# <u>Java で XMPP #1</u>

# Smack の使い方(1): EchoBot

XGMTK では遠隔セッション実現のため XMPP という通信プロトコルを利用します。本稿では Java 言語 による XMPP プログラミングを扱った日本語ドキュメントが少ないことから、サンプル・コードを挙げて使 い方を簡単に説明してみたいと思います。

今回は第一回として Java 言語で XMPP のクライアントを作成するためのライブラリである Smack につい てサンプル・コードを挙げて使い方を簡単に説明してみたいと思います。サンプルとしては非常に簡単な自動 応答プログラム(ボットと呼ばれる)の例として EchoBot というプログラムを作成します。

ここで扱うサンプル・プログラムのコードは<u>http://xgmtk.org/</u>からダウンロードできます。開発言語としては Java (Ver.6)<sup>1</sup>、XMPPライブラリとしてはSmack<sup>2</sup>、XMPPサーバとしてはOpenfire<sup>3</sup>を想定しています。動 作確認はWindows 7 環境、ブラウザはFirefox 5 で行いました。

<sup>&</sup>lt;sup>1</sup> <u>http://java.sun.com/javase/ja/6/download.html</u>

<sup>&</sup>lt;sup>2</sup> <u>http://www.igniterealtime.org/projects/smack/index.jsp</u>

 $<sup>\ \ \, {}^3 \ \</sup>underline{http://www.igniterealtime.org/projects/openfire/index.jsp}$ 

# 目次

Ja	va で XMPP #1	1
1	対象読者	3
2	環境設定	3
	Openfire	3
	Smack	8
3	EchoBot	8
	3-1 EchoBot の動作	8
	3・2 EchoBot のプログラム1	1
	プログラムの構成1	1
	main()メソッド1	3
	プロパティの読み込み1	5
	EchoBot コンストラクタ1	7
	新規チャットの開設	0
	プレゼンス情報の更新2	1
	時報の発信2	4
	EchoBot クラスの全ソース2	6
4	次回予告	0

# 1<u>対象読者</u>

本稿は Java のプログラミングを一通り取得し、開発環境の設定が一通りできる人を対象とします。開発に 利用した Java の SDK は Version 6 です。

# 2環境設定

#### Openfire

まず Openfire サーバをインストールし、ユーザ・アカウントを 2 つ以上作成します。一つは通常のログイン用でもう一つは EchoBot 用です。Openfire のインストールでは、小規模テスト環境と言うことで外部 DB ではなく Openfire サーバに組み込まれている DB を利用しています。設定は Web ブラウザから行うことができます。

実際に画面を見てみましょう、設定が全て終えた後、サーバを再起動してみます。Windows 環境の場合 openfire.exe というプログラムからサーバの起動、再起動、終了、管理画面(Web ブラウザが立ちあがります) ができます。「Launch Admin」ボタンで管理画面を立ち上げると以下の様な画面が表示されます:

W Openfire Admin Console - Mozilla Firefox				A Designation	-	×
ファイル(E) 編集(E) 表示(V) 履歴(S) ブックマーク(B) い	ノール(工) ヘルプ(日	1)	1 -			
🗋 🔔 🏨 🌆 🖾 💆 🎄 👩 Openfire Admin Console	× +		1	a series of the	ALC: N	- 2
	ogin.jsp?url=962F	index.jsp		☆ • @ 🛃 • Google	۹ ۹	
🤟 WEBラジオ   「俺の 🧧 よく見るページ 🧐 はじめよう	INIQLO CALE	NDAR 🗋 goo	検索ブックマーク	👩 国語辞典 英和辞典 和		39
6	openfire <sup>-</sup>	Admini	stration Cons	ale		
· · · · · · · · · · · · · · · · · · ·	opennie	AMITTI	structori e oris.			
	admin		Login			
	username	password				
			Openfre, Version 37,0			

#### 図 1: Openfire ログイン画面

サーバのインストール時に設定した admin アカウントについてパスワードを入力してログインすると以下の様な画面が表示されます:

Openfire Admin Console	: Server Informatio	n - Mozil	lla Firefox	States and	A DESCRIPTION OF	
アイル(E) 編集(E) 表示(	(⊻) 履歴(5) ブック	マーク(目)	) ツール(I) ヘル	7(日)	1000	
🗅 🛕 🛦 🖾 🔟	🔔 👩 Openfire Ad	min Cons	sole: × +		And Street, St.	
	http://127.0	0.0.1:909	90/index.jsp		🗄 • 😋 🛃 • Google	P 🏫 🛢
WEBラジオ L 「優の」	■ よく見るページ ●	a LELCAD		ALENDAR D goo 検索ブック	マーク (7) 国語辞典 革和辞典	和
		140000	and the ounders of	interior in Ann points a s	A NUL (D) INDEAL ACTING ACTING	
openfire <sup>-</sup>					L	Openfire 3.7.0 ogged in as admin - Logout
Server Users/Groups	Sessions Group	Chat	Plugins			
Server Manager	Settings   Media Ser	rices				
Server Information	Conver Infor	mation				
System Properties	Server infor	mauon				
Language and Time	Barbara solar and fi	and manipulate land	and the second being one	ed and island among about Operation		
Clustering	Delow you will in	na server in	ormation, pures being us	eu anu latest news abbut Openine.		
Cache Summary	Server Prope	erties			Ignite Realtime News	5
Database	Server Up	time: 5 min	utes - started Aug 11, 2	011 4:31:27 PM	ignite Realtine News	
Logs	Ver Server Direc	sion: Open	ITE 3.7.0 CXMPPionenfine 3.7.0		The Ignite Realtime feed is currently un	navailable.
Email Settings	Server N	ame: ostrici	b		The Ignite Realtime feed is currently u	navailable.
Security Audit Viewer						
	Environment	t.				
	Java Ver	sion: 1.6.0	26 Sun Microsystems In	ic Java HotSpot(TM) Client VM		
	Appse	rver: jetty/7	0.2-SNAPSHOT			
	OS / Hard	ware: Windo	ows 7 / x86			
	Locale / Times	tone: en/J	apan Standard Time (9 0	GMT)		
	Java Me	mory L	11.82 MB	of 247.50 MB (4.8%) used		
	Server Port	5				
	Interface	Port	Туре	Description		
	All addresses	5222	Client to Server	The standard port for clients be encrypted. You can upda	to connect to the server. Connections te the <u>security settings</u> for this port.	may or may not
	All addresses	5223	Client to Server	The port used for clients to o old SSL method is not an XI future. You can update the s	onnect to the server using the old SSI IPP standard method and will be depr ocurity settings for this port.	. method. The recated in the
	All	9090	Admin Console	The port used for unsecured	Admin Console access	

図 2: Openfire 初期画面

ここでデータベース設定を見ることができます。左の Database という項目をクリックすると以下の様な画面が表示されます:

	http://127.0.0.1.9090/server-	dh isn	A . a Google	0	
			and the stored to		 -
Server Users/Groups	Sessions Group Chat Plugins	IQLO CALENDAK 🗋 goo 検索クックマーク 医	) 国团研典 央和研典 和…		1
Conner Manager	Settions Media Semicar				
Server runager Server:	Petungs Previo Services				7
Server Information	Database Properties				
Language and Time					
Clustering	Below is a list of properties for your databa	ise and the JDBC driver.			
Cache Summary	Database Connection Info				
Database	Database and Version:	HSQL Database Engine 1.8.0			
Logs	JDBC Driver:	HSQL Database Engine Driver			
Email Settings	JDBC Driver Version:	1.8.0			
Security Audit Viewer	DB Connection URL:	idbc:hsaldb:C:\usr\XMPP\openfire 3 7 0\embedd	ded-db\openfire		
	DB User:	SA			
	Transaction Support	Yes			
	Transaction Isolation Level	TRANSACTION READ COMMITTED			
	Supports multiple connections open at once:	Yes			
	In read-only mode:	No			

図 3: Openfire データベース設定の確認

コネクション URL を見ると組み込み DB を利用していることが確認できます。インストール時に組み込み (Embedded) DB を選択するとパフォーマンスがよくないと警告が出ますが、プログラミングの練習用に自

分のマシンにサーバをインストールして利用する程度なら組み込み DB で問題はないでしょう。

実際にクライアント・ソフトウェアから接続するにはまずアカウントを作る必要があります。上に並んだ項 目から「Users/Groups」を選んで、右の項目から「Create New User」を選ぶと以下の様な新規ユーザ作成 画面が現れます:

() Openfire Admin Console	e: Create User - Mozilla Firefox	and the second se	_ <b>_</b> ×
ファイル(E) 編集(E) 表示(	(V) 履歴(S) ブックマーク(B) ツール(I) ヘルブ(H)		
🗋 🕭 🎄 🏧 🔟	▲ Openfire Admin Console: × +		- 11
+ + • • • • • •	http://127.0.0.1:9090/user-create.jsp	🏫 - 🖉 🚷 - Google	P 🛖 .
🍕 WEBラジオ   「俺の…	🧧 よく見るページ 🥹 はじめよう 🎛 UNIQLO CALENDAR 🗋 goo 検索	ブックマーク 👩 国語辞典 英和辞典 和	,
openfire <sup>.</sup>			Openfire 3.7.0 Logged in as admin - Logged
Server Users/Groups	Sessions Group Chat Plugins		
Users Groups			
User Summary Create New User	Create User		
User Search Advanced User Search	Use the form below to create a new user.		
	Create New User		
	Username: *		
	Password: *		
	Confirm Password, *		
	Is Administrator? (Grants admin access to Openfire)		
	Create User Create & Create Another Cancel		
	* Required fields		

図 4: Openfire 新規ユーザ作成

ここで適当なユーザ ID とパスワードを設定してユーザを作成します。一通りユーザを作成したら作成した ユーザの連絡先リスト (ロースター)を設定します。ロースターの設定はメッセンジャー・クライアントから 行うこともできますが、ロースターはサーバに保存されていますのでサーバの管理画面からも編集できます。 プログラミングの際には作成中のプログラムがロースターの内容の操作を誤って壊すこともあり得るので、サ ーバ上での編集に慣れておきましょう。一通りアカウントを作成して「Users/Groups」の「User Summary」 画面を見ると作成したユーザの一覧が見られます:

	A 6 http://	127.0.0.1:9090/use	er-summary, isp	ė.	- a Socale		0 0	
WEBラジオ L 「優の」		-ジ 👜 はじめよう 💷	UNIOLO CALENDAR D goo #	#素ブックマーク (図)目	115.25曲 芯和22曲 和			Ĩ
openfire Server Users/Groups	Sessions (	Group Chat Plugin	s		Logg	ped in as i	Opentire 3 admin - <u>Log</u> a	0.1 Inter
Users Groups								
User Summary								
Create New User	User Su	mmary						
Create New User User Search Advanced User Search	User Sul Total User	mmary	e Users per page: 100 💌					
Create New User User Search Advanced User Search	User Sul Total User Onlin	mmary rs: 5 Sorted by Usernam ne: Username	e Users per page: 100 💌 Name	Created	Last Logout	Edit	Delete	
Create New User User Search Advanced User Search	Total User Onlin 1 &	mmary rs: 5 Sorted by Usernam ne Username admin 🚖	e – Users per page: 100 🔹 Name Administrator	Created May 3, 2011	Last Logout	Edit 2	Delete	
Create New User User Search Advanced User Search	Total User Total User 1 & 2 &	mmary rs: 5 Sorted by Usernam ne Username admin 🚖 emilik	e – Users per page: 100 🔹 Name Administrator Emiri Kimidori	Created May 3, 2011 Jun 4, 2011	Last Logout 11 days, 23 hours, 14 minutes	Edit 17 17	Delete O	
Create New User User Search Advanced User Search	User Sul Total User 1 & 2 & 3 &	mmary rs: 5 - Sorted by Usemann e Username admin 📬 emit (k kando	e – Users per page: 100 🔹 Name Administrator Emiri Kimidori Takayuki Kando	Created May 3, 2011 Jun 4, 2011 May 3, 2011	Last Logout 11 days, 23 hours, 14 minutes 10 days, 5 hours, 44 minutes	Edit 17 17 17	Delete O	
Cruate New User User Search Advanced User Search	Total User Sul Total User 1 & 2 & 3 & 4 &	mmary s: 5 Sorted by Usernam Username admin * emiil.k kando yyoko.a	e – Users per page: 100 💌 Name Administrator Emiri Kimidori Takayuki Kando Ryoko Asakura	Created May 3, 2011 Jun 4, 2011 May 3, 2011 Jun 4, 2011	Last Logout 11 days, 23 hours, 14 minutes 10 days, 5 hours, 44 minutes 19 days, 10 hours, 40 minutes	Edit 12 12 12 12 12 12 12 12 12 12 12 12 12	Delete O O O	

図 5: Openfire ユーザー覧

一覧の「Username」の列にあるユーザ名をクリックすると左側の詳細メニューに項目が増えます。「Roster」 を選ぶと各ユーザの連絡先リスト(ロースター)を編集できます。ここでは emiri.k の連絡先を編集します。 ロースターへの追加元来承認制なので一方のユーザがロースターに追加しただけでは終わりませんので、各ユ ーザについて右側の Edit 列のアイコンをクリックして subscription を both(お互いに承認済みの意味)にし ておきましょう。一通り編集してロースターに他のユーザを追加して subscription も変更し終わった状態が下 の画面です:

	Openfire Admin Console: .					
	bttp://127.0.0.1:0000/usi	ar-rostar isn2usarnama=amiri k	Å.	a Coorde		
	0 map.//127.0.0.1.9090/da	a roster.jsprosename-emm.k	96.	C C C C C C C C C C C C C C C C C C C		( m
WEBラシオ   1歳の	👌 よく見るペーシ 🔮 はしめよう 🚻	UNIQLO CALENDAR [] goo 検索ノッ	/クマーク (1) 国	總辞典 央和辞典 和		
anonfire					Loosed in a	Openf
opentire					colling to a	- purchase
erver Users/Groups	Sessions Group Chat Plugin					
sers Groups						
ser Summary	User Roster					
Iser Options						
User Properties			2010/03/07/07/07/07	dinad links to an according of		
and the second	Below is the list of roster items for us	ter emiri.k. Shared groups are represented in the t	Groups column as unde	LIDIO A REINS OF ALL RESOLUTION	group edit screen.	Roster
Roster	Below is the list of roster items for us items provided by shared groups ma	ser emirit.k. Shared groups are represented in the ay not be deleted via this interface.	Groups column as unde	nineu initis eo an associareu (	group edit screen.	Roster
Roster Password	Below is the list of roster items for ur items provided by shared groups ma Total items: 3 – Sorted by JID	ser emini.k. Shared groups are represented in the iy not be deleted via this interface.	Groups column as unde		group edit screen.	Roster
Roster Password Lock Out	Below is the list of roster items for ur items provided by shared groups ma Total Items: 3 – Sorted by JID items per page: 15 👻 – Show:	ser emirit.k. Shared groups are represented in the involve deleted via this interface.	Groups column as unde	ninga alika ta an academina i	group edit screen. Add Ne	Roster witem <b>(</b>
Roster Password Lock Out Delete User	Below is the list of roster items for ut items provided by shared groups ma Total items: 3 – Sorted by JID items per page: 15 • - Show: 31D	ore emit.A. Shared groups are represented in the ey not be deleted via this interface. All roster items  Nickname	Groups column as unde Groups	Subscription	group edit screen Add Ne Edit	w item ( Delete
Roster Password Lock Out Delefe User reate New User	Below is the kill of froster items for unit items provided by shared groups ma Total items: 3 – Sorted by JID Items per page: 15 • - Show: JID 1 kando@costich	ere emirit.k. Shared groups are represented in the i ey not be deleted via this interface. All roster items • Nickname Takayuki Kando	Groups column as unde Groups Friends	Subscription both	group edit screen. Add Ne Edit	w item ( Delete
Roster Password Lock Out Delete User reate New User Iser Search	Below is the kild of roster items for u tems provided by shared groups ma Total items: 3 – Sorted by JID Items per page: 15 👻 – Show JID 1 kandod@postrich 2 ryoko.e@postrich	or emit.k. Shared groups are represented in the i ty not be deleted via this interface. All roster items	Groups column as unde Groups Friends None	Subscription both both	Add Ne Edet	w item ( Delete
Roster Password Lock Out Delete User reate New User Iser Search dvanced User Search	Betow is the kild of roster items for u tems provided by shared groups ma Total items: 3 – Sorted by JID Items per page: 15 👻 – Show: JID 1 kanda@ostrich 2 rysko.a@ostrich 3 yyki.n@ostrich	or emil.k. Shared groups are represented in the i y not be deleted via this interface. All rooter items Nickname Takayuki Kando Ryoko Asakura Yuki Nagato	Groups column as unde Groups Friends None None	Subscription both both	Add Ne Edit 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	w Item ( Delete 0 0 0
Roster Password Lock Out Dolete User reate New User Iser Search dvancod User Search	Betow is the kill of roster items for up tems provided by shared groups ma Total items: 3 – Sorted by JID Items per page: 15 – – Show: JID 1 kando@ostrich 2 ryoko.a@ostrich 3 yukin@iostrich	ore emit.k. Shared groups are represented in the ry not be deleted via this interface. All roder items • Nickname Takayuki Kando Ryoko Asakura Yuki Nagato	Groups Groups Friends None None	Subscription both both	Add Ne Edat	w Item ( Delete 3 3

図 6: Openfire ユーザの連絡先一覧

この状態でクライアントを 3 つ立ち上げてみましょう。Openfire の開発元が提供しているクライアントで ある Spark を使ってもいいのですが、Spark は Windows 環境では一つしか立ちあげられないので、ここでは 筆者が製作した Sample XMPP Client (リソース名は Sample Messenger) というクライアントを起動してい ます (このクライアントは現在開発中ですが、次号で製作記事を予定しています。)。以下のように yuki.n、 ryoko.a、emiri.k についてクライアントを起動、ログインすると以下のように 3 つのクライアントが起動され ます:

Emiri Kimidori(chat):		
Takayuko kanuolonninej.	Sample XMPP Client: ry	roko.a@ostrich/S 😑 📼 🄀
Ryoko Asakura(away):	Ryoko Asakura away	
	Emiri Kimidori[chat]:	
	Takayuki Kando(offline):	Sample XMPP Client: emin.k@ostrich/
	Yuki Nagato(dnd):	Emiri Kimidori
		Takayuki Kando(offline): -
		Rycko Asakura[away]:
		Yuki Nagato[dnd]:
		-

図 7:3ユーザについて Sample Messenger を起動した様子

各クライアントのウィンドウには一番上に自分のステイタスを表示&操作する部分があり、その下には連絡 先リストに登録されているユーザのリストが表示されています。yuki.n は Don't Disturb(邪魔しないで)を 意味する"dnd"を、ryoko.a は離席を意味する"away"を、emiri.k はチャット歓迎を意味する"chat"を現在の mode として選択した状態になっています。それぞれのユーザ・リストも各ユーザの選択を反映した表示にな っています。

**Openfire**の管理画面から「Sessions」の「Client Sessions」を選ぶと、以下の様な画面で各クライアント がサーバに接続している情報を見ることができます:

<ul> <li>② Openfire Admin Console:</li> <li>ファイル(E) 福集(E) 表示(V</li> <li>□ ① ① ① ① ① ② ☑ ①</li> </ul>	Client Sessions - M ) 頑麼(S) ブックマ o Openfire Adm	lozilla Firefox (ーク( <u>B</u> ) ツール( <u>T</u> nin Console: ×	) ヘルプ(H) +		P			k		_ 0 _*	
< > 0 0 0 0	o http://127.0.	0.1:9090/session	-summary.jsp			ର - ୯ 😽	<ul> <li>Google</li> </ul>			P 🟦 🛢	•
🍬 WEBラジオ   「俺の… 🧧	よく見るページ 🥹	はじめよう 🔛 UNI	QLO CALENDA	R	] goo 検索ブックマーク.	🚺 国語辞典	英和辞典	和			10
openfire <sup>.</sup>									Logges	Openfire 3.7 List an admin - Logo	0
Server Users/Groups	Sessions Group (	Chat Plugins									
Active Sessions Tools											
Client Sessions     Server Sessions	Client Sessio	ns									
Component Sessions	Active Client Sess	ions: 3 – Sessions per p	age: 15 👻					Refresh	60	<ul> <li>(seconds)</li> </ul>	
	Name 👔	Resource	Status		Presence		Priority	Client IP	Close	Connection	
	1 amiri.k	SampleMessenger	Authenticated	9	<ul> <li>Available to Chat</li> </ul>		1	127.0.0.1		0	
	2 ryoko.a	SampleMessenger	Authenticated	9	•		1	127.0.0.1		0	
	3 yuki.n	SampleMessenger	Authenticated	9	•		1	127.0.0.1		0	
	List last updated	Aug 11, 2011 4:47:36 PM	E.								
Server I Users/Groups I Sessions	I Otous Chat I Plusms					Built by	Jtre Softw	are and the jo	anteRes	allime.org commun	ty

図 8: Openfire セッション管理画面

クライアントの設定したリソース名"SampleMessenger"、接続の認証、暗号化、状態(各ユーザの mode を反映した緑、黄、赤色のアイコンで表示されている)、各接続の優先順位、各クライアントが稼働している マシンの IP アドレス(サーバと同じマシンの上から接続しているので localhost を意味する特別な IP アドレ ス 127.0.0.1 になっている)などを見ることができます。一番右の〇に×のアイコンはサーバ側から強制的に 接続を切断する際にクリックします。右上の Refresh を設定すると表示されている状況が定期的に更新されま す。

#### Smack

Smack ライブラリをインストールして javac や java コマンドを実行する際のクラスパス指定に追加します。 Eclipse のような IDE(統合開発環境)を利用している場合は適宜設定してください。Eclipse の場合はプロ ジェクトのプロパティを選んで、ビルドパスに外部 jar ファイルとしてそのパスを追加します。

# 3 EchoBot

## 3-1 EchoBot の動作

EchoBot は非常に簡単な自動応答型のクライアント・プログラム(ボットと呼ばれる)で、以下のような動作をします:

- 1) プロパティ・ファイルに指定された設定でサーバに接続&ログインする
- 接続すると available という mode、現在時刻を status にしたプレゼンス情報を一定時間(30秒)毎 に発信する
- 3) 通常のチャットで他のクライアントから話しかけられると以下の動作をする:

(ア)話しかけてきたアカウントが時報発信用のリストになければ追加

(イ) チャットで送られてきたテキストをそのまま返す

4) 一定時間(5分)毎に時報発信用のリストに登録された相手に時報メッセージを発信

以下、実際に動作を見てみましょう。ここでは emiri.k のユーザとして EchoBot をログインさせ、ryoko.a について先ほどの SampleMessenger というメッセンジャー・クライアントを立ち上げています。メッセンジャー・クライアントのロースターは以下のように見えます:

🙎 Sar	mple XMPP Client: ryoko.a@ost 👝 💷 🗮 🏹
RO BAAGE	ryoko.a available 💌
RO BANE OH	Emiri Kimidori[chat]: Date: 2011/08/11 16:56:32
HO IMAGE	Takayuki Kando[offline]: -
RO IMAGE	Yuki Nagato[offline]: -

#### 図 9: クライアントから見える EchoBot のステイタス

emiri.k としてログインしている EchoBot から送信されたプレゼンス情報を反映して、emiri.k の mode としては"chat"が、status としては現在の日付と時刻が表示されているのが分ります。(そしてこの原稿が如何にギリギリに作成されているかも分ります。この本の発行予定日は8月13日です。)

この時、Openfire の管理画面でセッションを眺めると以下のように表示されています:



図 10: Openfire から見る EchoBot のステイタス

EchoBot はリソース名として"EchoBot"を設定するのでそれが反映されています。また mode が"chat"になっていることもわかります。

その状態で実際に ryoko.a としてログインしているメッセンジャー・クライアントから話しかけた場合のロ グは以下のようになります:



図 11: クライアント側のチャット・ログ

チャットで ryoko.a が話しかけると同じテキストがオウム返しされています。また最初に話しかけて以降、 時刻の分の末尾が5になる時に時報メッセージが発信されているのも分ります。

では以下の節でこのプログラムについて見ていくことにしましょう。

### 3-2EchoBot のプログラム

#### プログラムの構成

EchoBotは非常に単純なプログラムなので、一個のトップレベル・クラスに全体がまとめられています。このトップレベル・クラスの中に定期的なプレゼンス情報の更新や時報メッセージの発信のためにTimerTaskインターフェースを実装する無名クラス<sup>4</sup>の定義が二つ含まれています。

<sup>4</sup>名前の付いていないクラス、匿名クラスとも呼ばれる。インナークラスの一種。Java ではクラスの中でクラスを入れ 子に定義することができる。入れ子の「内側」に定義されたクラスは(static として宣言されていない限り)「外側」のク ラスのオブジェクトの値を参照したり、メソッドを呼び出したりすることができる。無名クラスも同様(無名クラスは static にはできない)。



図 12: EchoBot.java の位置

パッケージは org.xgmtk.xmpp.samples です。xgmtk.org は XGMTK プロジェクトが保持しているドメイ ンですので、Java におけるパッケージ名の慣習に則って org.xgmtk パッケージは XGMTK プロジェクトで専 用に利用することができます。

下の図は Eclipse IDE で EchoBot クラスのアウトラインを表示したものです。



図 13: EchoBot クラスのアウトライン

EchoBot クラスはそれ自身が Java アプリケーションとして動作するよう、アプリケーション開始の起点となる main()メソッドを含んでいます。まずはこの main()メソッドから見て行きましょう。

main()メソッド

```
public static void main(String[] args) throws Exception{
    Properties settings = loadSettings();
    String service = settings.getProperty("service");
    String id = settings.getProperty("id");
    String passwd = settings.getProperty("passwd");
```

```
BOT = new EchoBot(service, id, passwd);
System out.println("Type ¥""+QUIT_COMMAND
        +"¥" and enter key to quit");
BufferedReader lineReader = new BufferedReader(
        new InputStreamReader(System in));
String lineInput = lineReader.readLine();
while(lineInput != null){
        if(lineInput.equals(QUIT_COMMAND)){
           break;
        }
        lineInput = lineReader.readLine();
    }
System.exit(0);
}
```

図 14: EchoBot  $\mathcal{O}$  main0メソッド

Java アプリケーションの実行は main0メソッドから開始されます。この main0メソッドではまず loadSettings0というメソッドでプロパティ・ファイルから情報を読み込んで Properties クラスのオブジェク トを作成しています。loadSettings0メソッドについては「プロパティの読み込み」(p.15)を参照してくださ い。java.util.Properties クラスは Java の標準 API に含まれるクラスで、Java アプリケーションの設定情報 ファイルを取り扱う目的でよく利用されます。

このプロパティ・ファイルは以下のような形式のテキストファイルです。各行の行頭から"="の前までの文 字列がキー、"="の後ろから開業の前までの文字列がそのキーに対応する値となります。今回の例ではプロパ ティ・ファイルの中身は以下の様になっています:

servi ce= <u>l ocal host</u>	
id=emiri.k	
passwd= <u>Haruhi</u>	

#### $\boxtimes$ 15: echo. properties

その後、service、id、passwd という String 型変数へをそれぞれ"service"、"id"、"passwd"というキーを使って Properties オブジェクトから値を取り出しています。プロパティ・ファイルの中身が図 18のようになっているので service、id、passwd という変数の値はそれぞれ、"localhost"、"emiri.k"、"Haruhi"となります。これらはこの後 EchoBot クラスのコンストラクタの中でコネクションの接続とログインに使われますので、「環境設定」の「Openfire」(p.3) で設定したものと一致している必要があります。

echo.properties ファイルはサンプル・コードでは誰でも読めるような設定のファイルになっていますが、 パスワードが含まれることから、実用に供するボットでは最低限、権限を適切に設定してボットと管理者以外 には読み取れないようにしておくべきでしょう。

そして main()のもっとも重要な仕事として EchoBot クラスのオブジェクトを作成します。EchoBot クラス のコンストラクタについては「EchoBot コンストラクタ」(p.17) で詳細を述べますが、このコンストラクタ の実行が終わると EchoBot は外からのチャットを受け付ける準備ができたことになります。

ここで BOT は EchoBot 型の static 変数です。処理が終わるまで GC されてしまわないよう念のため static 変数に保管しています。

その後は終了処理です。EchoBotオブジェクトが行うXMPP関係の処理はmain()メソッドとは別のスレッド で行われます。よってmain()にはもう終了する以外にやることはないのですが、そのまま最終行の System.exit(0)に突っ込んでしまうと当然全てのスレッドが終了してしまうので困ります。かといって System.exit(0)をなくしてただmain()を終了させた場合、もしXMPPの処理をするスレッドがデーモン・スレ ッド 5と設定されていた場合はやはり終了してしまいますし、そうでない場合は穏やかに終了させる簡単な方 法がありません。XMPPのイベントを処理するスレッドはXMPPConnectionクラスのオブジェクトが管理して いますが、それがデーモン・スレッドであるかどうかはマニュアル等のドキュメントで仕様として明記されて はいませんので、それがデーモン・スレッドであるにせよないにせよ、そのことに依存することは避けた方が よいでしょう。

そこでここでは終わるべき時がくる前にmain()のスレッドが終わってしまうことがないよう、条件付きの無限ループをまわすこととしました。そのためまず、System.inオブジェクトをBufferedReaderクラスのオブジェクトlineReaderでラップします。これはBufferedReaderクラスが持っている一行入力のためのreadLine()メソッドを利用するためです。ループではlineReader.readLine()で一行読み込み、その結果が"quit"(このクラスで設定している定数QUIT\_COMMANDの値です。)と一致するまで無限にループが回ります 6。ここで"quit"に続いて改行が入力されると無限ループが終わり、System.exit(0);が実行されてプログラム全体が終了します。

#### プロパティの読み込み

プロパティの読み込みは下に示す loadSettings()メソッドで行います:

#### private static Properties loadSettings()

<sup>&</sup>lt;sup>5</sup> デーモン・スレッドはプログラムの補助的な機能を担うスレッドに使われる機能で、スレッドを作成後、デーモン・ スレッドだと設定することでスレッドはデーモン・スレッドになります。Java ではプログラムで動いているスレッドがデ ーモン・スレッドだけになるとプログラム全体が終了することになっています。

<sup>6</sup>入力がない場合は lineReader.readLine0のところで main0のスレッドはブロックされ、入力が一行あるまで待ちますの でこの無限ループは CPU 時間を無駄遣いはしません。

```
throws FileNotFoundException, IOException {
    InputStream is = new BufferedInputStream(
        new FileInputStream(SETTING_FILE));
    Properties settings = new Properties();
    settings.load(is);
    is.close();
    return settings;
}
```

#### 図 16: loadSettings()メソッド

SETTING\_FILE 定数が指すファイルを InputStream 型オブジェクト is を通じて読むために FileInputStream を作成し、それを BufferedInputStream でラップして is 変数を初期化しています。そして Properties 型の settings 変数にオブジェクトを作成し、settings.load0メソッドへ InputStream 型オブジェ クト is を渡して読も込ませています。終わったら is.close0し、settings を返り値にしてリターンしています。

ここで、SETTING\_FILE は"resource/echo.properties"ファイルを指す File 型定数です:

private static final File SETTING\_FILE = new File("resource/echo.properties");

#### 図 17:SETTING\_FILE 定数

Eclipse IDE で標準的なプロジェクト設定している場合の resource/echo.properties ファイルの位置はここです:



図 18: echo.properties ファイルの位置

#### EchoBot コンストラクタ

EchoBot クラスのコンストラクタは EchoBot の機能を実現する中心部分です。3 つの部分に分けて説明します。

```
public EchoBot(String service, String id, String passwd)
    throws XMPPException {
    this.chats = new ArrayList<Chat>();
```

#### 図 19: EchoBot コンストラクタ序盤

最初に Chat オブジェクトを管理するためのリストを作成し、chats フィールドを初期化します。Chat オブ ジェクトはメッセージが送られてくるたびに渡されるもので、返事を返すために必要です。

ただ、オウム返しに返事を返すだけならメッセージが送られてくるたびにチャット・オブジェクトが渡され るので保管する必要はありませんが、EchoBotは話しかけてくれた人全員に自発的に時報のメッセージを送る ためリストに集めて覚えておく必要があります。 

#### 図 20: EchoBot コンストラクタ中盤

続いて XMPP サーバに接続し、ログインします。クライアントから発信されるあらゆる XMPP パケット (XMPP 用語ではスタンザ)はまずサーバに送られてサーバが処理あるいは適切な宛先に配送します。逆に クライアントが受信するあらゆるパケットはサーバからやってきます。また一つリクエストを送ってレスポン スを得るたびに接続が切れる HTTP とは異なり、XMPP ではセッションの開始から終了までずっと一つのコ ネクションを利用します。このため XMPP ではサーバとの接続を表現する XMPPConnection クラスのオブジ ェクトがあらゆる処理の中心になります。また XMPP の接続は基本的にユーザ認証されるので、ログインが 必要です。

手順は簡単で、まず XMPPConnection クラスのオブジェクトを作成します。この時に接続先のサービスを 選択します。通常はサーバ名 (多くの場合は DNS で引ける名前)です。ここでは main()から渡された"localhost" になります。"localhost"は特別な名前で発信側と同一のマシンの上の XMPP サーバを指します。実際にその マシンがどういう名前であっても同一マシン内からの接続は"localhost"を指定すれば接続できま す。"localhost"で接続しても XMPPConnection の getUser()メソッドでユーザ名を取得した際のサービス名の 部分は Openfire インストール時に設定した名前(通常はサーバが稼働しているマシンの名前)になります。

次いで、作成した XMPPConnection オブジェクトの connect()メソッドを呼び出して接続します。指定した サービス名でサービスが見つからない場合はこの段階で失敗します。

そしてユーザ名とパスワード、リソース名を指定して login()メソッドを呼び出してログインします。ここで ユーザ名とパスワードはそれぞれ main()から渡された"emiri.k"と"Haruhi"になります。リソース名には EchoBot クラスの単純名"EchoBot"を渡しています。ここで直接"EchoBot"という文字列リテラルで指定しな かった理由は EchoBot のクラス名を変更した時にリソース名を変え忘れないようにするためです。login()に はリソース名を省略した版のメソッドもありますが、その場合"Smack\_"+*Smack のバージョン番号*というリ ソース名になります。

ログインが終わったら ChatManager に自分自身を ChatManagerListener として登録します。これを可能 にするために EchoBot は ChatManagerListener インターフェースをインプリメントして chatCreated 0メソ ッドを備えています。ChatManager はある他のクライアントからメッセージが初めて送られてきたとき、こ こで登録した ChatManagerListener 型オブジェクトの chatCreated0メソッドを呼び出します。 chatCreated0メソッドでの処理については「最後にタイマーを二つ設定して、準備完了を標準出力に報告し ます。一つめの setTimeSignal()メソッドでは時報の間隔を5分に、二つ目の setPresenceUpdater()メソッド ではプレゼンス情報の更新間隔を30秒に設定しています。この二つのメソッドの時間指定の範囲はミリ秒で す。EchoBot クラスでは読みやすくするため定数 MINUTE と SECOND 以下の様に定義しています:

private static final lo	ong SECOND = 1000;
private static final lo	ong MI NUTE = $60 *$ SECOND;

#### 図 25:定数 SECOND と MINUTE

setTimeSignal()メソッドについては「時報の発信」(p.54) で、setPresenceUpdater()メソッドについては「プレゼンス情報の更新」(p.51) で詳しく説明します。

新規チャットの開設」(p.19) で詳しく述べます。

XMPPConnection はマルチ・スレッド対応です。そして XMPPConnection オブジェクト自身が複数のスレ ッドを持ち並行動作しています。ソースをざっと眺めて確認した範囲だけでも、XMPPConnection は内部に サーバからのパケットを読むスレッド、サーバに向けてパケットを書きだすスレッド、そして(例えば ChatManager 経由で ChatListener を呼び出すといった形で)ユーザにイベントを伝えるメソッドという少 なくとも三つのスレッドを備えています。この EchoBot のような簡単なアプリケーションではスレッドをそ れほどはっきり意識することはありません(せいぜい先に述べた main0の終了の部分くらいです)が、GUI を備えたメッセンジャー・クライアントなど複雑なアプリケーションではそれを意識する必要があるでしょう。

一般に XMPP でメッセージを処理する機能を拡張する際はその拡張された処理を行うオブジェクトに XMPPConnection オブジェクトを渡して初期化する場合が多いですがチャットの開設を司る ChatManager や連絡先リストの管理を司る Roster は基本的な機能なので XMPPConnection オブジェクトに内蔵されており、 それぞれ getChatManager()や getRoster ()というメソッドでアクセスすることができます。

this.setTimeSignal(5\*MINUTE);
this.setPresenceUpdater(30\*SECOND);
System.out.println("Echo back service started.(user: ¥""+
 this.connection.getUser()+"¥")");

}

#### 図 21: EchoBot コンストラクタ終盤

最後にタイマーを二つ設定して、準備完了を標準出力に報告します。一つめのsetTimeSignal()メソッドで は時報の間隔を5分に、二つ目のsetPresenceUpdater()メソッドではプレゼンス情報の更新間隔を30秒に設 定しています 7。この二つのメソッドの時間指定の範囲はミリ秒です。EchoBotクラスでは読みやすくするた

<sup>7</sup> ここではあまり複雑にしても分りにくいと考え、実施しませんでしたが、これらの時間間隔も echo.properties ファイル

#### め定数MINUTEとSECOND以下の様に定義しています:

private static final long SECOND = 1000; private static final long MINUTE = 60 \* SECOND;

#### 図 22:定数 SECOND と MINUTE

setTimeSignal()メソッドについては「時報の発信」(p.24) で、setPresenceUpdater()メソッドについては「プレゼンス情報の更新」(p.21) で詳しく説明します。

#### 新規チャットの開設

「EchoBot コンストラクタ」(p.17) で述べたように、EchoBot クラスは ChatManagerListener インター フェースをインプリメントして ChatManager に EchoBot オブジェクト自身を渡しています。その結果他の クライアントから初めてメッセージが送られてきたタイミングで下のような chatCreated()メソッドが ChatManager によって呼び出されます:

@Overri de
<pre>public void chatCreated(Chat chat, boolean createdLocally) {</pre>
chat. addMessageLi stener(thi s);
this.chats.add(chat);
}

#### 図 23:chatCreated()メソッド

第一引数はChatオブジェクトでどこのどんなアドレスのクライアントからの新規Chatかという情報が収め られています <sup>8</sup>。またChatオブジェクトはこのChatの一環として送られてくる実際のメッセージが到着する度 に通知をしてくれるマネージャーとしての機能も備えています。

従って chatCreated0メソッドでは 2 つのことを行います。1 つめは Chat オブジェクトに個々のメッセー ジの到着を知らせてもらうために MessageListner 型のオブジェクトを登録することです。ここでは ChatManager に EchoBot オブジェクトを登録したのと同じ方法を再び使います。このため EchoBot クラス は MessageListener インターフェースも実装します。

2つ目は話しかけてきてくれたアカウントに時報を発信するために Chat オブジェクトを EchoBot オブジェ クトが保持する Chat のリストである chats に追加することです。

に記述し、それを読み込んで設定する方がボットとしての使い勝手はよくなるでしょう。

<sup>&</sup>lt;sup>8</sup> Smack はスレッドをサポートしていて、Chat オブジェクトは実はスレッド毎に別になっています。このため実際には 同一のクライアントから複数の Chat が開設される場合があり得ます。

さて EchoBot クラスが MessageListner インターフェースを実装するには、EchoBot クラスに下の様な processMessage()メソッドを定義する必要があります:

# @Override public void processMessage(Chat chat, Message message) { String sender = chat.getParticipant(); System.out.println("Recieved a message from: ¥"" +sender+"¥", message text:¥""+message.getBody()); try { chat.sendMessage(message.getBody()); } catch (XMPPException e) { System.out.flush(); System.err.println( "\*\* Failed to send a message to ¥"" +chat.getParticipant()+"¥". \*\*"); e.printStackTrace(); } }

図 24: processMessage0メソッド

processMessage(スレッドは対象となる Chat にメッセージが到着する度に呼び出されます。ここでは、第 一引数はメッセージが到着した Chat オブジェクトで、第二引数は到着したメッセージのパケットを表す Message オブジェクトです。

ここではまず標準出力へメッセージの受信ログを出すために Chat オブジェクトの getParticipant()メソッ ドで送り主のアドレスを取り出して String 型の sender 変数に設定します。そして送られてきたメッセージの 本文であるテキストを取り出すために Message オブジェクトの getBody()メソッドを呼び出します。

ついでこのメソッド本来の仕事であるオウム返しを実現するため Chat オブジェクトの sendMessage()メソ ッドへ、送られてきた Message オブジェクトの本文(再び getBody()メソッドを使います。)を渡してメッセ ージを発信します。Chat オブジェクトは発信元のアドレスを知っているので、Chat オブジェクトは正しい宛 先として送られてきたメッセージの送り主を設定したメッセージ・パケットをサーバに送ることができます。

sendMessage()メソッドは失敗して例外が投げられる場合があり得るので、キャッチしてエラー・メッセージとスタックトレースを標準エラー出力へ表示しています。

#### プレゼンス情報の更新

プレゼンス情報を定期的に発信すると言った定期的な動作を実現するには Java の標準 API の Timer オブ ジェクトを利用します。下は EchoBot クラスにある Timer 型の static 変数 TIMER の定義です: private static final Timer TIMER = new Timer();

#### 図 25: static 変数 TIMER

「EchoBot コンストラクタ」(p.17) で述べたようにプレゼンス情報を定期発信するように設定をするメソ ッドが下の setPresenceUpdater()です。

<pre>private void setPresenceUpdater(long interval) {</pre>
<pre>TIMER. schedul eAtFi xedRate(new TimerTask() {</pre>
@0verri de
<pre>public void run() {</pre>
String status = "Date: "
+FORMAT.format(Calendar.getInstance().getTime());
<pre>sendPresence(status);</pre>
}
<pre>}, Calendar.getInstance().getTime(), interval);</pre>
}

#### 図 26:setPresenceUpdater()メソッド

先ほどの TIMER 変数に格納された Timer 型オブジェクトの scheduleAtFixedRate()メソッドを使って TimerTask 型オブジェクトを登録することで定期気的にタスクが実行されます。scheduleAtFixedRate()メソ ッドの第一引数は実行すべきタスクを表す TimerTask 型オブジェクト、第二引数は開始時刻を表す Date オブ ジェクト、第三引数は実行間隔を表す long 型整数で単位はミリ秒です。

ここで第一引数には TimerTask インターフェースに基づく無名クラスのオブジェクトを定義&作成し、そ れを渡しています。こうすると時間が来るたびにこの無名クラスの run()メソッドが呼び出されます。無名ク ラスはその無名クラスが含まれるクラスのフィールド、メソッド、static 変数、定数を利用することができま す。ここでは FORMAT 変数に格納された日付と時刻をフォーマットする DateFormat オブジェクトを利用し て、run()メソッドが実行された際の現在時刻を表す文字列を作成して String 型の status 変数に格納し、 EchoBot オブジェクトの sendPresence()メソッドを呼び出してそれを渡しています。

EchoBot クラスの static 変数 FORMAT の定義は以下の通りです:

# private static final DateFormat FORMAT = DateFormat.getDateTimeInstance();

#### 図 27: static 変数 FORMAT

これによって Date 型オブジェクトから日付と時刻を表す文字列が作成できます。

scheduleAtFixedRate()メソッドの第二引数には現在時刻、第三引数には EchoBot コンストラクタから渡さ

れた値(ここでは30\*60\*1000ミリ秒)を渡しています。

TimerTask オブジェクトの run()メソッドから呼び出される sendPresence()メソッドは下の様になります:

```
public void sendPresence(String status) {
    Presence. Mode m = Presence. Mode. chat;
    Presence. Type type = Presence. Type. available;
    Presence p = new Presence(type, status, 1, m);
    this. connection. sendPacket(p);
}
```

#### 図 28: sendPresence()メソッド

sendPresence0メソッドはプレゼンス情報を表す Presence オブジェクトを作成して、XMPPConnection オ ブジェクトである EchoBot のフィールド connection について sendPacketo0メソッドを呼び出し、プレゼン ス・パケットをサーバに送りつけるという仕事をします。プレゼンス・パケットはサーバが適当に必要なクラ イアントへ配送するのであて先を指定する必要はありません。

Presence オブジェクトのコンストラクタにはユーザのサーバへの登録状況を表す Presence.Type 列挙型の 値(ここではサーバ利用可能を表す available)、ユーザのクライアント利用状況を示す Presence.Mode 列挙 型の値(ここではチャット歓迎を表す chat)、接続クライアントの優先順位(1、よく使われるデフォルト値)、 ユーザが設定した短いテキストである status (ここでは TimerTask から渡された現在時刻を表す文字列)を 渡しています。

#### 時報の発信

「EchoBot コンストラクタ」(p.17) で述べたように時報を定期発信するように設定をするメソッドが下の setTimeSignal()です:

```
private void setTimeSignal(long interval) {
   Calendar local_calendar = Calendar.getInstance();
   local_calendar.set(Calendar.MINUTE, 0);
   local_cal endar. set(Cal endar. SECOND, 0);
   local_calendar.set(Calendar.MILLISECOND, 0);
   Date startTime = local_calendar.getTime();
   TIMER. schedul eAtFi xedRate(new TimerTask() {
       @Overri de
      public void run() {
          String message
             = "* Time signal: "
             +FORMAT. format(Calendar.getInstance().getTime());
          sendMessageToAll(message);
      }
   }, startTime, interval);
   System.out.println(
       "Start time signale(interval: "
       +((double)interval / MINUTE)+" minutes) at : "
       +FORMAT.format(startTime));
}
```

#### 図 29:setTimeSignal()メソッド

setTimeSignalOの動作は TIMER や FORMAT、TimerTask 型の無名クラスの使い方に関して言えば「プ レゼンス情報の更新」(p.21)の setPresenceUpdater()メソッドとほとんど同じです。違いは3つだけです。

1 つめは Presence の定期更新は開始時刻をあまり気にせず現在時刻にしていましたが、時報はきっちりした時間にしたいので開始時刻を、現在時刻の分、秒、ミリ秒を0に設定した時刻に変えていることです。例えば11時48分25秒と225ミリ秒が現在時刻であったとすると、11時ジャストが開始時刻になります。このようにして scheduleAtFixedRate0メソッドを使うと11時50分から以降5分毎(この例では interval が5分に設定されているので)に TimerTask オブジェクトの run0メソッドが起動されるようになります。

2 つめは TimerTask 型のオブジェクトの run()メソッドで呼び出される EchoBot のメソッドが sendPresence()ではなくて sendMessageToAll()メソッドであることです、これは目的がプレゼンス情報の更 新でなくてこれまでに話しかけてきた全員に時報のメッセージを送ることなので当然だと言えます。 3つめは些細なことですが、ログとしてタイマー設定終了後に開始時刻を標準出力に出力していることです。 時報を実際に送るために呼び出される EchoBot クラスのメソッド sendMessageToAll()は下のようになりま す:

```
public void sendMessageToAll(String message) {
    for(Chat c : this.chats){
        try {
            c.sendMessage(message);
        } catch (XMPPException e) {
            System.out.flush();
            System.err.println(
                "** Failed to send a message to ¥""
                +c.getParticipant()+"¥". **");
            e.printStackTrace();
        }
    }
}
```

#### 図 30: sendMessageToAll()メソッド

既に「」(p.) で説明した方法により、これまでに開設された全ての Chat オブジェクトが chats フィールド のリストに記録されているのでこれを利用します。即ち chats の全要素をイテレートしてそれら Chat オブジ ェクトの sendMessage()メソッドを呼び出すことで、全員に向かって TimaerTask の run()メソッドから渡さ れたメッセージ (つまり現在時刻を表す文字列)を送信します。

sendMessage()メソッドは失敗して例外が投げられる場合があり得るので、キャッチしてエラー・メッセージとスタックトレースを標準エラー出力へ表示して、引き続き残りの Chat オブジェクトに対する送信を行います。

```
EchoBot クラスの全ソース
```

最後に EchoBot クラスの全ソース(EchoBot.java)を載せておきます。

```
package org. xgmtk. xmpp. samples;
import java.io.*;
import java.text.DateFormat;
import java.util.*;
import org.jivesoftware.smack.*;
public class EchoBot implements ChatManagerListener, MessageListener{
   static EchoBot BOT;
   private static final String QUIT_COMMAND = "quit";
   private static final File SETTING_FILE
       = new File("resource/echo. properties");
   private static final Timer TIMER = new Timer();
   private static final long SECOND = 1000;
   private static final long MINUTE = 60 * SECOND;
   private static final DateFormat FORMAT
       = DateFormat.getDateTimeInstance();
   private final XMPPConnection connection;
   private List<Chat> chats;
   public EchoBot(String service, String id, String passwd)
          throws XMPPException {
       this.chats = new ArrayList<Chat>();
       this.connection = new XMPPConnection(service);
       this.connection.connect();
       this.connection.login(id, passwd,
          EchoBot. cl ass. getSi mpl eName());
       this.connection.getChatManager().addChatListener(this);
       this.setTimeSignal (5*MINUTE);
```

```
this.setPresenceUpdater(30*SECOND);
   System.out.println("Echo back service started.(user: ¥""+
       this.connection.getUser()+"¥")");
}
private void setPresenceUpdater(long interval) {
   TIMER. schedul eAtFi xedRate(new TimerTask() {
       @Override
       public void run() {
          String status = "Date: "
              +FORMAT. format(Calendar.getInstance().getTime());
          sendPresence(status);
       }
   }, Calendar.getInstance().getTime(), interval);
}
private void setTimeSignal(long interval) {
   Calendar local_calendar = Calendar.getInstance();
   local_calendar.set(Calendar.MINUTE, 0);
   local_cal endar. set(Cal endar. SECOND, 0);
   local_calendar.set(Calendar.MILLISECOND, 0);
   Date startTime = local_calendar.getTime();
   TIMER. schedul eAtFi xedRate(new TimerTask() {
       @Override
       public void run() {
          String message
              = "* Time signal: "
                 +FORMAT. format(Calendar.getInstance().getTime());
          sendMessageToAll(message);
       }
   }, startTime, interval);
   System.out.println(
       "Start time signale(interval: "
       +((double)interval / MINUTE)+" minutes) at : "
       +FORMAT.format(startTime));
```

}

```
28
```

```
public void sendMessageToAll(String message) {
   for(Chat c : this.chats){
       try {
          c. sendMessage(message);
       } catch (XMPPException e) {
          System.out.flush();
          System.err.println(
              "** Failed to send a message to ¥""
              +c.getParticipant()+"¥". **");
          e. printStackTrace();
      }
   }
}
public void sendPresence(String status) {
   Presence. Mode m = Presence. Mode. chat;
   Presence. Type type = Presence. Type. available;
   Presence p = new Presence(type, status, 1, m);
   this.connection.sendPacket(p);
}
@Override
public void chatCreated(Chat chat, boolean createdLocally) {
   chat.addMessageListener(this);
   this.chats.add(chat);
}
@Override
public void processMessage(Chat chat, Message message) {
   String sender = chat.getParticipant();
   System.out.println("Recieved a message from: ¥""
       +sender+"¥", message text: ¥""+message.getBody());
   try {
       chat.sendMessage(message.getBody());
   } catch (XMPPException e) {
```

```
System.out.flush();
       System.err.println(
          "** Failed to send a message to ¥""
          +chat.getParticipant()+"\"". **");
      e. printStackTrace();
   }
}
public static void main(String[] args) throws Exception{
   Properties settings = loadSettings();
   String service = settings.getProperty("service");
   String id = settings.getProperty("id");
   String passwd = settings.getProperty("passwd");
   BOT = new EchoBot(service, id, passwd);
   System.out.println("Type ¥""+QUIT_COMMAND
       +"¥" and enter key to quit");
   BufferedReader lineReader = new BufferedReader(
      new InputStreamReader(System.in));
   String lineInput = lineReader.readLine();
   while(lineInput != null){
      if(lineInput.equals(QUIT_COMMAND)){
          break;
      }
      lineInput = lineReader.readLine();
   }
   System.exit(0);
}
private static Properties loadSettings()
   throws FileNotFoundException, IOException {
   InputStream is = new BufferedInputStream(
      new FileInputStream(SETTING_FILE));
   Properties settings = new Properties();
   settings.load(is);
```

```
is.close();
return settings;
}
```

#### 図 31: EchoBot.java

次回はメッセンジャーアプリケーションらしい GUI を備え実際にメッセージをやり取りできるクライアント をサンプル・コードについて解説します。規模が大きいので2回に分けて紹介しますが、前編では特に GUI とメッセージのやり取りではスレッドを意識する必要があるのでそのあたりを詳しく解説する予定です。後編 ではユーザを表すアイコンであるユーザ・アバター (User Avatar)を例に Pub/Sub プロトコルの簡易版でユ - バターを始めリッチ・プレゼンスの送受信によく利用されている PEP (Personal Eventing Protocol) の実装について紹介する予定です。

}