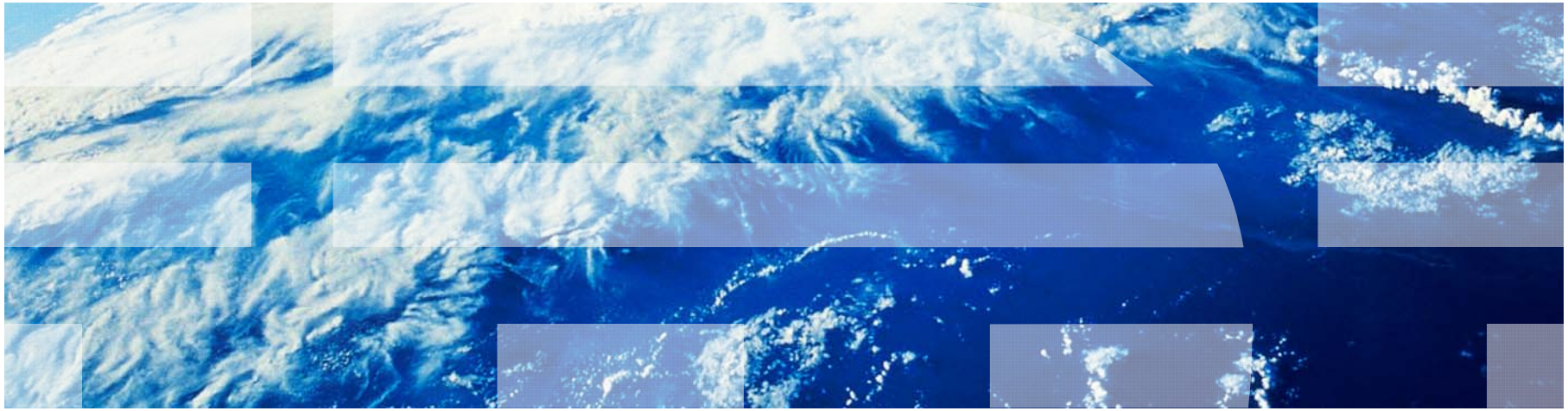
Health Center（IBM Java診断ツール）利用ガイド



更新履歴

- 2010/09/30　第1版公開
- 2011/02/25　改定第2版公開
  - Health Center V1.3 の情報を追加
  - Windows で Java6 I/O パースペクティブを使用する際の注意点を追加
  - GCポリシーがgenconの場合の、GCパースペクティブの例を追加
  - 複数JVMモニタリングのチャートを追加
  - 参照情報を追加
- 2012/10/31　改訂版第3版公開
  - Health Center V2.0の情報を追加

目次

- この資料では、IBM Javaの診断ツールであるHealth Centerについてご紹介します

1. Health Center概要

Health Centerの機能概要

- Health Centerは、IBM Java の稼動状況をモニタリング、プロファイリングできる診断ツールです
  以下のようなJavaアプリケーションの問題解析に役立てることができます
  - パフォーマンス分析
    - Javaメソッドプロファイリング
    - ロック分析
    - ガーベッジ・コレクション（以下GC）分析（GCの稼動状況の確認）
  - メモリ使用状況（メモリリークがネイティブ・ヒープで発生しているかJVMヒープで発生しているか、など）
  - JVMの稼働環境（稼動するOS、JVMの引数、など）
  - クラスのロード、オブジェクト・アロケーションの状況の確認
  - ファイルI/Oのボトルネックの有無の確認
  - スレッド数やスレッド状態の確認(V2.0～)
  - メソッド呼び出し時間の確認やメソッド使用状況の分析(V2.0～)
  - WebSphere Real Time の状況確認

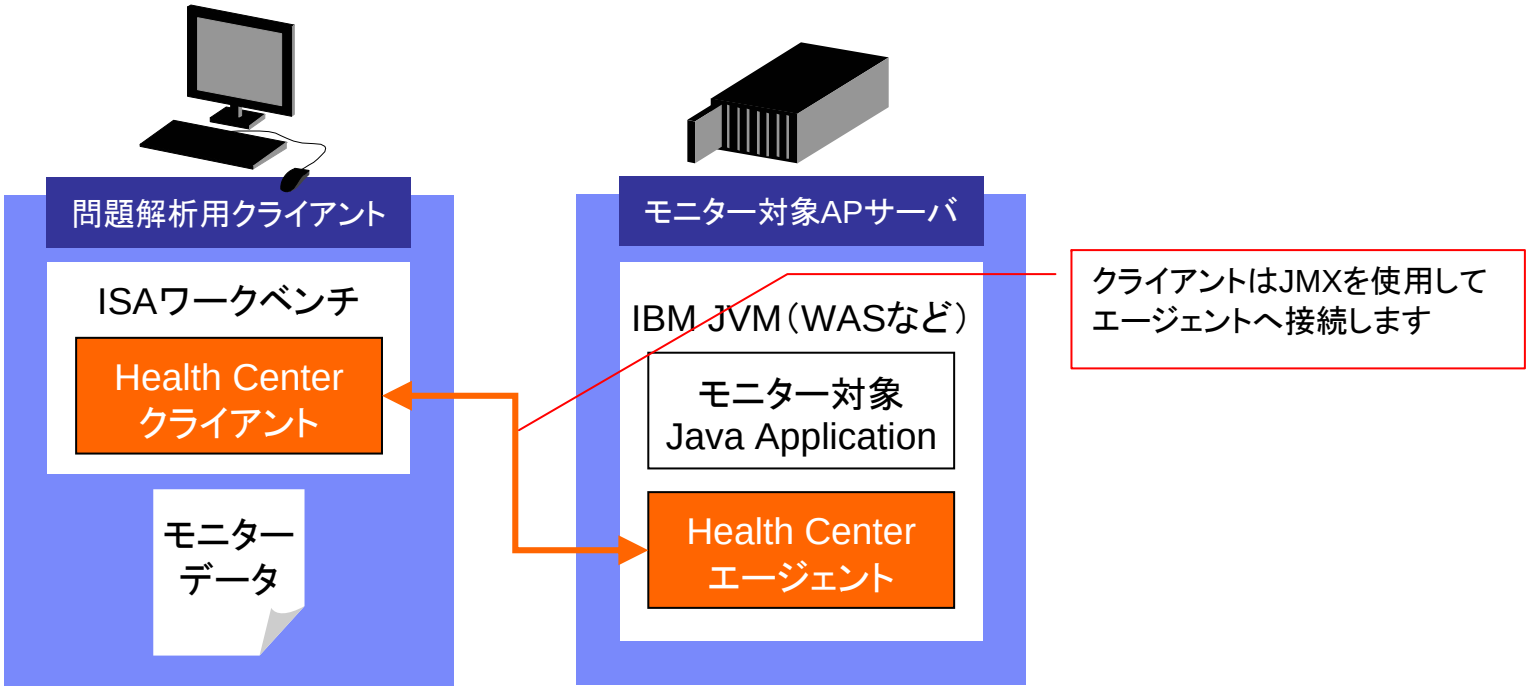
- モニター結果から考えられる、推奨設定やチューニング方法も表示されます

- モニターの対象のJavaの稼動状況をリアルタイムでモニタリングできます
  また、稼動状況をファイルに保管することも可能です

- モニター対象は、WebSphere上のアプリケーション以外にも、IBM Javaであれば製品は限定されません

Health Center のコンポーネント

- Health Centerは、以下の二つのコンポーネントから構成されます
  - Health Centerクライアント
    - IBM Support Assistant（ISA：IBMの無償問題判別ツール）ワークベンチの一部としてインストールします
    - JVMの状況をGUIベースで確認できるEclipseベースのツールです
  - Health Centerエージェント
    - モニター対象のJavaアプリケーションの情報をHealth Centerクライアントへ提供します



Health Center V2.0 前提条件

- プラットフォーム要件
  - Health Centerクライアント
    - Microsoft® Windows® またはLinux® x86
    - Health Centerクライアントは、Eclipse RCP(Rich Client Platform)をベースにしているため、最小オペレーティング・システム要件はEclipse RCPプロジェクトと同じです。詳細はhttp://www.eclipse.org/documentation/ を参照してください
  - Health Centerエージェント
    - Health Centerエージェントのバージョンとモニター対象JVMのバージョンによって、使用できるHealth Centerの機能が異なります
    - モニター対象JVMのバージョン
      - 以下のバージョンが、モニター対象JVMとして推奨されています
        - Java 5 SR10 以降
        - Java 6 SR5 以降
        - Java 6 with the IBM J9 2.6 virtual machine
        - Java 6 with the IBM J9 2.6 virtual machine, SR1
        - Java 6.0.1 for z/OS
        - Java 6.0.1 for z/OS, SR1
        - Java 7
      - 以下のバージョンは、パフォーマンスに影響があり、かつエージェントがファイルを生成するため、実働環境での使用は推奨されていません
        - Java 5 SR8 ～ Java 5 SR10 より下位
        - Java 6 SR1 ～ Java 6 SR5 より下位
      - WebSphere® Real Time for Linux® V2 SR2 (APAR IZ61672 以降のサービス更新適用済み)
      - SUNとHPは対象外です
    - Health Centerエージェントのバージョン
      - JREのレベルによってはデフォルトでエージェントが同梱されていますが、より多くの機能を使用するためには最新のエージェントをインストールしてください。JREレベルでの機能の違いは、マニュアルの「プラットフォーム要件」をご確認ください

- Health Center V1.2.1では、z/OS® 31 ビットまたは z/OS 64 ビットのプラットフォーム用のネイティブ・メモリー・パースペクティブ・ビューは用意されていません
- Health Center V2.0では、AIX® 32 ビットまたは AIX 64 ビットの PPC プラットフォーム、あるいは z/OS® 31 ビットまたは z/OS 64 ビットのプラットフォーム用のネイティブ・メモリー・パースペクティブ・ビューは用意されていません

- 2010/11 に、Health Center V1.3 がリリースされました
- 2011/12に、Health Center V2.0がリリースされました

Health Center V2.0 新機能

- **スレッドパースペクティブが新たに追加**
  - 現在のライブスレッド数、スレッドの状態（ブロックなど）を表示できます
- **メソッド・トレースパースペクティブが新たに追加**
  - メソッド呼び出しの正確な時間、および実行中のスレッドにおけるメソッド使用状況の分析を表示できます
- **ネイティブ・メモリーパースペクティブの機能拡張**
  - ネイティブ・メモリーを使用している JVM 領域の明細（クラスローダーやJIT等）を確認できます
- **ガーベッジ・コレクションパースペクティブの機能拡張**
  - 過剰なオブジェクト割り当てやサイズの大きなオブジェクトを割り当てているコードを検出できます
- **Health Centerエージェントのlate attach機能**
  - 稼働中のJVMに対して、Health Centerエージェントを開始できます
- **ヘッドレスモードによるデータ収集の機能拡張**
  - クライアント接続なしでサーバー側でデータを収集する場合に、データ収集時間や休止時間などを構成することで部分的にデータを収集できます
- **Health Centerクライアントによる冗長GCデータのファイル書き込み機能**
  - Health Centerクライアントから対象JVMに対して冗長GCデータ収集をONにすることができます
- **Health Centerクライアントとエージェント間のSSL接続**
  - クライアントとエージェント間の接続を暗号化することができます

2. Health Center が提供するパースペクティブ

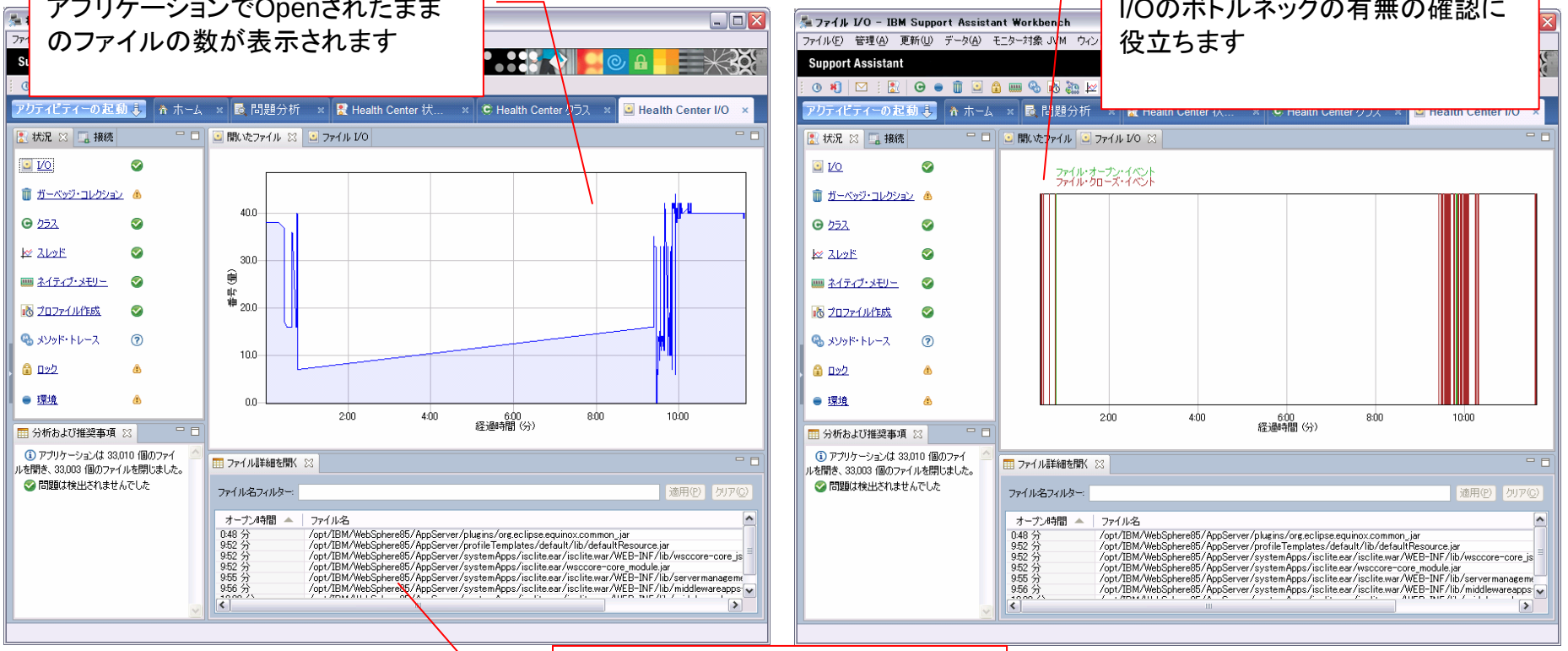
Health Center が提供するパースペクティブ

- Health Center クライアントには、以下のようなパースペクティブが用意されており、それぞれの情報をグラフィカルに確認することができます
- Health Center クライアントを起動すると表示される「状況」ビューから、各パースペクティブを起動できます
- グラフで表示されるものは全て、右クリック⇒「単位の変更」で、X軸/Y軸の単位を変更できます

	1	I/O
	2	ガーベッジ・コレクション
	3	クラス
V2.0～	4	スレッド
	5	ネイティブ・メモリー
	6	プロファイル作成
V2.0～	7	メソッド・トレース
	8	ロック
	9	環境
	10	WebSphere Real Time(本ドキュメントではご紹介していません)

1. I/Oパースペクティブ

- I/Oパースペクティブでは、ファイルI/O状況がモニタリングできます。アプリケーションでファイル・ハンドルのクローズ有無を確認することができます
- I/Oパースペクティブを選択すると以下の画面が表示されます



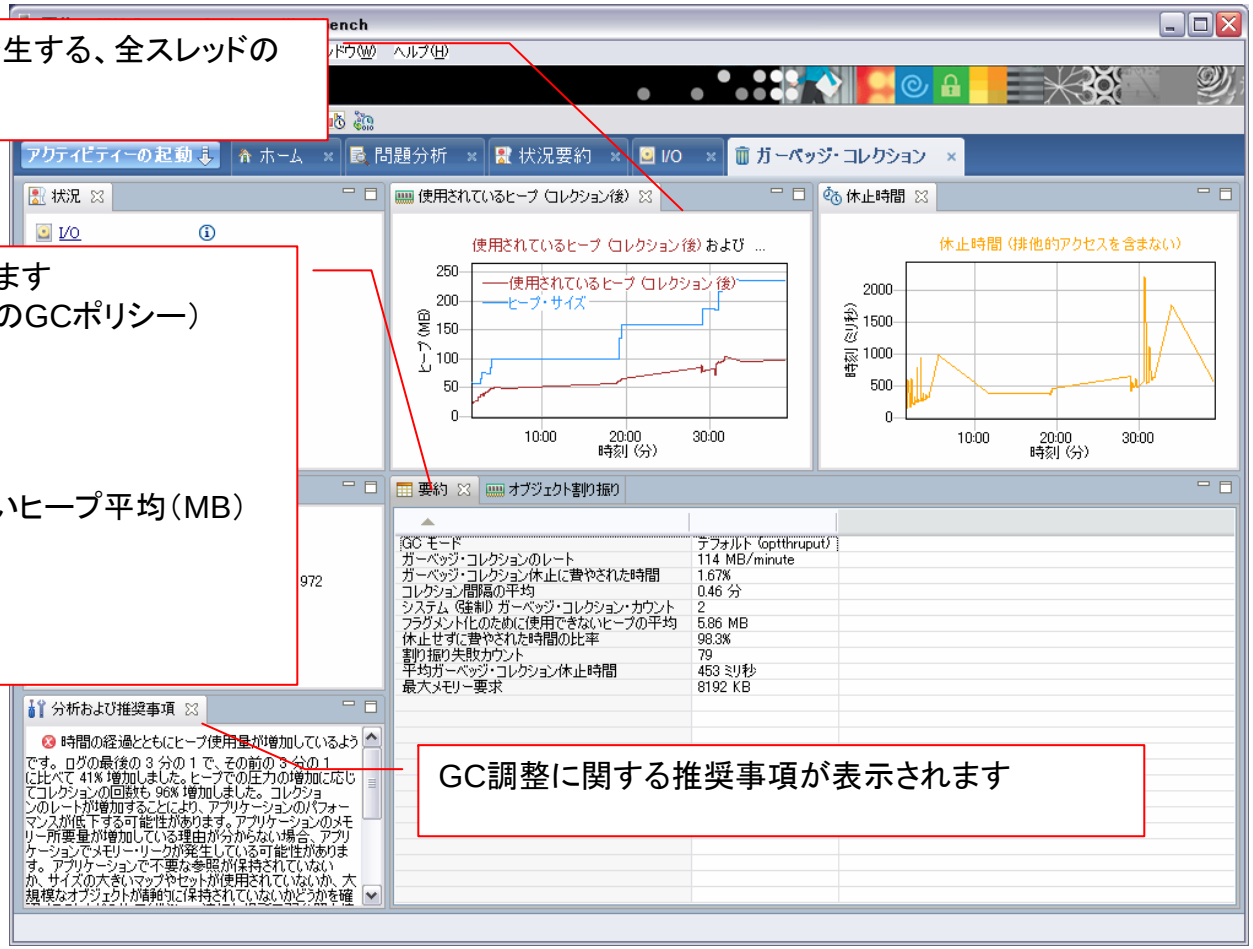
2. ガーベッジ・コレクション・パースペクティブ

- ガーベッジ・コレクション・パースペクティブでは、ガーベッジ・コレクション（以下、GC）の実行状況を確認することができます

■GCポリシーがoptthruputのケース

・ヒープの使用量やGC実行により発生する、全スレッドの停止時間がグラフで表示されます

GC実行状況のサマリーが表示されます
・GCモード（optthruput/genconなどのGCポリシー）
・GCのレート（MB/minute）
・GC休止に費やされた時間（%）
・GC実行間隔の平均（分）
・強制GC回数
・フラグメント化のために使用できないヒープ平均（MB）
・休止しない時間の比率（%）
・割り振り失敗回数
・平均GC休止時間（ミリ秒）
・最大メモリー要求（KB）



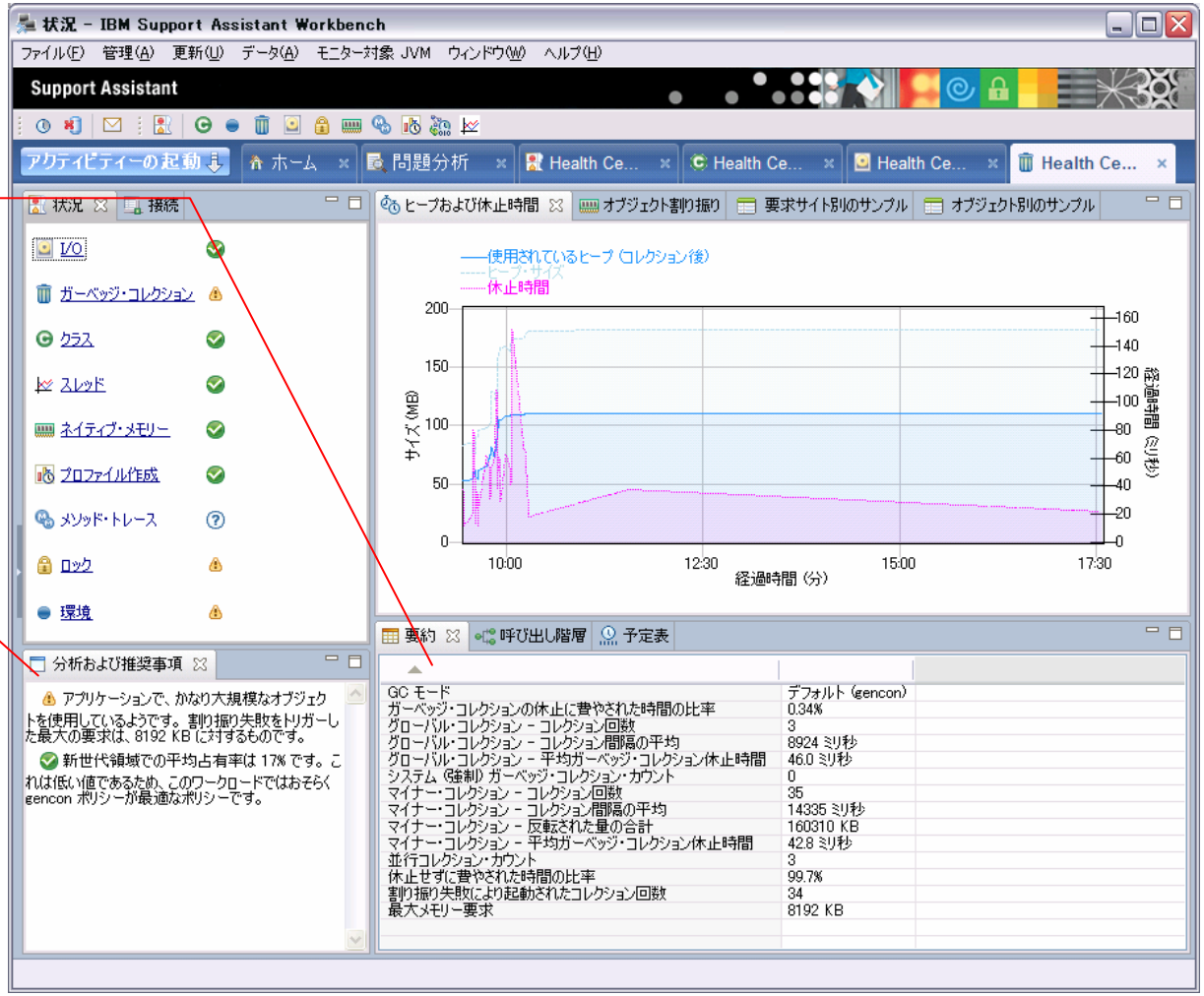
GC調整に関する推奨事項が表示されます

2. ガーベッジ・コレクション・パースペクティブ

■GCポリシーがgenconのケース

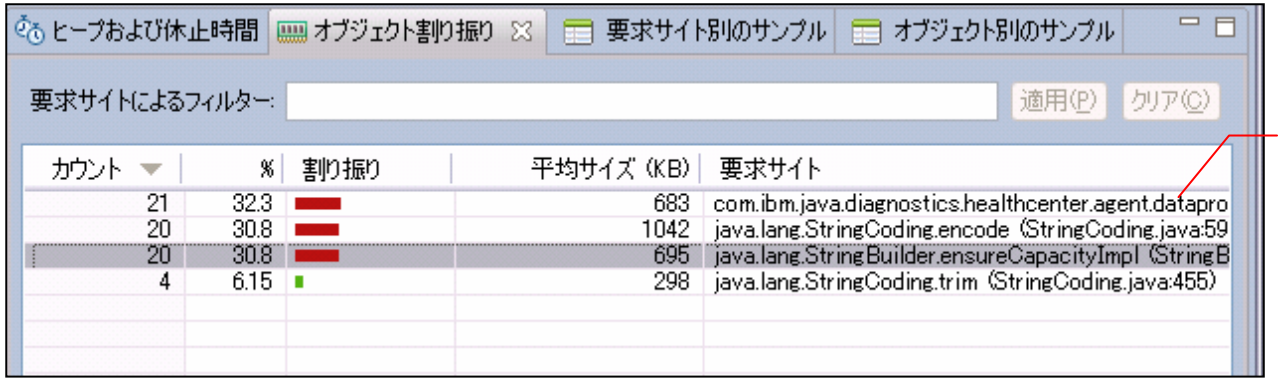
GCポリシーがgenconの場合は、グローバル・コレクションとマイナー・コレクションが別々にサマライズされます。

GCポリシーがgenconの場合は、旧世代領域（Old領域）、新世代領域（New領域）の分析、推奨事項が表示されます

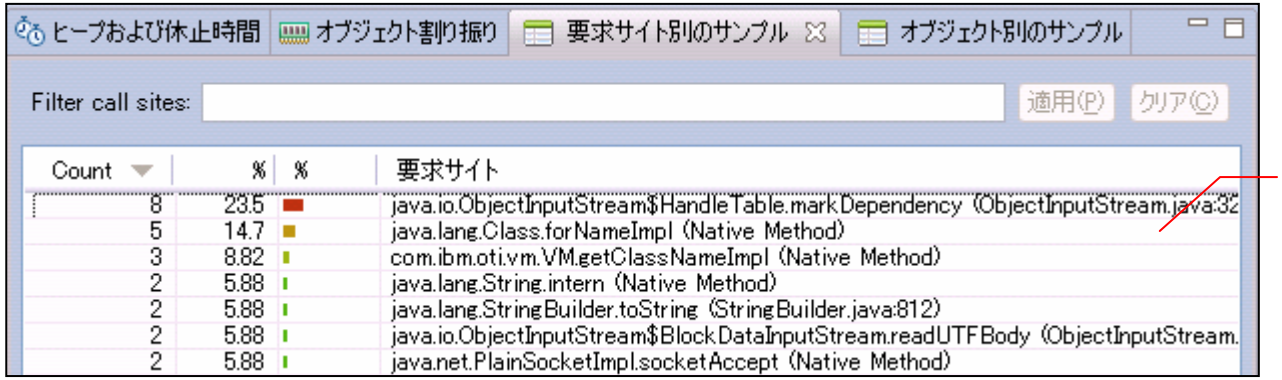


2. ガーベッジ・コレクション・パースペクティブ

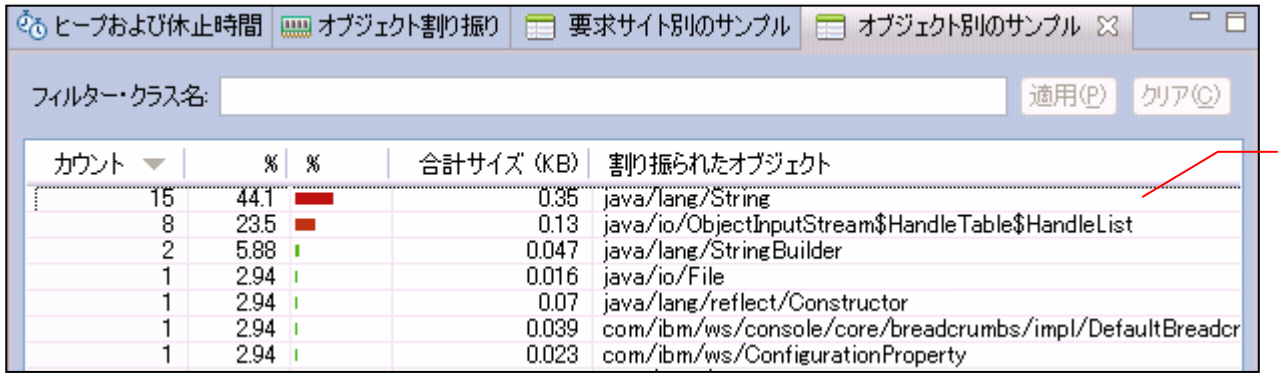
V2.0～



サイズの大きいオブジェクトを割り振っているコードを検出できます



割り振り要求を最も頻繁に実行しているコードを検出できます

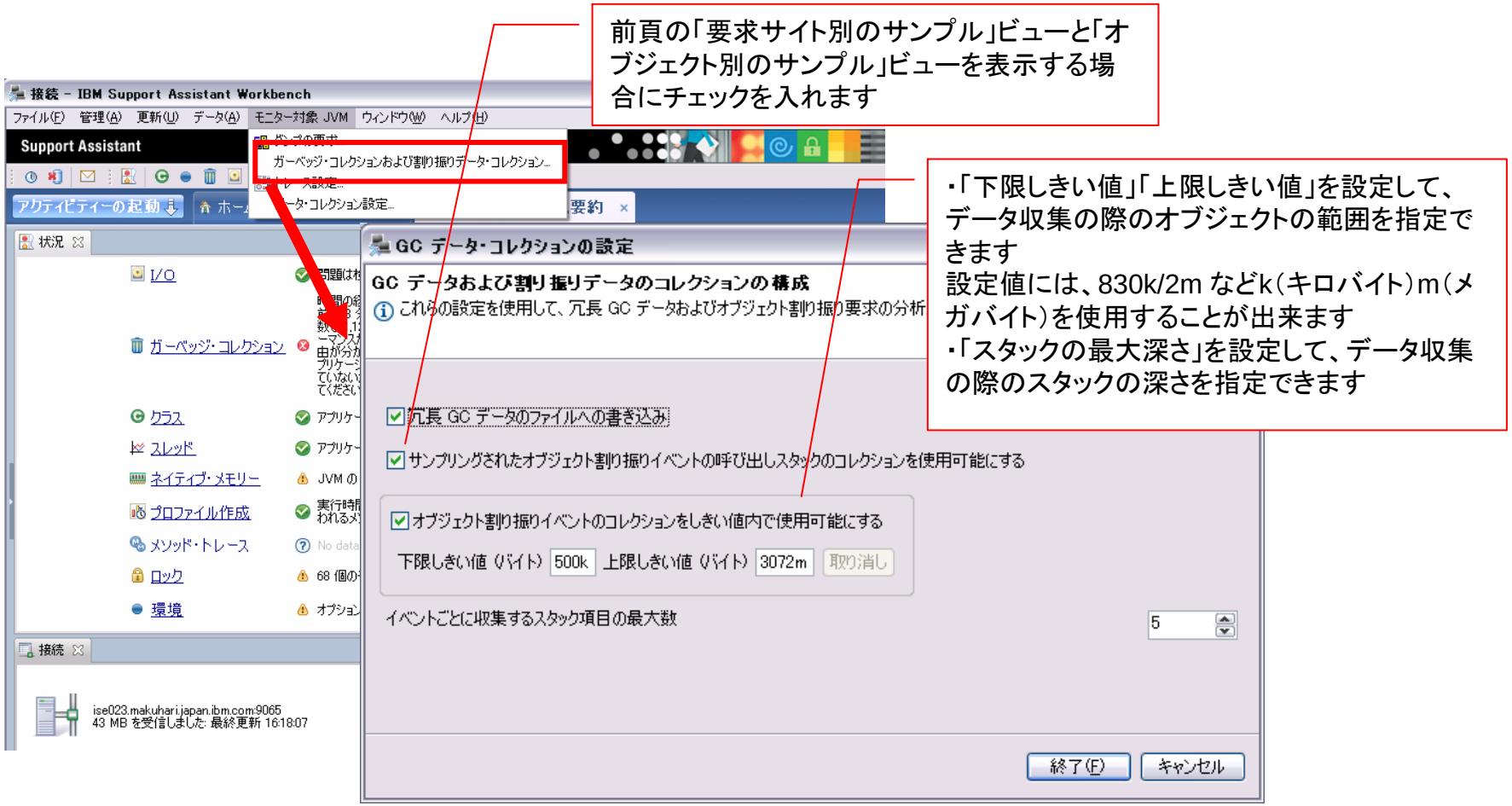


最も頻繁に割り振られるオブジェクトを検出できます

2. ガーベッジ・コレクション・パースペクティブ

V2.0～

- 前頁のオブジェクト割り振りのデータを表示するためには、モニター対象JVMの割り振りデータのコレクションの構成を行う必要があります



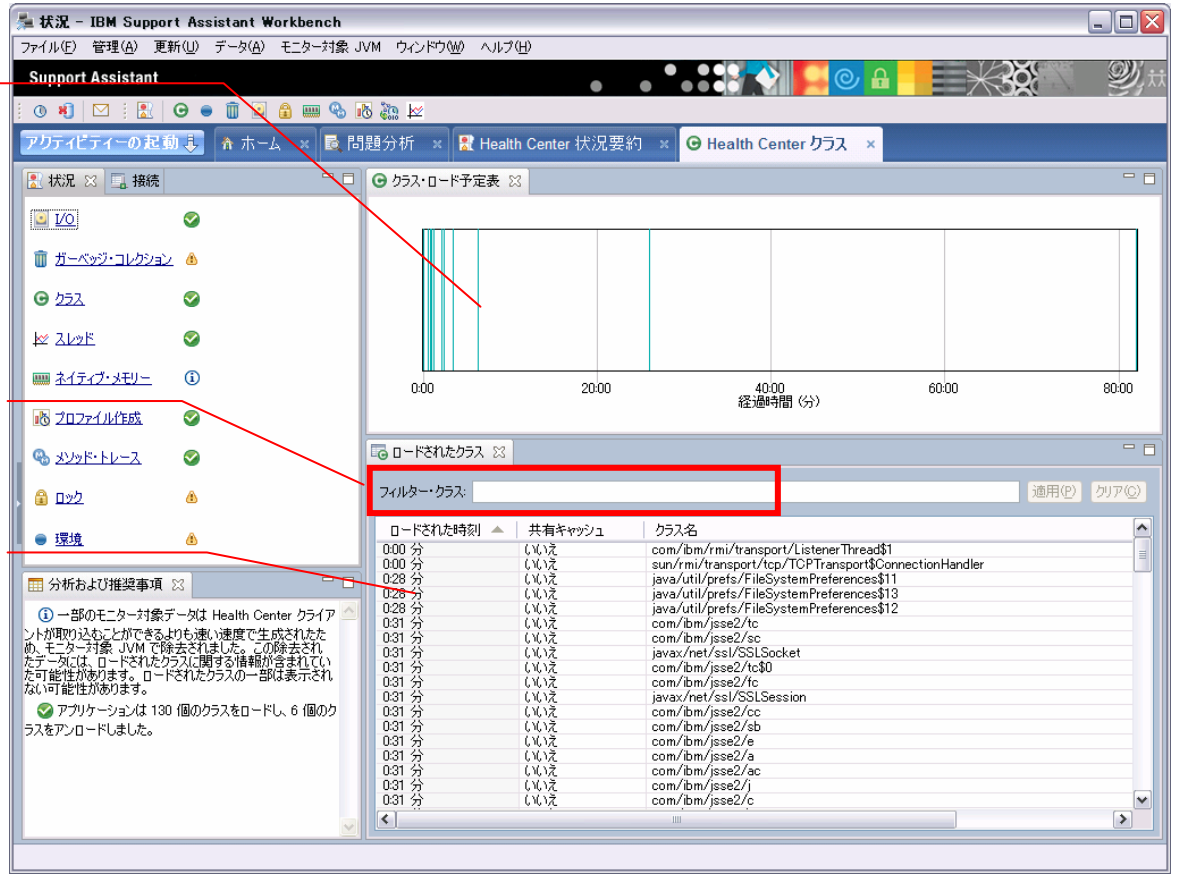
3. クラス・パースペクティブ

- クラス・パースペクティブでは、クラスがロードされた時刻、共有キャッシュの有無が確認できます。過剰なクラスローディングによるパフォーマンス問題の有無などを確認する場合に向いています
- 類似したツールとして、WAS管理コンソールのクラス・ローダー・ビューアーがあります。クラス・ローダー・ビューアーは、クラスをロードしたクラスローダーやクラスパスが確認できます。アプリケーションが正しくクラスを検索できているか？などのクラス・ローダーに関するトラブルシューティングに向いています

ロードされたクラスの数を、時系列で視覚的に確認できます
いつ予期しないレートでロードされたかを確認できます

クラスロードの詳細表示をフィルタリングできます

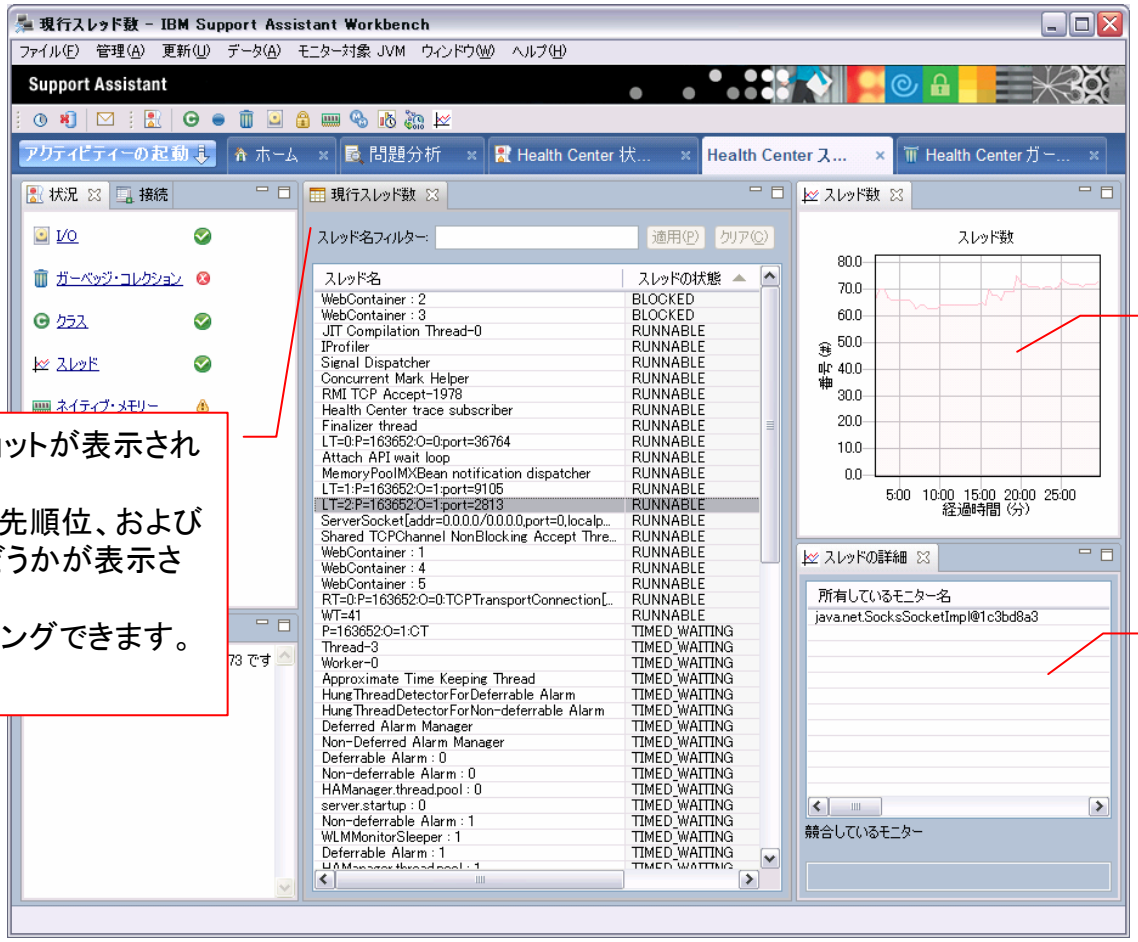
クラスがロードされた時刻が、JVMの始動時刻からの経過時間で表されます
また、クラスが共用クラス・キャッシュからロードされたかどうかを確認できます



4. スレッド・パースペクティブ

V2.0～

- スレッド・パースペクティブでは、アプリケーション内のすべての進行中スレッドの状況やロックでブロックされているかどうかを表示することができます
- アプリケーションのパフォーマンス悪化をもたらすロック競合問題を特定するときに便利です



スレッドのスナップショットが表示されます。
各スレッドの状態と優先順位、およびデーモン・スレッドかどうかが表示されます。
スレッド名でフィルタリングできます。

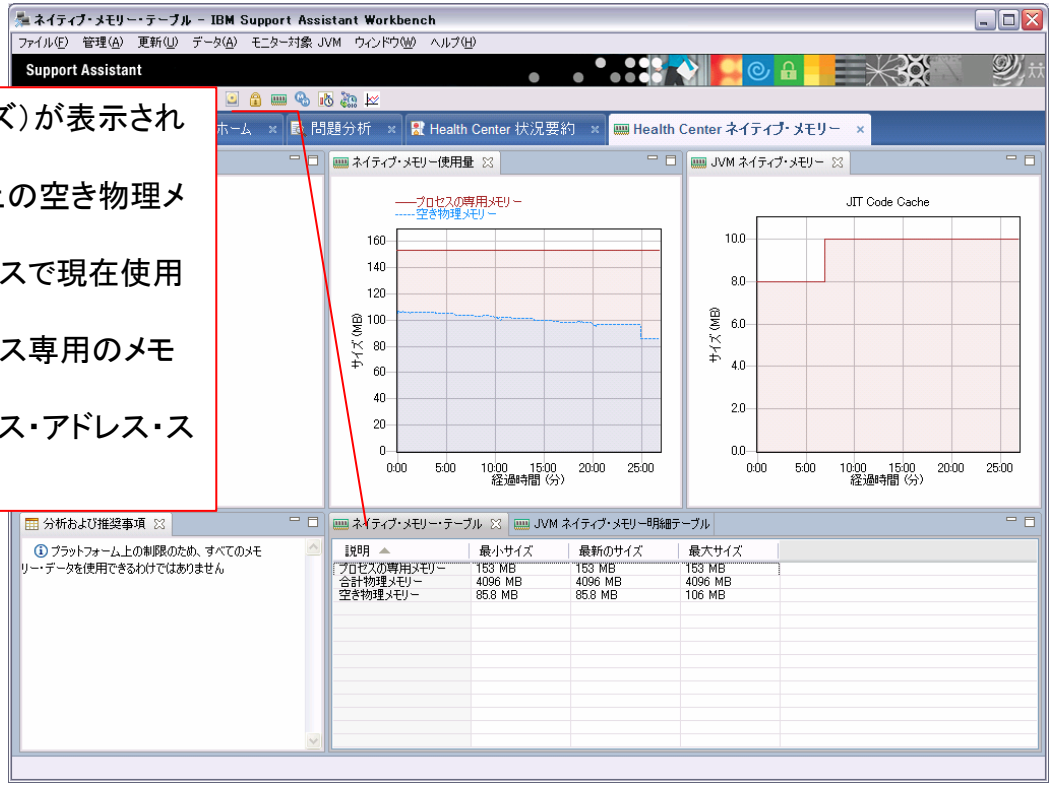
実行中のスレッド数が表示されます。
スレッド数が増加しているかどうかを確認できます

スレッドの詳細情報が表示されます

5. ネイティブ・メモリー・パースペクティブ

- ネイティブ・メモリー・パースペクティブ（またはメモリー・パースペクティブ）では、モニターするJavaプロセスとシステムのネイティブ・メモリー使用量に関する情報が表示されます
- このビューで確認できる情報はプラットフォームによって異なります

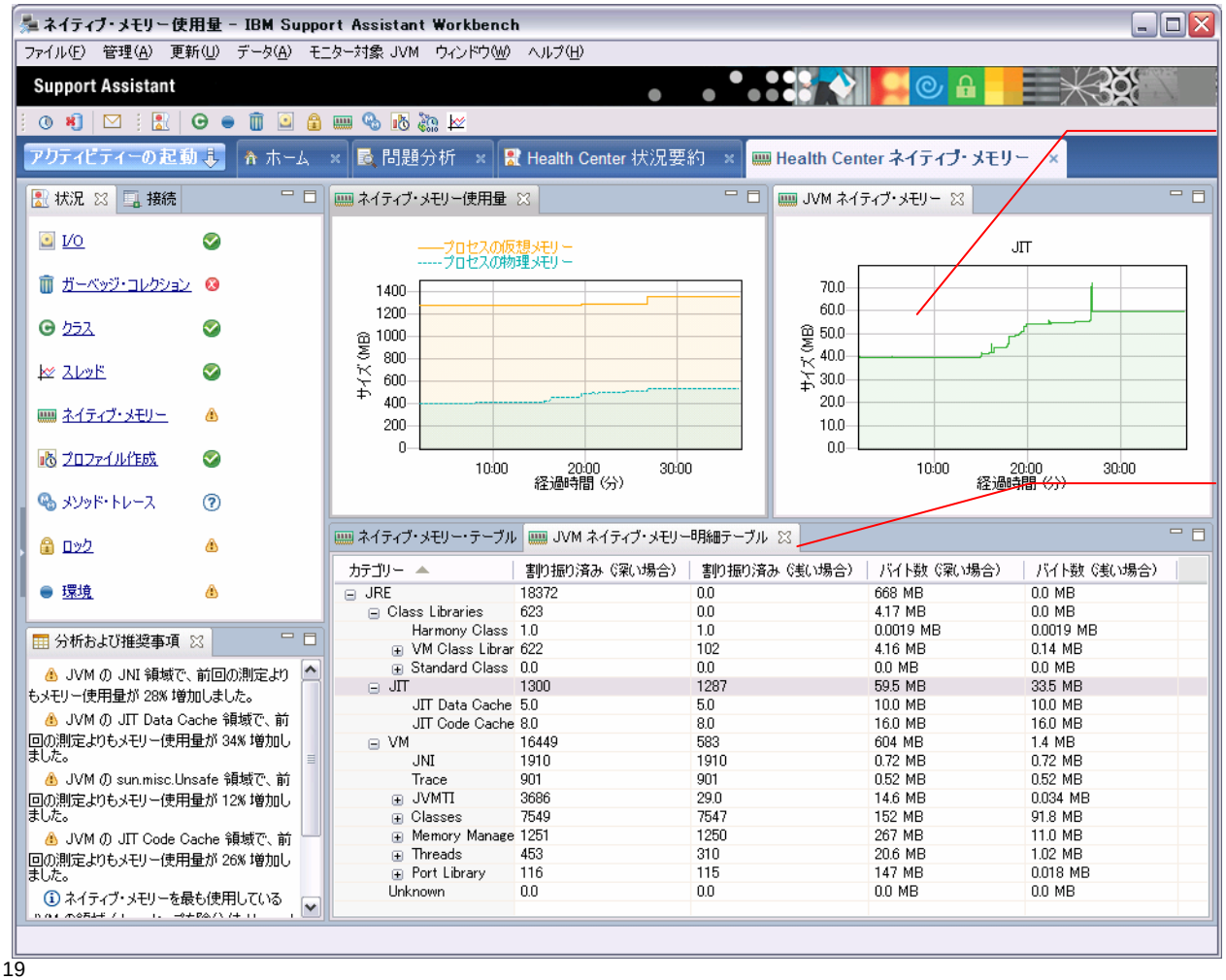
メモリー使用状況（最大/最小/最新のサイズ）が表示されます
空き物理メモリー：モニター対象システム上の空き物理メモリー量 (RAM)
プロセス物理メモリー：モニター対象プロセスで現在使用中の物理メモリー量 (RAM)
プロセス専用メモリー：モニター対象プロセス専用のメモリー量
プロセス仮想メモリー：使用しているプロセス・アドレス・スペースの合計



5. ネイティブ・メモリー・パースペクティブ

V2.0～

- ネイティブ・メモリーを使用している JVM 領域の明細を確認できます
- 例えばクラスローダーやJITによってどの程度ネイティブメモリーを使用しているかを確認できます



JVMネイティブ・メモリーでは「JVMネイティブ・メモリー明細テーブル」で選択された行の項目の、ネイティブ・メモリー使用量を時系列でグラフ表示します

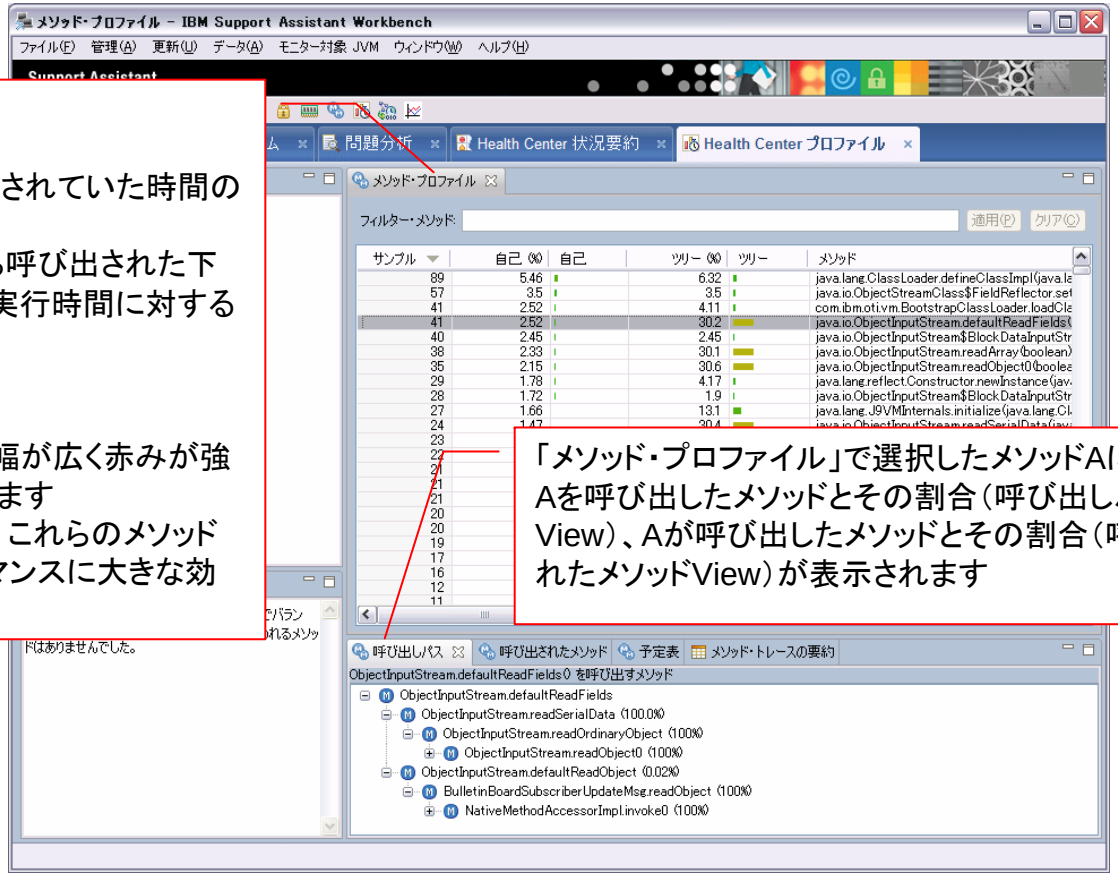
JVMネイティブ・メモリー明細テーブルでは、ネイティブ・メモリーを使用している JVM の領域の明細が表示されます

6. プロファイル作成・パースペクティブ

- メソッド・プロファイル・パースペクティブを使用すると、リソースを消費しているメソッドを確認できます
- プロファイラーは全ての実行メソッドを記録するのではなく、定期的にサンプルを取得して実行中のメソッドを調べます。したがって、頻繁に呼び出されるメソッドおよび処理時間が長いメソッドの情報が表示され、実行頻度が低いメソッドや短時間で実行されるメソッドは、表示されない可能性があります

メソッド実行状況が表示されます
・サンプル：取得されたサンプル数
・自己(%)：このメソッドのサンプルが実行されていた時間の全体の実行時間に対する割合
・ツリー(%)：このメソッドとこのメソッドから呼び出された下位のメソッドが実行された時間の全体の実行時間に対する割合
・メソッド：メソッドの完全修飾表記

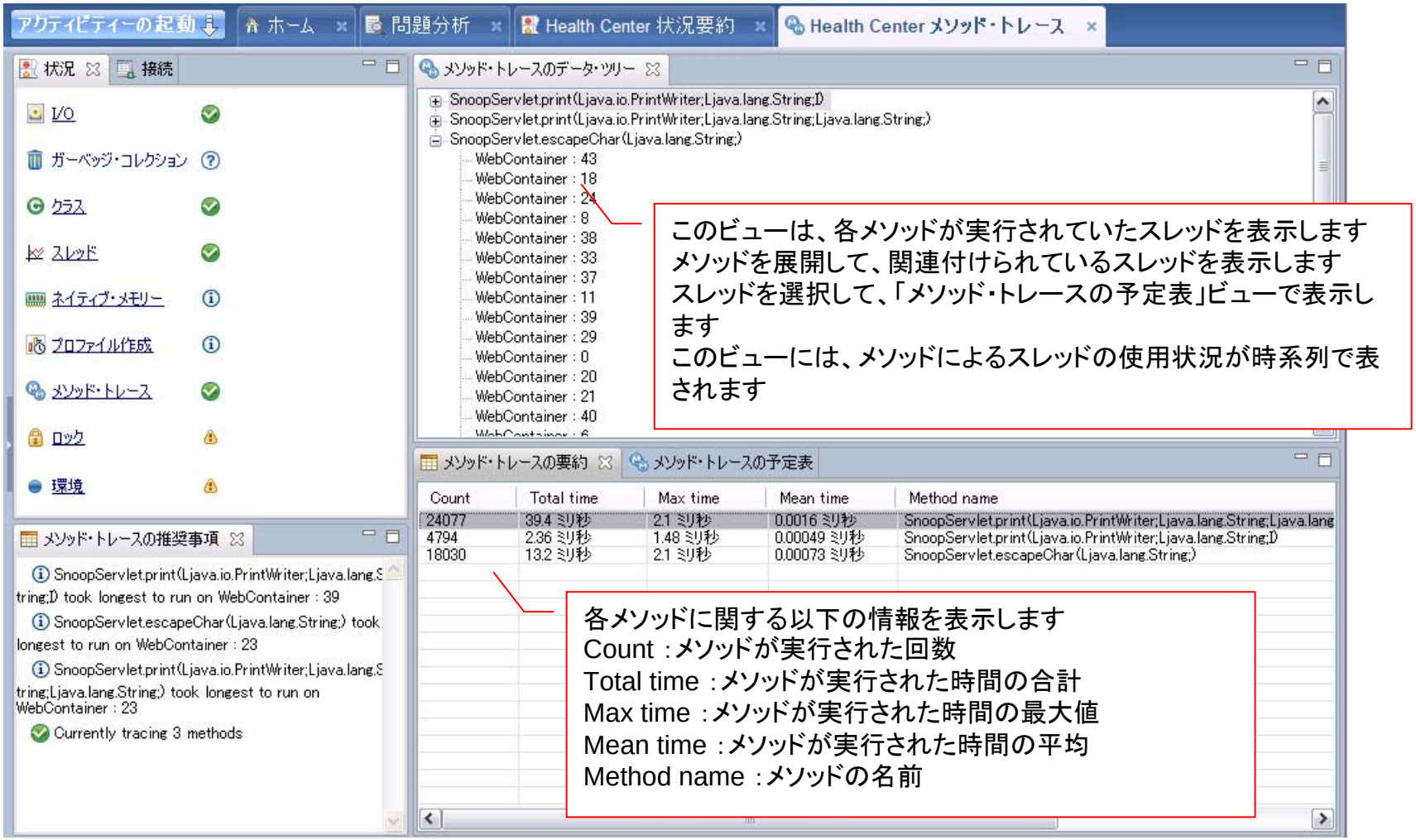
自己、ツリーとも棒グラフで表され、棒の幅が広く赤みが強いほど「ホット」なメソッドであることを示します
「ホット」なメソッドは最適化の候補として、これらのメソッドの効率を少し改善するだけで、パフォーマンスに大きな効果が出ると考えられます



「メソッド・プロファイル」で選択したメソッドAについて、Aを呼び出したメソッドとその割合（呼び出しパスView）、Aが呼び出したメソッドとその割合（呼び出されたメソッドView）が表示されます

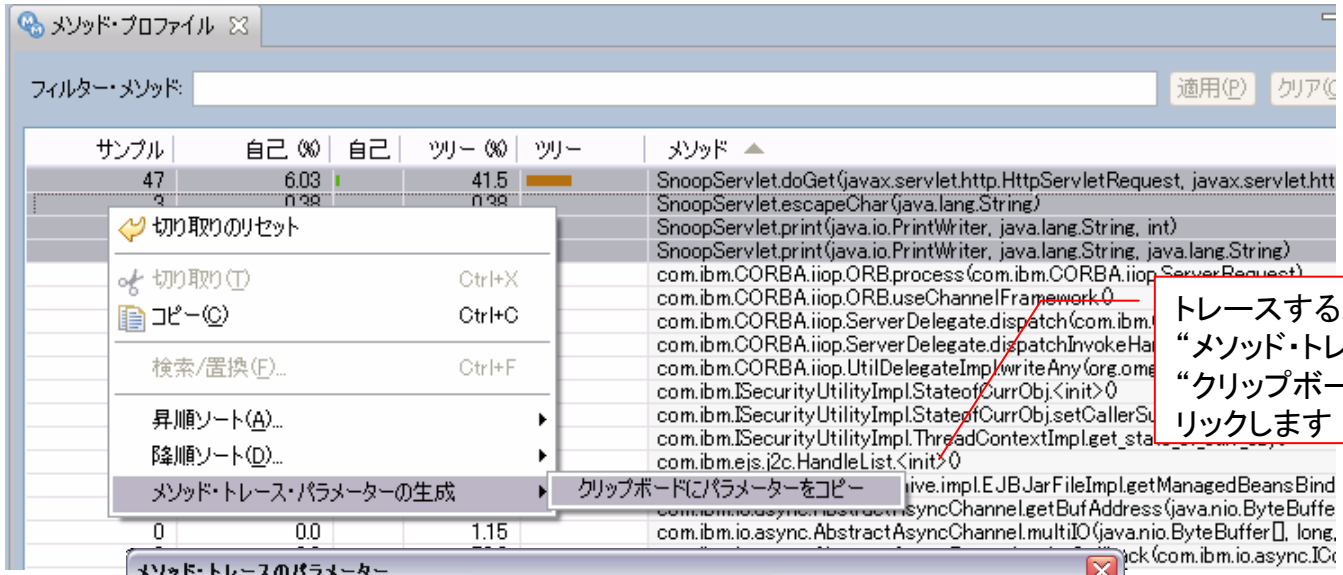
7. メソッド・トレース・パースペクティブ

- メソッド呼び出しの正確な時間、および実行中のスレッドにおけるメソッド使用状況の分析を表示します



7. メソッド・トレース・パースペクティブ

- デフォルトでは有効になっておらず、以下の事前設定が必要です
- プロファイル作成 パースペクティブでトレース対象にしたいメソッドを選択してメソッド・トレース・パラメータを生成し、生成したパラメータを対象JVMの汎用JVM引数に追加してサーバー再始動を行います



トレースするメソッドを選択して、右クリックし、“メソッド・トレース・パラメータの生成“→“クリップボードにパラメーターをコピー”をクリックします

WASの場合

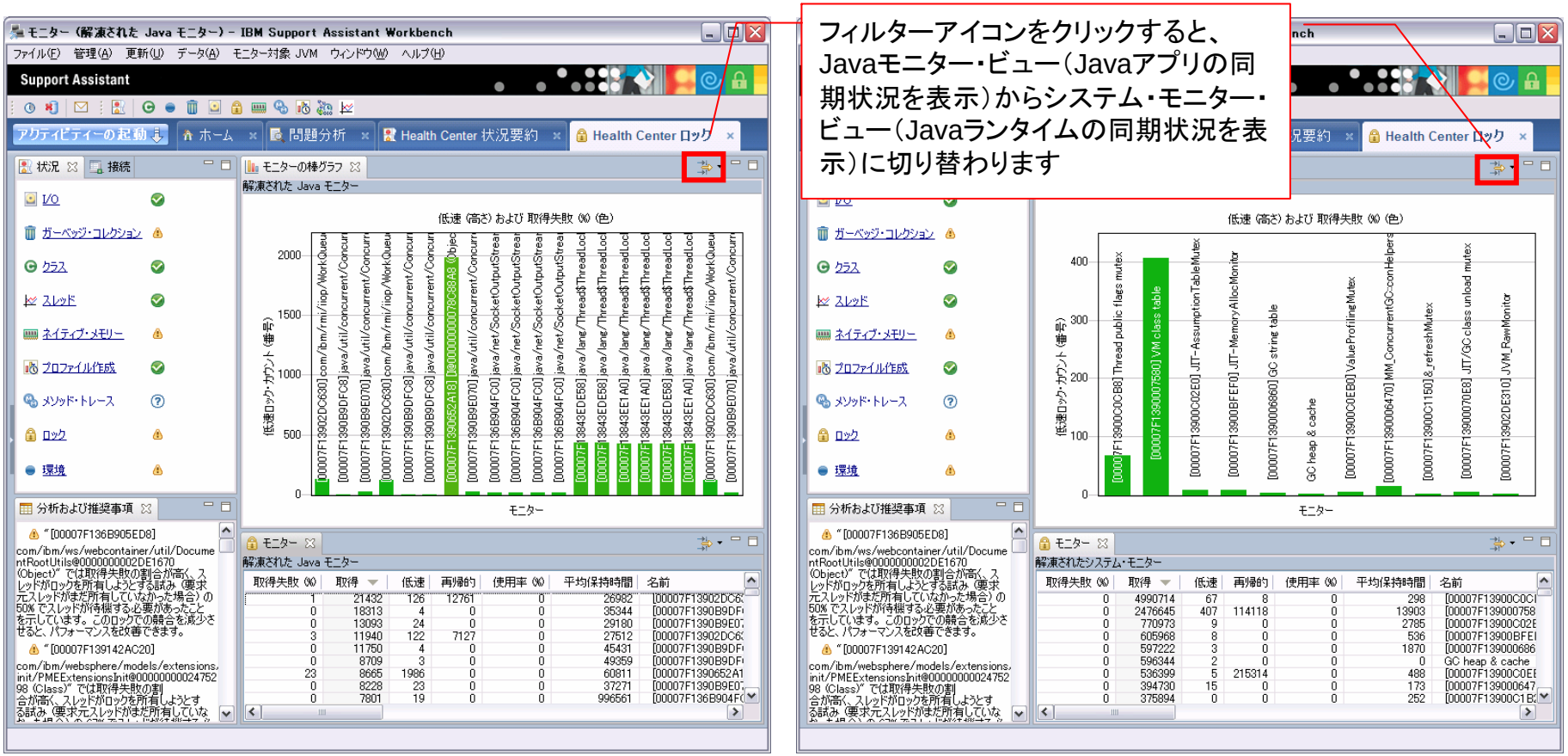
汎用 JVM 引数

```
-Xhealthcenter:port=1978 -Xtrace:maximal=mt,methods=
{"SnoopServlet.doGet,SnoopServlet.print,SnoopServlet.escapeCha
r,SnoopServlet.print"}
```

WASの場合、対象JVMの汎用JVM引数に追加して、サーバー再始動を行います

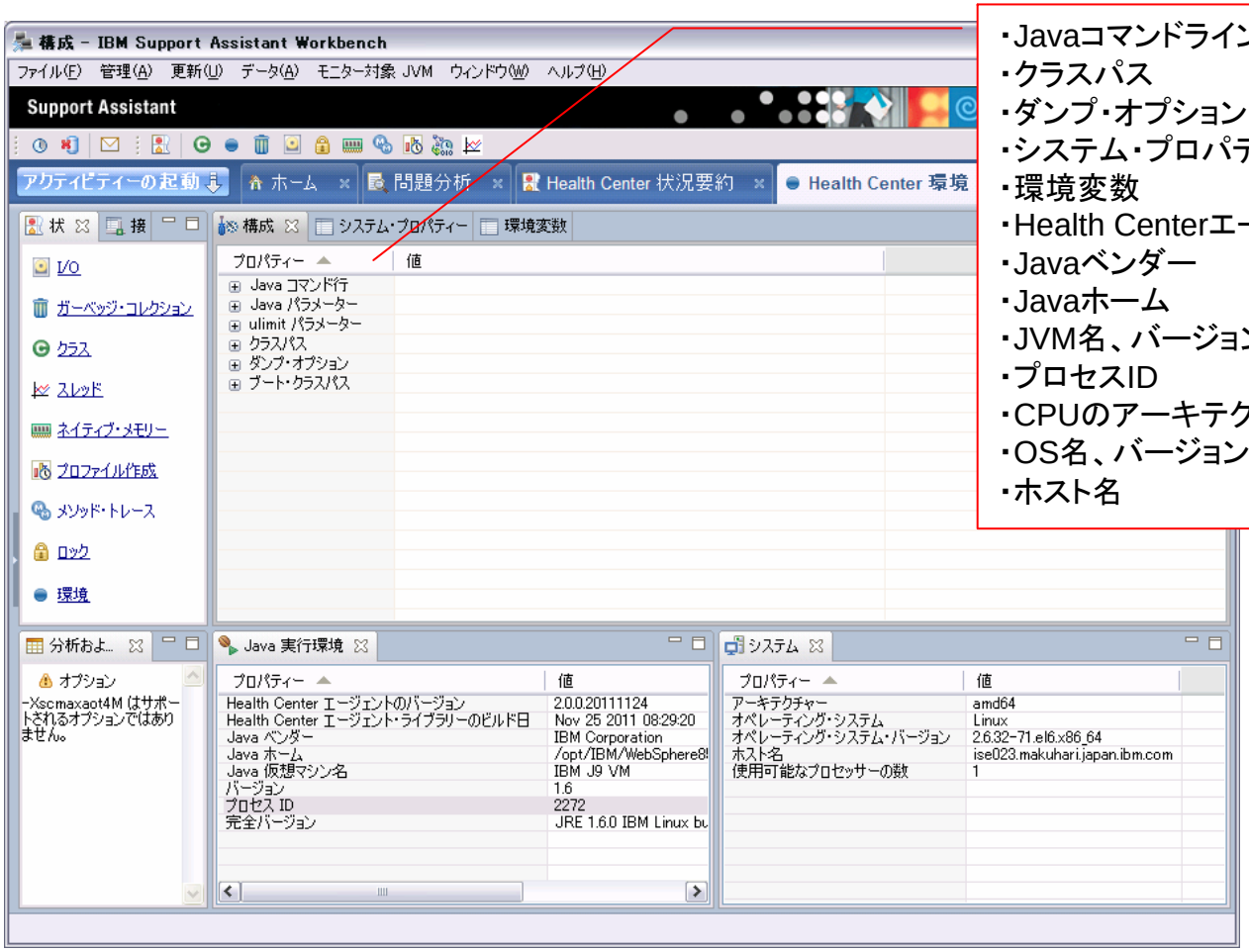
8. ロック・パースペクティブ

- Javaはマルチスレッド・アプリケーションのため、リソースの整合性を保つために、コードの synchronized 等で共用リソースをロック（同期）しなければならない場合があります。これがパフォーマンスのボトルネックとなる場合もあり、特に複数CPUマシンで稼動しているアプリケーションでロックが多用されると、CPUを効率的に使用できなくなる可能性があります
- ロック・パースペクティブでは、ロックの状況を確認できます



9. 環境パースペクティブ

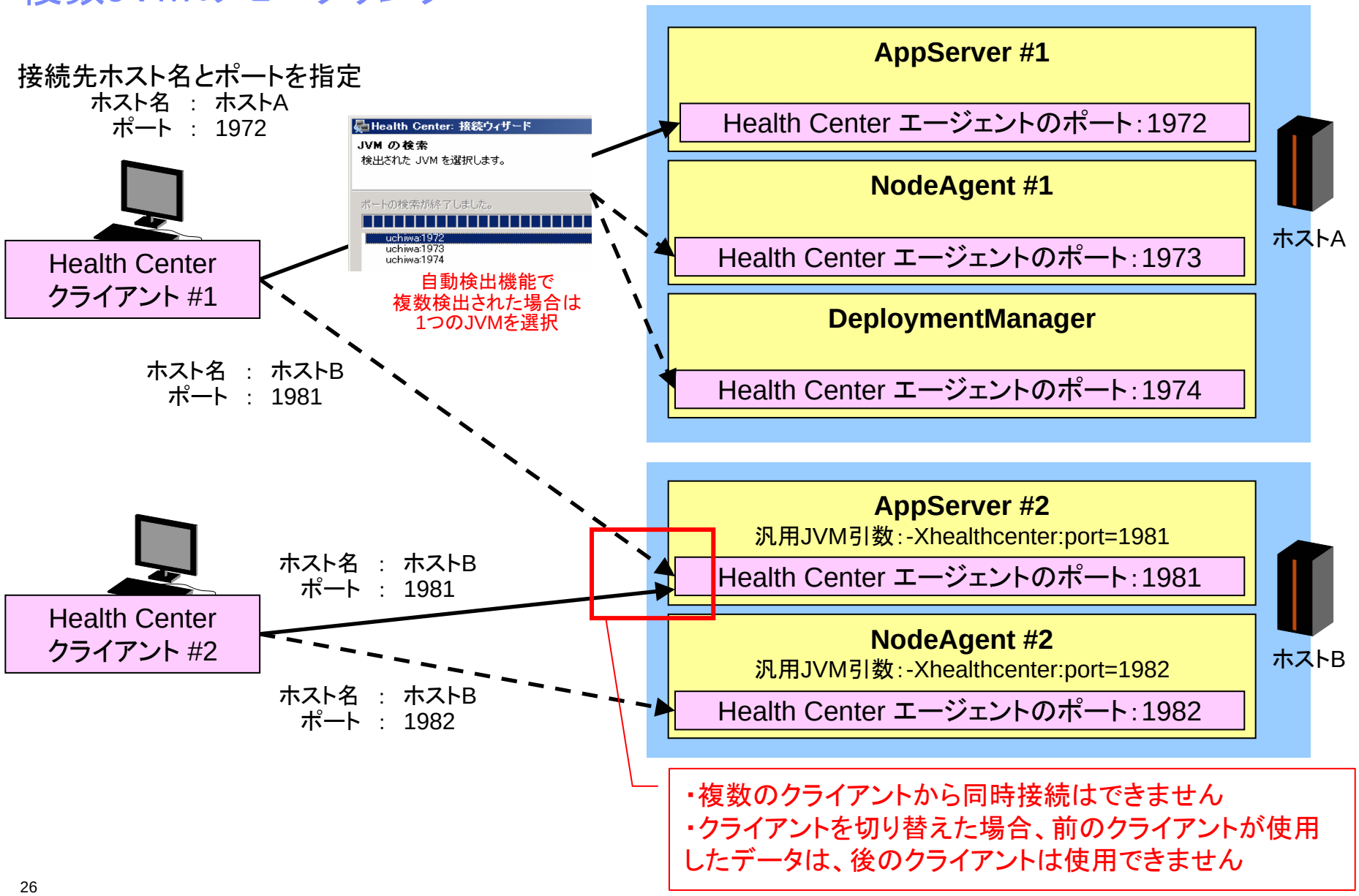
- 環境パースペクティブでは、モニター対象のJVMに関する様々なシステム情報、構成情報が確認できます



- Javaコマンドライン引数
- クラスパス
- ダンプ・オプション
- システム・プロパティー
- 環境変数
- Health Centerエージェントのバージョン
- Javaベンダー
- Javaホーム
- JVM名、バージョン
- プロセスID
- CPUのアーキテクチャ、CPU数
- OS名、バージョン
- ホスト名

3. Health Centerの運用のポイント

複数JVMのモニタリング



パフォーマンス

- Health Centerエージェントがアプリケーションに与える影響
  - Health Centerエージェントが使用するメモリやCPUは少なく、パフォーマンスに影響を与えることはほとんどないとマニュアルに記載されています
  - 以下のフォーラムでは、Health Centerのオーバーヘッドはアプリケーションのワークロード（1JVMあたり）の3%以下を目指していると記載されています
    - http://www.ibm.com/developerworks/forums/thread.jspa?threadID=320417&tstart=0

- オーバーヘッド削減モード
  - モニター対象アプリケーションが収集するデータ量を削減します
    - メソッド・プロファイル・パースペクティブの「呼び出しパス」と「呼び出されたメソッド」ビューのスタックを収集しない
  - 以下のような、大量のネイティブ・メモリーを Health Center エージェントが消費し、異常終了するようなケースで有効です
    - 多数のプロセッサーがある
    - 対象アプリケーションのスタック・トレースが深い
  - 以下のどちらかの方法で指定します
    - JVMに -Xhealthcenter:level=low を指定する
      （ Java 5 JRE SR10 以降、Java6 SR5以降を使用した場合）
    - healthcenter.properties の以下のプロパティをlowに書き換える(デフォルトはfullです)
      com.ibm.java.diagnostics.healthcenter.data.collection.level

- スリープモード
  - Health Center V1.2 から、実行しているアプリケーションに影響を与えずにクライアント接続を待機することができます。
  - 通常は、クライアントが接続していなくてもHealth Centerエージェントが開始されます。スリープモードを使用すると、Health Centerエージェントは、Health Centerクライアントが接続してからはじめてデータ収集を開始します
  - healthcenter.properties の以下のプロパティをoffに書き換えると、スリープモードとなります(デフォルトはfullです)
    - com.ibm.java.diagnostics.healthcenter.data.collection.level

- パフォーマンス検証結果
  - 下記資料（P.27）では、サンプルアプリケーションを使用したパフォーマンス測定結果が紹介されています
    - http://public.dhe.ibm.com/software/dw/jp/websphere/was/was7_update/wasv7updatews06systemmanagement_rv.pdf

Listenポートの変更とログファイル

- Listenポートの変更
  - デフォルトで、Health Center エージェントはポート 1972 を使用して通信を行います。ポート 1972 を使用できない場合、エージェントはポート番号をインクリメントして、最大 100 回まで再試行します
  - ポートの変更は以下のどちらかで可能です
    - JVM引数で指定します

      -Xhealthcenter:port=<*ポート番号*>
    - healthcenter.properties の以下のプロパティを変更します

      com.ibm.java.diagnostics.healthcenter.agent.port

- Health Centerエージェントのログファイル
  - Health Centerエージェントは、一時ファイルディレクトリーにhealthcenter.<pid>.log ファイル　を出力します。
    これらのファイルは自動で削除されないため、手動で運用してください
  - 一時ファイルディレクトリーは、環境パースペクティブ⇒システム・プロパティー⇒java.io.tmpdir で確認可能です。

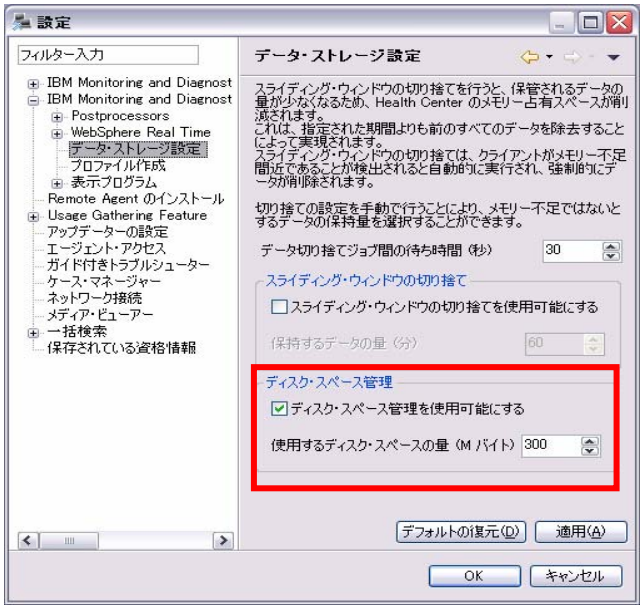
- Health Centerクライアントの一時ファイル
  - Health Centerクライアントは、一時ファイルディレクトリーに一時ファイルを作成します。
  - これらのファイルは自動で削除されません。一時ファイルですので、手動で削除してください

モニターデータの調整

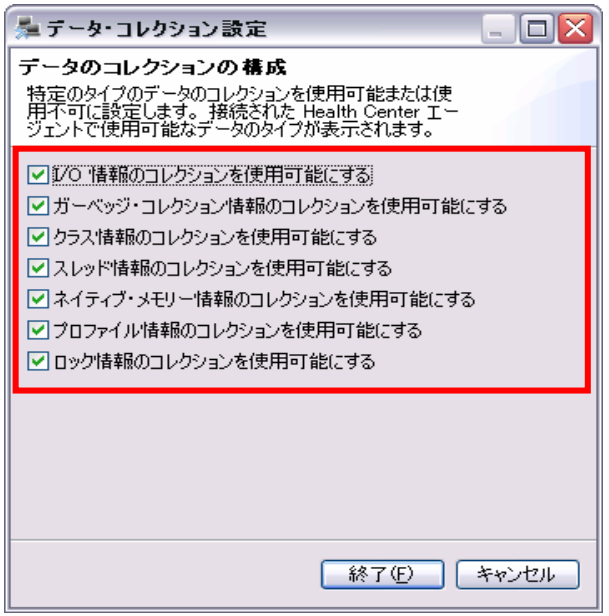
- 保管データのサイズ指定
  - Health Centerクライアントは、デフォルトで300MBのデータ（.hcdファイル）を保管します。従ってモニター対象JVMを停止してもデータは確認可能です
  - 保管するデータのサイズは、メニューの「ファイル」⇒「プリファレンス」⇒「IBM Monitoring and Diagnostic Tools for Java – Health Center」⇒データ・ストレージ設定⇒ディスク・スペース設定　で調整可能です

- 収集対象データの選択
  - 不必要なパースペクティブをオフにすると、必要なデータのみを収集できます
  - メニューの「モニター対象JVM」⇒「データ・コレクション設定...」で調整してください

■保管データのサイズ指定



■収集対象データの選択

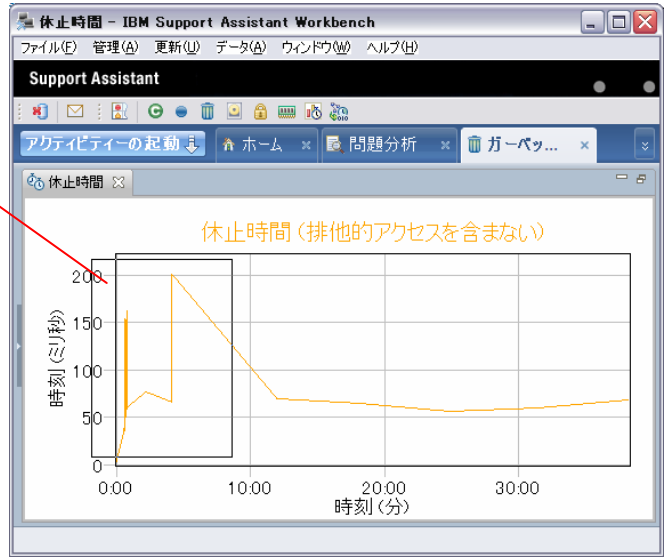


モニターデータの調整

- データの切り取りと表示間隔
  - GC・パースペクティブおよびクラス・パースペクティブでは、データの表示間隔を調整することができます

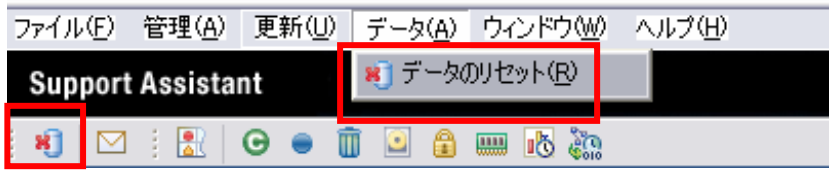
グラフ内のドラッグで範囲を指定することができます
元の状態に戻すには、右クリックで「切り取りのリセット」を選択するか、グラフ上の任意の場所をダブルクリックします

■データの切り取り



- データのリセット
  - Health Centerクライアントに表示されたデータは、メニューの「データ」⇒「データのリセット」を選択すると、それまでのデータは削除されます

■データのリセット



モニターデータの保存とオープン

- Health Centerのデータを保管して、別のHealth Centerクライアント（同じバージョン）で開くことができます。データを取得して、他の技術者に見てもらう、という作業も可能です

- データの保存
  - Health Centerクライアントで分析中のデータを.hcd ファイルとして保管できます（旧リリースでは.zip）
    - メニューの「ファイル」⇒「データの保存」
  - デフォルトの保存容量は300MBです。モニターデータ量によって、300MBに含まれる時間幅は大きく変わりますので注意が必要です
  - 保管されるデータは、モニターされた最新の情報です
  - ファイルのサイズ制限を解除できますが、ディスクスペースに注意が必要です

- データのオープン
  - Health CenterクライアントはHealth Centerエージェントに接続していなくても、.hcdファイルを分析できます
    - メニューの「ファイル」⇒「ファイルを開く」

クライアント接続レベルの設定

- Health Center構成プロパティーでクライアント接続レベル（収集レベル）を設定できます
  com.ibm.java.diagnostics.healthcenter.data.collection.level=off / full / headless

off：クライアント接続なし	エージェントは開始されるが、クライアントがエージェントに接続するまで待機する。接続後にデータ収集を開始する。
full：クライアント接続（デフォルト）	エージェントは直ちにデータ収集を開始する。クライアントが接続するまではデータを表示できない。
headless：ヘッドレスモード(*)	エージェントは直ちにデータ収集を開始。データをクライアントに送信するのではなく、サーバー側にファイルとして格納する。 Health Centerクライアントがエージェントに接続できないシステムの場合に有用。

(*)ヘッドレスモードの場合、下記オプションを指定できます
詳細はヘルプ「Health Center の構成プロパティー」を参照してください

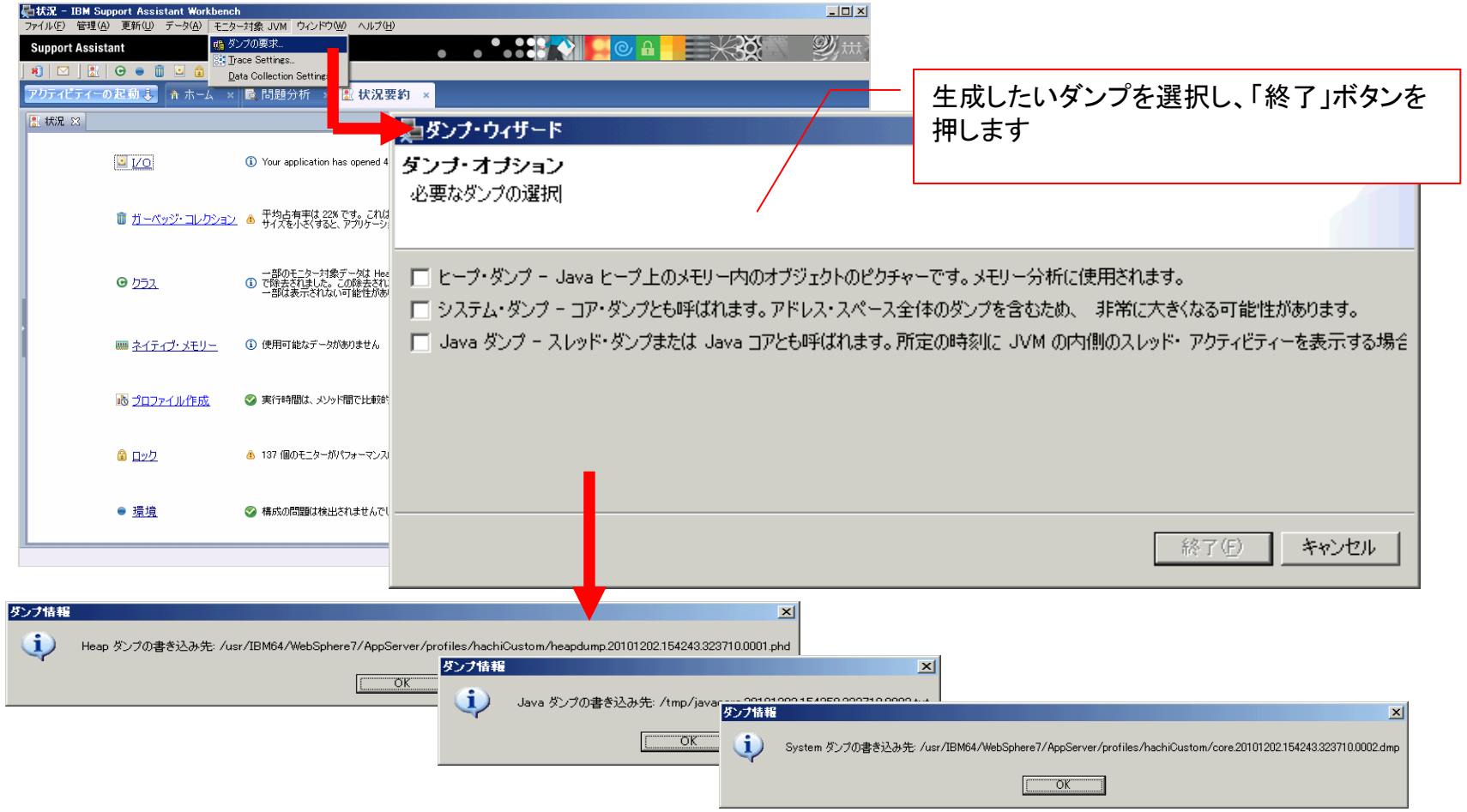
- 出力ファイルの場所 ：デフォルトはカレント作業ディレクトリ(WASの場合、プロファイルルート)に保管
- (V2.0～)出力ファイルのキープ数 ：デフォルトは10ファイル保持
- (V2.0～)出力ファイルの最大サイズ ：2GBまで
- (V2.0～)遅延開始時間 ：エージェントがデータの収集を開始するまでに待機する時間（分）
- (V2.0～)実行時間 ：データを収集する時間（分）
- (V2.0～)休止時間 ：エージェントがデータ収集の間に休止する時間（分）
- (V2.0～)実行数変更 ：収集を実行する数

例）休止時間を 5、実行数を 10、実行時間を 2 と設定した場合、エージェントは 2 分間データを収集し、5 分間休止し、その後再び 2 分間データを収集し、という収集サイクルを 10 回繰り返します

モニター対象JVMのダンプ・ファイルの取得

V1.3～

- Health Centerクライアントから、ヒープ・ダンプ、システム・ダンプ、Javaダンプ（Javaコア）を生成できます

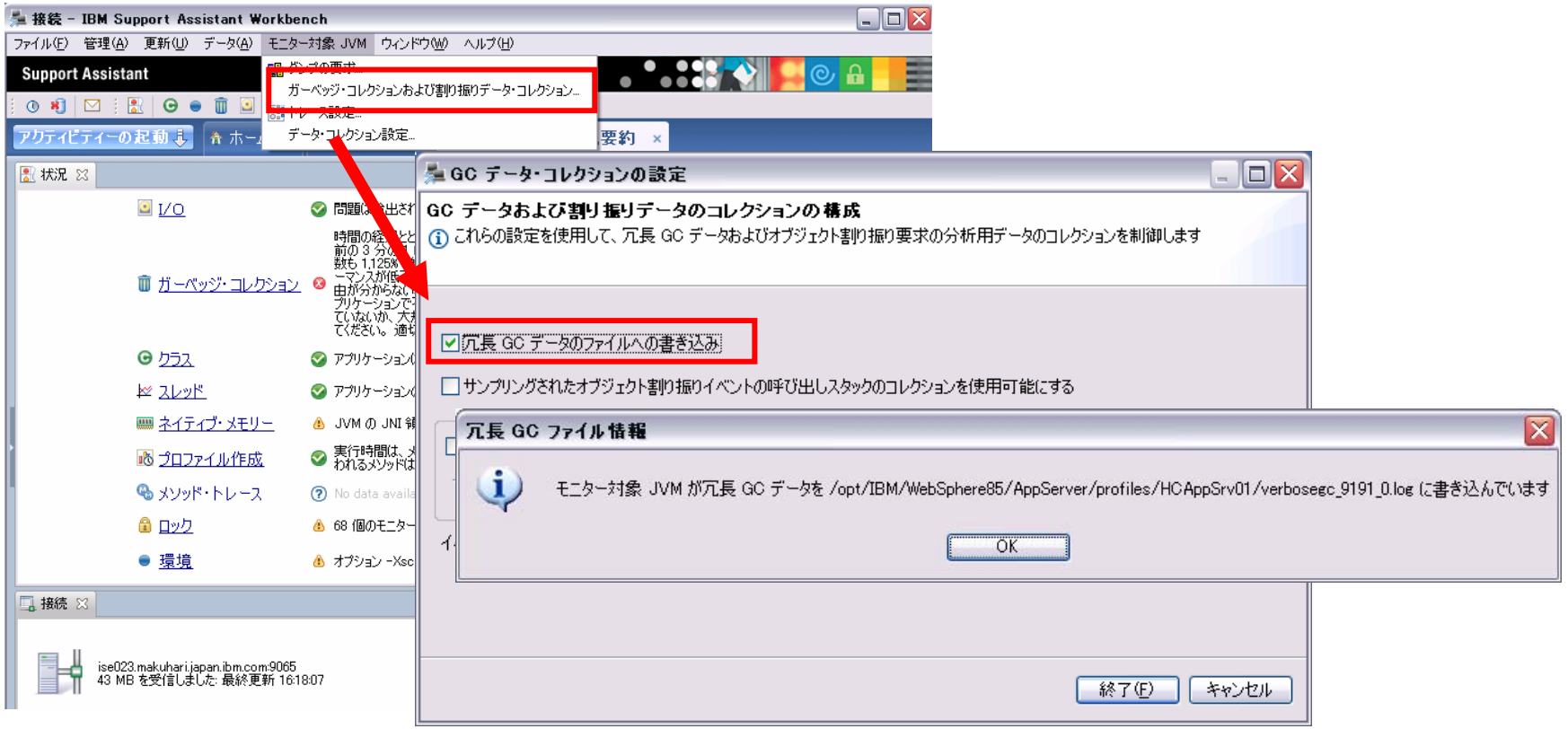


モニター対象JVMの冗長GCデータ収集

V2.0～

- Health Centerクライアントから稼働中のJVMに対してverbosegcを設定することが可能です
  - モニター対象JVM側に保管される
  - ファイル名：verbosegc_<JVMのプロセスID>_<シーケンス番号>.log
  - 保管先ディレクトリ：com.ibm.java.diagnostics.healthcenter.output.folder プロパティーで指定

- アプリケーションの再起動はしたくないが、GCMVでGCの実施状況を分析したい場合に有用



【参考】JVMの状況を把握するダンプ・ファイル

- Health Centerクライアントから生成できるヒープ・ダンプ、Javaダンプからは、以下のようなJVMの状況を把握できます

  – ヒープ・ダンプ
    - ヒープ・ダンプは、Javaヒープ上で存続している（アプリケーションで使用中の）すべてのオブジェクトをダンプしたファイルです。**メモリを多く使用しているオブジェクト等**が確認できます
    - デフォルトでは、PHD（Portable Heap Dump）ファイルというバイナリフォーマットで出力されるため、何らかのツールを使用する必要があります。オプショナルでテキスト・ファイルの出力（古いフォーマット）も可能です

  – Javaダンプ
    - Javaダンプは、取得時の Java ランタイムの内部状態のサマリーを記載したテキスト・ファイルです。その時点の**実行スレッドや実行メソッド**を確認できます
    - Javaコアとも呼ばれます
    - OSが出力するシステムダンプとは異なります

【参考】Javaダンプ

- IBM JDKのJavaダンプは、1.4以前と5.0以上には大きな違いがあります。ここでは、IBM JDK 5.0のjavacoreについて説明します。
  - ファイル名は、javacore.%Y%m%d.%H%M%S.%pid.%seq.txt　です。
    - %pid　･･･JVMのPID
    - %seq　･･･このPIDのJVMが出力したjavacore/heapdumpのシーケンス番号（JDK5から）
    - 例　･･･　javacore.20100601.135015.3986.0001.txt
  - 出力には数百ミリ秒の時間がかかります。
  - ファイルサイズは2～3MB程度です。
  - 以下のような情報が出力されます。
    - Javacore生成のトリガー（シグナル）
      クラッシュ、OOM、手動など
      GPINFO･･･どのコンポーネントがクラッシュしたか
    - 日付、javaバージョン、パス、クラスパス
    - JVM上の全てのスレッド（スレッドの状態、優先度、スレッドID、名前）
    - クラスローダー、クラス
    - ネイティブ・メモリーの状態
    - ストレージ
    - ヒープの使用量とフリーの量
    - 最後数回のGCサイクル

【参考】ヒープ・ダンプ/Javaダンプの出力先

- WebSphereの場合、デフォルトで、/<Profile Root>/<Profile Name>/ 直下に出力されます。
- native_stderr.logに出力状況が書き込まれるため確認することができます。
  - 例:
    - JVMDUMP010I Java ダンプは /opt/IBM/WebSphere70/AppServer/profiles/AP01/javacore.20100601.135015.3986.0001.txt に書き込まれました
    - JVMDUMP010I Heap ダンプは /opt/IBM/WebSphere70/AppServer/profiles/AP01/heapdump.20100601.161235.13688.0002.phd に書き込まれました
- 出力先
  - 出力先は以下の順序で決定されます
    - JVMコマンドライン（-Xdump）で指定したファイル名
      例）-Xdump:java:file=/core/javacore/javacore.%Y%m%d.%H%M%S.%pid.%seq.txt
      -Xdump:heap:file=/core/heapdump/heapdump.%Y%m%d.%H%M%S.%pid.%seq.phd
      -Xdump:snap:file=/core/snap/Snap.%Y%m%d.%H%M%S.%pid.%seq.trc
    - 以下の環境変数で指定されたディレクトリ
      （WAS管理コンソールの場合、[アプリケーション・サーバー]>[アプリケーションサーバー名]>[Javaおよびプロセス管理]>[プロセス定義]>[環境エントリー] で設定）
      IBM_JAVACOREDIR ・・・javacore出力先
      IBM_HEAPDUMPDIR ・・・heapdump出力先
      IBM_COREDIR ・・・systemdump.snap出力先
    - カレント作業ディレクトリ（WASの場合、/<Profile Root>/<Profile Name>/ 直下）
  - 空き領域不足、権限不足など、上記のディレクトリに書き込めない場合は、以下の順序で書き込まれます
    - Windowsの場合、C:¥WINDOWS ディレクトリ
    - TMPDIR環境変数で指定されたディレクトリ
    - /tmpディレクトリ
    - Windowsの場合、C:¥Temp ディレクトリ

その他の注意点

- データの更新間隔
  - Health Centerクライアントの表示データは10秒毎に更新されます
  - この値は変更できません

- トレース・オプション設定
  - Health Center は、トレース・オプション -Xtrace:none と互換性がありません
  - このオプションを設定すると、GC、またはプロファイル・データが使用できなくなります

- JITコンパイラー
  - プロファイル作成の対象となるアプリケーションの JIT コンパイラーが無効になっている場合は、プロファイル・データを使用できません

- Java Debug Wire Protocol (JDWP)
  - プロファイル対象アプリケーションで JDWP を使用してデバッグしている場合は、プロファイル・データを使用できません

- Health Center クライアントのJavaヒープサイズ
  - Health Center クライアントが稼動するISA 4.1のJava最大ヒープサイズは、デフォルトで256MBになっています。しかし、データ処理量によってはOutOfMemoryエラーが発生する場合があるため、512MBが推奨されています
  - ISAのヒープサイズは以下のように設定します
    - <home drive>¥<home path>¥IBM¥ISAv41¥.config¥rcpinstall.properties にvmarg.Xmx=-Xmx512m を設定

- Windows 上のJava6 のI/O パースペクティブの設定
  - Windows　Java6 のI/O を確認する場合には、起動時の引数に-Xtrace:maximal=io が必要です

4. Health Center導入と設定

Health Center の導入手順

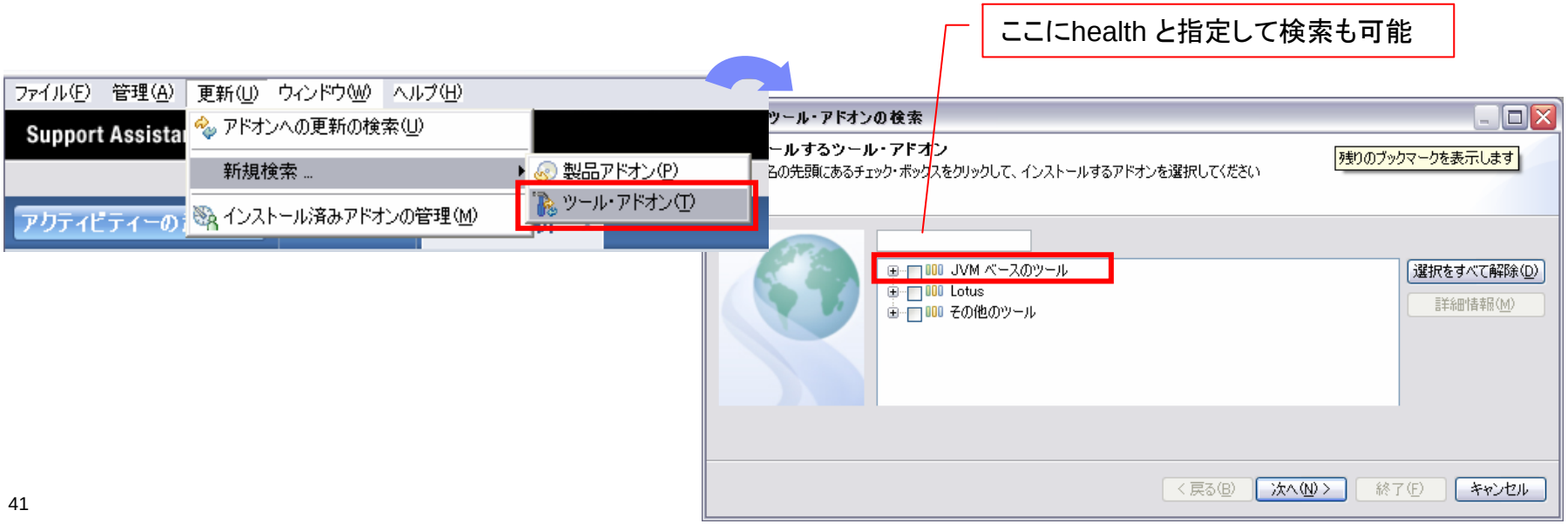
- 一部の IBM® Java™ ランタイム環境 (JRE) には、Health Center エージェントが既にインストールされています。その場合でも、最新の更新を確実に組み込むために、この手順に従ってエージェントをインストールしてください

1	ISAにHealth Centerクライアントの導入
2	Health Center エージェントのダウンロード
3	モニタリング対象のサーバへHealth Center エージェントを展開
4	モニタリング対象のJavaアプリケーションに対するHealth Center有効化の設定
5	Health Centerクライアントからモニタリング対象のJavaアプリケーションへの接続

1. ISAにHealth Centerクライアントの導入

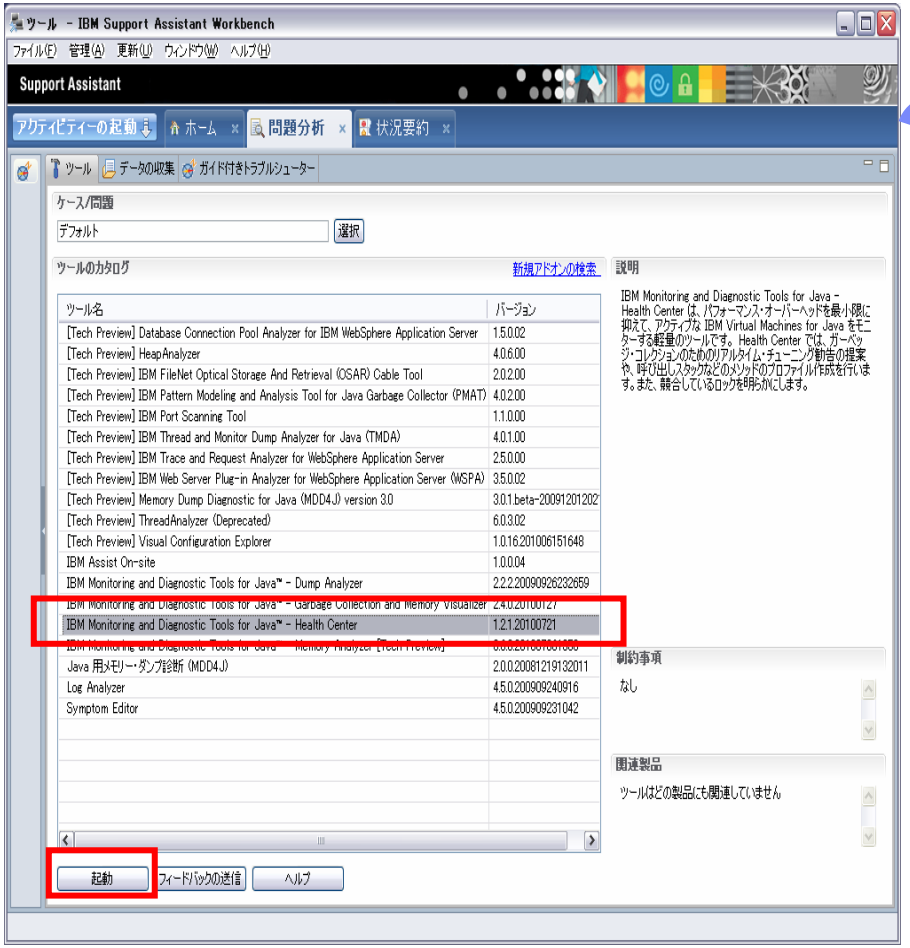
- Health Center クライアントのインストールの前に、ISAの導入が必要です
  - 以下を参考にしてISAを導入してください
  - IBM Support Assistant 4.1 利用ガイド⇒1.ISA 導入と概要
  - http://download.boulder.ibm.com/ibmdl/pub/software/dw/jp/websphere/was/isa41_guide/wasisa01_install.pdf

- ISAの導入後、Health Centerクライアントを、ISAのツール・アドオンとして追加します
  - 以下を参考にしてHealth Centerクライアントを導入してください
  - IBM Support Assistant 4.1 利用ガイド⇒2.ISA 構成と管理⇒P.7　3. ツール・アドオンの導入と管理
    - http://download.boulder.ibm.com/ibmdl/pub/software/dw/jp/websphere/was/isa41_guide/wasisa02_config.pdf
  - ISAアドオンのオフラインインストール方法は以下URLを参照してください
    - http://www-01.ibm.com/support/docview.wss?uid=jpn1J1009241



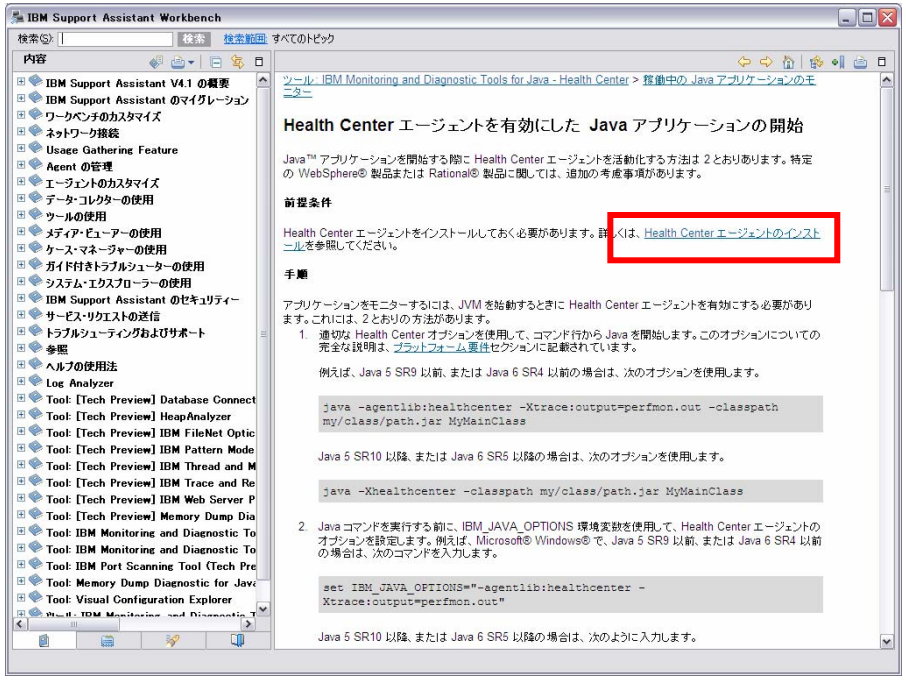
2. Health Center エージェントのダウンロード

- ISA からHealth Centerを起動し、「Health Center:接続ウィザード」で「アプリケーションでモニターを使用可能にする」を選択します

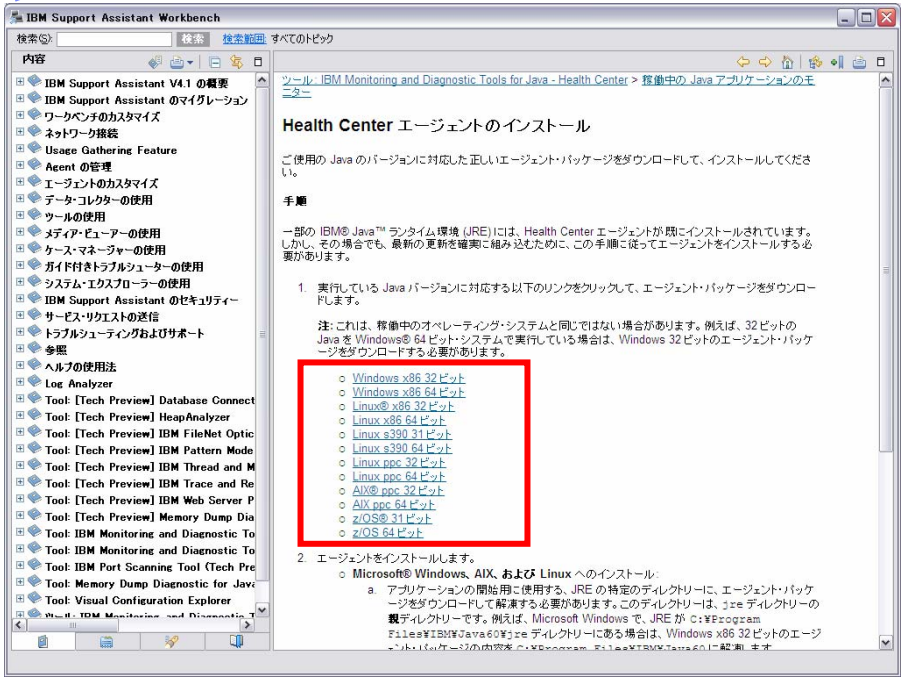


2. Health Center エージェントのダウンロード

- ISA のヘルプ「Health Center エージェントを有効にした Java アプリケーションの開始」が起動するので、「Health Center エージェントのインストール」をクリックします



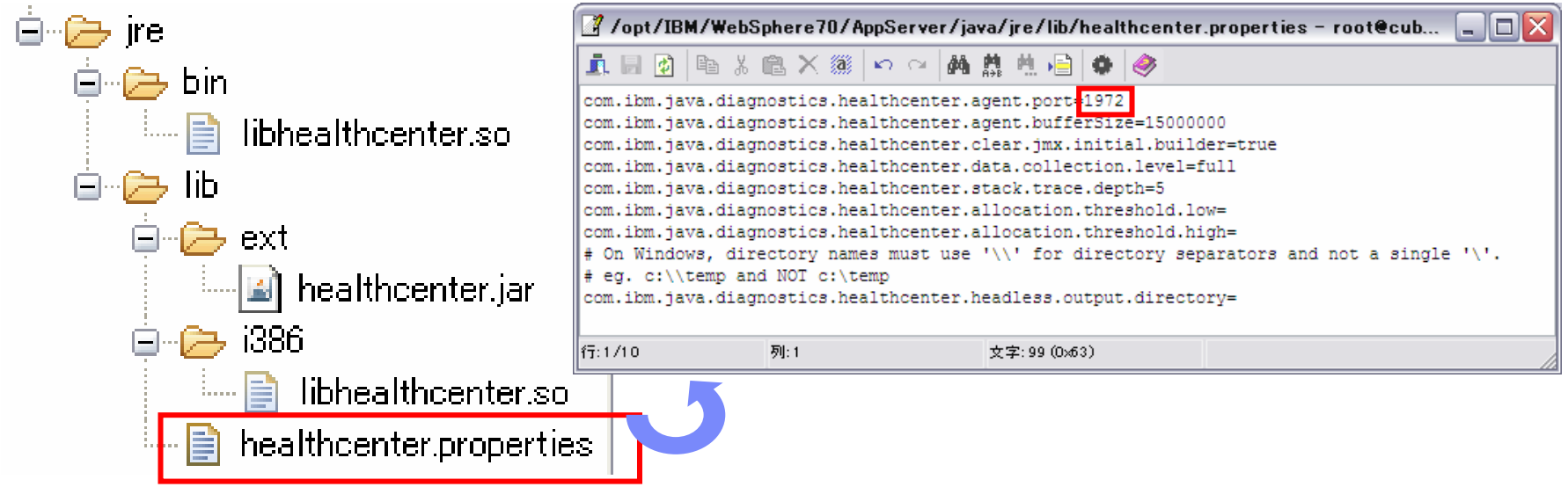
- プラットフォームに応じたエージェントをダウンロードします



3.モニタリング対象のサーバへHealth Center エージェントを展開

- 必要であればJavaディレクトリをバックアップします
- ダウンロードしたzipには以下のようなファイルが含まれているため、これらをJavaのディレクトリ（例：<WebSphereインストールルート>/java/jre）にコピーします
  - healthcenter.properties、healthcenter.jar、libhealthcenter.soなどのファイルが存在していれば上書きします
- healthcenter.propertiesには、クライアントから接続するポート（デフォルト1972）が定義されています

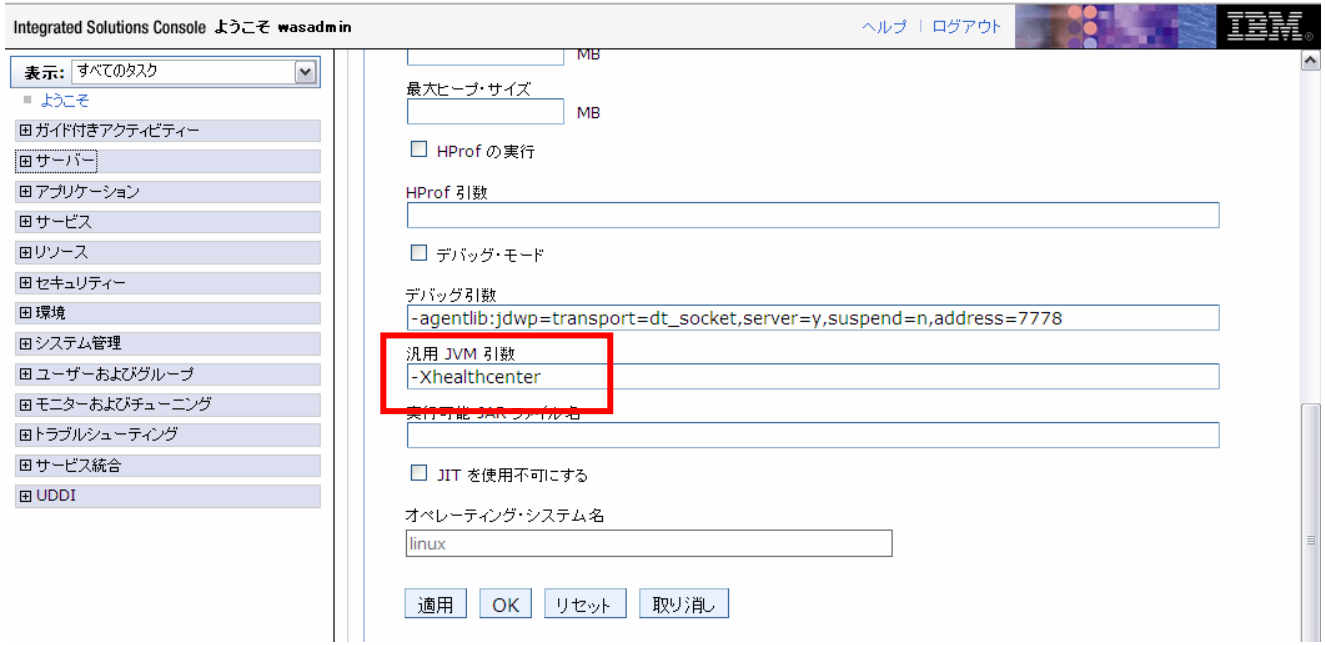
以下はLinuxの例で、含まれるファイルはプラットフォームで異なります



4.モニタリング対象のJavaアプリケーションに対するHealth Center有効化の設定　～JVM起動時に有効化する方法～

- 指定方法は、Javaのバージョンやアプリケーション（単体のJavaアプリケーション/WAS/WID/RAD等）によって異なるため、詳細は、ISAヘルプの「Health Center エージェントを有効にした Java アプリケーションの開始 」を参照してください
- Java6 SR5以降を使用した WASは、汎用JVM引数に「-Xhealthcenter 」を指定します
- 無効化にはサーバ再始動が必要です

管理コンソール⇒サーバ⇒<該当サーバ>⇒プロセス定義⇒Java仮想マシン⇒汎用JVM引数



4.モニタリング対象のJavaアプリケーションに対するHealth Center有効化の設定　～JVM起動後に有効化する方法～

V2.0～

- Health Center V1.3以前のリリースまではアプリケーション開始時にHealth Centerエージェントを有効化する必要がありました（前項）
- V2.0より実行中のアプリケーションに対してHealth Centerエージェントを有効化できます
- アプリケーション稼働期間中ずっと有効化するのではなく、問題が発生した時にHealth Centerを有効化して分析を行いたい場合に有用です

【設定方法】
下記コマンドを実行します。

- JVMのプロセスIDを指定する場合

  #<WAS_Root>/java/bin/java –jar <WAS_Root>/java/jre/lib/ext/healthcenter.jar ID:46375 port=1999

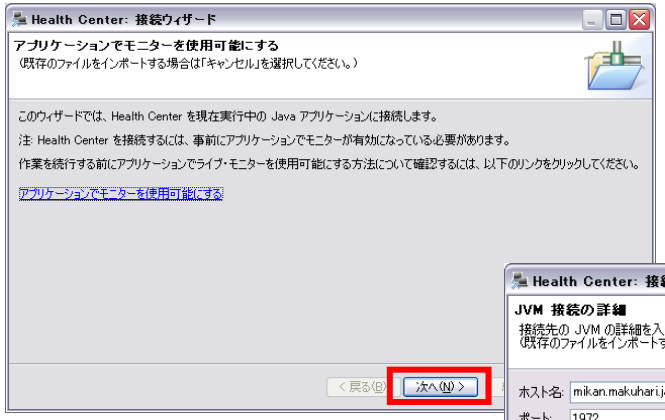
- JVMのプロセスIDを指定しない場合

  コマンド実行結果として実行中の JVM のリストが表示されますので、リストから JVM を選択します

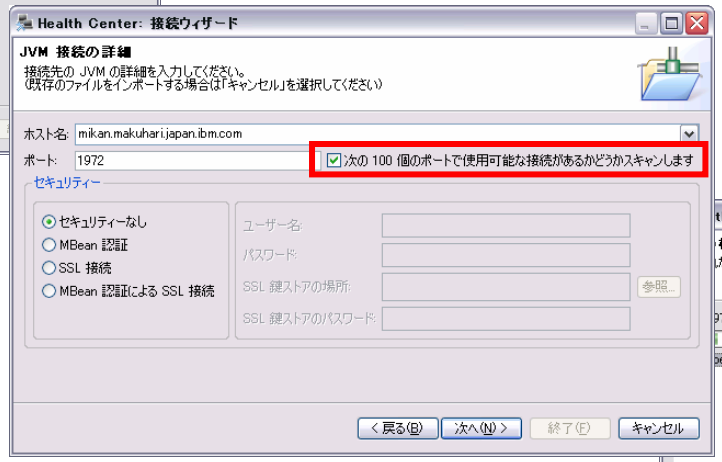
  #<WAS_Root>/java/bin/java -jar <WAS_Root>/java/jre/lib/ext/healthcenter.jar
  -Dcom.ibm.java.diagnostics.healthcenter.agent.port=1999

5. Health Centerクライアントからモニタリング対象のJavaアプリケーションへの接続

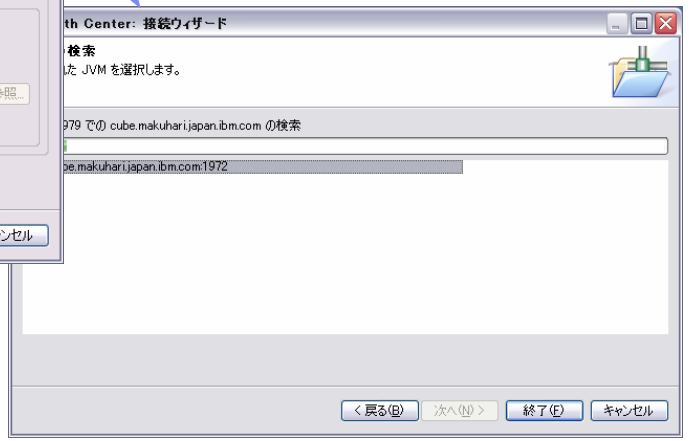
- Health Center接続ウィザードから、サーバへ接続します



- デフォルトのポート番号は 1972 です
- 「次の 100 個のポートで使用可能な接続があるかどうかスキャンします」を選択するとHealth Centerのエージェントが待機している可能性があるポートをスキャンしますので、ポート番号がわかっている場合はチェックをはずします



- 検出されたホスト:ポートを選択して「終了」ボタンをクリックします



V2.0～

- クライアントとエージェント間の通信をSSLで暗号化することが可能です
（設定方法は次項参照）

5. Health Centerクライアントからモニタリング対象のJavaアプリケーションへの接続

V2.0～

SSL接続の設定方法

1. エージェントの鍵ストアと鍵ペアを作成して、公開鍵をエクスポートする
   #<WAS_ROOT>/java/bin/keytool -keystore HCAgentKeystore -genkey -keyalg RSA -keysize 2048 -validity 200 -alias HCAgentKey
   #<WAS_ROOT>/java/bin/keytool -keystore HCAgentKeystore -export -alias HCAgentKey -rfc -file HCAgentKey.pub

2. クライアントの鍵ストアと鍵ペアを作成して、公開鍵をエクスポートする
   #<WAS_ROOT>/java/bin/keytool -keystore HCClientKeystore -genkey -keyalg RSA -keysize 2048 -validity 200 -alias HCClientKey
   #<WAS_ROOT>/java/bin/keytool -keystore HCClientKeystore -export -alias HCClientKey -rfc -file HCClientKey.pub

3. 公開鍵を交換する
   #<WAS_ROOT>/java/bin/keytool -keystore HCAgentKeystore -import -file HCClientKey.pub
   #<WAS_ROOT>/java/bin/keytool -keystore HCClientKeystore -import -file HCAgentKey.pub

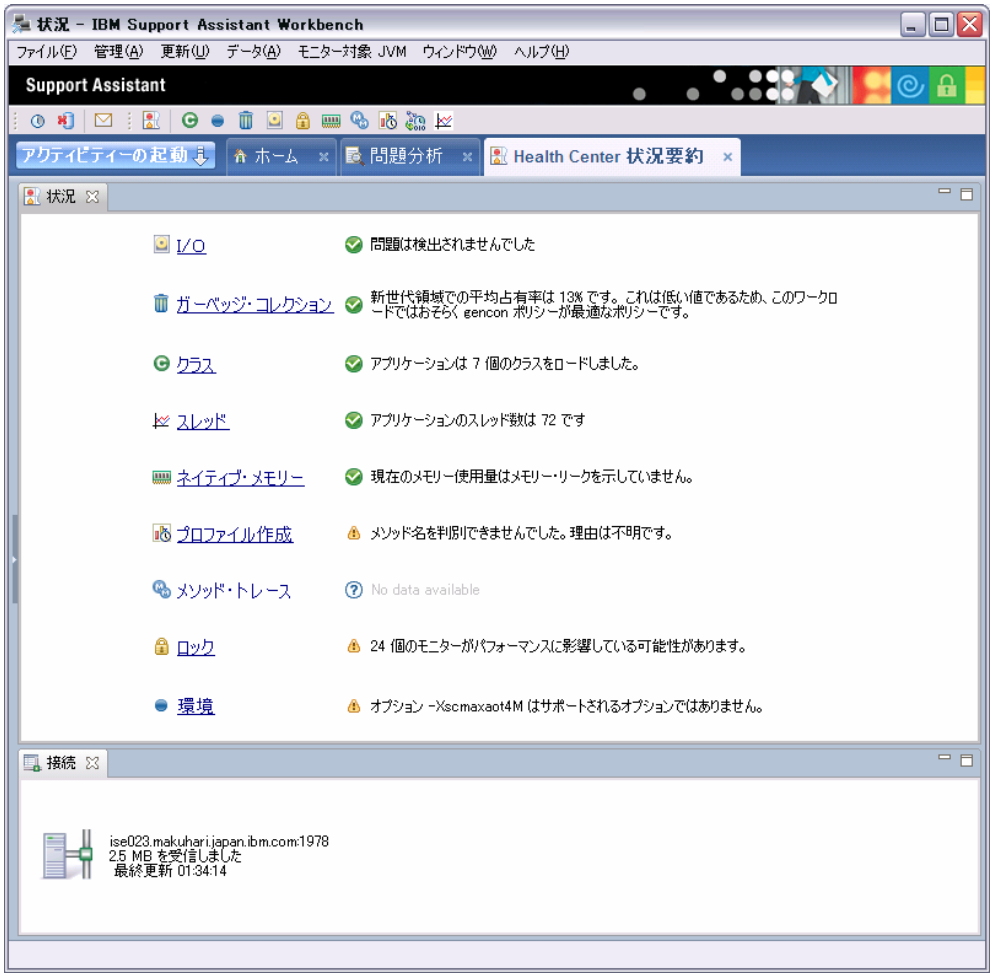
4. クライアントの鍵ストア(HCClientKeystore)をクライアントの任意のディレクトリにコピーする

5.エージェント側の設定
   <WAS_ROOT>/java/jre/lib/healthcenter.propertiesにエージェント鍵ストアの場所とパスワードを指定する
   (javaコマンドにて-D オプションを使用して設定することも可能)
   com.ibm.java.diagnostics.healthcenter.agent.ssl.keyStore=/work/HCAgentKeystore
   com.ibm.java.diagnostics.healthcenter.agent.ssl.keyStorePassword=XXXXXX

6.クライアント側の設定
   Health Center接続ウィザードでSSL接続を選択し、クライアントの鍵ストアとパスワードを指定して接続する

5. Health Centerクライアントからモニタリング対象のJavaアプリケーションへの接続

- 接続が完了すると、「状況要約」ページが表示されます



Health Center の一般設定1

- メニューの「ファイル」⇒「プリファレンス」⇒「IBM Monitoring and Diagnostic Tools for Java – Health Center」から、HealthCenter の一般設定ができます

Health Center の一般設定2

FAQ

Q. Health Centerのサポートはどのようになっていますか？

A. 例えば、WASのJVM問題判別のためにHealth Centerを使用している場合など、パスポート・アドバンテージの契約があればサポートを受けることが可能です。WAS以外のJVMを対象にしている場合でもパスポート・アドバンテージの契約があれば相談することが可能です。契約がない場合は、Health CenterのForumなどに投稿することで情報を得られます。

IBM Monitoring and Diagnostic Tools for Java™ - Health Center Forum

http://www.ibm.com/developerworks/forums/forum.jspa?forumID=1461

Q. 過去のデータを分析することは可能ですか？

A. はい、可能です。モニターデータを保存しておくことで過去のデータを分析可能です。詳細はP31を参照してください。ヘッドレスモードを使用している場合はサーバー側にデータが保管されています。

Q. システムの制約上、Health Centerクライアントをエージェントに接続することができない場合の対応方法は？

A. P32に記載しているヘッドレスモードを使用します。ヘッドレスモードであれば、クライアント接続が不要で、サーバー側でデータを収集することが可能です。

Q.あるシステムにHealthCenterが入っているかどうか？どのバージョンが入っているか？について、どのように確認すればよいでしょうか？

A. 対象となるjavaのディレクトリにhealthcenter.jarが存在しているか確認してください。存在すればエージェントが入っています。healthcenter.jarを展開するとversion.propertiesというファイルがあります。このファイルにバージョンが明記されています。

参考資料

- **Health Center Homepage**
  - https://www.ibm.com/developerworks/java/jdk/tools/healthcenter/
- **Information Center : IBM Monitoring and Diagnostic Tools for Java**
  - http://publib.boulder.ibm.com/infocenter/hctool/v1r0/topic/com.ibm.java.diagnostics.healthcenter.doc/homepage/plugin-homepage-hc.html
- **IBM Monitoring and Diagnostic Tools for Java - Getting started with Health Center**
  - https://www.ibm.com/developerworks/java/jdk/tools/healthcenter/getting_started.html
- **IBM Monitoring and Diagnostic Tools for Java™ - Health Center Forum**
  - http://www.ibm.com/developerworks/forums/forum.jspa?forumID=1461
- **Java の診断を IBM スタイルで: 第 5 回 Health Center によってアプリケーションを最適化する**
  - http://www.ibm.com/developerworks/jp/java/library/j-ibmtools5/index.html?ca=drs-
- **Java Health Center- a low overhead monitoring tool**
  - http://www-01.ibm.com/support/docview.wss?uid=swg21413628
- **Webcast replay: Low overhead performance monitoring for your JVM with IBM Monitoring and Diagnostic Tools for Java - Health Center**
  - http://www-01.ibm.com/support/docview.wss?uid=swg27016069
- **IBM Support Assistant 4.1 利用ガイド**
  - http://www.ibm.com/developerworks/jp/websphere/library/was/isa41_guide/
- **WAS V7 最新動向ワークショップ「システム運用」**
  - http://public.dhe.ibm.com/software/dw/jp/websphere/was/was7_update/wasv7updatews06systemmanagement_rv.pdf
- **Update: Health Center 2.0 provides several major enhancements**
  - http://www-304.ibm.com/support/docview.wss?uid=swg21573878