B'
who = arr[0] + arr[1] + arr[2]

print('Hello, ' + who)
```

タプル

```
arr = ('C', 'i', 'sco')  
arr[0] = 'B'  
who = arr[0] + arr[1] + arr[2]  
  
print('Hello, ' + who)
```

変更しようとする
失敗します

```
arr[0] = 'B'  
TypeError: 'tuple' object does not support item assignment
```

タプル

```
arr = ('C', 'i', 'sco')  
who = arr[0] + arr[1] + arr[2]  
  
print('Hello, ' + who)
```



```
Hello, Cisco
```

変更している個所をなくすと
こんな感じ

ディクショナリ

```
dic = { 'name' : 'Tohru', 'email' : 'tohzono@example.com' }  
  
print(dic['name'])  
print(dic['email'])
```


ディクショナリ

{ }で囲んで、カンマ(,)で区切って
ディクショナリ
というものを作ることができる

```
dic = { 'name' : 'Tohru', 'email' : 'tohzono@example.com' }  
  
print(dic['name'])  
print(dic['email'])
```

ディクショナリ

```
dic = { 'name' : 'Tohru', 'email' : 'tohzono@example.com' }
```

```
print(dic['name'])  
print(dic['email'])
```

キーとなる値

:

キーに対応する値

のペアになっていて

,

で区切られる

JSONに
非常に似ている！！

ディクショナリ

```
dic = { 'name' : 'Tohru', 'email' : 'tohzono@example.com' }  
  
print(dic['name'])  
print(dic['email'])
```

name

Tohru

email

tohzono@e...

値にラベル付けするようなイメージ

ディクショナリ

```
dic = { 'name' : 'Tohru', 'email' : 'tohzono@example.com' }  
  
print(dic['name'])  
print(dic['email'])
```

アクセスするときは、


変数名[キーとなる値]

でアクセスする



ディクショナリ

```
dic = { 'name' : 'Tohru', 'email' : 'tohzono@example.com' }  
  
print(dic['name'])  
print(dic['email'])
```



```
Tohru  
tohzono@example.com
```

実行するところなる

分岐処理

```
a = 2

if a == 1:
    print('aは1です。')
    print('aは2ではありません。')

if a == 2:
    print('aは2です。')
    print('aは1ではありません。')
```

分岐処理

```
a = 2

if a == 1:
    print('aは1です。')
    print('aは2ではありません。')

if a == 2:
    print('aは2です。')
    print('aは1ではありません。')
```

条件に応じて、
処理を分けたい場合があります

分岐処理

```
a = 2
```

```
if a == 1:
```

```
    print('aは1です。')  
    print('aは2ではないです。')
```

```
if a == 2:
```

```
    print('aは2です。')  
    print('aは1ではないです。')
```

こんな感じで

if 条件:

という書き方をする

最後のコロン(:)を
忘れないように

分岐処理

```
a = 2
```

```
if a == 1:
```

```
    print('aは1です。')
```

```
    print('aは2ではないです。')
```

```
if a == 2:
```

```
    print('aは2です。')
```

```
    print('aは1ではないです。')
```

左辺と右辺が等しいか
どうかを確認する記号(演算子)

分岐処理

```
a = 2
```

条件が成立する場合

```
if a == 1:
```

```
    print('aは1です。')  
    print('aは2ではないです。')
```

この処理が実行される

```
if a == 2:
```

```
    print('aは2です。')  
    print('aは1ではないです。')
```

分岐処理

```
a = 2

if a == 1:
    print('aは1です。')
    print('aは2ではないです。')
```

このインデントはかなり重要

インデントによって、
条件成立時に実行される
範囲が決まる。
実行したい処理は、
同じインデントを使って書く

分岐処理

```
a = 2  
  
if a == 1:  
    print('aは1です。')  
    print('aは2ではないです。')
```

基本的には、
半角スペース4つを入れよう!!

分岐処理

```
a = 2

if a == 1:
    print('aは1です。')
    print('aは2ではないです。')

if a == 2:
    print('aは2です。')
    print('aは1ではないです。')
```



実行するところなる

```
aは2です。
aは1ではないです。
```

繰り返し処理

```
for i in range(3):  
    print('Hello, Python!!')  
    print('See you again!!')
```

繰り返し処理

```
for i in range(3):  
    print('Hello, Python!!')  
    print('See you again!!')
```

同じ処理を繰り返したい
ことがあります

繰り返し処理

```
for i in range(3):  
    print('Hello, Python!!')  
    print('See you again!!')
```

今日は、詳細な理解はあきらめて、
形でとらえる

繰り返し処理

```
for i in range(3):  
    print('Hello, Python!!')  
    print('See you again!!')
```

こんな感じで

```
for i in range(繰り返したい回数):
```

という書き方をする

最後のコロン(:)を
忘れないように


繰り返し処理

```
for i in range(3):  
    print('Hello, Python!!')  
    print('See you again!!')
```

インデントされている
この範囲が繰り返される

繰り返し処理

```
for i in range(3):  
    print('Hello, Python!!')  
    print('See you again!!')
```



```
Hello, Python!!  
See you again!!  
Hello, Python!!  
See you again!!  
Hello, Python!!  
See you again!!
```

実行するようになる

繰り返し処理

```
arr = ['C', 'i', 'sco']  
  
for value in arr:  
    print(value)
```

配列などを
取り出しながら
繰り返すことも
できる

こちらが正しく理解できれば、
for i in range(繰り返したい回数):
も適切に理解できるはずだが、
初めてプログラミングを学ぶ場合は、
難しいと思うので、とりあえずは、
こんな感じという捉え方でよいと思います

繰り返し処理

```
arr = ['C', 'i', 'sco']
```

```
for value in arr:  
    print(value)
```

配列の中身を取り出しながら、
繰り返します。

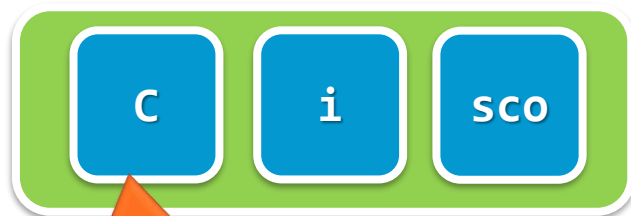
繰り返し処理

```
arr = ['C', 'i', 'sco']  
for value in arr:  
    print(value)
```

取り出した値は、
指定した変数に代入される

繰り返し処理

```
arr = ['C', 'i', 'sco']  
for value in arr:  
    print(value)
```

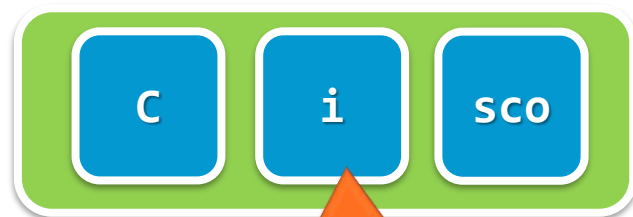


繰り返しの
1回目

これがvalue変数に代入されて、
print(value)が実行される

繰り返し処理

```
arr = ['C', 'i', 'sco']  
for value in arr:  
    print(value)
```



繰り返しの
2回目

これがvalue変数に代入されて、
print(value)が実行される

繰り返し処理

```
arr = ['C', 'i', 'sco']  
for value in arr:  
    print(value)
```



繰り返しの
3回目

これがvalue変数に代入されて、
print(value)が実行される

繰り返し処理

```
arr = ['C', 'i', 'sco']  
  
for value in arr:  
    print(value)
```



```
C  
i  
SCO
```

実行するようになる

関数

```
def add(a, b):  
    answer = a + b  
    return answer
```

```
ans1 = add(1, 2)  
ans2 = add(7, 8)
```

```
print(ans1)  
print(ans2)
```

関数

```
def add(a, b):  
    answer = a + b  
    return answer
```

```
ans1 = add(1, 2)  
ans2 = add(7, 8)
```

```
print(ans1)  
print(ans2)
```

よく使う処理は
関数というものにしておくと便利

関数

```
def add(a, b):  
    answer = a + b  
    return answer
```

```
ans1 = add(1, 2)  
ans2 = add(7, 8)
```

```
print(ans1)  
print(ans2)
```

これが関数の定義

def 関数名(関数に渡す引数のリスト):

の後のインデントで区切られた範囲が、
関数の定義範囲

関数

```
def add(a, b):  
    answer = a + b  
    return answer
```

```
ans1 = add(1, 2)  
ans2 = add(7, 8)
```

```
print(ans1)  
print(ans2)
```

定義されているだけなので、
この時点では実行されない

関数

```
def add(a, b):  
    answer = a + b  
    return answer
```

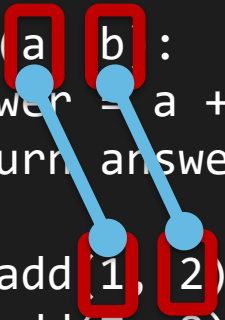
```
ans1 = add(1, 2)  
ans2 = add(7, 8)
```

```
print(ans1)  
print(ans2)
```

定義された名前で
呼び出しています

関数

```
def add(a, b):  
    answer = a + b  
    return answer  
  
ans1 = add(1, 2)  
ans2 = add(7, 8)  
  
print(ans1)  
print(ans2)
```



対応する順番の引数に
値が代入された後に
関数が実行される

関数

```
def add(a, b):  
    answer = a + b  
    return answer
```

```
ans1 = add(1, 2)  
ans2 = add(7, 8)
```

```
print(ans1)  
print(ans2)
```

a + b
を計算して、
answer変数に代入

関数

```
def add(a, b):  
    answer = a + b  
    return answer
```

```
ans1 = add(1, 2)  
ans2 = add(7, 8)
```

```
print(ans1)  
print(ans2)
```

answer変数の値を
結果として返す(return)
という意味

結果は変数に代入して
受け取れます

関数

```
def add(a, b):  
    answer = a + b  
    return answer
```

```
ans1 = add(1, 2)  
ans2 = add(7, 8)
```

```
print(ans1)  
print(ans2)
```

渡す値を変えて、
同じように何度でも呼び出せる

関数

```
def add(a, b):  
    answer = a + b  
    return answer
```

```
ans1 = add(1, 2)  
ans2 = add(7, 8)
```

```
print(ans1)  
print(ans2)
```



実行するところなる

```
3  
15
```

便利なモジュールの利用

モジュール

今日の段階では、
いろんな関数が含まれていて
プログラミングの時に
便利に使えるもの程度の理解で良いと思います

頭の良い誰かの作ったモジュールを活用すると、
本来複雑な処理がとんでもなく簡単に！

便利なモジュールの利用

`https://server.example.com/contents`

サーバから
リソース取得する、
この処理、何行で書けますか？

`GET /contents HTTP/1.1`



アプリ



`server.example.com`

便利なモジュールの利用

```
import requests  
  
res = requests.get('https://server.example.com/contents')  
  
print(res.text)
```

成功したかのチェックなどを
しなければ3行

(本来は必ずチェックすべきだが)

便利なモジュールの利用

```
import requests
```

誰かが作ったモジュールを取り込む

import 取り込むモジュール名

で取り込みます

```
res = requests.get('http://www.python.org')
```

```
print(res.text)
```


便利なモジュールの利用

```
import requests  
res = requests.get('https://server.example.com/contents')  
print(res.text)
```

取り込んだモジュールの関数を呼び出す

モジュール名.関数名(引数のリスト)

のように呼び出します

便利なモジュールの利用

```
import requests  
res = requests.get('https://server.example.com/contents')
```

結果は変数に代入して
受け取れます

便利なモジュールの利用

```
import requests  
  
res = requests.get('https://server.example.com/contents')  
  
print(res.text)
```

結果の中身にアクセスするために、
こんな感じで、ドット(.)付きでアクセスする場合があります

res.status_code

HTTPステータスコード

res.text

レスポンスのテキスト

などなど、
モジュール側の
実装による

便利なモジュールの利用

```
import requests  
  
res = requests.get('https://server  
print(res.text)
```

今日の範囲外である、
“クラス”というものを
理解していないと
正確な理解は難しい。
本日の範囲では、
こんなものかで受け入れる

結果の中身にアクセスするために、
こんな感じで、ドット(.)付きでアクセスする場合があります

res.status_code

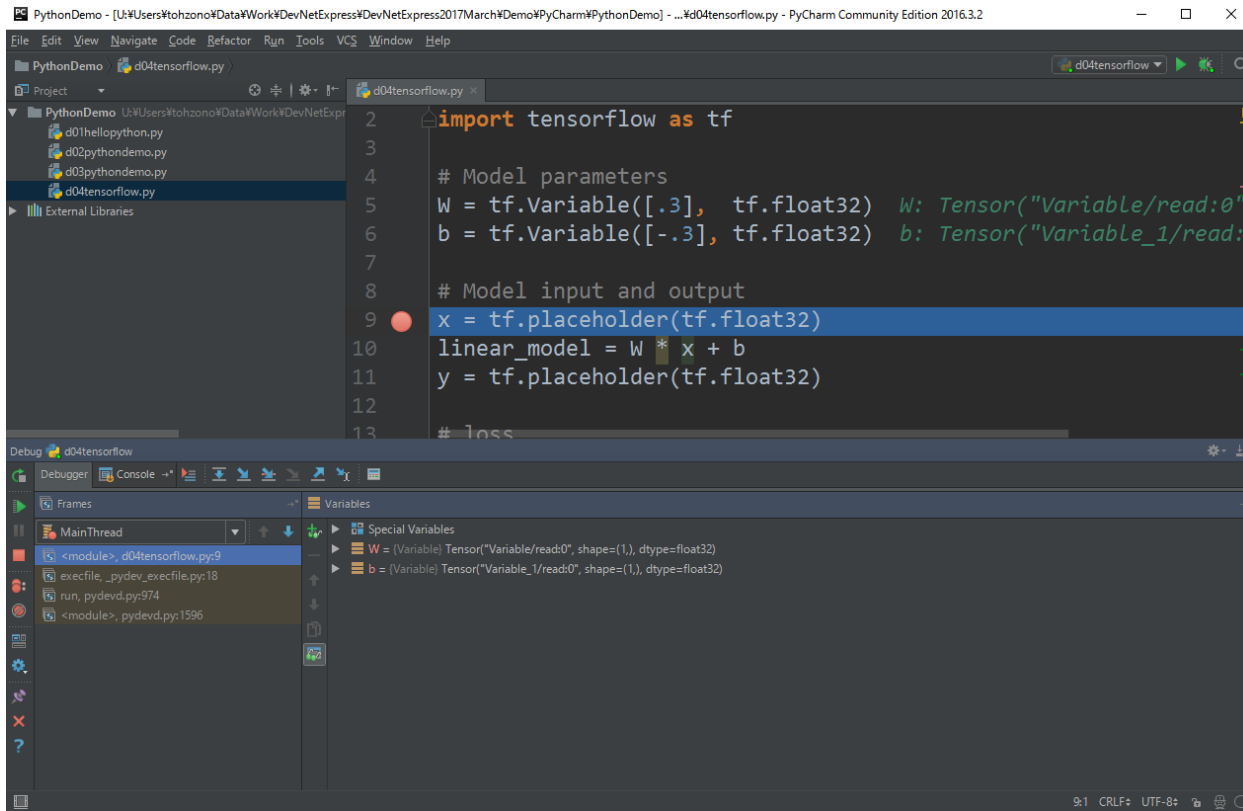
HTTPステータスコード

res.text

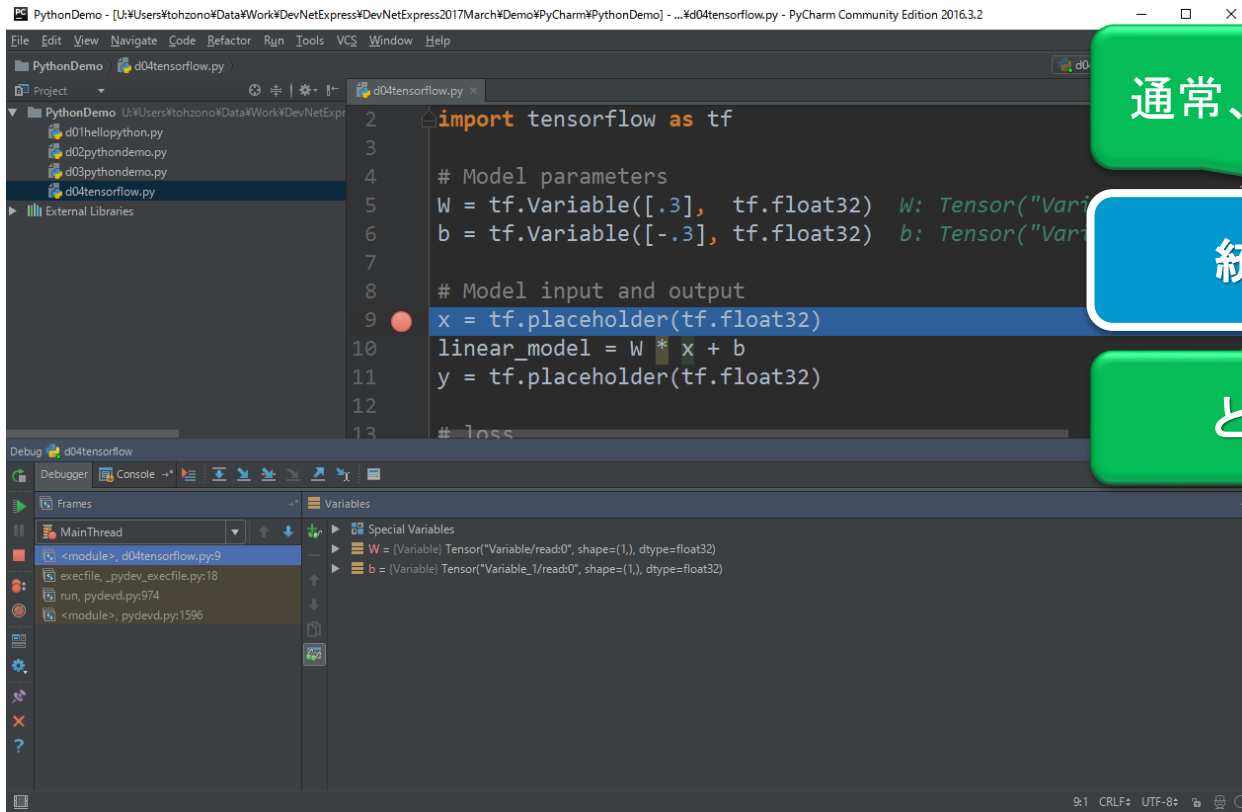
レスポンスのテキスト

などなど、
モジュール側の
実装による

慣れてきたらIDEを活用しよう



慣れてきたらIDEを活用しよう



通常、プログラミングなどには

統合開発環境 (IDE)

というツール群を使う

慣れてきたらIDEを活用しよう

統合開発環境 (IDE)

ソースコードを書く際に
次に書く候補を教えたり、
間違いを指摘してくれたり、
様々なコード作成支援機能

デバッグの支援機能

単体試験の支援機能

Git / GitHubなどの
バージョン管理システムとの連携

などなど、
アプリ開発を支援する
強力な機能が提供される

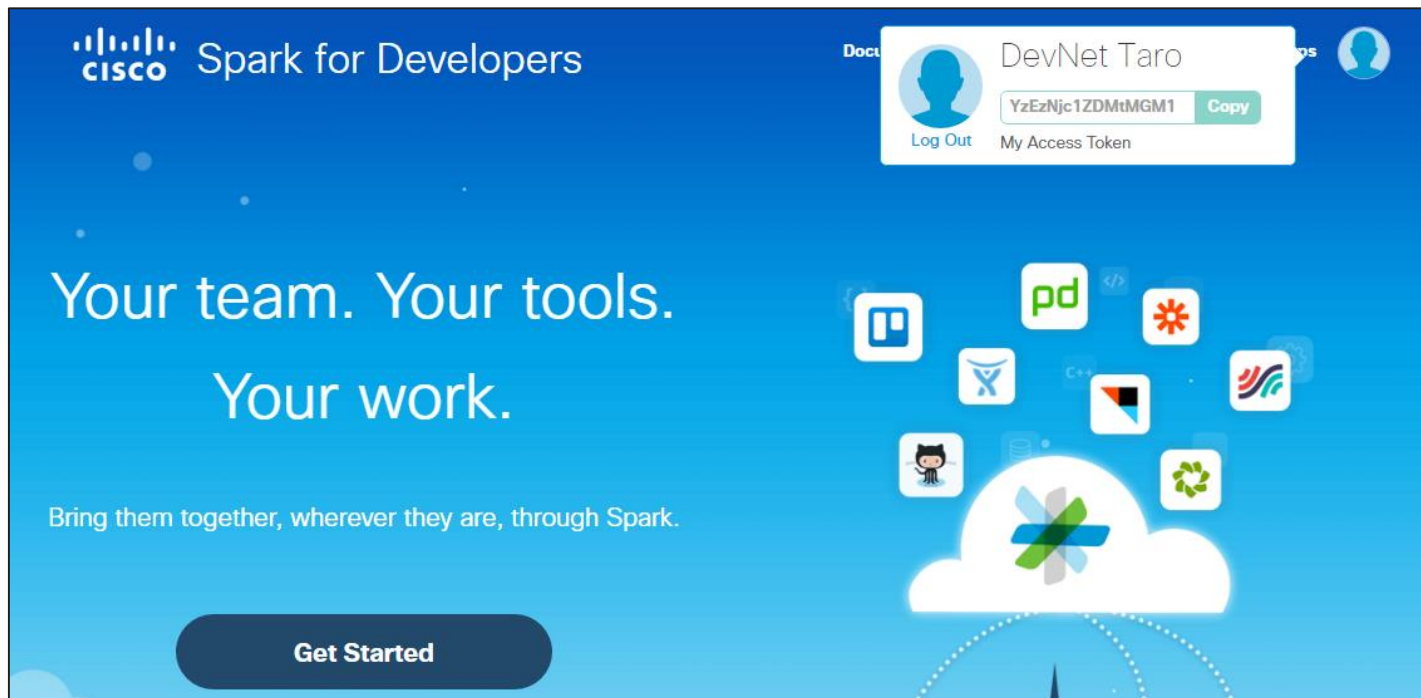
慣れてきたらIDEを活用しよう

統合開発環境(IDE)

PyCharm
Eclipse + PyDev
Visual Studio
などPythonをサポートするIDEは多く存在する

補足

開発者用トークンの注意



The screenshot displays the Cisco Spark for Developers user interface. At the top left is the Cisco logo and the text "Spark for Developers". In the top right corner, a user profile dropdown menu is open, showing a profile picture, the name "DevNet Taro", and a "Log Out" button. Below the name, the text "My Access Token" is displayed above a text box containing the token "YzEzNjc1ZDMtMGM1" and a "Copy" button. The main content area features the text "Your team. Your tools. Your work." and "Bring them together, wherever they are, through Spark." Below this is a "Get Started" button. On the right side, there is a cluster of various application icons including Slack, Jira, GitHub, and others, all connected to a central cloud icon with a plus sign.

開発者用トークンの注意




The screenshot shows the Cisco Spark for Developers interface. At the top left is the Cisco logo and the text "Spark for Developers". Below this is the slogan "Your team. Your tools. Your work." and the tagline "Bring them together, wherever they are, through Spark." with a "Get Started" button. On the right side, there is a user profile for "DevNet Taro" with a "Log Out" button. Below the name, the access token "YzEzNjc1ZDMtMGM1" is displayed with a "Copy" button. A red box highlights the user profile and the access token. A red octagonal warning sign with a white 'X' is overlaid on the bottom right of the screenshot.


ここで取得できるトークンは
開発者トークンです

開発、実験時など
限られた用途でのみ利用可能です。
運用環境などでは
絶対に利用しないでください

アプリケーション用のトークン



Spark for Developers

[Documentation](#) [Blog](#) [Support](#) [Haus](#) [Fund](#) [My Apps](#) 


GUIDES

- [Getting Started](#)
- [Quick Reference](#)
- [Pagination](#)
- [Message Attachments](#)
- [Formatting Messages](#)
- [Webhooks Explained](#)
- [Admin API](#)

APPS

- [Integrations \(OAuth\)](#)
- [Bots](#)
- [Depot](#)

New App




Integration

Create an integration to customize how teams on Spark interact with other applications.

[Learn More](#)

[Create an Integration](#)



Bot

Create rich messaging experiences on Spark with your intelligent Bots.

[Learn More](#)

[Create a Bot](#)

アプリケーション用のトークン

My Apps

Documentation Blog Support Haus Fun

GUIDES

- Getting Started
- Quick Reference
- Pagination
- Message Attachments
- Formatting Messages
- Webhooks Explained
- Admin API

APPS

- Integrations (OAuth)
- Bots
- Depot

New App

Integration

Create an integration to customize how teams on Spark interact with other applications.

[Learn More](#)

[Create an Integration](#)

Bot

Create rich messaging experiences on Spark with your intelligent Bots.

[Learn More](#)

[Create a Bot](#)

My Appsメニューからアプリ用のアカウントが作成できます

アプリケーション用のトークン

The screenshot shows the Cisco Spark for Developers website. The header is blue with the Cisco logo and the text 'Spark for Developers'. On the right side of the header, there are links for 'Documentation', 'Blog', and 'Support'. The main content area is divided into two columns. The left column contains a 'GUIDES' section with links for 'Getting Started', 'Quick Reference', 'Pagination', 'Message Attachments', 'Formatting Messages', 'Webhooks Explained', and 'Admin API'. Below that is an 'APPS' section with links for 'Integrations (OAuth)', 'Bots', and 'Depot'. The right column is titled 'New App' and contains two main options: 'Integration' and 'Bot'. The 'Integration' option features an icon of two interlocking rings, the text 'Integration', a description 'Create an integration to customize how teams on Spark interact with other applications.', a 'Learn More' link, and a 'Create an Integration' button. The 'Bot' option features a robot icon, the text 'Bot', a description 'Create rich messaging experiences on Spark with your intelligent Bots.', a 'Learn More' link, and a 'Create a Bot' button. A red rounded rectangle highlights the 'Bot' option.

Botアカウントの
作成は簡単!!
Bot用の
トークンも取得できます

アプリケーション用のトークン

The screenshot shows the Cisco Spark for Developers website. The header includes the Cisco logo and the text 'Spark for Developers', along with links for 'Documentation', 'Blog', and 'Support'. A left sidebar lists 'GUIDES' (Getting Started, Quick Reference, Pagination, Message Attachments, Formatting Messages, Webhooks Explained, Admin API) and 'APPS' (Integrations (OAuth), Bots, Depot). The main content area is titled 'New App' and features two options: 'Integration' (with a link icon) and 'Bot' (with a robot icon). The 'Bot' option is highlighted with a red rounded rectangle. Below the 'Integration' option is a 'Create an Integration' button, and below the 'Bot' option is a 'Create a Bot' button.

ハンズオンを
Botトークンで
やってみたい人は要相談

Botアカウント

The screenshot shows the Cisco Spark for Developers website. The top navigation bar includes the Cisco logo, the text 'Spark for Developers', and links for 'Documentation', 'Blog', 'Support', 'Haus', 'Fund', and 'My Apps'. A user profile icon is visible in the top right. The main content area features a 'Bot' section with a robot icon, the title 'Bot', and the text 'Create rich messaging experiences on Spark with your intelligent Bots.' Below this is a 'Learn More' link and a 'Create a Bot' button. A blue callout bubble points to the 'Bot' section with the text 'Botアカウント'. An orange callout bubble contains the text: 'プライバシー保護などの目的のため、スペース内で該当Botに明示的に@mentionされたメッセージのみ取得できます'.

トークンを保護しよう

トークンは流出しないよう適切に保護しましょう

トークンが保存される設定ファイルへの
アクセス権限を適切に
(Linuxの場合、パーミッションを“600”にするなど)

例

トークンは暗号化して保存を推奨

verify = False?

```
res = requests.post('https://example.com/', ... , verify = False)
```

verify = False?

```
res = requests.post('https://example.com/', ... , verify = False)
```

本来は、このような
コードを書いてはいけません

verify = False?

```
res = requests.post('https://example.com/', ... , verify = False)
```

本来は、このような
コードを書いてはいけません

正しく理解した上で、
意図的にやっている場合にのみ
(したくないけど、)許容される

verify = False?

```
res = requests.post('https://example.com/', ... , verify = False)
```

TLSの証明書検証などを
無効にするって意味？

verify = False?

```
res = requests.post('https://example.com/', ... , verify = False)
```

TLSの証明書などを
無効にする意味？

通信経路上で
認証情報やメッセージ本文中の
重要な情報などを含めて
盗み見られても構わないという宣言です

verify = False?

```
res = requests.post('https://example.com/', ... , verify = False)
```

認証関連の情報を
盗られると
大抵のシステムは
大損害を受けます

TLSの証明書などを
無効にする意味？

通信経路上で
認証情報やメッセージ本文中の
重要な情報などを含めて
盗み見られても構わないという宣言です

API利用者の責任

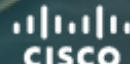
認証関連の情報を
盗られると
大抵のシステムは
大損害を受けます

APIを使えるということは、
システム側から、
それなりに信頼されて、
機能や情報を
受け取っているので、
使う側の責任も
理解はしておく
必要がある


```
name = input('Input Your Name: ')
print('Thanks, {0}!!'.format(name))
```

DevNet Express [Tokyo]

March 15, 2017 • Tokyo, Japan

 **DevNet**
developer.cisco.com