CHAPTER 7

Cisco Unified JTAPI のアラームとサービス

Cisco Unified JTAPI のアラームとサービスは、JTAPI 1.2 の仕様ではまだ公開されていないが Cisco Unified Communications Manager では使用可能な追加機能を公開するクラスとインターフェイスのセットで構成されます。開発者はこのクラスやインターフェイスを使用して新しいアプリケーションを開発したり、既存のクラスやインターフェイスを変更して新しいメソッドを作成できます。

この章では、Cisco Unified Communications Manager の実装で使用可能なアラームとサービスについて説明します。次のような構成になっています。

Cisco Unified JTAPI 拡張の詳細については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。

アラーム クラスの階層

次のクラス階層は、com.cisco.services.alarm パッケージに含まれています。

java.lang.Object

- com.cisco.services.alarm.AlarmManager
- com.cisco.services.alarm.DefaultAlarm（com.cisco.services.alarm.Alarm を実装）
- com.cisco.services.alarm.DefaultAlarmWriter（com.cisco.services.alarm.AlarmWriter を実装）
- com.cisco.services.alarm.ParameterList

AlarmManager

AlarmManager は、Alarm オブジェクトの作成に使用します。AlarmManager は、ファシリティ名および AlarmService ホスト名とポートとともに作成されます。ファクトリにより作成されるアラームはすべて、このファシリティ名に関連付けられます。このクラスでは、システム全体で使用できる、単一の AlarmWriter への参照が維持されます。アプリケーションで、この AlarmWriter を利用できます。AlarmManager には、AlarmWriter のデフォルトの実装が用意されています。ユーザ定義の AlarmWriter を実装して、この AlarmWriter をオーバーライドすることが可能です。

使用法：
AlarmManager AlarmManager = new AlarmManager(facilityName, alarmServiceHost, alarmServicePort, debugTrace, errorTrace);

Alarm は、alarmName (mnemonic)、サブファシリティ、および重大度を指定することで、ファクトリによって作成されます。Alarm は、アプリケーションのさまざまな部分での使用に備えてキャッシュしておくことができます。アラームの送信の際、アプリケーションでは、AlarmService に特定の情報を提供する変数パラメータを指定できます。

使用法：
一般に、アプリケーションは、それ自体の AlarmManager インスタンスを維持しています。また、アプリケーションでは、アラーム トレースが既存のトレース宛先にも送信されるように、デバッグおよびエラー トレースを設定する必要があります。

Manager および Writer クラスの設定：
AlarmWriter alarmWriter = new DefaultAlarmWriter(port, alarmServiceHost);
AlarmManager alarmManager = new AlarmManager(“AA_IVR“, alarmWriter, debugTrace, errorTrace);

アラームの生成：
サブファシリティとデフォルトの重大度でアラームを作成します。
Alarm alarm = alarmManager.createAlarm(“HTTPSS“, Alarm.INFORMATIONAL);
alarm.send(“090T“) は、ニーモニック付きでアラームを送信します。
alarm.send(“090T“, “Port is stuck“, “CTIPort01“) またはニーモニックおよびパラメータ付き。

宣言

```
public class AlarmManager

java.lang.Object
      |
      +--com.cisco.services.alarm.AlarmManager
```

（注）　複数のパラメータの送信は、ParameterList を指定すれば可能です。

メンバの概要	
コンストラクタ	
	AlarmManager(String, AlarmWriter, Trace, UnconditionalTrace) ファシリティに対して AlarmManager のインスタンスを作成します。
メソッド	
Alarm	createAlarm(String, int) subFacility に要求された重大度の Alarm を作成します。
AlarmWriter	getAlarmWriter()
void	setAlarmWriter(AlarmWriter) アプリケーションが、この AlarmManager によって使用される AlarmWriter を、ユーザ定義の AlarmWriter でオーバーライドすることを可能にします。

継承メンバの概要
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

コンストラクタ

AlarmManager(String, AlarmWriter, Trace, UnconditionalTrace)

```
public AlarmManager(java.lang.String facility,
    com.cisco.services.alarm.AlarmWriter writer,
    com.cisco.services.tracing.Trace debugTrace_,
    com.cisco.services.tracing.UnconditionalTrace errorTrace_)
```

ファシリティに対して AlarmManager のインスタンスを作成します。アプリケーションで、AlarmService に Alarm を送信するための、この AlarmManager により使用される AlarmWriter を指定します。

メソッド

createAlarm(String, int)

```
public com.cisco.services.alarm.Alarm createAlarm(java.lang.String
    subfacility, int severity)
```

subFacility に要求された重大度の Alarm を作成します。

戻り値：

アラーム インターフェイスを実装するオブジェクト。

getAlarmWriter()

```
public com.cisco.services.alarm.AlarmWriter getAlarmWriter()
```

戻り値：

AlarmWriter オブジェクト。

setAlarmWriter(AlarmWriter)

```
public void setAlarmWriter(com.cisco.services.alarm.AlarmWriter
    writer)
```

アプリケーションが、この AlarmManager によって使用される AlarmWriter を、ユーザ定義の AlarmWriter でオーバーライドすることを可能にします。

AlarmWriter

AlarmWriter は、アラーム メッセージを受け取り、TCP リンク上にある受信側の AlarmService にそのメッセージを転送します。このインターフェイスは、com.cisco.service.alarm の実装で使用する他の AlarmWriter を実装するために使用できます。DefaultAlarmWriter は、この実装に提供されていて、AlarmManager から取得できます。

宣言

```
public interface AlarmWriter
```

既知の実装クラスの一覧

DefaultAlarmWriter

メンバの概要

メンバの概要	
メソッド	
void	close() AlarmWriter をクローズします。
java.lang.String	getDescription()
boolean	getEnabled()
java.lang.String	getName()
void	send(String) アラーム メッセージを AlarmService に送信します。
void	setEnabled(boolean) AlarmWriter を有効または無効にします。

メソッド

close()

```
public void close()
```

AlarmWriter をクローズします。

getDescription()

```
public java.lang.String getDescription()
```

戻り値：

AlarmWriter の説明。

getEnabled()

```
public boolean getEnabled()
```

戻り値：

AlarmWriter の現在の有効または無効の状態。

getName()

```
public java.lang.String getName()
```

戻り値：

AlarmWriter の名前。

send(String)

```
public void send(java.lang.String alarmMessage)
```

アラーム メッセージを AlarmService に送信します。

パラメータ：

`the`：送信する Alarm。

setEnabled(boolean)

```
public void setEnabled(boolean enable)
```

AlarmWriter を有効または無効にします。

パラメータ：

`enable`：AlarmWriter を有効または無効にします。

DefaultAlarm

Alarm インターフェイスの実装。AlarmManager は、createAlarm() メソッドが呼び出されると、これらの Alarm を作成します。

宣言

```
public class DefaultAlarm implements Alarm

java.lang.Object
      |
      +--com.cisco.services.alarm.DefaultAlarm
```

実装インターフェイスの一覧

Alarm

メンバの概要

メンバの概要	
コンストラクタ	
	DefaultAlarm(String, String, int, AlarmWriter))
メソッド	
java.lang.String	getFacility()
int	getSeverity()
java.lang.String	getSubFacility()
void	send(String) 指定されたニーモニック付きでアラームを送信します。

メンバの概要 （続き）	
void	send(String, ParameterList) 指定された名前とパラメータ リスト付きでアラームを送信します。
void	send(String, String, String) 指定された名前とパラメータ付きでアラームを送信します。

継承メンバの概要
インターフェイス Alarm から継承したフィールド
ALERTS, CRITICAL, DEBUGGING, EMERGENCIES, ERROR, HIGHEST_LEVEL, INFORMATIONAL, LOWEST_LEVEL, NOTIFICATION, NO_SEVERITY, UNKNOWN_MNEMONIC, WARNING
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

コンストラクタ

DefaultAlarm(String, String, int, AlarmWriter)

```
public DefaultAlarm(java.lang.String facility,
    java.lang.String subFacility, int severity,
    com.cisco.services.alarm.AlarmWriter alarmWriter)
```

メソッド

getFacility()

```
public java.lang.String getFacility()
```

定義：

インターフェイス Alarm の getFacility

getSeverity()

```
public int getSeverity()
```

定義：

インターフェイス Alarm の getSeverity

getSubFacility()

```
public java.lang.String getSubFacility()
```

定義：

インターフェイス Alarm の getSubFacility

send(String)

```
public void send(java.lang.String mnemonic)
```

指定されたニーモニック付きでアラームを送信します。

定義：

インターフェイス Alarm の send

send(String, ParameterList)

```
public void send(java.lang.String mnemonic,
    com.cisco.services.alarm.ParameterList paramList)
```

指定された名前とパラメータ リスト付きでアラームを送信します。

定義：

インターフェイス Alarm の send

send(String, String, String)

```
public void send(java.lang.String mnemonic,
    java.lang.String paramName, java.lang.String paramValue)
```

指定された名前とパラメータ付きでアラームを送信します。

定義：

インターフェイス Alarm の send

DefaultAlarmWriter

AlarmWriter インターフェイスの DefaultAlarmWriter 実装。
DefaultAlarmWriter は、アラームが書き込まれる、固定サイズのキューを保持しています。アラーム サービスへのアラームの送信は、別のスレッド上で実行されます。このキューは、固定サイズです。

宣言

```
public class DefaultAlarmWriter implements AlarmWriter

java.lang.Object
      |
      +--com.cisco.services.alarm.DefaultAlarmWriter
```

実装インターフェイスの一覧

AlarmWriter

メンバの概要

メンバの概要	
コンストラクタ	
	DefaultAlarmWriter(int, String) AlarmService のホスト名、ポートを指定できる DefaultAlarmWriter のコンストラクタ。キューサイズは 50 に初期化されます。

メンバの概要 （続き）	
	DefaultAlarmWriter(int, String, int) AlarmService のホスト名、ポートおよびキュー サイズを指定できる、DefaultAlarmWriter のコンストラクタ。
	DefaultAlarmWriter(int, String, int, ConditionalTrace, UnconditionalTrace) AlarmService のホスト名、ポート、キュー サイズ、デバッグトレースレベル、エラートレースレベルを指定できる、DefaultAlarmWriter のコンストラクタ。
メソッド	
void	close() スレッドをシャットダウンしてソケットを閉じます。
java.lang.String	getDescription()
boolean	getEnabled()
java.lang.String	getName()
static void	main(String[])
void	send(String) Alarm をアラーム サービスに送信します。
void	setEnabled(boolean) アプリケーションは、AlarmWriter を動的に有効または無効にできます。

継承メンバの概要
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

コンストラクタ

DefaultAlarmWriter(int, String)

```
public DefaultAlarmWriter(int port,
    java.lang.String alarmServiceName)
    throws UnknownHostException
```

AlarmService のホスト名、ポートを指定できる、DefaultAlarmWriter のコンストラクタ。キューサイズは 50 になります。AlarmService は、このポートで Alarm メッセージを受信します。

パラメータ：

port：アラーム サービスがアラームを受信するポート。

alarmServiceName：アラーム サービスを所有するマシンのホスト名。

例外：

java.net.UnknownHostException

DefaultAlarmWriter(int, String, int)

```
public DefaultAlarmWriter(int port,
    java.lang.String alarmServiceName, int queueSize)
    throws UnknownHostException
```

AlarmService のホスト名、ポート、キュー サイズ、デバッグ トレース レベル、エラー トレース レベルを指定できる、DefaultAlarmWriter のコンストラクタ。AlarmService は、このポートで Alarm メッセージを受信します。

パラメータ：

`port`：アラーム サービスがアラームを受信するポート。

`alarmServiceName`：アラーム サービスを所有するマシンのホスト名。

`queueSize`：アラーム ライターで保持されるキューのサイズ。

`debugTrace`：デバッグ トレース レベル。

`errorTrace`：エラー トレース レベル。

例外：

java.net.UnknownHostException

DefaultAlarmWriter(int, String, int, ConditionalTrace, UnconditionalTrace)

```
public DefaultAlarmWriter(int port,
    java.lang.String alarmServiceName, int queueSize,
    com.cisco.services.tracing.ConditionalTrace debugTrace_,
    com.cisco.services.tracing.UnconditionalTrace errorTrace_)
    throws UnknownHostException
```

AlarmService のホスト名、ポートおよびキュー サイズを指定できる、DefaultAlarmWriter のコンストラクタ。AlarmService は、このポートで Alarm メッセージを受信します。

パラメータ：

`port`：アラーム サービスがアラームを受信するポート。

`alarmServiceName`：アラーム サービスを所有するマシンのホスト名。

`queueSize`：アラーム ライターで保持されるキューのサイズ。

例外：

java.net.UnknownHostException

メソッド

close()

```
public void close()
```

送信スレッドをシャットダウンし、ソケットをクローズします。

定義：

インターフェイス `AlarmWriter` の `close`

getDescription()

```
public java.lang.String getDescription()
```

定義：

インターフェイス `AlarmWriter` の `getDescription`

戻り値：

AlarmWriter の簡単な説明。

getEnabled()

```
public boolean getEnabled()
```

定義：

インターフェイス AlarmWriter の getEnabled

戻り値：

AlarmWriter の有効状態。

getName()

```
public java.lang.String getName()
```

定義：

インターフェイス AlarmWriter の getName

戻り値：

AlarmWriter の名前。

main(String[])

```
public static void main(java.lang.String[] args)
```

send(String)

```
public void send(java.lang.String alarmMessage)
```

Alarm をアラーム サービスに送信します。

定義：

インターフェイス AlarmWriter の send

setEnabled(boolean)

```
public void setEnabled(boolean enable)
```

アプリケーションは、AlarmWriter を動的に有効または無効にできます。

定義：

インターフェイス AlarmWriter の setEnabled

ParameterList

ParameterList は、追加の（かつオプションの）ユーザ定義パラメータを AlarmService に送るのに使用される、名前と値のペアのリストです。これらのパラメータには、Alarm の詳細を保持できます。

たとえば、LowResourceAlarm には、次のように、どのリソースが低レベルであるかをサービスに通知するパラメータを保持できます。

name="CPUUsage"
value="0.9"

これらのパラメータはユーザ定義可能ですが、事前に AlarmService カタログに定義しておく必要があります。

宣言

```
public class ParameterList

java.lang.Object
      |
      +--com.cisco.services.alarm.ParameterList
```

メンバの概要

メンバの概要	
コンストラクタ	
	ParameterList() ParameterList のデフォルト コンストラクタ。
	ParameterList(String, String) 名前と値のペアを持つコンストラクタ。
メソッド	
void	addParameter(String, String) 名前と値のペア（パラメータ）をリストに追加するメソッド。
java.lang.String[]	getParameterNames() パラメータの値を取得します。
java.lang.String	getParameterValue(String) パラメータの値を取得します。
void	removeAllParameters() リストにあるすべてのパラメータを除去します。
void	removeParameter(String) 特定のパラメータを除去します（リストにある場合）。
java.lang.String	toString()

継承メンバの概要
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

コンストラクタ

ParameterList()

```
public ParameterList()
```

ParameterList のデフォルト コンストラクタ。

ParameterList(String, String)

```
public ParameterList(java.lang.String name,
    java.lang.String value)
```

名前と値のペアを持つコンストラクタ。

メソッド

addParameter(String, String)

```
public void addParameter(java.lang.String name,
    java.lang.String value)
```

名前と値のペア（パラメータ）をリストに追加するメソッド。

getParameterNames()

```
public java.lang.String[] getParameterNames()
```

リストのパラメータ名を取得します。

戻り値：

パラメータの配列。

getParameterValue(String)

```
public java.lang.String getParameterValue(java.lang.String
    parameterName)
```

パラメータの値を取得します。

戻り値：

パラメータの値。

removeAllParameters()

```
public void removeAllParameters()
```

リストにあるすべてのパラメータを除去します。

removeParameter(String)

```
public void removeParameter(java.lang.String parameterName)
```

特定のパラメータを除去します（リストにある場合）。

toString()

```
public java.lang.String toString()
```

オーバーライド：

クラス Object 内の toString

アラーム インターフェイスの階層

次のインターフェイス階層は、com.cisco.services.alarm パッケージに含まれています。

com.cisco.services.alarm.Alarm

com.cisco.services.alarm.AlarmWriter

Alarm

Alarm インターフェイスは、Alarm の定義に使用します。アラームには、アラーム サービスにより認識されるために、次に示す DTD を使用した XML 表現に遵守する必要があります。アプリケーションでは、このインターフェイスを実装するかまたは AlarmFactory を使用して、正しいフォーマットの Alarm を生成できます。Alarm は AlarmService に送られる仕様であり、AlarmService は Alarm に基づいて必要な動作を実行します。この仕様を使用して、AlarmService はカタログで利用可能な定義にアクセスします。このカタログは、Alarm について適切な動作を実行する Alarm 関数を必要とするユーザによって維持されます。Alarm に指定した重大度は、カタログ内にあるこの Alarm に関連付けられた重大度をオーバーライドできます。Alarm に重大度を指定しない場合、カタログ重大度が使用されます。

アラームの重大度は、Syslog から取得され、次のように定義されます。

0 = EMERGENCIES、システム使用不可
1 = ALERTS、ただちに処置が必要
2 = CRITICAL、重大な状態
3 = ERROR、エラー状態
4 = WARNING、警告状態
5 = NOTIFICATION、動作は通常であるが重大な状態
6 = INFORMATIONAL、情報メッセージだけ
7 = DEBUGGING、デバッグ用メッセージ

宣言

```
public interface Alarm
```

既知の実装クラスの一覧

DefaultAlarm

メンバの概要

メンバの概要	
フィールド	
static int	ALERTS アプリケーションはタスクの作業を継続できるが、すべての機能が動作可能なわけではない（リストの 1 つ以上のデバイスがアクセス不能であるが、他はアクセス可能）。 Syslog 重大度 = 1
static int	CRITICAL 重大な障害であり、アプリケーションはこの障害が原因で要求されたタスクを達成できない。たとえば、アプリケーションが、データベースを開いてデバイス リストを読むことができない。 Syslog 重大度 = 2
static int	DEBUGGING エラーまたはプロセスの状態に関する詳細な情報で、DEBUG モードが有効になっているときだけに生成される。 Syslog 重大度 = 7

メンバの概要 （続き）	
static int	EMERGENCIES 緊急状態であり、システムのシャットダウンが必要。 Syslog 重大度 = 0
static int	ERROR 何らかのエラーの状態が発生し、ユーザは、この障害の性質を理解する必要がある。 Syslog 重大度 = 3
static int	HIGHEST_LEVEL 最高位のトレース レベルであり、現在は、トレース レベル 7 の DEBUGGING。
static int	INFORMATIONAL エラー、警告、監査、またはデバッグに関係しない形式の情報。 Syslog 重大度 = 6
static int	LOWEST_LEVEL 最低位のトレース レベルであり、現在は、トレース レベル 0 の EMERGENCIES。
static int	NO_SEVERITY 重大度のない Alarm の生成には、アプリケーションでこのレベルを設定できる。
static int	NOTIFICATION NOTIFICATION は正常であるが、重大な状態を示す。 Syslog 重大度 = 5
static java.lang.String	UNKNOWN_MNEMONIC Alarm の送信の際、ニーモニックが設定されていない場合に使用される文字列。
static int	WARNING なんらかの問題が存在するが、アプリケーションのタスクの実行が妨げられてはいないという警告。 Syslog 重大度 = 4
メソッド	
java.lang.String	getFacility()
int	getSeverity()
java.lang.String	getSubFacility()
void	send(String) 指定されたニーモニック付きで Alarm を送信します。
void	send(String, ParameterList) 指定されたニーモニックとパラメータ リスト付きで Alarm を送信します。
void	send(String, String, String) 指定されたニーモニックと 1 つのパラメータ付きでアラームを送信します。

フィールド

ALERTS

```
public static final int ALERTS
```

アプリケーションはタスクの作業を継続できるが、すべての機能が動作可能なわけではない（リストの 1 つ以上のデバイスがアクセス不能であるが、他はアクセス可能）。
Syslog 重大度 = 1

CRITICAL

```
public static final int CRITICAL
```

重大な障害であり、アプリケーションはこの障害が原因で要求されたタスクを達成できない。たとえば、アプリケーションが、データベースを開いてデバイス リストを読むことができない。
Syslog 重大度 = 2

DEBUGGING

```
public static final int DEBUGGING
```

エラーまたはプロセスの状態に関する詳細な情報で、DEBUG モードが有効になっているときだけに生成される。

Syslog 重大度 = 7

EMERGENCIES

```
public static final int EMERGENCIES
```

緊急状態であり、システムのシャットダウンが必要。
Syslog 重大度 = 0

ERROR

```
public static final int ERROR
```

何らかのエラーの状態が発生し、ユーザは、この障害の性質を理解する必要がある。
Syslog 重大度 = 3

HIGHEST_LEVEL

```
public static final int HIGHEST_LEVEL
```

最高位のトレース レベルであり、現在は、トレース レベル 7 の DEBUGGING。

INFORMATIONAL

```
public static final int INFORMATIONAL
```

エラー、警告、監査、またはデバッグに関係しない形式の情報。
Syslog 重大度 = 6

LOWEST_LEVEL

```
public static final int LOWEST_LEVEL
```

最低位のトレース レベルであり、現在は、トレース レベル 0 の EMERGENCIES。

NO_SEVERITY

```
public static final int NO_SEVERITY
```

重大度のない Alarm の生成には、アプリケーションでこのレベルを設定できる。注意：これは、アプリケーションで、カタログにあるアラームに関連する重大度を AlarmService に使用させる場合だけを想定しています。

NOTIFICATION

```
public static final int NOTIFICATION
```

NOTIFICATION は正常であるが、重大な状態を示す。

Syslog 重大度 = 5

UNKNOWN_MNEMONIC

```
public static final java.lang.String UNKNOWN_MNEMONIC
```

Alarm の送信の際、ニーモニックが設定されていない場合に使用される文字列。

WARNING

```
public static final int WARNING
```

なんらかの問題が存在するが、アプリケーションのタスクの実行が妨げられてはいないという警告。

Syslog 重大度 = 4

メソッド

getFacility()

```
public java.lang.String getFacility()
```

戻り値：

この Alarm のファシリティ名。

getSeverity()

```
public int getSeverity()
```

戻り値：

アラームの重大度。[0-7] の範囲の整数。

getSubFacility()

```
public java.lang.String getSubFacility()
```

戻り値：

この Alarm のサブファシリティ。

send(String)

```
public void send(java.lang.String mnemonic)
```

指定されたニーモニック付きで Alarm を送信します。null または空文字列が渡された場合、ニーモニック UNK が送信されます。

send(String, ParameterList)

```
public void send(java.lang.String mnemonic,
    com.cisco.services.alarm.ParameterList parameterList)
```

指定されたニーモニックとパラメータ リスト付きで Alarm を送信します。

send(String, String, String)

```
public void send(java.lang.String mnemonic,
    java.lang.String parameterName, java.lang.String parameterValue)
```

指定されたニーモニックと 1 つのパラメータ付きでアラームを送信します。

AlarmWriter

AlarmWriter は、アラーム メッセージを受け取り、TCP リンク上にある受信側の AlarmService にそのメッセージを転送します。このインターフェイスは、com.cisco.service.alarm の実装で使用する他の AlarmWriter を実装するために使用できます。DefaultAlarmWriter は、この実装に提供されていて、AlarmManager から取得できます。

宣言

```
public interface AlarmWriter
```

既知の実装クラスの一覧

DefaultAlarmWriter

メンバの概要

メンバの概要	
メソッド	
void	close() AlarmWriter をクローズします。
java.lang.String	getDescription()
boolean	getEnabled()
java.lang.String	getName()
void	send(String) アラーム メッセージを AlarmService に送信します。
void	setEnabled(boolean) AlarmWriter を有効または無効にします。

メソッド

close()

```
public void close()
```

AlarmWriter をクローズします。

getDescription()

```
public java.lang.String getDescription()
```

戻り値：

AlarmWriter の説明。

getEnabled()

```
public boolean getEnabled()
```

戻り値：

AlarmWriter の現在の有効または無効の状態。

getName()

```
public java.lang.String getName()
```

戻り値：

AlarmWriter の名前。

send(String)

```
public void send(java.lang.String alarmMessage)
```

アラーム メッセージを AlarmService に送信します。

パラメータ：

`the`：送信する Alarm。

setEnabled(boolean)

```
public void setEnabled(boolean enable)
```

AlarmWriter を有効または無効にします。

パラメータ：

`enable`：AlarmWriter を有効または無効にします。

サービス トレース クラスの階層

次のクラス階層は、com.cisco.services.tracing パッケージに含まれています。

java.lang.Object

- com.cisco.services.tracing.BaseTraceWriter（com.cisco.services.tracing.TraceWriter を実装）
  - com.cisco.services.tracing.ConsoleTraceWriter
  - com.cisco.services.tracing.LogFileTraceWriter
  - com.cisco.services.tracing.OutputStreamTraceWriter
  - com.cisco.services.tracing.SyslogTraceWriter
- com.cisco.services.tracing.TraceManagerFactory

BaseTraceWriter

この抽象クラスは、デフォルトで非出力の `TraceWriter` を `TraceWriterManager` に提供するのに有効です。トレースする機能をさまざまなストリームに提供するには、このクラスを拡張する必要があります。拡張側のクラスによって、`doPrintln ()` メソッドを実装する必要があります。

宣言

```
public abstract class BaseTraceWriter implements TraceWriter
```

```
java.lang.Object
      |
      +--com.cisco.services.tracing.BaseTraceWriter
```

実装インターフェイスの一覧

TraceWriter

直系の既知のサブクラス

ConsoleTraceWriter, LogFileTraceWriter, OutputStreamTraceWriter, SyslogTraceWriter

メンバの概要

メンバの概要	
コンストラクタ	
protected	BaseTraceWriter(int[], String, String) Trace.LOWEST_LEVEL および Trace.HIGHEST_LEVEL の範囲を超えた traceLevels 配列に渡された、トレース レベル付き BaseTraceWriter は、無視されます。
protected	BaseTraceWriter(int, String, String) maxTraceLevel までのすべてのレベルをトレースする BaseTraceWriter。トレース レベルは、[Trace.HIGHEST_LEVEL, Trace.LOWEST_LEVEL] の範囲に維持されます。
protected	BaseTraceWriter(String, String)) 重大度レベルが Trace.LOWEST_LEVEL である最低レベルのメッセージだけをトレースする BaseTraceWriter。
メソッド	
void	close()
protected void	doClose()
protected void	doFlush()
protected abstract void	doPrintln(String, int) 特定のトレース機能を実現するために、BaseTraceWriter を拡張するさまざまな TraceWriter に実装する必要があります。
void	flush()
java.lang.String	getDescription()
boolean	getEnabled()
java.lang.String	getName()
int[]	getTraceLevels()
void	println(String, int)
void	setTraceLevels(int[])
java.lang.String	toString()

継承メンバの概要
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

コンストラクタ

BaseTraceWriter(int[], String, String)

```
protected BaseTraceWriter(int[] traceLevels,
    java.lang.String name, java.lang.String description)
```

Trace.LOWEST_LEVEL および Trace.HIGHEST_LEVEL の範囲を超えた traceLevels 配列に渡された、トレース レベル付き BaseTraceWriter は、無視されます。

パラメータ：

traceLevels：トレース レベルの配列。

関連項目：

Trace

BaseTraceWriter(int, String, String)

```
protected BaseTraceWriter(int maxTraceLevel,
    java.lang.String name, java.lang.String description)
```

maxTraceLevel までのすべてのレベルをトレースする BaseTraceWriter。トレース レベルは、[Trace.HIGHEST_LEVEL, Trace.LOWEST_LEVEL] の範囲に維持されます。

関連項目：

Trace

BaseTraceWriter(String, String)

```
protected BaseTraceWriter(java.lang.String name,
    java.lang.String description)
```

重大度レベルが Trace.LOWEST_LEVEL である最低レベルのメッセージだけをトレースする BaseTraceWriter。

関連項目：

Trace

メソッド

close()

```
public final void close()
```

次のインターフェイスからコピーされた記述：

com.cisco.services.tracing.TraceWriter

この TraceWriter によって関連付けられたすべてのリソースを解放します。

定義：

インターフェイス TraceWriter の close

doClose()

```
protected void doClose()
```

doFlush()

```
protected void doFlush()
```

doPrintln(String, int)

```
protected abstract void doPrintln(java.lang.String message,
    int messageNumber)
```

特定のトレース機能を実現するために、BaseTraceWriter を拡張するさまざまな TraceWriter に実装する必要があります。

flush()

```
public final void flush()
```

次のインターフェイスからコピーされた記述：com.cisco.services.tracing.TraceWriter

println メソッドを使用して、出力されたすべてのメッセージを強制出力します。

定義：

インターフェイス TraceWriter の flush

getDescription()

```
public final java.lang.String getDescription()
```

定義：

インターフェイス TraceWriter の getDescription

getEnabled()

```
public boolean getEnabled()
```

次のインターフェイスからコピーされた記述：com.cisco.services.tracing.TraceWriter

println メソッドによって出力されるものがあるかどうかを返します。クローズされた TraceWriter のこのメソッドは、常に false を返します。

定義：

インターフェイス TraceWriter の getEnabled

getName()

```
public final java.lang.String getName()
```

定義：

インターフェイス TraceWriter の getName

getTraceLevels()

```
public final int[] getTraceLevels()
```

定義：

インターフェイス TraceWriter の getTraceLevels

println(String, int)

```
public final void println(java.lang.String message, int severity)
```

次のインターフェイスからコピーされた記述：com.cisco.services.tracing.TraceWriter

指定された文字列とそれに続く改行を出力します。具象 TraceWriter クラスでは、重大度を使用して、特定のストリームからのメッセージをブロックします。各 TraceWriter は、実行する必要のある最高位レベルのトレースを認知しています。

定義：

インターフェイス TraceWriter の println

setTraceLevels(int[])

```
public final void setTraceLevels(int[] levels)
```

次のインターフェイスからコピーされた記述：com.cisco.services.tracing.TraceWriter

この TraceWriter によりトレースされるトレース レベルを設定します。

定義：

インターフェイス TraceWriter の setTraceLevels

toString()

```
public final java.lang.String toString()
```

オーバーライド：

クラス Object 内の toString

ConsoleTraceWriter

トレースするコンソール TraceWriter を System.out. に提供します。

関連項目：
Trace

宣言

```
public final class ConsoleTraceWriter extends BaseTraceWriter

java.lang.Object
      |
      +--com.cisco.services.tracing.BaseTraceWriter
                |
                +--com.cisco.services.tracing.ConsoleTraceWriter
```

実装インターフェイスの一覧

TraceWriter

メンバの概要

メンバの概要	
コンストラクタ	
	ConsoleTraceWriter() デフォルトのコンストラクタ。すべての重大度レベルをトレースします。
	ConsoleTraceWriter(int) トレースする最大レベルを設定するコンストラクタ。
	ConsoleTraceWriter(int[]) トレース レベルの配列を使用して ConsoleTraceWriter を構築します。トレース レベル配列に重大度のあるトレースだけが、トレースされます。
メソッド	
protected void	doFlush()
protected void	doPrintln(String, int)
static void	main(String[])

継承メンバの概要
クラス BaseTraceWriter から継承したメソッド
close(), doClose(), flush(), getDescription(), getEnabled(), getName(), getTraceLevels(), println(String, int), setTraceLevels(int[]), toString()
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

コンストラクタ

ConsoleTraceWriter()

```
public ConsoleTraceWriter()
```

デフォルトのコンストラクタ。すべての重大度レベルをトレースします。

ConsoleTraceWriter(int)

```
public ConsoleTraceWriter(int maxTraceLevel)
```

トレースする最大レベルを設定するコンストラクタ。

関連項目：

Trace

ConsoleTraceWriter(int[])

```
public ConsoleTraceWriter(int[] traceLevels)
```

トレース レベルの配列を使用して ConsoleTraceWriter を構築します。トレース レベル配列に重大度のあるトレースだけが、トレースされます。

パラメータ：

int：[] traceLevels

関連項目：

Trace

メソッド

doFlush()

```
protected final void doFlush()
```

オーバーライド：

クラス BaseTraceWriter 内の doFlush

doPrintln(String, int)

```
protected final void doPrintln(java.lang.String message,
    int messageNumber)
```

次のクラスからコピーされた記述：com.cisco.services.tracing.BaseTraceWriter

特定のトレース機能を実現するために、BaseTraceWriter を拡張するさまざまな TraceWriter に実装する必要があります。

オーバーライド：

クラス BaseTraceWriter 内の doPrintln

main(String[])

```
public static void main(java.lang.String[] args)
```

LogFileTraceWriter

このクラスは、BaseTraceWriter クラスを拡張します。ログ ファイルのセットに書き込みを行い、各ログ ファイルに指定された容量が満たされると、そのログ ファイルを指定のディレクトリに保存し、順番に次のファイルに書き込む TraceWriter が実装されます。

各ログ ファイルは、CurrentFile、FileNameBase、および FileExtension の 3 つのプロパティによって制御されるパターンに基づいて命名されます。CurrentFile プロパティは、序数でどのログ ファイルに現在書き込みが行われているのかを、FileNameBase プロパティは、各ログ ファイル名のプレフィクスを、また FileExtension プロパティは、「txt」などのサフィクスを決定します。これらのプロパティをもとに、ログ ファイルには **FileNameBase** *LeadingZeroPadding* CurrentFile.FileExtension という名前が割り当てられます。CurrentFile プロパティは、1 から MaxFiles プロパティまでの値を取ります。CurrentFile プロパティは、文字列に変換されると、MaxFiles および CurrentFile プロパティの値に応じて、先行するゼロでパディングされます。インデックス ファイルが、最後に書き込まれたファイルのインデックスを記録しています。logFileWriter が（アプリケーションがリスタートされた場合などで）再作成されると、最後の書き込みインデックスに続いて新規ファイルが作成されます。

ログ ファイルの保管場所は、path、dirNameBase、および useSameDir により決定されます。path を指定しない場合、デフォルトとして、現在のパスが使用されます。dirNameBase を指定しない場合、パスにログ ファイルが書き込まれます。LogFileTraceWriter インスタンスが作成されるたびに、useSameDir が true であるか false であるかに従って、ログ ファイルが同じディレクトリまたは

新規のディレクトリに書き込まれます。毎回新規のディレクトリが作成される場合、ディレクトリ名は、dirNameBase および「_」の後に番号が付いた名前になります。番号は、パスにある同じ dirNameBame のディレクトリに関連付けられた最大数よりも 1 大きい数です。パスの指定には、「/」または「¥¥」は使用できますが、「¥」は使用できません。

LogFileTraceWriter は、現在のログ ファイルに書き込まれたバイト数を記録しています。この数が LogFileTraceWriter.ROLLOVER_THRESHOLD バイトに到達すると次のファイルにログが書き込まれます。次のファイルは CurrentFile が MaxFiles に等しくない場合は CurrentFile + 1、または CurrentFile が MaxFiles に等しい場合は 1 です。

（注）　このクラスのすべてのプロパティは、コンストラクタに指定されます。これを動的に変更する方法はありません。**Caveat** : LogFileTraceWriter の 2 つのインスタンスが、同じ path と dirNameBase で作成され、useSameDir が true である場合、この 2 つは同じファイルに書き込まれる可能性があります。

例

次のコードでは、「MyLog01.log」から「MyLog12.log」のログ ファイルを作成する、LogFileTraceWriter をインスタンス化しています。各ファイルが、およそ 100K バイトのサイズまで大きくなると、次のファイルが作成されます。

LogFileTraceWriter out = new LogFileTraceWriter ("MyLog", "log", 12, 100 * 1024); と指定すると、ログ ファイルの TraceWriter が作成され、Mylog01.log から Mylog12.log までの 12 ファイルに、順番に 100K バイト分のトレースが書き込まれます。デフォルトでは、トレースは HIGHEST_LEVEL に設定されます。

例

次のコードでは、パス「c:/LogFiles」のサブディレクトリ「Run」にログ ファイルを保存する LogFileTraceWriter を構築します。ファイルは、MyLogXX.log. と命名されます。順に保存するファイル数は 12 で、サイズは 100 KB です。アプリケーションの各インスタンスで同じディレクトリを使用します。

LogFileTraceWriter out = new LogFileTraceWriter ("c:/logFiles", "Run", "MyLog", "log", 12, 100*1024, true);

関連項目：

Trace

宣言

```
public final class LogFileTraceWriter extends BaseTraceWriter

java.lang.Object
      |
      +--com.cisco.services.tracing.BaseTraceWriter
                |
                +--com.cisco.services.tracing.LogFileTraceWriter
```

実装インターフェイスの一覧

TraceWriter

メンバの概要

メンバの概要	
フィールド	
static java.lang.String	DEFAULT_FILE_NAME_BASE
static java.lang.String	DEFAULT_FILE_NAME_EXTENSION
static char	DIR_BASE_NAME_NUM_SEPERATOR
static int	MIN_FILE_SIZE
static int	MIN_FILES
static int	ROLLOVER_THRESHOLD
コンストラクタ	
	LogFileTraceWriter(String, String, int, int) すべてのレベルのトレースに、任意の数のファイルを順に処理する LogFileTraceWriter のデフォルトのコンストラクタ。
	LogFileTraceWriter(String, String, String, String, int, int, boolean) すべてのレベルのトレースに、任意の数のファイルを順に処理する LogFileTraceWriter のデフォルトのコンストラクタ。
	LogFileTraceWriter(String, String, String, String, int, int, int, boolean) 任意の数のファイルを順に処理し、指定のディレクトリに保存する LogFileTraceWriter を構築します。
メソッド	
protected void	doClose() この OutputStream をクローズします。
protected void	doFlush()
protected void	doPrintln(String, int)
int	getCurrentFile() CurrentFile プロパティ。
java.lang.String	getFileExtension() FileExtension プロパティ。
java.lang.String	getFileNameBase() FileNameBase プロパティ。
java.lang.String	getHeader() 各ログ ファイルの先頭に書き込まれるヘッダー文字列を取得します。
int	getMaxFiles() MaxFiles プロパティ。
int	getMaxFileSize() MaxFileSize プロパティ。
void	setHeader(String) 各ファイルの先頭に書き込まれるヘッダー定数を設定します。トレースの書き込みは、ヘッダーが書き込まれた次の行から継続されます。setHeader がファイル出力の開始後に呼び出された場合、次に書き込まれるファイルから有効になります。

継承メンバの概要
クラス BaseTraceWriter から継承したメソッド
close(), doClose(), flush(), getDescription(), getEnabled(), getName(), getTraceLevels(), println(String, int), setTraceLevels(int[]), toString()
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

フィールド

DEFAULT_FILE_NAME_BASE

```
public static final java.lang.String DEFAULT_FILE_NAME_BASE
```

DEFAULT_FILE_NAME_EXTENSION

```
public static final java.lang.String DEFAULT_FILE_NAME_EXTENSION
```

DIR_BASE_NAME_NUM_SEPERATOR

```
public static final char DIR_BASE_NAME_NUM_SEPERATOR
```

MIN_FILE_SIZE

```
public static final int MIN_FILE_SIZE
```

MIN_FILES

```
public static final int MIN_FILES
```

ROLLOVER_THRESHOLD

```
public static final int ROLLOVER_THRESHOLD
```

コンストラクタ

LogFileTraceWriter(String, String, int, int)

```
public LogFileTraceWriter(java.lang.String fileNameBase,
    java.lang.String fileNameExtension, int maxFiles, int maxFileSize)
    throws IOException
```

すべてのレベルのトレースに、任意の数のファイルを順に処理する LogFileTraceWriter のデフォルトのコンストラクタ。path および Directory Base 名を指定しないので、ファイルは、サブディレクトリを作成せずに現行ディレクトリに書き込まれます。

例外：

java.io.IOException

LogFileTraceWriter(String, String, String, String, int, int, boolean)

```
public LogFileTraceWriter(java.lang.String path,
    java.lang.String dirNameBase, java.lang.String fileNameBase,
    java.lang.String fileNameExtension, int maxFiles, int maxFileSize,
    boolean useSameDir)
    throws IOException
```

すべてのレベルのトレースに、任意の数のファイルを順に処理する LogFileTraceWriter のデフォルトのコンストラクタ。

例外：

java.io.IOException

LogFileTraceWriter(String, String, String, String, int, int, int, boolean)

```
public LogFileTraceWriter(java.lang.String path,
    java.lang.String dirNameBase, java.lang.String fileNameBase,
    java.lang.String fileNameExtension, int maxFiles, int maxFileSize,
    int maxTraceLevel, boolean useSameDir)
    throws IOException
```

任意の数のファイルを順に処理し、指定のディレクトリに保存する LogFileTraceWriter を構築します。

例外：

java.io.IOException

メソッド

doClose()

```
protected void doClose()
```

この OutputStream をクローズします。現在オープンされているすべてのログ ファイルも、同様にクローズされます。

オーバーライド：

クラス BaseTraceWriter 内の doClose

doFlush()

```
protected void doFlush()
```

オーバーライド：

クラス BaseTraceWriter 内の doFlush

doPrintln(String, int)

```
protected void doPrintln(java.lang.String message,
    int messageNumber)
```

次のクラスからコピーされた記述： com.cisco.services.tracing.BaseTraceWriter

特定のトレース機能を実現するために、BaseTraceWriter を拡張するさまざまな TraceWriter に実装する必要があります。

オーバーライド：

クラス BaseTraceWriter 内の doPrintln

getCurrentFile()

```
public int getCurrentFile()
```

戻り値：

CurrentFile プロパティ。

getFileExtension()

```
public java.lang.String getFileExtension()
```

戻り値：

FileExtension プロパティ。

getFileNameBase()

```
public java.lang.String getFileNameBase()
```

戻り値：

FileNameBase プロパティ。

getHeader()

```
public java.lang.String getHeader()
```

各ログ ファイルの先頭に書き込まれるヘッダー文字列を取得します。

戻り値：

Header プロパティ。

getMaxFiles()

```
public int getMaxFiles()
```

戻り値：

MaxFiles プロパティ。

getMaxFileSize()

```
public int getMaxFileSize()
```

戻り値：

MaxFileSize プロパティ。

setHeader(String)

```
public void setHeader(java.lang.String header)
```

各ファイルの先頭に書き込まれるヘッダー定数を設定します。トレースの書き込みは、ヘッダーが書き込まれた次の行から継続されます。`setHeader` がファイル出力の開始後に呼び出された場合、次に書き込まれるファイルから有効になります。

使用法：

```
tm = TraceManagerFactory.registerModule( this );
tw = new LogFileTraceWriter ( "trace", "log", 10, 1024*1024);
tw.setHeader ( header );
tm.getTraceWriterManager ().addTraceWriter (tw);
```

OutputStreamTraceWriter

OutputStreamTraceWriter は、TraceWriter の出力ストリームのラッパー クラスです。これにより、他の TraceWriter と共存可能なカスタム トレース クラスを簡単に追加できます。

宣言

```
public final class OutputStreamTraceWriter extends BaseTraceWriter

java.lang.Object
      |
      +--com.cisco.services.tracing.BaseTraceWriter
                |
                +--com.cisco.services.tracing.OutputStreamTraceWriter
```

実装インターフェイスの一覧

TraceWriter

メンバの概要

メンバの概要	
コンストラクタ	
	OutputStreamTraceWriter(int, OutputStream) オートフラッシュするデフォルトのコンストラクタ。
	OutputStreamTraceWriter(int, OutputStream, boolean) OutputStreamTraceWriter を作成します。
メソッド	
protected void	doClose()
protected void	doFlush()
protected void	doPrintln(String, int)
java.io.OutputStream	getOutputStream()

継承メンバの概要
クラス BaseTraceWriter から継承したメソッド
close(), doClose(), flush(), getDescription(), getEnabled(), getName(), getTraceLevels(), println(String, int), setTraceLevels(int[]), toString()
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

コンストラクタ

OutputStreamTraceWriter(int, OutputStream)

```
public OutputStreamTraceWriter(int maxTraceLevel,
    java.io.OutputStream outputStream)
```

オートフラッシュするデフォルトのコンストラクタ。

関連項目：

Trace

OutputStreamTraceWriter(int, OutputStream, boolean)

```
public OutputStreamTraceWriter(int maxTraceLevel,
    java.io.OutputStream outputStream, boolean autoFlush)
```

OutputStreamTraceWriter を作成します。

関連項目：

Trace

メソッド

doClose()

```
protected void doClose()
```

オーバーライド：

クラス BaseTraceWriter 内の doClose

doFlush()

```
protected void doFlush()
```

オーバーライド：

クラス BaseTraceWriter 内の doFlush

doPrintln(String, int)

```
protected void doPrintln(java.lang.String message,
    int messageNumber)
```

次のクラスからコピーされた記述：com.cisco.services.tracing.BaseTraceWriter

特定のトレース機能を実現するために、BaseTraceWriter を拡張するさまざまな TraceWriter に実装する必要があります。

オーバーライド：

クラス BaseTraceWriter 内の doPrintln

getOutputStream()

```
public java.io.OutputStream getOutputStream()
```

戻り値：

TraceWriter に関連付けられた出力ストリーム。

SyslogTraceWriter

SyslogTraceWriter は、BaseTraceWriter を改良したもので、トレースを syslog に送ることが可能です。Cisco の syslog 仕様では、低レベルのトレースを UDP メッセージの形式で syslog コレクタに送ります。TraceWriter では、バッファリングは行われません。SyslogTraceWriter には、println() メソッドに対して、システムで規定されているメッセージ パケット末尾の行区切りの代わりに、「¥0」を使用する点で例外があります。

宣言

```
public final class SyslogTraceWriter extends BaseTraceWriter

java.lang.Object
      |
      +--com.cisco.services.tracing.BaseTraceWriter
                |
                +--com.cisco.services.tracing.SyslogTraceWriter
```

実装インターフェイスの一覧

TraceWriter

メンバの概要

メンバの概要	
コンストラクタ	
	SyslogTraceWriter(int, String) 最大トレース レベル INFORMATIONAL の付けられたデフォルトの SyslogTraceWriter。
	SyslogTraceWriter(int, String, int) 最大トレース レベルの指定された SyslogTraceWriter。
	SyslogTraceWriter(int, String, int[]) トレース レベルの配列を指定できる SyslogTraceWriter。
メソッド	
void	doClose() ソケットをクローズします。
protected void	doPrintln(String, int) SyslogTraceWriter には、println() メソッドに対して、システムで規定されているメッセージ パケット末尾の行区切りの代わりに、「¥0」を使用する点で例外があります。メッセージの、「¥r」または「¥n」より後の部分は無視されます。
static void	main(String[])

継承メンバの概要
クラス BaseTraceWriter から継承したメソッド
close(), doClose(), flush(), getDescription(), getEnabled(), getName(), getTraceLevels(), println(String, int), setTraceLevels(int[]), toString()
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

コンストラクタ

SyslogTraceWriter(int, String)

```
public SyslogTraceWriter(int port, java.lang.String collector)
```

最大トレース レベル INFORMATIONAL の付けられたデフォルトの SyslogTraceWriter。

関連項目：

Trace

SyslogTraceWriter(int, String, int)

```
public SyslogTraceWriter(int port, java.lang.String collector,
    int maxTraceLevel)
```

最大トレース レベルの指定された SyslogTraceWriter。

関連項目：

Trace

SyslogTraceWriter(int, String, int[])

```
public SyslogTraceWriter(int port, java.lang.String collector,
    int[] traceLevels)
```

トレース レベルの配列を指定できる SyslogTraceWriter。

関連項目：

Trace

メソッド

doClose()

```
public void doClose()
```

ソケットをクローズします。

オーバーライド：

クラス BaseTraceWriter 内の doClose

doPrintln(String, int)

```
protected void doPrintln(java.lang.String message,
```

```
int messageNumber)
```

SyslogTraceWriter には、println() メソッドに対して、システムで規定されているメッセージパケット末尾の行区切りの代わりに、「¥0」を使用する点で例外があります。メッセージの、「¥r」または「¥n」より後の部分は無視されます。

オーバーライド：

クラス BaseTraceWriter 内の doPrintln

main(String[])

```
public static void main(java.lang.String[] args)
```

TraceManagerFactory

TraceManagerFactory クラスは、アプリケーションが TraceManager オブジェクトの取得に使用するクラスです。コンストラクタに渡された TraceModule は、リストに登録されます。リストは、getModules() メソッドを使用して、列挙できます。

宣言

```
public class TraceManagerFactory

java.lang.Object
      |
      +--com.cisco.services.tracing.TraceManagerFactory
```

メンバの概要

メンバの概要	
メソッド	
static java.util.Enumeration	getModules() このファクトリに登録されている TraceModules の列挙を返します。
static TraceManager	registerModule(TraceModule) TraceManager オブジェクトのインスタンスを返します。
static TraceManager	registerModule(TraceModule, String[], TraceWriterManager)) TraceManager オブジェクトのインスタンスを返します。
static TraceManager	registerModule(TraceModule, TraceWriterManager) TraceManager オブジェクトのインスタンスを返します。

継承メンバの概要
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

メソッド

getModules()

```
public static java.util.Enumeration getModules()
```

このファクトリに登録されている TraceModules の列挙を返します。

registerModule(TraceModule)

```
public static com.cisco.services.tracing.TraceManager
    registerModule(com.cisco.services.tracing.TraceModule module)
```

TraceManager オブジェクトのインスタンスを返します。包含される TraceWriterManager には、デフォルトの TraceWriter がありません。

registerModule(TraceModule, String[], TraceWriterManager)

```
public static com.cisco.services.tracing.TraceManager
    registerModule(com.cisco.services.tracing.TraceModule module,
    java.lang.String[] subFacilities,
    com.cisco.services.tracing.TraceWriterManager traceWriterManager)
```

TraceManager オブジェクトのインスタンスを返します。Trace 出力は、指定された TraceWriterManager オブジェクトへリダイレクトされます。

registerModule(TraceModule, TraceWriterManager)

```
public static com.cisco.services.tracing.TraceManager
    registerModule(com.cisco.services.tracing.TraceModule module,
    com.cisco.services.tracing.TraceWriterManager traceWriterManager)
```

TraceManager オブジェクトのインスタンスを返します。Trace 出力は、指定された TraceWriterManager オブジェクトへリダイレクトされます。

サービス トレース インターフェイスの階層

次のインターフェイス階層は、com.cisco.services.tracing パッケージに含まれています。

com.cisco.services.tracing.Trace

　com.cisco.services.tracing.ConditionalTrace

　com.cisco.services.tracing.UnconditionalTrace

com.cisco.services.tracing.TraceManager

com.cisco.services.tracing.TraceModule

com.cisco.services.tracing.TraceWriter

com.cisco.services.tracing.TraceWriterManager

Trace

Trace インターフェイスは、アプリケーションのトレースを可能にするメソッドを定義します。Trace では、Syslog Trace Logging で指定される標準トレース タイプも定義しています。Syslog では、現在 8 レベルのトレースを定義しています。メッセージの重大度は、[0-7]（0 および 7 を含む）間

の範囲の数値で、トレース内に示されます。現在、7 が HIGHEST_LEVEL で、0 が LOWEST_LEVEL のトレースです。ここでは、トレース サブシステム上で参照される全 8 レベルを static int 型として事前定義しています。

次に、トレースされる重大度を示します。

0 = EMERGENCIES、システム使用不可
1 = ALERTS、ただちに処置が必要
2 = CRITICAL、重大な状態
3 = ERROR、エラー状態
4 = WARNING、警告状態
5 = NOTIFICATION、動作は通常であるが重大な状態
6 = INFORMATIONAL、情報メッセージだけ
7 = DEBUGGING、デバッグ用メッセージ

宣言

```
public interface Trace
```

すべての既知のサブインターフェイス

ConditionalTrace, UnconditionalTrace

メンバの概要

メンバの概要	
フィールド	
static int	ALERTS アプリケーションはタスクの作業を継続できるが、すべての機能が動作可能なわけではない（リストの 1 つ以上のデバイスがアクセス不能であるが、他はアクセス可能）。 Syslog 重大度 = 1
static java.lang.String	ALERTS_TRACE_NAME ALERTS トレース レベルの文字列記述子。
static int	CRITICAL 重大な障害であり、アプリケーションはこの障害が原因で要求されたタスクを達成できない。たとえば、アプリケーションが、データベースを開いてデバイス リストを読むことができない。 Syslog 重大度 = 2
static java.lang.String	CRITICAL_TRACE_NAME CRITICAL トレース レベルの文字列記述子。
static int	DEBUGGING エラーまたはプロセスの状態に関する詳細な情報で、DEBUG モードが有効になっているときだけに生成される。 Syslog 重大度 = 7
static java.lang.String	DEBUGGING_TRACE_NAME DEBUGGING トレース レベルの文字列記述子。
static int	EMERGENCIES 緊急状態であり、システムのシャットダウンが必要。 Syslog 重大度 = 0

メンバの概要 （続き）	
static java.lang.String	EMERGENCIES_TRACE_NAME EMERGENCIES トレース レベルの文字列記述子。
static int	ERROR 何らかのエラーの状態が発生し、ユーザは、この障害の性質を理解する必要がある。 Syslog 重大度 = 3
static java.lang.String	ERROR_TRACE_NAME ERROR トレース レベルの文字列記述子。
static int	HIGHEST_LEVEL 最高位のトレース レベルであり、現在は、トレース レベル 7 の DEBUGGING。
static int	INFORMATIONAL エラー、警告、監査、またはデバッグに関係しない形式の情報。 Syslog 重大度 = 6
static java.lang.String	INFORMATIONAL_TRACE_NAME INFORMATIONAL トレース レベルの文字列記述子。
static int	LOWEST_LEVEL 最低位のトレース レベルであり、現在は、トレース レベル 0 の EMERGENCIES。
static int	NOTIFICATION NOTIFICATION は正常であるが、重大な状態を示す。 Syslog 重大度 = 5
static java.lang.String	NOTIFICATION_TRACE_NAME NOTIFICATION トレース レベルの文字列記述子。
static int	WARNING なんらかの問題が存在するが、アプリケーションのタスクの実行が妨げられてはいないという警告。 Syslog 重大度 = 4
static java.lang.String	WARNING_TRACE_NAME WARNING トレース レベルの文字列記述子。
メソッド	
java.lang.String	getName() この Trace オブジェクトの名前を返します。
java.lang.String	getSubFacility() トレースの subFacility のタイプ。
int	getType() Syslog に指定されているトレースのタイプ。
boolean	isEnabled() この Trace オブジェクトの状態を返します。
void	println(Object) Object.toString() メソッドによって返された文字列を出力し、システムでの定義に従って行を終了します。
void	println(String) Trace.print() と同じフォーマットでメッセージを出力し、システムでの定義に従って行を終了します。
void	println(String, Object) Object.toString() メソッドによって返された文字列を出力し、システムでの定義に従って行を終了します。

メンバの概要 （続き）	
void	println(String, String) Trace.print() と同じフォーマットでメッセージを出力し、システムでの定義に従って行を終了します。
void	setDefaultMnemonic(String) このトレースに出力されるすべてのメッセージにデフォルトのニーモニックを設定します。

フィールド

ALERTS

```
public static final int ALERTS
```

アプリケーションはタスクの作業を継続できるが、すべての機能が動作可能なわけではない（リストの 1 つ以上のデバイスがアクセス不能であるが、他はアクセス可能）。
Syslog 重大度 = 1

ALERTS_TRACE_NAME

```
public static final java.lang.String ALERTS_TRACE_NAME
```

ALERTS トレース レベルの文字列記述子。

CRITICAL

```
public static final int CRITICAL
```

重大な障害であり、アプリケーションはこの障害が原因で要求されたタスクを達成できない。たとえば、アプリケーションが、データベースを開いてデバイス リストを読むことができない。
Syslog 重大度 = 2

CRITICAL_TRACE_NAME

```
public static final java.lang.String CRITICAL_TRACE_NAME
```

CRITICAL トレース レベルの文字列記述子。

DEBUGGING

```
public static final int DEBUGGING
```

エラーまたはプロセスの状態に関する詳細な情報で、DEBUG モードが有効になっているときだけに生成される。
Syslog 重大度 = 7

DEBUGGING_TRACE_NAME

```
public static final java.lang.String DEBUGGING_TRACE_NAME
```

DEBUGGING トレース レベルの文字列記述子。

EMERGENCIES

```
public static final int EMERGENCIES
```

緊急状態であり、システムのシャットダウンが必要。
Syslog 重大度 = 0

EMERGENCIES_TRACE_NAME

```
public static final java.lang.String EMERGENCIES_TRACE_NAME
```

EMERGENCIES トレース レベルの文字列記述子。

ERROR

```
public static final int ERROR
```

何らかのエラーの状態が発生し、ユーザは、この障害の性質を理解する必要がある。
Syslog 重大度 = 3

ERROR_TRACE_NAME

```
public static final java.lang.String ERROR_TRACE_NAME
```

ERROR トレース レベルの文字列記述子。

HIGHEST_LEVEL

```
public static final int HIGHEST_LEVEL
```

最高位のトレース レベルであり、現在は、トレース レベル 7 の DEBUGGING。

INFORMATIONAL

```
public static final int INFORMATIONAL
```

エラー、警告、監査、またはデバッグに関係しない形式の情報。
Syslog 重大度 = 6

INFORMATIONAL_TRACE_NAME

```
public static final java.lang.String INFORMATIONAL_TRACE_NAME
```

INFORMATIONAL トレース レベルの文字列記述子。

LOWEST_LEVEL

```
public static final int LOWEST_LEVEL
```

最低位のトレース レベルであり、現在は、トレース レベル 0 の EMERGENCIES。

NOTIFICATION

```
public static final int NOTIFICATION
```

NOTIFICATION は正常であるが、重大な状態を示す。
Syslog 重大度 = 5

NOTIFICATION_TRACE_NAME

```
public static final java.lang.String NOTIFICATION_TRACE_NAME
```

NOTIFICATION トレース レベルの文字列記述子。

WARNING

```
public static final int WARNING
```

なんらかの問題が存在するが、アプリケーションのタスクの実行が妨げられてはいないという警告。
Syslog 重大度 = 4

WARNING_TRACE_NAME

```
public static final java.lang.String WARNING_TRACE_NAME
```

WARNING トレース レベルの文字列記述子。

メソッド

getName()

```
public java.lang.String getName()
```

戻り値：

この Trace オブジェクトの名前を返します。

getSubFacility()

```
public java.lang.String getSubFacility()
```

戻り値：

トレースの subFacility のタイプ。

getType()

```
public int getType()
```

戻り値：

Syslog に指定されているトレースのタイプ。DEBUGGING、INFORMATIONAL、WARNING など。

isEnabled()

```
public boolean isEnabled()
```

この Trace オブジェクトの状態を返します。デフォルトでは、Trace オブジェクトは有効になっています。つまり、常に println() メソッドのトレースを実行します。このインターフェイスでは状態を変更できませんが、このオブジェクトに、状態を変更可能な追加のインターフェイスを実装することができます。

戻り値：

トレースが有効の場合に true、そうでない場合に false。

関連項目：

ConditionalTrace

println(Object)

```
public void println(java.lang.Object object)
```

Object.toString() メソッドによって返された文字列を出力し、システムでの定義に従って行を終了します。

パラメータ：

object：出力されるオブジェクト。

println(String)

```
public void println(java.lang.String message)
```

Trace.print() と同じフォーマットでメッセージを出力し、システムでの定義に従って行を終了します。

パラメータ：

message：出力されるメッセージ。

println(String, Object)

```
public void println(java.lang.String mnemonic,
    java.lang.Object object)
```

Object.toString() メソッドによって返された文字列を出力し、システムでの定義に従って行を終了します。

パラメータ：

object：出力されるオブジェクト。

mnemonic：出力されるメッセージにマップされるニーモニック。

println(String, String)

```
public void println(java.lang.String mnemonic,
    java.lang.String message)
```

Trace.print() と同じフォーマットでメッセージを出力し、システムでの定義に従って行を終了します。

パラメータ：

message：出力されるメッセージ。

mnemonic：出力されるメッセージにマップされるニーモニック。

setDefaultMnemonic(String)

```
public void setDefaultMnemonic(java.lang.String mnemonic)
```

このトレースに出力されるすべてのメッセージにデフォルトのニーモニックを設定します。

パラメータ：

mnemonic：ニーモニック文字列。

ConditionalTrace

ConditionalTrace インターフェイスは、Trace インターフェイスを拡張したもので、この特定条件のトレースの有効または無効を可能にするメソッドを定義しています。

一般に、アプリケーションは、条件当たり 1 つの ConditionalTrace オブジェクトを取得します。この条件は、常にではなく、特定の状況（たとえば、AUDIT、INFO など）においてトレースする必要があります。

宣言

```
public interface ConditionalTrace extends Trace
```

すべてのスーパーインターフェイス

Trace

メンバの概要

メンバの概要	
メソッド	
void	disable() このトレース条件を無効にします。
void	enable() このトレース条件を有効にします。

継承メンバの概要
インターフェイス Trace から継承したフィールド
ALERTS, ALERTS_TRACE_NAME, CRITICAL, CRITICAL_TRACE_NAME, DEBUGGING, DEBUGGING_TRACE_NAME, EMERGENCIES, EMERGENCIES_TRACE_NAME, ERROR, ERROR_TRACE_NAME, HIGHEST_LEVEL, INFORMATIONAL, INFORMATIONAL_TRACE_NAME, LOWEST_LEVEL, NOTIFICATION, NOTIFICATION_TRACE_NAME, WARNING, WARNING_TRACE_NAME
インターフェイス Trace から継承したメソッド
getName(), getSubFacility(), getType(), isEnabled(), println(Object), println(String), println(String, Object), println(String, String), setDefaultMnemonic(String)

メソッド

disable()

```
public void disable()
```

このトレース条件を無効にします。

enable()

```
public void enable()
```

このトレース条件を有効にします。

UnconditionalTrace

UnconditionalTrace インターフェイスは、Trace インターフェイスを拡張したものです。このオブジェクトは Trace を拡張しているため、その状態はデフォルトで有効になっていて、変更できないことに注意してください。

一般に、アプリケーションは、条件当たり 1 つの UnconditionalTrace オブジェクトを取得します。この条件は、常にすべての状況（ERROR、FATAL など）においてトレースする必要があります。

宣言

```
public interface UnconditionalTrace extends Trace
```

すべてのスーパーインターフェイス

Trace

メンバの概要

継承メンバの概要
インターフェイス Trace から継承したフィールド
ALERTS, ALERTS_TRACE_NAME, CRITICAL, CRITICAL_TRACE_NAME, DEBUGGING, DEBUGGING_TRACE_NAME, EMERGENCIES, EMERGENCIES_TRACE_NAME, ERROR, ERROR_TRACE_NAME, HIGHEST_LEVEL, INFORMATIONAL, INFORMATIONAL_TRACE_NAME, LOWEST_LEVEL, NOTIFICATION, NOTIFICATION_TRACE_NAME, WARNING, WARNING_TRACE_NAME
インターフェイス Trace から継承したメソッド
getName(), getSubFacility(), getType(), isEnabled(), println(Object), println(String), println(String, Object), println(String, String), setDefaultMnemonic(String)

TraceManager

TraceManager インターフェイスは、アプリケーションのトレース管理を可能にするメソッドを定義します。

一般に、アプリケーションは、1 つの TraceManager オブジェクトだけを取得します。すべての Trace オブジェクトがデフォルトで作成されます。次に、Syslog の定義に基づいた事前定義の Trace を示します。

```
ConditionalTraces: INFORMATIONAL, DEBUGGING, NOTIFICATION, WARNING
UnconditionalTraces: ERROR, CRITICAL, ALERTS, EMERGENCIES
```

ファシリティ /サブファシリティ：

- **Facility**：メッセージが参照するファシリティを示す、2 つ以上の大文字からなるコードです。ファシリティは、ハードウェア デバイス、プロトコル、またはシステム ソフトウェアのモジュールなどです。

- **SubFacility**：メッセージが参照するサブファシリティを示す、2 つ以上の大文字からなるコードです。サブファシリティは、ハードウェア デバイス コンポーネント、プロトコル ユニット、またはシステム ソフトウェアのサブモジュールなどです。

デフォルトでは、8 つの Conditional と UnConditional Trace がすべて Facility に作成されます。各 subFacility にも 8 つ作成されます。たとえば、親 FACILITY で DEBUGGING トレースを使用するには、アプリケーションは、このオブジェクトの getConditionalTrace("DEBUGGING") メソッドを使用する必要があります。

たとえば、SUBFACILITY で DEBUGGING トレースを使用するには、アプリケーションで、このオブジェクトの getConditionalTrace(SUBFACILITY + "_" + "DEBUGGING") メソッドを使用するか、getConditionalTrace(SUBFACILITY, "DEBUGGING") メソッドを使用する必要があります。

システム全体の TraceWriterManager は、このインターフェイスが提供する setTraceWriterManager メソッドにより設定されます。

また、Trace Manager オブジェクトでは、enableAll() および disableAll() メソッドを使用して、すべてのトレースをアプリケーションによって有効または無効にすることが可能です。

宣言

```
public interface TraceManager
```

メンバの概要

メンバの概要	
メソッド	
void	addSubFacilities(String[]) この TraceManager/Facility の sub-Facility のセットを設定します。
void	addSubFacility(String) この TraceManager/Facility の単一の subFacility を追加します。
void	disableAll() この TraceManager により管理されるすべての Trace オブジェクトのトレースを無効にします。
void	disableTimeStamp() この TraceManager によって出力されるすべてのメッセージについて、タイムスタンプにプレフィクスを付けることを無効にします。
void	enableAll() この TraceManager によって管理されるすべての Trace オブジェクトのトレースを有効にします。
void	enableTimeStamp() この TraceManager によって出力されるすべてのメッセージについて、タイムスタンプにプレフィクスを付けることを有効にします。
ConditionalTrace	getConditionalTrace(int) この条件について、新規の ConditionalTrace オブジェクトを作成する、または既存の ConditionalTrace オブジェクトを取得します。
ConditionalTrace	getConditionalTrace(String, int) この条件および subFacility について、新規の ConditionalTrace オブジェクトを作成する、または既存の ConditionalTrace オブジェクトを取得します。

メンバの概要 （続き）	
java.lang.String	getName() この TraceManager の Facility 名を返します。
java.lang.String[]	getSubFacilities() この TraceManager/Facility の subFacility 名を返します。
java.util.Enumeration	getTraces() この TraceManager により管理される Trace オブジェクトの列挙を返します。
TraceWriterManager	getTraceWriterManager() この TraceManager により使用される TraceWriter を返します。
UnconditionalTrace	getUnconditionalTrace(int) この条件について、新規の UnconditionalTrace オブジェクトを作成するか、または既存の UnconditionalTrace オブジェクトを取得します。
UnconditionalTrace	getUnconditionalTrace(String, int) この条件および subFacility について、新規の UnconditionalTrace オブジェクトを作成するか、または既存の UnconditionalTrace オブジェクトを取得します。
void	removeTrace(Trace) Trace オブジェクトを削除します（オブジェクトがある場合）。
void	setSubFacilities(String[]) この TraceManager/Facility の sub-Facility のセットを設定します。
void	setSubFacility(String) この TraceManager/Facility の単一の subFacility を追加します。
void	setTraceWriterManager(TraceWriterManager) この TraceManager で使用する TraceWriter を設定します。

メソッド

addSubFacilities(String[])

```
public void addSubFacilities(java.lang.String[] names)
```

この TraceManager/Facility の sub-Facility のセットを設定します。

addSubFacility(String)

```
public void addSubFacility(java.lang.String name)
```

この TraceManager/Facility の単一の subFacility を追加します。

disableAll()

```
public void disableAll()
```

この TraceManager により管理されるすべての Trace オブジェクトのトレースを無効にします。

disableTimeStamp()

```
public void disableTimeStamp()
```

この TraceManager によって出力されるすべてのメッセージについて、タイムスタンプにプレフィクスを付けることを無効にします。

enableAll()

```
public void enableAll()
```

この TraceManager によって管理されるすべての Trace オブジェクトのトレースを有効にします。

enableTimeStamp()

```
public void enableTimeStamp()
```

この TraceManager によって出力されるすべてのメッセージについて、タイムスタンプにプレフィクスを付けることを有効にします。

getConditionalTrace(int)

```
public com.cisco.services.tracing.ConditionalTrace
    getConditionalTrace(int severity)
```

この条件について、新規の ConditionalTrace オブジェクトを作成する、または既存の ConditionalTrace オブジェクトを取得します。

getConditionalTrace(String, int)

```
public com.cisco.services.tracing.ConditionalTrace
    getConditionalTrace(java.lang.String subFacility, int severity)
```

この条件および subFacility について、新規の ConditionalTrace オブジェクトを作成する、または既存の ConditionalTrace オブジェクトを取得します。

getName()

```
public java.lang.String getName()
```

この TraceManager の Facility 名を返します。

getSubFacilities()

```
public java.lang.String[] getSubFacilities()
```

この TraceManager/Facility の subFacility 名を返します。

getTraces()

```
public java.util.Enumeration getTraces()
```

この TraceManager により管理される Trace オブジェクトの列挙を返します。

getTraceWriterManager()

```
public com.cisco.services.tracing.TraceWriterManager
    getTraceWriterManager()
```

この TraceManager により使用される TraceWriter を返します。

getUnconditionalTrace(int)

```
public com.cisco.services.tracing.UnconditionalTrace
    getUnconditionalTrace(int severity)
```

この条件について、新規の UnconditionalTrace オブジェクトを作成するか、または既存の UnconditionalTrace オブジェクトを取得します。

getUnconditionalTrace(String, int)

```
public com.cisco.services.tracing.UnconditionalTrace
    getUnconditionalTrace(java.lang.String subFacility, int severity)
```

この条件および subFacility について、新規の UnconditionalTrace オブジェクトを作成するか、または既存の UnconditionalTrace オブジェクトを取得します。

removeTrace(Trace)

```
public void removeTrace(com.cisco.services.tracing.Trace tc)
```

Trace オブジェクトを削除します（オブジェクトがある場合）。

setSubFacilities(String[])

```
public void setSubFacilities(java.lang.String[] names)
```

推奨されません。

TraceManager.addSubFacilities メソッドに置き換えられました。

この TraceManager/Facility の sub-Facility のセットを設定します。

setSubFacility(String)

```
public void setSubFacility(java.lang.String name)
```

推奨されません。

TraceManager.addSubFacility メソッドに置き換えられました。

この TraceManager/Facility の単一の subFacility を追加します。

setTraceWriterManager(TraceWriterManager)

```
public void
    setTraceWriterManager(com.cisco.services.tracing.TraceWriterManager twm)
```

この TraceManager で使用する TraceWriter を設定します。

TraceModule

TraceModule インターフェイスは、2 つの目的で使用されます。第 1 に、アプリケーション上で使用中の他のパッケージが使用している TraceManager オブジェクトを、アプリケーションが検出できるようにします。第 2 に、TraceManagerFactory に登録するアプリケーションは、このインターフェイスを実装して、アプリケーション自体の ID を示す必要があります。

宣言

```
public interface TraceModule
```

すべての既知のサブインターフェイス

com.cisco.jtapi.extensions.CiscoJtapiPeer

メンバの概要

メンバの概要	
メソッド	
TraceManager	getTraceManager() オブジェクトがトレースに使用している TraceManager を返します。
java.lang.String	getTraceModuleName() モジュール名を返します。

メソッド

getTraceManager()

```
public com.cisco.services.tracing.TraceManager getTraceManager()
```

オブジェクトがトレースに使用している TraceManager を返します。

getTraceModuleName()

```
public java.lang.String getTraceModuleName()
```

モジュール名を返します。

TraceWriter

TraceWriter インターフェイスは、トレース メッセージ出力の abstruct クラスです。TraceWriter は、enabled メソッドを使用して、print メソッドや println メソッドでの出力を有効化・無効化します。TraceWriter のユーザは、print メソッドおよび println メソッドを呼び出す必要があるかどうかの指標として、getEnabled メソッドが返す値を使用する必要があります。

宣言

```
public interface TraceWriter
```

すべての既知のサブインターフェイス

TraceWriterManager

既知の実装クラスの一覧

BaseTraceWriter

メンバの概要

メンバの概要	
メソッド	
void	close() この TraceWriter によって関連付けられたすべてのリソースを解放します。
void	flush() println メソッドを使用して、出力されたすべてのメッセージを強制出力します。
java.lang.String	getDescription()
boolean	getEnabled() println メソッドによって出力されるものがあるかどうかを返します。
java.lang.String	getName()
int[]	getTraceLevels()
void	println(String, int) 指定された文字列とそれに続く改行を出力します。TraceWriter の具象 クラスでは、重大度を使用して、特定のストリームからのメッセージをブロックします。
void	setTraceLevels(int[]) この TraceWriter によりトレースされるトレース レベルを設定します。

メソッド

close()

```
public void close()
```

この TraceWriter によって関連付けられたすべてのリソースを解放します。

flush()

```
public void flush()
```

println メソッドを使用して、出力されたすべてのメッセージを強制出力します。

getDescription()

```
public java.lang.String getDescription()
```

戻り値：

この TraceWriter の簡単な説明。

getEnabled()

```
public boolean getEnabled()
```

println メソッドによって出力されるものがあるかどうかを返します。クローズされた TraceWriter のこのメソッドは、常に false を返します。

戻り値：

TraceWriter が有効の場合に true、そうでない場合に false。

getName()

```
public java.lang.String getName()
```

戻り値：

この TraceWriter の名前を返します。

getTraceLevels()

```
public int[] getTraceLevels()
```

戻り値：

この TraceWriter によりトレースされるトレース レベルの配列。

println(String, int)

```
public void println(java.lang.String message, int severity)
```

指定された文字列とそれに続く改行を出力します。TraceWriter の具象クラスでは、重大度を使用して、特定のストリームからのメッセージをブロックします。各 TraceWriter は、自分が実行する必要のある最高位レベルのトレース レベルを認知しています。

パラメータ：

`message`：出力されるメッセージ。

`severity`：トレースの重大度。

関連項目：

Trace

setTraceLevels(int[])

```
public void setTraceLevels(int[] levels)
```

この TraceWriter によりトレースされるトレース レベルを設定します。

パラメータ：

`int[]`：レベル。

関連項目：

Trace

TraceWriterManager

TraceWriterManager には、トレースの実装に使用される TraceWriter オブジェクトのリストを保持します。リストは、起動時に読み込まれる .ini ファイルの設定によって初期化されます。LogFileTraceWriter、ConsoleTraceWriter および SyslogTraceWriter が利用可能です。ユーザは、ユーザ実装の TraceWriter[] を設定して、あるいは既存の TraceWriter に追加して、既存の TraceWriter をオーバーライドすることができます。これにより、既存の TraceWriter と併存して機能する、他の TraceWriter を追加することが可能です。

宣言

```
public interface TraceWriterManager extends TraceWriter
```

すべてのスーパーインターフェイス

TraceWriter

メンバの概要

メンバの概要	
メソッド	
void	addTraceWriter(TraceWriter) 別の TraceWriter を配列に追加します。
TraceWriter[]	getTraceWriters()
void	removeTraceWriter(TraceWriter) Manager の配列から TraceWriter を取り除きます。
void	setTraceWriters(TraceWriter[]) 実装では、このメソッドを使用して、提供された TraceWriter をオーバーライドまたは拡張できます。

継承メンバの概要
インターフェイス TraceWriter から継承したメソッド
close(), flush(), getDescription(), getEnabled(), getName(), getTraceLevels(), println(String, int), setTraceLevels(int[])

メソッド

addTraceWriter(TraceWriter)

```
public void
    addTraceWriter(com.cisco.services.tracing.TraceWriter traceWriter)
```

別の TraceWriter を配列に追加します。

パラメータ：

TraceWriter：リストに追加される TraceWriter。

getTraceWriters()

```
public com.cisco.services.tracing.TraceWriter[] getTraceWriters()
```

戻り値：

Manager の TraceWriter の配列。

removeTraceWriter(TraceWriter)

```
public void
    removeTraceWriter(com.cisco.services.tracing.TraceWriter traceWriter)
```

Manager の配列から TraceWriter を取り除きます。

setTraceWriters(TraceWriter[])

```
public void
    setTraceWriters(com.cisco.services.tracing.TraceWriter[] traceWriters)
```

実装では、このメソッドを使用して、提供された TraceWriter をオーバーライドまたは拡張できます。

パラメータ：

`set`：TraceWriter の配列を設定します。

トレースの実装クラスの階層

次のトレースの実装クラスの階層は、com.cisco.services.tracing.implementation パッケージに含まれています。

java.lang.Object

com.cisco.services.tracing.implementation.TraceImpl（com.cisco.services.tracing.Trace を実装）

com.cisco.services.tracing.implementation.ConditionalTraceImpl（com.cisco.services.tracing.ConditionalTrace を実装）

com.cisco.services.tracing.implementation.UnconditionalTraceImpl（com.cisco.services.tracing.UnconditionalTrace を実装）

com.cisco.services.tracing.implementation.TraceManagerImpl（com.cisco.services.tracing.TraceManager を実装）

com.cisco.services.tracing.implementation.TraceWriterManagerImpl（com.cisco.services.tracing.TraceWriterManager を実装）

TraceImpl

宣言

public abstract class **TraceImpl**

extends java.lang.Object

implements Trace

実装インターフェイスの一覧

Trace

メソッド

println

public final void **println**(java.lang.String message)

次のインターフェイスからコピーされた記述：Trace

Trace.print() と同じフォーマットでメッセージを出力し、システムでの定義に従って行を終了します。

定義：

インターフェイス Trace の println

パラメータ：

message：出力されるメッセージ。

println

public final void println(java.lang.String mnemonic,
java.lang.String message)

次のインターフェイスからコピーされた記述：Trace

Trace.print() と同じフォーマットでメッセージを出力し、システムでの定義に従って行を終了します。

定義：

インターフェイス Trace の println

パラメータ：

mnemonic：出力されるメッセージにマップされるニーモニック。

message：出力されるメッセージ。

println

public final void **println**(java.lang.Object object)

次のインターフェイスからコピーされた記述：Trace

Object.toString() メソッドによって返された文字列を出力し、システムでの定義に従って行を終了します。

定義：

インターフェイス Trace の println

パラメータ：

object：出力されるオブジェクト。

println

public final void **println**(java.lang.String mnemonic,
java.lang.Object object)

次のインターフェイスからコピーされた記述：Trace

Object.toString() メソッドによって返された文字列を出力し、システムでの定義に従って行を終了します。

定義：

インターフェイス Trace の println

パラメータ：

mnemonic：出力されるメッセージにマップされるニーモニック。

object：出力されるオブジェクト。

getName

public final java.lang.String **getName**()

次のインターフェイスからコピーされた記述：Trace

この Trace オブジェクトの名前を返します。

定義：

インターフェイス Trace の getName

戻り値：

この Trace オブジェクトの名前を返します。

setDefaultMnemonic

public final void **setDefaultMnemonic**(java.lang.String mnemonic)

次のインターフェイスからコピーされた記述：Trace

このトレースに出力されるすべてのメッセージにデフォルトのニーモニックを設定します。

定義：

インターフェイス Trace の setDefaultMnemonic

パラメータ：

mnemonic：ニーモニック文字列。

getType

public int **getType**()

次のインターフェイスからコピーされた記述：Trace

トレースのタイプを返します。

定義：

インターフェイス Trace の getType

戻り値：

Syslog に指定されているトレースの重大度。DEBUGGING、INFORMATIONAL、WARNING など。

getSubFacility

public java.lang.String **getSubFacility**()

次のインターフェイスからコピーされた記述：Trace

トレースの subFacility を返します。

定義：

インターフェイス Trace の getSubFacility

戻り値：

トレースの subFacility のタイプ。

継承したメソッド

isEnabled()

ConditionalTraceImpl

宣言

public final class **ConditionalTraceImpl**
extends TraceImpl
implements ConditionalTrace

実装インターフェイスの一覧

ConditionalTrace、Trace

メソッド

enable

public void **enable**()

次のインターフェイスからコピーされた記述：ConditionalTrace

このトレース条件を有効にします。

定義：

インターフェイス ConditionalTrace の enable

disable

public void **disable**()

次のインターフェイスからコピーされた記述：ConditionalTrace

このトレース条件を無効にします。

定義：

インターフェイス ConditionalTrace の disable

isEnabled()

public boolean **isEnabled**()

次のインターフェイスからコピーされた記述：Trace

この Trace オブジェクトの状態を返します。デフォルトでは、Trace オブジェクトは有効になっています。つまり、常に println() メソッドのトレースを実行します。このインターフェイスでは状態を変更できませんが、このオブジェクトに、状態を変更可能な追加のインターフェイスを実装することができます。

定義：

インターフェイス Trace の isEnabled

戻り値：

トレースが有効の場合に true、そうでない場合に false。

関連項目：

ConditionalTrace

継承したメソッド

クラス java.lang.Object から継承したメソッドは、clone、equals、finalize、getClass、hashCode、notify、notifyAll、toString、wait、wait、wait です。

UnconditionalTraceImpl

宣言

public final class **UnconditionalTraceImpl**

extends TraceImpl

implements UnconditionalTrace

実装インターフェイスの一覧

Trace、UnconditionalTrace

メソッド

isEnabled()

public boolean **isEnabled**()

次のインターフェイスからコピーされた記述：Trace

この Trace オブジェクトの状態を返します。デフォルトでは、Trace オブジェクトは有効になっています。つまり、常に println() メソッドのトレースを実行します。このインターフェイスでは状態を変更できませんが、このオブジェクトに、状態を変更可能な追加のインターフェイスを実装することができます。

定義：

インターフェイス Trace の isEnabled

戻り値：

トレースが有効の場合に true、そうでない場合に false。

関連項目：

ConditionalTrace

継承したメソッド

クラス java.lang.Object から継承したメソッドは、clone、equals、finalize、getClass、hashCode、notify、notifyAll、toString、wait、wait、wait です。

TraceManagerImpl

TraceManagerImpl クラスは TraceManager インターフェイスを実装しています。

宣言

public class **TraceManagerImpl** extends java.lang.Object

java.lang.Object
|
+--**com.cisco.services.tracing.TraceManagerImpl**

実装インターフェイスの一覧

TraceManager

コンストラクタ

public **TraceManagerImpl**(java.lang.String moduleName, java.lang.String[] subFacilities, TraceWriterManager traceWriterManager)

public **TraceManagerImpl**(java.lang.String moduleName, TraceWriterManager traceWriterManager)

メソッド

getConditionalTrace

public ConditionalTrace **getConditionalTrace**(int severity)

次のインターフェイスからコピーされた記述：TraceManager

この条件について、新規の ConditionalTrace オブジェクトを作成する、または既存の ConditionalTrace オブジェクトを取得します。

定義：

インターフェイス TraceManager の getConditionalTrace

getConditionalTrace

public ConditionalTrace getConditionalTrace(java.lang.String subFacility, int severity)

次のインターフェイスからコピーされた記述：TraceManager

この条件および subFacility について、新規の ConditionalTrace オブジェクトを作成する、または既存の ConditionalTrace オブジェクトを取得します。

定義：

インターフェイス TraceManager の getConditionalTrace

getUnconditionalTrace

public UnconditionalTrace getUnconditionalTrace(int severity)

次のインターフェイスからコピーされた記述：TraceManager

この条件について、新規の UnconditionalTrace オブジェクトを作成するか、または既存の UnconditionalTrace オブジェクトを取得します。

定義：

インターフェイス TraceManager の getUnconditionalTrace

getUnconditionalTrace

public UnconditionalTrace getUnconditionalTrace(java.lang.String subFacility, int severity)

次のインターフェイスからコピーされた記述：TraceManager

この条件および subFacility について、新規の UnconditionalTrace オブジェクトを作成するか、または既存の UnconditionalTrace オブジェクトを取得します。

定義：

インターフェイス TraceManager の getUnconditionalTrace

getTraceWriterManager

public TraceWriterManager getTraceWriterManager()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager により使用される TraceWriter を返します。

定義：

インターフェイス TraceManager の getTraceWriterManager

setTraceWriterManager

public void setTraceWriterManager(TraceWriterManager out)

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager で使用する TraceWriter を設定します。

定義：

インターフェイス TraceManager の setTraceWriterManager

removeTrace

public void removeTrace(Trace tc)

次のインターフェイスからコピーされた記述：TraceManager

Trace オブジェクトを削除します（オブジェクトがある場合）。

定義：

インターフェイス TraceManager の removeTrace

getTraces()

public java.util.Enumeration getTraces()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager により管理される Trace オブジェクトの列挙を返します。

定義：

インターフェイス TraceManager の getTraces

enableAll

public void enableAll()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager によって管理されるすべての Trace オブジェクトのトレースを有効にします。

定義：

インターフェイス TraceManager の enableAll

disableAll

public void disableAll()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager により管理されるすべての Trace オブジェクトのトレースを無効にします。

定義：

インターフェイス TraceManager の disableAll

getName

public java.lang.String getName()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager の Facility 名を返します。

定義：

インターフェイス TraceManager の getName

enableTimeStamp

public void enableTimeStamp()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager によって出力されるすべてのメッセージについて、タイムスタンプにプレフィクスを付けることを有効にします。

定義：

インターフェイス TraceManager の enableTimeStamp

disableTimeStamp

public void disableTimeStamp()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager によって出力されるすべてのメッセージについて、タイムスタンプにプレフィクスを付けることを無効にします。

定義：

インターフェイス TraceManager の disableTimeStamp

getSubFacilities

public java.lang.String[] getSubFacilities()

この TraceManager/Facility の subFacility 名を返します。

定義：

インターフェイス TraceManager の getSubFacilities

addSubFacilities

public void addSubFacilities(java.lang.String[] names)

この TraceManager/Facility の subFacility を追加します。

定義：

インターフェイス TraceManager の addSubFacilities

addSubFacility

public void addSubFacility(java.lang.String name)

この TraceManager/Facility の subFacility を追加します。

定義：

インターフェイス TraceManager の addSubFacility

非推奨のメソッド

getSubFacilities(java.lang.String[] names)

addSubFacilties(String[]) に置き換えられました。

setSubFacility(java.lang.String name)

addSubFacility(String) に置き換えられました。

継承したメソッド

クラス java.lang.Object から継承したメソッドは、clone、equals、finalize、getClass、hashCode、notify、notifyAll、toString、wait、wait、wait です。

TraceWriterManagerImpl

TraceWriterManager には、トレースの実装に使用される TraceWriter オブジェクトのリストを保持します。リストは、始動時に、.ini ファイル内のスイッチによって入力されます。LogFileTraceWriter、ConsoleTraceWriter および SyslogTraceWriter が利用可能です。ユーザは、ユーザ実装の TraceWriter[] を設定して、あるいは既存の TraceWriter に追加して、既存の TraceWriter をオーバーライドすることができます。これにより、既存の TraceWriter と併存して機能する、他の TraceWriter を追加することが可能です。

（注） クラス java.lang.Object から継承したメソッドは、clone、equals、finalize、getClass、hashCode、notify、notifyAll、toString、wait、wait、wait です。

宣言

public class TraceWriterManagerImpl extends java.lang.Object implements TraceWriterManager

java.lang.Object

com.cisco.services.tracing.implementation.TraceWriterManagerImpl

実装インターフェイスの一覧

TraceWriter、TraceWriterManager

コンストラクタ

TraceWriterManagerImpl

public TraceWriterManagerImpl()

長さゼロの TraceWriter 配列で TraceWriterManagerImpl を作成します。

メソッド

setTraceWriters

public void setTraceWriters(TraceWriter[] traceWriters)

既存の TraceWriter を新しいユーザが指定したセットで上書きします。

定義：

インターフェイス TraceWriterManager の setTraceWriters

パラメータ：

traceWriters：TraceWriter の配列です。

getTraceWriters()

public TraceWriter[] getTraceWriters()

現在使用されている TraceWriter の配列を返します。

定義：

インターフェイス TraceWriterManager の getTraceWriters

戻り値：

Manager の TraceWriter の配列。

addTraceWriter

public void addTraceWriter(TraceWriter tw)

この TraceWriter を TraceWriter の配列に追加します。

定義：

インターフェイス TraceWriterManager の addTraceWriter

パラメータ：

tw：リストに追加される TraceWriter です。

removeTraceWriter

public void removeTraceWriter(TraceWriter tw)

TraceWriter の配列から Tracewriter を削除します。

定義：

インターフェイス TraceWriterManager の removeTraceWriter

println

public void println(java.lang.String message, int severity)

すべてのトレースがこのメソッドを呼び出します。トレースによって、メッセージとともに重大度が指定されます。トレースが TraceWriter の重大度のしきい値以下になる場合があります。例：重大度のしきい値が INFORMATIONAL (level = 6) DEBUG に設定されている場合、トレースは TraceWriter によって渡されません。重大度レベルは TraceWriter のコンストラクタで設定されます。

定義：

インターフェイス TraceWriter の println

パラメータ：

message：出力されるメッセージ。

severity：トレースの重大度。

関連項目：

Trace

フラッシュ

public void flush()

次のインターフェイスからコピーされた記述：TraceWriter

println メソッドを使用して、出力されたすべてのメッセージを強制出力します。

定義：

インターフェイス TraceWriter の flush

close()

public void close()

次のインターフェイスからコピーされた記述：TraceWriter

この TraceWriter によって関連付けられたすべてのリソースを解放します。

定義：

インターフェイス TraceWriter の close

getEnabled

public boolean getEnabled()

基になる TraceWriter のいずれかが有効になっている場合は true を返し、そうでない場合は false を返します。

定義：

インターフェイス TraceWriter の getEnabled

戻り値：

TraceWriter が有効の場合に true、そうでない場合に false。

getName

public java.lang.String getName()

定義：

インターフェイス TraceWriter の getName

戻り値：

この TraceWriter の名前を返します。

getDescription

public java.lang.String getDescription()

定義：

インターフェイス TraceWriter の getDescription

戻り値：

この TraceWriter の簡単な説明。

setTraceLevels

public void setTraceLevels(int[] levels)

TraceWriterManager はこのメソッドに対して何も行いません。

定義：

インターフェイス TraceWriter の setTraceLevels

パラメータ：

Levels：トレース レベルの配列。

関連項目：

Trace

getTraceLevels

public int[] getTraceLevels()

traceLevel が個々の TraceWriter で維持されるため、TraceWriterManager は null を返します。

定義：

インターフェイス TraceWriter の getTraceLevels

戻り値：

null